Lecture 2

Course information

- 20pts, lots of work
- Think of highly condensed lectures that you need to supplement by a lot of reading
- Goal of lectures: tell you about what's hot in foundational research on data management (some of it will be reflected in products soon, some already is)
- Goal of your work: make sure you can follow and understand what's hot
- Way to show it: read papers, present their summaries, and in one case show that really understand everything (project)

Course info cont'd

- Guidelines for essays: 5-7 pages, should present a summary of the paper understandable to someone who has not read the paper. Key definitions must be complete. Cut-and-paste should be limited, it is *your* essay. Crucially it must have *your thoughts* and/or *your analysis*
- Guidelines for projects: same but more detail and a piece of your own work. Possibilities are:
  - Extending some of the results in the paper you read
  - Implementing an algorithm and analyzing performance
  - Discovering special cases, heuristics, etc that improve solutions presented in the paper.
  - The list is not exhaustive.

Big data challenges

Big data is big. But not only. One speaks of 4Vs:

- Volume – size does matter
- Variety – many data formats
- Veracity – data is often incomplete/inconsistent
- Velocity – data often arrives at fast speed, updates are frequent

Volume challenges

- If data is big, many standard algorithms for data processing don't scale
- If data is VERY BIG, we may not even have what can realistically be called an algorithm
  - Classical assumption: data must be at least scanned
  - Hence the best case is $O(n)$ — linear time
  - But take a linear scan on the best available device (around 6GB/s)
  - 1PB is scanned in about 2 days
  - 1EB is scanned in over 5 years
  - We have PB data sets, and EB data sets are not far away

Approaches

- Scale independence: find queries that can be answered regardless of scale
- Approximation
- Parallelization

Scale Independence

Scale independence: desiderata

- We need *sublinear* algorithms: those that run faster than scanning the input
- How can one achieve this? Two ways.
- First way: probabilistic guarantees.
  - e.g., the average number of friends in a social network graph can be found in time $O(\sqrt{n})$, up to a factor arbitrarily close to 2. More precisely, we can estimate it up to a factor of $2 + \varepsilon$ using a randomized algorithm running in time $O(\sqrt{n}/\varepsilon)$.
  - But typical database queries are different, e.g. 'Find friends of John who live in London and like the same restaurants'
- Second way: use auxiliary information about data. Should not be too surprising: indexing! But what can it be?

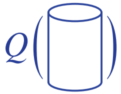
Query answering on big data

Queries can be **slow** on big data due the **size of the data**:

$Q($⛁$)$

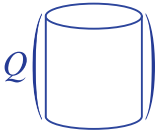
Query answering on big data

Queries can be **slow** on big data due the **size of the data**:



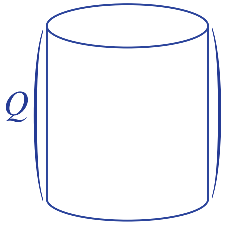
Query answering on big data

Queries can be **slow** on big data due the **size of the data**:



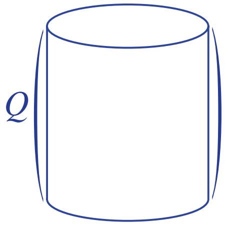
Query answering on big data

Queries can be **slow** on big data due the **size of the data**:



Query answering on big data

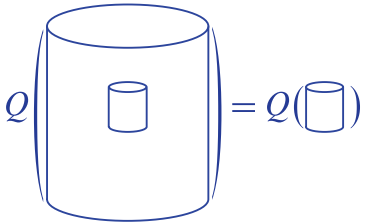
Queries can be **slow** on big data due the **size of the data**:



Current database technology tells us how to **quickly** answer queries on **“normal” sized data**.

Query answering on big data

Queries can be **slow** on big data due the **size of the data**:



Current database technology tells us how to **quickly** answer queries on **"normal" sized data**.

Wouldn't it be great if we can answer Q on a **big database** using a **small database** inside it?

Can this be done?

Can this be done? Yes!

Armbrust et al. considered the notion of **scale independence**:

> *"The evaluation of queries using a number of "operations" that is* **independent** *of the size of data."*

- M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh. Scads: Scale-independent storage for social computing applications. In CIDR, 2009.
- M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: Success-tolerant query processing in the cloud. In VLDB, 2011.
- M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. Patterson. Generalized scale independence through incremental precomputation. In SIGMOD, 2013.

Scale independence: Example

Relation $\text{person}(\text{id}, \text{name}, \text{city})$

- key constraint: id is a key of person relation.

Relation $\text{friend}(\text{id}_1, \text{id}_2)$ (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\text{p}, \text{name}) = \exists \text{id}(\text{friend}(\text{p}, \text{id}) \wedge \text{person}(\text{id}, \text{name}, \text{NYC})).$$

Scale independence: Example

Relation person(id, name, city)

- key constraint: id is a key of person relation.

Relation friend(id_1, id_2) (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\mathrm{p}, \mathrm{name}) = \exists \mathrm{id}(\mathrm{friend}(\mathrm{p}, \mathrm{id}) \wedge \mathrm{person}(\mathrm{id}, \mathrm{name}, \mathrm{NYC})).$$

For a given person p_0

Scale independence: Example

Relation person(id, name, city)

- key constraint: id is a key of person relation.

Relation friend(id_1, id_2) (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\text{p}, \text{name}) = \exists \text{id}(\text{friend}(\text{p}, \text{id}) \wedge \text{person}(\text{id}, \text{name}, \text{NYC})).$$

For a given person p_0 there at most 5000 friends

at most 5 000 tuples needed from friends

Scale independence: Example

Relation $\mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{city})$

- key constraint: id is a key of person relation.

Relation $\mathsf{friend}(\mathsf{id}_1, \mathsf{id}_2)$ (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id}(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathrm{NYC})).$$

For a given person p_0 there at most 5000 friends

For each such friend f

at most 5 000 tuples needed from friends

Scale independence: Example

Relation $\mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{city})$

- key constraint: id is a key of person relation.

Relation $\mathsf{friend}(\mathsf{id}_1, \mathsf{id}_2)$ (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id}(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathrm{NYC})).$$

For a given person p_0 there at most 5000 friends

For each such friend f we have a unique name.

at most 5 000 tuples needed from friends + at most 1 tuple from person =

Scale independence: Example

Relation $\mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{city})$

- key constraint: id is a key of person relation.

Relation $\mathsf{friend}(\mathsf{id}_1, \mathsf{id}_2)$ (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id}(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathrm{NYC})).$$

For a given person p_0 there at most 5000 friends

For each such friend f we have a unique name.

at most 5 000 tuples needed from friends + at most 1 tuple from person =

10, 000 tuples in total are needed.

Scale independence: Example

Relation person(id, name, city)

- key constraint: id is a key of person relation.

Relation friend(id_1, id_2) (Facebook graph)

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (aka Facebook constraint).

Query:

$$Q(\text{p}, \text{name}) = \exists \text{id}(\text{friend}(\text{p}, \text{id}) \wedge \text{person}(\text{id}, \text{name}, \text{NYC})).$$

For a given person p_0 there at most 5000 friends

For each such friend f we have a unique name.

at most 5 000 tuples needed from friends + at most 1 tuple from person =
10, 000 tuples in total are needed.

For a **given** person, this query can be answered with a **bounded** number of tuples, **independent** of the size of the Facebook graph.

Scale independent queries: Definition

A query Q is scale independent in a database D if

- $Q(D) = Q(D_Q)$ for some **part** $D_Q \subseteq D$; and
- the size $|D_Q|$ of D_Q is **independent** of the size $|D|$ of D.

A query Q is scale independent if it is scale independent in **all** databases.

Checking scale independence

Let's start with some bad news first:

- For first-order queries: **undecidable**.
  - This is even true for Boolean FO queries.
  - The class of scale independent FO queries is not even recursively enumerable.
- For CQ/UCQ queries: are **never** scale independent, except when they are **trivial**.
  - This is due to monotonicity.
  - Example of trivial query: return a constant tuple on all database instances.

Checking scale independence on given database

Intractable or high complexity, except for some special cases.

$\mathcal{L}$	Data selecting		Boolean	
	combined	data	combined	data
CQ, UCQ	Σ_3^p-c	NP-c	O(1)-time	O(1)-time
FO	PSPACE-c	NP-c	PSPACE-c	NP-c
	Special case: when M is a constant			
	combined	data	combined	data
CQ, UCQ	Π_2^p-c	PTIME	O(1)-time	O(1)-time
FO	PSPACE-c	NP-c	PSPACE-c	PTIME

data complexity: schema and query are fixed, database and M may vary.

combined complexity: nothing is fixed.

Is scale independent a lost cause?

Let us reconsider the Facebook example.

Facebook example revisited

Relation $\mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{city})$

- key constraint: id is a key of person relation.

Relation $\mathsf{friend}(\mathsf{id}_1, \mathsf{id}_2)$

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation.

Query: $Q(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id} \big(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{NYC}) \big)$.

Scale independence was a result of the **interaction** between

- the **query** and
- **information about accessing the data** (constraints).

Facebook example revisited

Relation $\mathrm{person}(\mathrm{id}, \mathrm{name}, \mathrm{city})$

- key constraint: id is a key of person relation.

Relation $\mathrm{friend}(\mathrm{id}_1, \mathrm{id}_2)$

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation.

Query: $Q(\mathrm{p}, \mathrm{name}) = \exists \mathrm{id}\big(\mathrm{friend}(\mathrm{p}, \mathrm{id}) \wedge \mathrm{person}(\mathrm{id}, \mathrm{name}, \mathrm{NYC})\big)$.

Scale independence was a result of the **interaction** between

- the **query** and
- **information about accessing the data** (constraints).

How to specify information about access to the data? How to marry this with scale independence?

Access Schemas: Definition

For a relational schema $\mathcal{R} = (R_1, \dots, R_n)$, an access schema $\mathcal{A}$ over $\mathcal{R}$ is a set of tuples (R, X, N, T) where

- R is a relation name in $\mathcal{R}$,
- X is a set of attributes of R, and
- $N, T \in \mathbb{N}$.

A database D conforms to the access schema $\mathcal{A}$ if for each $(R, X, N, T) \in \mathcal{A}$:

- **Size bound**: for each tuple of values $\bar{a}$ of attributes of X, the set $\sigma_{X=\bar{a}}(D)$ has at most N tuples; and
- **Time bound**: $\sigma_{X=\bar{a}}(D)$ can be retrieved in time at most T.

Access Schemas: Example

Relation $\text{person}(\text{id}, \text{name}, \text{city})$

- key constraint: id is a key of person relation (**size bound**).
- suppose it takes time T_1 to retrieve a tuple based on its key value (**time bound**).

Relation $\text{friend}(\text{id}_1, \text{id}_2)$

- cardinality constraint: each user id_1 has at most 5 000 friends in friend relation (**size bound**).
- suppose it takes time T_2 to retrieve those $5\,000$ tuples (**time bound**).

Access schema: $\mathcal{A} = \{(\text{person}, \text{id}, 1, T_1), (\text{friend}, \text{id}_1, 5000, T_2)\}$.

Facebook Example revisited (once more)

Access schema $\mathcal{A} = \{(\mathsf{person}, \mathsf{id}, 1, T_1), (\mathsf{friend}, \mathsf{id}_1, 5000, T_2)\}$.

Query: $Q(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id}\big(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{NYC})\big)$.

For a given person, this query is scale independent based on the **size bounds** in the access schema.

- Only 10 000 tuples were needed from the database.

Furthermore, based on the **time bounds** (T_1 and T_2) in the access schema, we know that it takes $O(T_1 \cdot T_2)$ time to answer the query.

Facebook Example revisited (once more)

Access schema $\mathcal{A} = \{(\text{person}, \text{id}, 1, T_1), (\text{friend}, \text{id}_1, 5000, T_2)\}$.

Query: $Q(\text{p}, \text{name}) = \exists \text{id}\big(\text{friend}(\text{p}, \text{id}) \wedge \text{person}(\text{id}, \text{name}, \text{NYC})\big)$.

For a given person, this query is scale independent based on the **size bounds** in the access schema.

- Only 10 000 tuples were needed from the database.

Furthermore, based on the **time bounds** (T_1 and T_2) in the access schema, we know that it takes $O(T_1 \cdot T_2)$ time to answer the query.

By **only looking at the access schema** we can tell whether we can **efficiently answer** this query.

Scale-independence under access schemas

This motivates the following definition:

When $Q(\bar{x}, \bar{y})$ is fixed but the access schema $\mathcal{A}$ may vary, we say that

$Q(\bar{x}, \bar{y})$ is **efficiently** $\bar{x}$-scale-independent under $\mathcal{A}$

if the time to answer $Q(\bar{a}, D)$ is **polynomial** in $\mathcal{A}$, on all databases that conform to $\mathcal{A}$ and all instantiations $\bar{a}$ of $\bar{x}$.

Scale-independence under access schemas

This motivates the following definition:

When $Q(\bar{x}, \bar{y})$ is fixed but the access schema $\mathcal{A}$ may vary, we say that

$Q(\bar{x}, \bar{y})$ is **efficiently** $\bar{x}$-scale-independent under $\mathcal{A}$

if the time to answer $Q(\bar{a}, D)$ is **polynomial** in $\mathcal{A}$, on all databases that conform to $\mathcal{A}$ and all instantiations $\bar{a}$ of $\bar{x}$.

In other words, the time to evaluate $Q(\bar{a}, D)$ is **independent** of D!

The Facebook example query is efficiently person-scale-independent under $\mathcal{A} = \{(\mathrm{person}, \mathrm{id}, 1, T_1), (\mathrm{friend}, \mathrm{id}_1, 5000, T_2)\}$.

It also suggests that building on **index** on $\bar{x}$-attributes may be a good thing!

Can we characterise such queries?

Can we characterise such queries? No

This class of queries is not recursively enumerable.

Instead we provide a **syntactic class** of queries that

- is **sufficiently rich** to cover interesting queries; and
- **guarantees** that the queries are **efficiently scale independent**.

Can we characterise such queries? No

This class of queries is not recursively enumerable.

Instead we provide a **syntactic class** of queries that

- is **sufficiently rich** to cover interesting queries; and
- **guarantees** that the queries are **efficiently scale independent**.

Syntactic class: $\bar{x}$-**controllable** queries.

$\bar{x}$-controllability $\Longrightarrow$ efficiently $\bar{x}$-scale independence $\Longrightarrow$ efficient query answering on big data.

Example

Query: "Find all restaurants in NYC that are rated A and were visited in a given year yy_0 by a given person p_0 friends who lived in NYC"

$$
\begin{aligned}
Q(\mathrm{rn}, \mathrm{p}_0, \mathrm{yy}_0) = \exists \mathrm{id}, \mathrm{rid}, \mathrm{pn}, \mathrm{mm}, \mathrm{dd} \big(\mathrm{friend}(\mathrm{p}_0, \mathrm{id}) \\
\wedge \mathrm{visit}(\mathrm{id}, \mathrm{rid}, \mathrm{yy}_0, \mathrm{mm}, \mathrm{dd}) \wedge \ \mathrm{person}(\mathrm{id}, \mathrm{pn}, \mathrm{NYC}) \\
\wedge \mathrm{dates}(\mathrm{yy}_0, \mathrm{mm}, \mathrm{dd}) \wedge \ \mathrm{restr}(\mathrm{rid}, \mathrm{rn}, \mathrm{NYC}, A)\big).
\end{aligned}
$$

Access schema $\mathcal{A}$:

- $(\mathrm{person}, \mathrm{id}, 1, T_1)$ and $(\mathrm{friend}, \mathrm{p}, 5000, T_2)$;
- $(\mathrm{visit}, \{\mathrm{id}, \mathrm{yy}, \mathrm{dd}, \mathrm{mm}\}, 1, T_3)$: person visits only one restaurant a day;
- $(\mathrm{dates}, \mathrm{yy}, 366, T_4)$: a year consists of at most 366 years.
- $(\mathrm{restr}, \mathrm{rid}, 1, T_5)$: rid is a key.

One can check whether Q is $\{\mathrm{p}, \mathrm{yy}\}$-**controlled** based on the structure of the query and by applying some $\bar{x}$-**controllability rules**.

Example

Query: "Find all restaurants in NYC that are rated A and were visited in a given year $\mathsf{yy_0}$ by a given person $\mathsf{p_0}$ friends who lived in NYC"

$$\begin{aligned} Q(\mathsf{rn}, \mathsf{p_0}, \mathsf{yy_0}) = \exists \mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}, \mathsf{dd} \big(& \mathsf{friend}(\mathsf{p_0}, \mathsf{id}) \\ & \wedge \mathsf{visit}(\mathsf{id}, \mathsf{rid}, \mathsf{yy_0}, \mathsf{mm}, \mathsf{dd}) \wedge \ \mathsf{person}(\mathsf{id}, \mathsf{pn}, \mathsf{NYC}) \\ & \wedge \mathsf{dates}(\mathsf{yy_0}, \mathsf{mm}, \mathsf{dd}) \wedge \ \mathsf{restr}(\mathsf{rid}, \mathsf{rn}, \mathsf{NYC}, A)\big). \end{aligned}$$

Access schema $\mathcal{A}$:

- $(\mathsf{person}, \mathsf{id}, 1, T_1)$ and $(\mathsf{friend}, \mathsf{p}, 5000, T_2)$;
- $(\mathsf{visit}, \{\mathsf{id}, \mathsf{yy}, \mathsf{dd}, \mathsf{mm}\}, 1, T_3)$: person visits only one restaurant a day;
- $(\mathsf{dates}, \mathsf{yy}, 366, T_4)$: a year consists of at most 366 years.
- $(\mathsf{restr}, \mathsf{rid}, 1, T_5)$: rid is a key.

One can check whether Q is $\{\mathsf{p}, \mathsf{yy}\}$-**controlled** based on the structure of the query and by applying some $\bar{x}$-**controllability rules**.

$\bar{x}$-controllability: Atom Rule

If (R, X, N, T) is in $\mathcal{A}$, then $R(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$, where $\bar{x}$ is the subtuple of $\bar{y}$ corresponding to attributes in X.

For example

Atom	Access schema	controlling variables
friend(p, id)	(friend, p, 5000, T_2)	{p}
visit(id, rid, yy, mm, dd)	(visit, {id, yy, dd, mm}, 1, T_3)	{id, yy, dd, mm}
person(id, pn, NYC)	(person, id, 1, T_1)	{id}
dates(yy, mm, dd)	(dates, yy, 366, T_4)	{yy}
restr(rid, rn, NYC, A)	(restr, rid, 1, T_5)	{rid}

We write this as

friend($\underline{\text{p}}$, id), visit($\underline{\text{id}}$, rid, $\underline{\text{yy, mm, dd}}$), person($\underline{\text{id}}$, pn, NYC), dates($\underline{\text{yy}}$, mm, dd) and restr($\underline{\text{rid}}$, rn, NYC, A)

$\bar{x}$-controllability: Atom Rule

If (R, X, N, T) is in $\mathcal{A}$, then $R(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$, where $\bar{x}$ is the subtuple of $\bar{y}$ corresponding to attributes in X.

For example

Atom	Access schema	controlling variables
$\text{friend}(\text{p}, \text{id})$	$(\text{friend}, \text{p}, 5000, T_2)$	$\{\text{p}\}$
$\text{visit}(\text{id}, \text{rid}, \text{yy}, \text{mm}, \text{dd})$	$(\text{visit}, \{\text{id}, \text{yy}, \text{dd}, \text{mm}\}, 1, T_3)$	$\{\text{id}, \text{yy}, \text{dd}, \text{mm}\}$
$\text{person}(\text{id}, \text{pn}, \text{NYC})$	$(\text{person}, \text{id}, 1, T_1)$	$\{\text{id}\}$
$\text{dates}(\text{yy}, \text{mm}, \text{dd})$	$(\text{dates}, \text{yy}, 366, T_4)$	$\{\text{yy}\}$
$\text{restr}(\text{rid}, \text{rn}, \text{NYC}, A)$	$(\text{restr}, \text{rid}, 1, T_5)$	$\{\text{rid}\}$

We write this as

$\text{friend}(\underline{\text{p}}, \text{id})$, $\text{visit}(\underline{\text{id}}, \text{rid}, \underline{\text{yy}, \text{mm}, \text{dd}})$, $\text{person}(\underline{\text{id}}, \text{pn}, \text{NYC})$, $\text{dates}(\underline{\text{yy}}, \text{mm}, \text{dd})$ and $\text{restr}(\underline{\text{rid}}, \text{rn}, \text{NYC}, A)$

$\bar{x}$-controllability: Conjunction rule

If $Q_i(\bar{x}_i, \bar{y}_i)$ is $\bar{x}_i$-controlled under $\mathcal{A}$ for $i = 1, 2$, then $Q_1 \wedge Q_2$ is controlled under $\mathcal{A}$ by both $\bar{x}_1 \cup (\bar{x}_2 - \bar{y}_1)$ and $\bar{x}_2 \cup (\bar{x}_1 - \bar{y}_2)$.

$$\underbrace{\mathrm{visit}(\underline{\mathrm{id}}, \mathrm{rid}, \underline{\mathrm{yy}, \mathrm{mm}, \mathrm{dd}}) \wedge \mathrm{dates}(\underline{\mathrm{yy}}, \mathrm{mm}, \mathrm{dd})}_{e_1}$$

Controlling variables: $\{\mathrm{yy}\} \cup (\{\mathrm{id}, \mathrm{yy}, \mathrm{mm}, \mathrm{dd}\} \setminus \{\mathrm{mm}, \mathrm{dd}\}) = \{\mathrm{id}, \mathrm{yy}\}$
(or $\{\mathrm{id}, \mathrm{yy}, \mathrm{mm}, \mathrm{dd}\} \cup (\{\mathrm{yy}\} \setminus \{\mathrm{rid}\}) = \{\mathrm{id}, \mathrm{yy}, \mathrm{mm}, \mathrm{dd}\}$.)

$$\underbrace{e_1(\underline{\mathrm{id}}, \mathrm{rid}, \underline{\mathrm{yy}}, \mathrm{mm}, \mathrm{dd}) \wedge \mathrm{person}(\underline{\mathrm{id}}, \mathrm{pn}, \mathrm{NYC})}_{e_2}$$

Controlling variables: $\{\mathrm{id}, \mathrm{yy}\}$

$\bar{x}$-controllability: Conjunction rule

If $Q_i(\bar{x}_i, \bar{y}_i)$ is $\bar{x}_i$-controlled under $\mathcal{A}$ for $i = 1, 2$, then $Q_1 \wedge Q_2$ is controlled under $\mathcal{A}$ by both $\bar{x}_1 \cup (\bar{x}_2 - \bar{y}_1)$ and $\bar{x}_2 \cup (\bar{x}_1 - \bar{y}_2)$.

$$\underbrace{\mathrm{visit}(\underline{\mathrm{id}}, \mathrm{rid}, \underline{\mathrm{yy}, \mathrm{mm}, \mathrm{dd}}) \wedge \mathrm{dates}(\underline{\mathrm{yy}}, \mathrm{mm}, \mathrm{dd})}_{e_1}$$

Controlling variables: $\{\mathrm{yy}\} \cup (\{\mathrm{id}, \mathrm{yy}, \mathrm{mm}, \mathrm{dd}\} \setminus \{\mathrm{mm}, \mathrm{dd}\}) = \{\mathrm{id}, \mathrm{yy}\}$
(or $\{\mathrm{id}, \mathrm{yy}, \mathrm{mm}, \mathrm{dd}\} \cup (\{\mathrm{yy}\} \setminus \{\mathrm{rid}\}) = \{\mathrm{id}, \mathrm{yy}, \mathrm{mm}, \mathrm{dd}\}$.)

$$\underbrace{e_1(\underline{\mathrm{id}}, \mathrm{rid}, \underline{\mathrm{yy}}, \mathrm{mm}, \mathrm{dd}) \wedge \mathrm{person}(\underline{\mathrm{id}}, \mathrm{pn}, \mathrm{NYC})}_{e_2}$$

Controlling variables: $\{\mathrm{id}, \mathrm{yy}\}$

$$\underbrace{e_1(\underline{\mathsf{id}}, \mathsf{rid}, \underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd}) \wedge \mathsf{person}(\underline{\mathsf{id}}, \mathsf{pn}, \mathsf{NYC})}_{e_2}$$

Controlling variables: $\{\mathsf{id}, \mathsf{yy}\}$

$$\underbrace{e_2(\underline{\mathsf{id}}, \mathsf{rid}, \underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd}, \mathsf{pn}, \mathsf{NYC}) \wedge \mathsf{friend}(\underline{\mathsf{p}}, \mathsf{id})}_{e_3}$$

Controlling variables: $\{\mathsf{id}, \mathsf{yy}, \mathsf{p}\}$ or $\{\mathsf{p}, \mathsf{yy}\}$

$$\underbrace{e_3(\mathsf{id}, \mathsf{rid}, \underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd}, \mathsf{pn}, \mathsf{NYC}, \underline{\mathsf{p}}) \wedge \mathsf{restr}(\underline{\mathsf{rid}}, \mathsf{rn}, \mathsf{NYC}, A)}_{e_4}$$

Controlling variables: $\{\mathsf{p}, \mathsf{yy}\}$ and $\{\mathsf{p}, \mathsf{yy}, \mathsf{rid}\}$.

Note that
$Q(\mathsf{rn}, \mathsf{p}, \mathsf{yy}) = \exists \mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}, \mathsf{dd}\; e_4(\mathsf{id}, \mathsf{rid}, \underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd}, \mathsf{pn}, \mathsf{NYC}, \underline{\mathsf{p}}, \mathsf{rn}, A)$

$\bar{x}$-controllability: Existential quantification rule

If $Q(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$ and $\bar{z}$ is a subtuple of $\bar{y} - \bar{x}$, then $\exists \bar{z}\ Q$ is $\bar{x}$-controlled under $\mathcal{A}$.

$$Q(\mathsf{rn}, \mathsf{p}, \mathsf{yy}) = \exists \mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}, \mathsf{dd}\ e_4(\mathsf{id}, \mathsf{rid}, \underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd}, \mathsf{pn}, \mathsf{NYC}, \underline{\mathsf{p}}, \mathsf{rn}, A)$$

Observe that $\mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}$ and dd do not occur in the controlling variables $\{\mathsf{p}, \mathsf{yy}\}$ of e_4. Hence, Q's controlling variables are $\{\mathsf{p}, \mathsf{yy}\}$.

$\bar{x}$-controllability: Existential quantification rule

If $Q(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$ and $\bar{z}$ is a subtuple of $\bar{y} - \bar{x}$, then $\exists\bar{z}\ Q$ is $\bar{x}$-controlled under $\mathcal{A}$.

$$Q(\mathsf{rn}, \mathsf{p}, \mathsf{yy}) = \exists \mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}, \mathsf{dd}\ e_4(\mathsf{id}, \mathsf{rid}, \underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd}, \mathsf{pn}, \mathsf{NYC}, \underline{\mathsf{p}}, \mathsf{rn}, A)$$

Observe that $\mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}$ and dd do not occur in the controlling variables $\{\mathsf{p}, \mathsf{yy}\}$ of e_4. Hence, Q's controlling variables are $\{\mathsf{p}, \mathsf{yy}\}$.

Other rules

- Similar rules can be found for other operations:
  - disjunction
  - (safe) negation (i.e., $Q_1 \wedge \neg Q_2$)
  - universal quantification $\forall$
  - change of free variables (viewing $Q(\bar{x})$ as $Q(\bar{x}, \bar{y})$)
- In isolation, these rules are optimal

Main result on $\bar{x}$-controllability

It is a **sufficient condition** for scale independence:

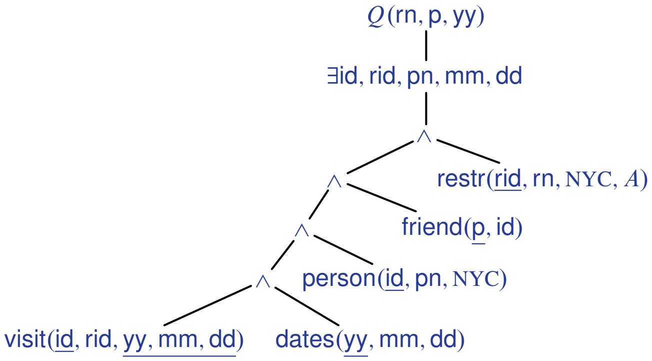
If an FO query Q is $\bar{x}$-controlled under an access schema $\mathcal{A}$, then it is efficiently $\bar{x}$-scale-independent under $\mathcal{A}$.

That is, by **filling the variables** $\bar{x}$ in Q by constants $\bar{a}$, $Q(\bar{a}, \bar{y})$ can be answered on **any** database D that is consistent with $\mathcal{A}$ in **time polynomial** in $\mathcal{A}$.

Furthermore, an **effective plan for identifying** D_Q such that $Q(\bar{,}D_Q) = Q(\bar{a}, D)$ can be obtained.

Example

Recall: Q is $\{\mathsf{p}, \mathsf{yy}\}$-controllable. We can get an effective plan to evaluate Q as follows:

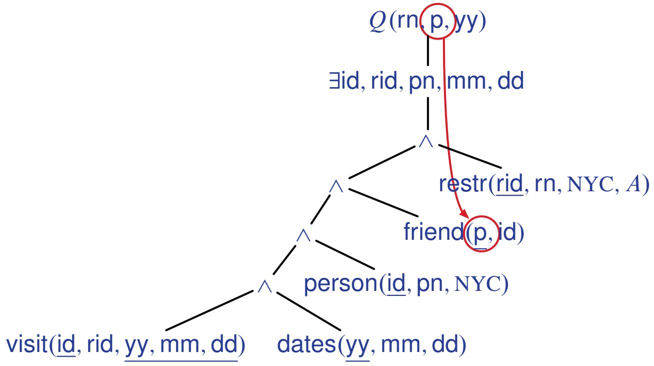


$Q(\mathsf{rn}, \mathsf{p}, \mathsf{yy})$
$\exists \mathsf{id}, \mathsf{rid}, \mathsf{pn}, \mathsf{mm}, \mathsf{dd}$
$\wedge$
$\wedge$
$\mathsf{restr}(\underline{\mathsf{rid}}, \mathsf{rn}, \mathrm{NYC}, A)$
$\wedge$
$\mathsf{friend}(\underline{\mathsf{p}}, \mathsf{id})$
$\wedge$
$\mathsf{person}(\underline{\mathsf{id}}, \mathsf{pn}, \mathrm{NYC})$
$\mathsf{visit}(\underline{\mathsf{id}}, \mathsf{rid}, \underline{\mathsf{yy}, \mathsf{mm}, \mathsf{dd}})$
$\mathsf{dates}(\underline{\mathsf{yy}}, \mathsf{mm}, \mathsf{dd})$

The time it takes to answer Q entirely depends on $\mathcal{A}$.

Example

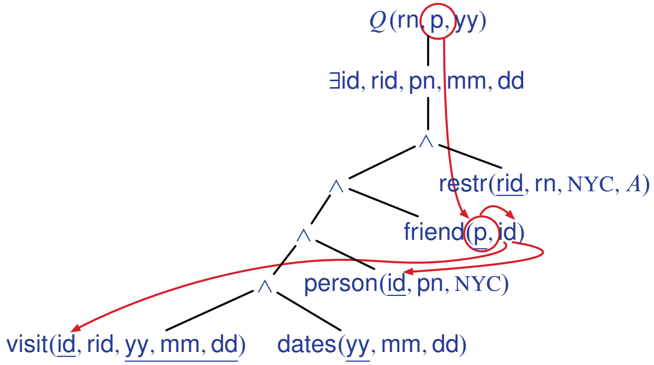
Recall: Q is $\{\text{p}, \text{yy}\}$-controllable. We can get an effective plan to evaluate Q as follows:



The time it takes to answer Q entirely depends on $\mathcal{A}$.

Example

Recall: Q is $\{\text{p}, \text{yy}\}$-controllable. We can get an effective plan to evaluate Q as follows:

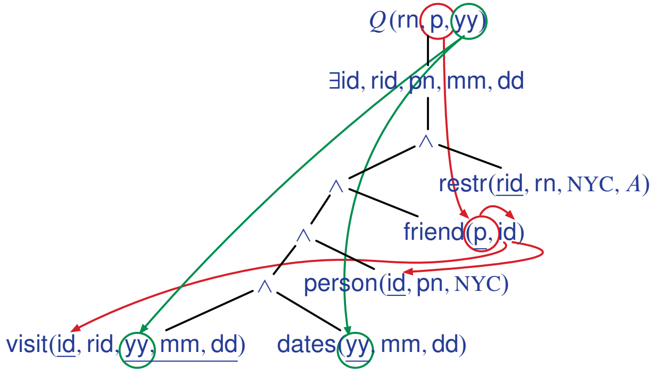


$Q(\text{rn}, \text{p}, \text{yy})$
$\exists \text{id}, \text{rid}, \text{pn}, \text{mm}, \text{dd}$
$\wedge$
$\wedge$
restr(rid, rn, NYC, A)
$\wedge$
friend(p, id)
$\wedge$
person(id, pn, NYC)
visit(id, rid, yy, mm, dd)
dates(yy, mm, dd)

The time it takes to answer Q entirely depends on $\mathcal{A}$.

Example

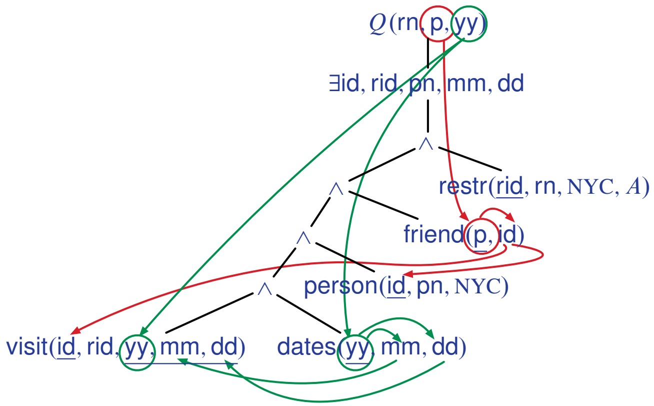
Recall: Q is $\{\mathsf{p}, \mathsf{yy}\}$-controllable. We can get an effective plan to evaluate Q as follows:



The time it takes to answer Q entirely depends on $\mathcal{A}$.

Example

Recall: Q is $\{\mathrm{p}, \mathrm{yy}\}$-controllable. We can get an effective plan to evaluate Q as follows:

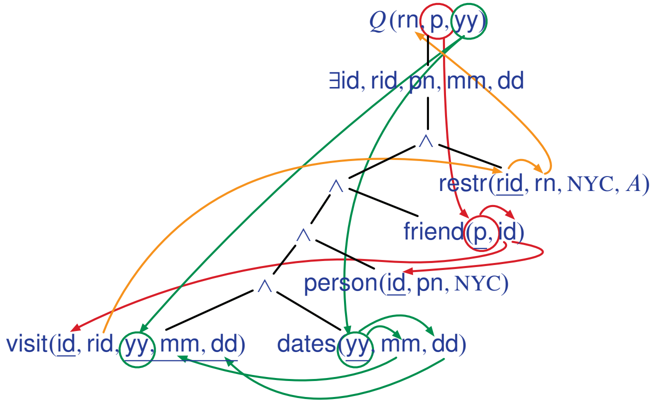


$Q(\mathrm{rn}, \mathrm{p}, \mathrm{yy})$
$\exists \mathrm{id}, \mathrm{rid}, \mathrm{pn}, \mathrm{mm}, \mathrm{dd}$
$\wedge$
$\mathrm{restr}(\underline{\mathrm{rid}}, \mathrm{rn}, \mathrm{NYC}, A)$
$\wedge$
$\mathrm{friend}(\underline{\mathrm{p}}, \mathrm{id})$
$\wedge$
$\mathrm{person}(\underline{\mathrm{id}}, \mathrm{pn}, \mathrm{NYC})$
$\wedge$
$\mathrm{visit}(\underline{\mathrm{id}}, \mathrm{rid}, \underline{\mathrm{yy}, \mathrm{mm}, \mathrm{dd}})$
$\mathrm{dates}(\underline{\mathrm{yy}}, \mathrm{mm}, \mathrm{dd})$

The time it takes to answer Q entirely depends on $\mathcal{A}$.

Example

Recall: Q is $\{\mathsf{p}, \mathsf{yy}\}$-controllable. We can get an effective plan to evaluate Q as follows:



The time it takes to answer Q entirely depends on $\mathcal{A}$.

Approximations

We look at conjunctive queries.

Why?

- because they are important: joins
- and because for them we know when queries can be evaluated fast.

We use this topic to discuss not only approximations but also give an overview of good classes of conjunctive queries

CQ evaluation

- Conjunctive queries (CQs) – probably the best studied class of database queries
    - capture select-project-join queries
- Complexity of evaluation:
    - data: very low (in AC^0)
    - combined: NP-complete
    - algorithmically: $|\mathrm{database}|^{O(|\mathrm{query}|)}$
- Prohibitively expensive for very large databases
    - new applications: scientific databases, social networks, etc may store up to several terabytes of information.
- What do we do when we cannot find an exact solution?
- APPROXIMATE!

Approximation idea

- Idea: Given
    - a database D
    - a query Q
- find an approximation Q' of Q so that:
    - Q' approximates Q well;
    - Q' is much faster to run (e.g., tractable)
- and then run Q' instead of Q

Approximation techniques

The idea is, of course, quite old.

Standard approaches include:

- Statistical methods, e.g. histograms, combined with data mining techniques
    - uses both data and query
- Semantic relaxations based on traditional data management techniques
    - uses only the query (static analysis)

Approximation techniques

The idea is, of course, quite old.

Standard approaches include:

- Statistical methods, e.g. histograms, combined with data mining techniques
  - uses both data and query
- Semantic relaxations based on traditional data management techniques
  - uses only the query (static analysis)

Our choice: static analysis.

Once a query is approximated, it can be run and re-run when updates are applied.

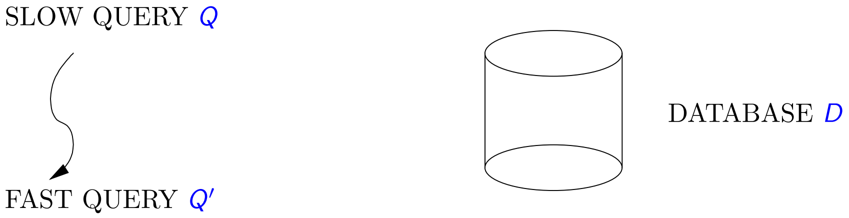
Approximation idea, by picture

SLOW QUERY Q

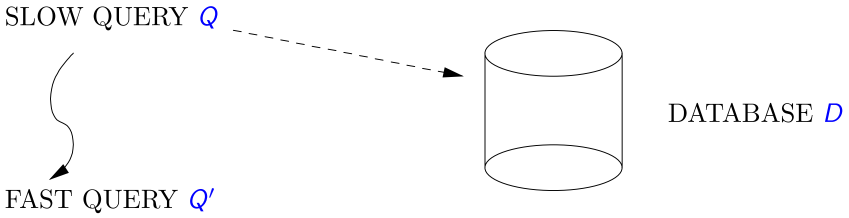
Approximation idea, by picture

$$\begin{tikzcd}
\text{SLOW QUERY } Q \arrow[d, squiggly] \\
\text{FAST QUERY } Q'
\end{tikzcd}$$

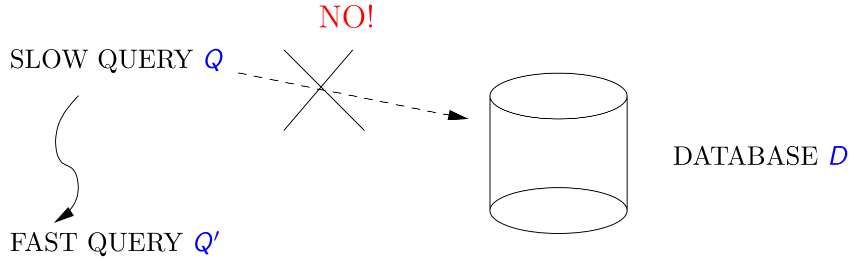
Approximation idea, by picture



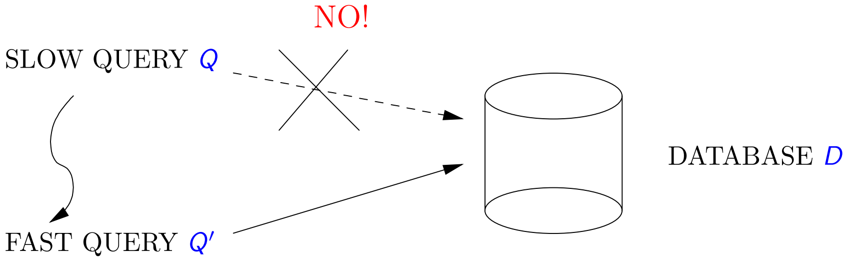
Approximation idea, by picture



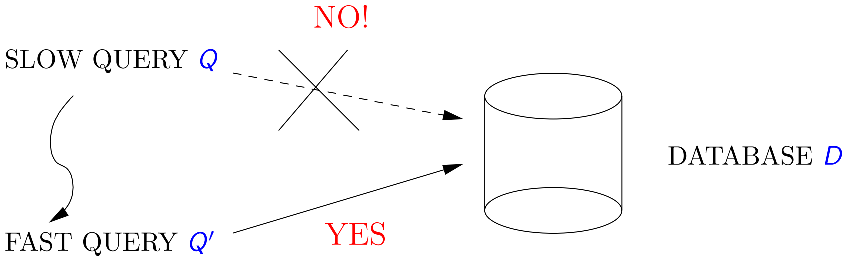
Approximation idea, by picture



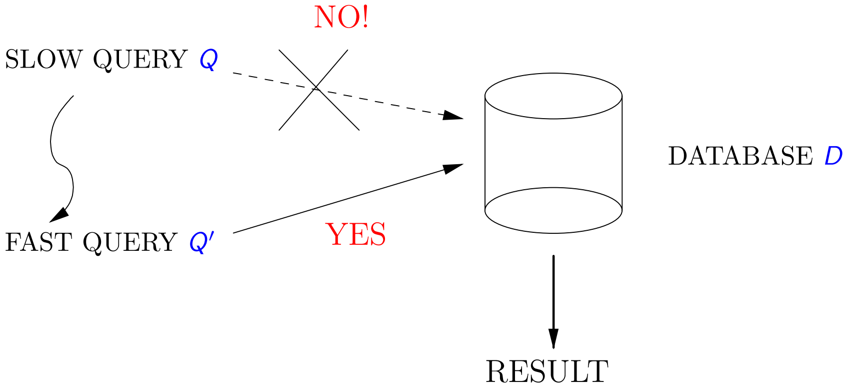
Approximation idea, by picture



Approximation idea, by picture



Approximation idea, by picture



Desiderata for approximations

- They must be fast
- They must be close to the queries they approximate

Next: formulate these two requirements for conjunctive queries.

Conjunctive queries (CQs)

$$\underbrace{Q(\bar{x})}_{\textit{head}} \quad \text{:-} \quad \underbrace{S_1(\bar{u}_1), \ldots S_n(\bar{u}_n)}_{\textit{body}}$$

- atoms $S_i(\bar{u}_i)$: each S_i is a relation symbol, each $\bar{u}_i$ a list of variables.
- variables $\bar{y}$ in the body but not in the head are existentially quantified:
  $Q(\bar{x}) \;=\; \exists y \; \big(S_1(\bar{u}_1) \wedge \cdots \wedge S_n(\bar{u}_n)\big)$.

Conjunctive queries (CQs)

$$\underbrace{Q(\bar{x})}_{\textit{head}} \quad :- \quad \underbrace{S_1(\bar{u}_1), \ldots S_n(\bar{u}_n)}_{\textit{body}}$$

- atoms $S_i(\bar{u}_i)$: each S_i is a relation symbol, each $\bar{u}_i$ a list of variables.
- variables $\bar{y}$ in the body but not in the head are existentially quantified:
  $Q(\bar{x}) \;=\; \exists y \;\big(S_1(\bar{u}_1) \wedge \cdots \wedge S_n(\bar{u}_n)\big)$.
- Why CQs?
  - a very important and common class of queries
  - equivalent to select-project-join queries
  - we know a lot about their evaluation.

Tableaux, evaluation, containment

$$Q(\bar{x}) \quad :- \quad S_1(\bar{u}_1), \ldots S_n(\bar{u}_n)$$

Its tableaux is the body of Q viewed as a database:

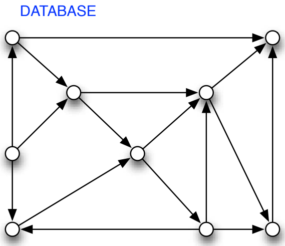
$$T_Q \quad = \quad (\{S_1(\bar{u}_1), \ldots, S_n(\bar{u}_n)\}, \quad \bar{x})$$

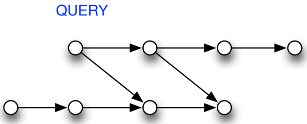
Evaluation and static analysis via tableaux and homomorphisms:

- $\bar{a} \in Q(D)$ $\Leftrightarrow$ there is a homomorphism $T_Q \rightarrow (D, \bar{a})$
- $Q \subseteq Q'$ $\Leftrightarrow$ there is a homomorphism $T_{Q'} \rightarrow T_Q$

CQ evaluation, graphically

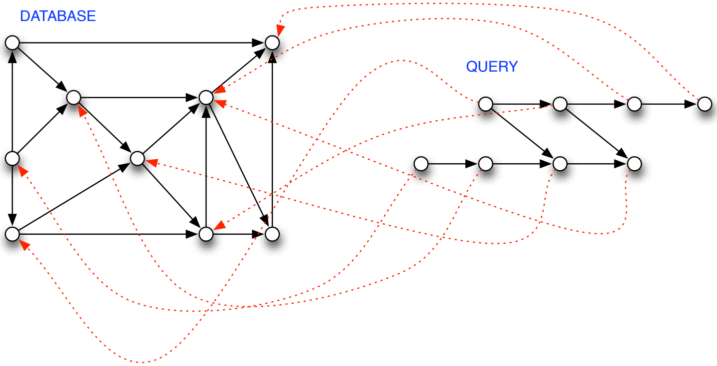
Goal: Find a homomorphism from T_Q into D.





CQ evaluation, graphically

Goal: Find a homomorphism from T_Q into D.

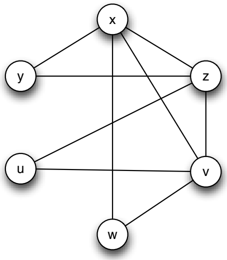


Good classes of CQs

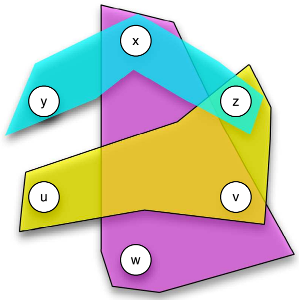
Two kinds: graph-based and hypergraph-based

$$Q \ :\!-\ R(x,y,z), R(z,u,v), R(v,w,x)$$

graph of the query, $G(Q)$



hypergraph of the query, $H(Q)$



Good classes of CQs – acyclicity and relatives

A general idea of tractable restrictions for CQs: acyclicity
(Yannakakis 1981 – linear-time evaluation)

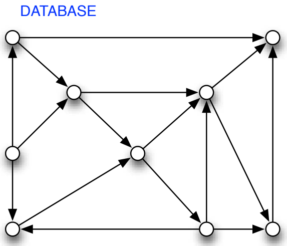
Extensions:

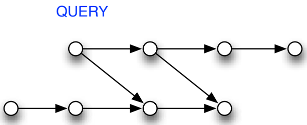
- for graph-based notions: bounded treewidth
- for hypergraph-based notions: bounded hypertree width
- Yannakakis' notion of acyclicity is actually hypertree width 1.
- For queries on graphs, treewidth 1 $=$ acyclicity.

Revised goal: approximate within tractable classes of CQs

Approximating CQs, graphically

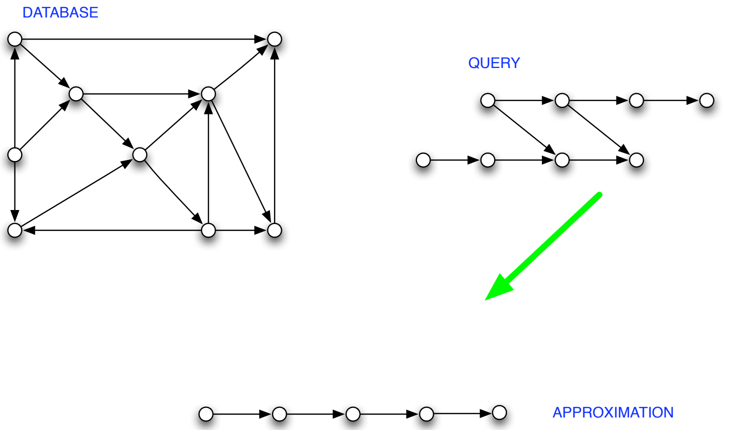
Given Q and D





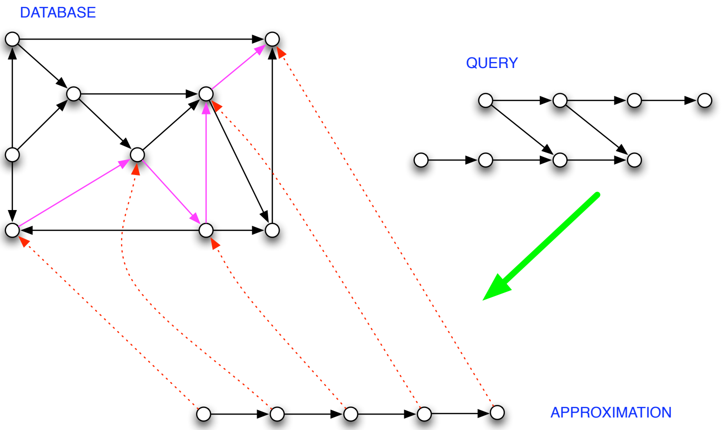
Approximating CQs, graphically

Given Q and D, find a tractable approximation



Approximating CQs, graphically

Given Q and D, find a tractable approximation and evaluate on D



Approximations: complexity analysis

- Evaluate Q on D: $|D|^{O(|Q|)}$
- Evaluate approximation Q' on D:

$$O(\text{time to compute } Q' \; + \; |D|^c \cdot p(|Q'|)$$

  - c is constant; p is polynomial
- Desiderata:
  - Q' at most polynomial in Q
  - time to compute Q' at most single-exponential in Q
  - hard to hope for more given the complexity of static analysis for CQs
- For example, using acyclic approximations we have complexity $2^{O(|Q|\cdot\log|Q|)} + |D| \cdot |Q|^k$
- Much faster than $|D|^{O(|Q|)}$ on large databases.

Approaches to approximations

- Qualitative (or look for best ones): Q' approximates Q if no other query Q'' does it better than Q'.
  - Better = disagrees with Q less often than Q' does.
- Quantitative (or look for good ones): define a measure $\mu(Q, Q')$ of disagreement; search for Q' with $\mu(Q, Q') \leq \text{threshold}$.
- Nobody's perfect:
  - best may not be good;
  - good need not be best.
- We look at qualitative approach.

CQ approximations: Definition

We want to approximate within a class $\mathcal{C}$ of good queries.

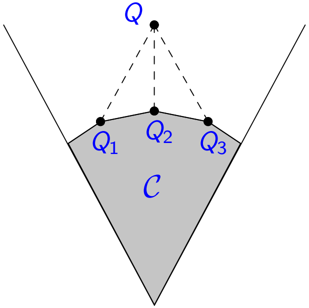
We also want to approximate without producing false results:

if Q' approximates Q then $Q' \subseteq Q$.

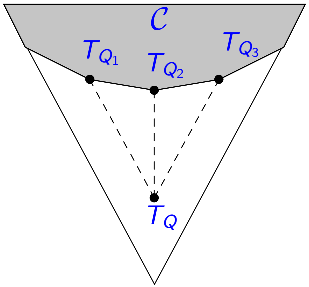
Definition A query $Q' \in \mathcal{C}$ is a $\mathcal{C}$-approximation of Q if:

- $Q' \subseteq Q$; and
- there is no $Q'' \in \mathcal{C}$ such that $Q' \subset Q'' \subseteq Q$.

CQ approximations, graphically



a query view



a tableau view

Why are these orderings important?

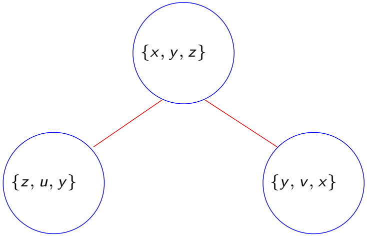
- The homomorphism (pre)ordering between graphs (and structures) has been actively studied in graph theory.
- P. Hell and J. Nešetřil. Graphs and Homomorphisms. Oxford University Press, 2004.
- The key object: lattice of graph cores and homomorphism ordering.
- Our job essentially boils down to finding upper bounds in subsets of that lattice.
- For those of you who love math, very nontrivial proof techniques involving mix of algebra and graph theory.

Hypergraph-based classes: acyclicity

A CQ Q is acyclic if its hypergraph $H(Q)$ is acyclic.

- hyperedges can be arranged in a tree in such a way that the part containing each variable is connected

Example: Decomposition for the query
$Q :– R(x, y, z), R(z, u, y), R(y, v, x)$:



Acyclicity and hypertreewidth

Theorem (Yannakakis 1981)

If Q is an acyclic conjunctive query, then checking whether $\bar{a} \in Q(D)$ can be done in time $O(|D| \cdot |Q|)$.

Generalization: hypertree width (Gottlob, Leone, Scarcello, 2000).

- Extends the notion of treewidth.
- Acyclicity = hypertree width 1.
- Guaranteed tractable evaluation of CQs:
  - Hypertree width determines the exponent.
  - $O(|D|^c \cdot p(|Q|))$.

Treewidth

- Measures how close a graph is to a tree
- A tree decomposition of a graph G with vertices V and edges E is a tree T and a map f from T to sets of vertices (i.e., $f(n) \subseteq V$) such that
  1. each edge is contained in one of $f(n)$
  2. for each $v \in V$, the set $\{n \in T \mid v \in f(v)\}$ is connected
- Width of a tree decomposition: $\max |f(n)|$ where n is a node in T
- Treewidth: minimal width of a tree decomposition, minus 1
- Why "minus 1"? So that tree width of a tree is 1.

Treewidth and tractable CQs

- If $G(Q)$ has treewidth k, then Q can be evaluated in polynomial time on a database D: the complexity is not $|D|^{O(|Q|)}$ but rather $O(|D|^k)$ + time to find a tree decomposition. This time is linear in $|Q|$ if k is fixed (although dependence on k is exponential).
- A class of CQs $\mathcal{C}$ is of bounded treewidth if there is a number k such that the treewidth of $G(Q)$ is $\leq k$ for each Q in $\mathcal{C}$.
- A very powerful Theorem (Grohe-Schwentick-Segoufin): Let $\mathcal{C}$ be a graph-based class of CQs
  - i.e., $\mathcal{C}$ is a class of graphs, and the condition is $G(Q) \in \mathcal{C}$

  If evaluation of CQs in $\mathcal{C}$ is in PTIME, then $\mathcal{C}$ has bounded treewidth.

Treewidth and tractable CQs

Lessons – CQ evaluation is polynomial for:

- Bounded-treewidth CQs for graph-based classes
- Acyclic CQs (and more general bounded hypertree width) for hypergraph-based classes
- Real aim: acyclicity, due to linear time.
- If a linear scan is affordable, we are ok!

Existence of approximations: graph-based case

A class $\mathcal{C}$ of queries is closed under substructures if:

$$Q \in \mathcal{C} \ \text{ and } \ T_{Q'} \subseteq T_Q \implies Q' \in \mathcal{C}$$

Example: classes of queries of bounded treewidth.

Existence theorem

Let $\mathcal{C}$ be closed under substructures.

Theorem

- *Every conjunctive query Q has a $\mathcal{C}$-approximation.*
- *Each approximation is (equivalent to) a query whose size does not exceed the size of Q.*
- *An approximation can be found in single-exponential time*
  - $2^{O(n \log n)}$ *to be precise*
- *There are at most exponentially many non-equivalent approximations.*

Example

- Take

$$Q \coloneq R(x, y, z),\ R(y, x, u),\ R(u, z, x)$$

- Its treewidth is 3 – maximum possible for a query with 4 variables; its graph is K_4.
- Has approximations of the smallest possible treewidth, i.e., 1:

$$Q' \coloneq R(x, y, y),\ R(y, x, y),\ R(y, y, x).$$

- Good to reduce treewidth as it determines the exponent in query evaluation.

Queries on graphs

$$Q :\!-\ R(x\ y\ z),\ R(y\ x\ u),\ R(u\ z\ x)$$

Queries on graphs

$$Q :\!-\ R(\quad y\ z),\ R(y\quad u),\ R(u\ z\quad)$$

Remove x

We get a graph query with $G(Q) = K_3$

This query has only a trivial treewidth 1/acyclic approximation: $Q :\!-\ R(x, x)$.

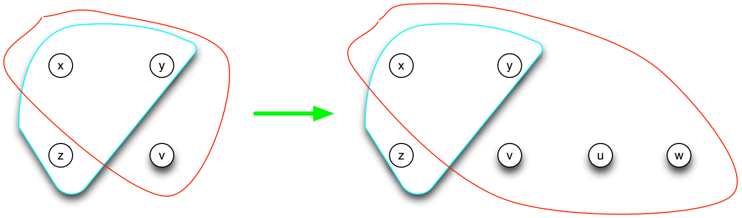
A Boolean query Q over graphs has a nontrivial acyclic approximation iff $G(Q)$ is 2-colorable.

Existence of approximation: closure conditions

Problem: previous existence conditions don't work – even acyclic hypergraphs are not closed under taking sub-hypergraphs.

Instead, we use two new closure conditions:

1. Closure under induced sub-hypergraphs;
2. Closure under edge extensions:



3. Holds for acyclic hypergraphs

Existence of approximations for hypergraph-based classes

$\mathcal{C}$ – a class of CQs whose hypergraphs are closed under induced subhypergraphs and edge extensions:

- for example, queries of hypertree width $\leq k$, or
- acyclic queries.

Theorem

- *Every conjunctive query Q has a $\mathcal{C}$-approximation.*
- *Each approximation is (equivalent to) a query whose size is at most polynomial in the size of Q.*
- *An approximation can be found in single-exponential time*
  - $2^{O(n \log n)}$ *to be precise*
- *There are at most exponentially many non-equivalent approximations.*

Example of acyclic approximations

A cyclic query:

$$Q \;:\!-\; R(x,y,z),\; R(z,u,v),\; R(v,w,x)$$

It has several acyclic approximations:

$$Q_1 \;:\!-\; R(x,y,z),\; R(z,u,y),\; R(y,v,x)$$

$$Q_2 \;:\!-\; R(x,y,z),\; R(z,u,v),\; R(v,w,x),\; R(x,z,v)$$

$$Q_3 \;:\!-\; R(x,y,x)$$

Acyclic approximations: how to choose

Quantitative analysis is needed.

Much work on estimating the size of CQs/joins, but no final answers yet.

Comments on papers

- Michael Armbrust, Kristal Curtis, Tim Kraska, Armando Fox, Michael J. Franklin, David A. Patterson: PIQL: Success-Tolerant Query Processing in the Cloud. PVLDB 5(3):181-192 (2011)
- Michael Armbrust, Armando Fox, David A. Patterson, Nick Lanham, Beth Trushkowsky, Jesse Trutna, Haruki Oh: SCADS: Scale-Independent Storage for Social Computing Applications. CIDR 2009

  Two early systems paper on scalability; what we saw in class was a formalization of their approach
- Michael Armbrust, Eric Liang, Tim Kraska, Armando Fox, Michael J. Franklin, David A. Patterson: Generalized scale independence through incremental precomputation. SIGMOD 2013:625-636

  Scalability under updates to the underlying data
- Wenfei Fan, Floris Geerts, Frank Neven: Making Queries Tractable on Big Data with Preprocessing. PVLDB 6(9): 685-696 (2013)

  New notions of complexity for handling large volumes of data
- Wenfei Fan, Floris Geerts, Leonid Libkin: On scale independence for querying big data. PODS 2014:51-62

  We saw the notion of controllability here. Eligible topics for an esseay are incremental computation and using views

Comments on papers

- Yang Cao, Wenfei Fan, Tianyu Wo, Wenyuan Yu: Bounded Conjunctive Queries. PVLDB 7(12): 1231-1242 (2014)
  Specialized algorithms for handling select-project-join queries over big data
- Foto N. Afrati, Jeffrey D. Ullman: Transitive closure and recursive Datalog implemented on clusters. EDBT 2012: 132-143
- Foto N. Afrati, Jeffrey D. Ullman: Optimizing Multiway Joins in a Map-Reduce Environment. IEEE Trans. Knowl. Data Eng. 23(9): 1282-1298 (2011)
  Two papers on parallelizing different types of queries (recursive and non-recursive)
- Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, Atri Rudra: Joins via Geometric Resolutions: Worst-case and Beyond. PODS 2015: 213-228
- Hung Q. Ngo, Christopher Ré, Atri Rudra: Skew strikes back: new developments in the theory of join algorithms. SIGMOD Record 42(4): 5-16 (2013)
- Hung Q. Ngo, Ely Porat, Christopher Ré, Atri Rudra: Worst-case optimal join algorithms. PODS 2012: 37-48
- Albert Atserias, Martin Grohe, Daniel Marx: Size Bounds and Query Plans for Relational Joins. SIAM J. Comput. 42(4): 1737-1767 (2013)
  A series of papers on worst-case performance of join algorithms

Comments on papers

- Yannis E. Ioannidis: Approximations in Database Systems. ICDT 2003: 16-30
- Minos N. Garofalakis, Phillip B. Gibbons: Approximate Query Processing: Taming the TeraBytes. VLDB 2001

  Approximation techniques that take into account both data and queries
- Pablo Barcelo, Leonid Libkin, Miguel Romero: Efficient Approximations of Conjunctive Queries. SIAM J. Comput. 43(3): 1085-1130 (2014)

  Eligible topics include static analysis of approximations
- Markus Frick, Martin Grohe: Deciding first-order properties of locally tree-decomposable structures. Journal of the ACM 48(6): 1184-1206 (2001)

  How to improve performance of relational queries on databases with special properties
- Joerg Flum, Martin Grohe: Fixed-Parameter Tractability, Definability, and Model-Checking. SIAM J. Comput. 31(1): 113-145 (2001)

  A different way of measuring complexity, and its full analysis
- Joerg Flum, Markus Frick, Martin Grohe: Query evaluation via tree-decompositions. Journal of the ACM 49(6): 716-752 (2002)

  Using tree decompositions to get faster query evaluation

Comments on papers

- Paul Beame, Paraschos Koutris, Dan Suciu: Communication steps for parallel query processing. PODS 2013: 273-284

  Models for correct parallel evaluation of queries
- Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, Thomas Schwentick: Parallel-Correctness and Transferability for Conjunctive Queries. PODS 2015: 47-58

  A detailed study of correct parallel evaluation of conjunctive queries