What we talk about when we talk about graphs

George Fletcher

Database Group
Eindhoven University of Technology

University of Waterloo

May 25, 2020

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Nikolay Yakovets

George Fletcher

Odysseas Papapetrou

Database Group

- design and engineering of graph query languages
- social network analytics
- data analytics over streaming and heterogeneous data
- big data infrastructure
- privacy-sensitive data analytics
- data integration

EINDHOVEN UNIVERSITY OF TECHNOLOGY

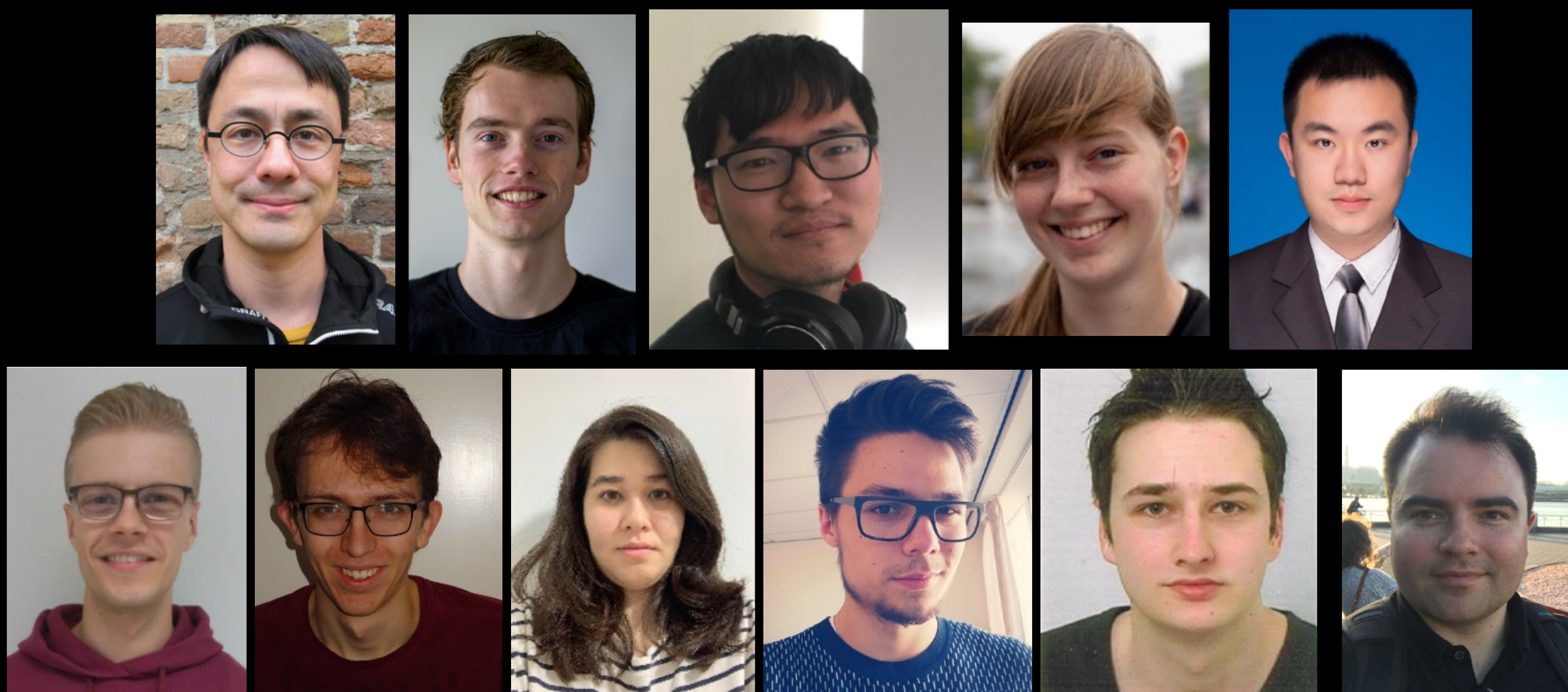
Ag
AvantGraph

Os Open Source	**Mm** Main Memory	**Ra** Recursive Analytics
Vc Vectorized Compiled	**Wco** Worst-case Optimal	**Fz** Factorized
Tm Temporal	**Re** Reachability	**Td** Topo+data

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Ag AvantGraph

Open source release coming in 2020!



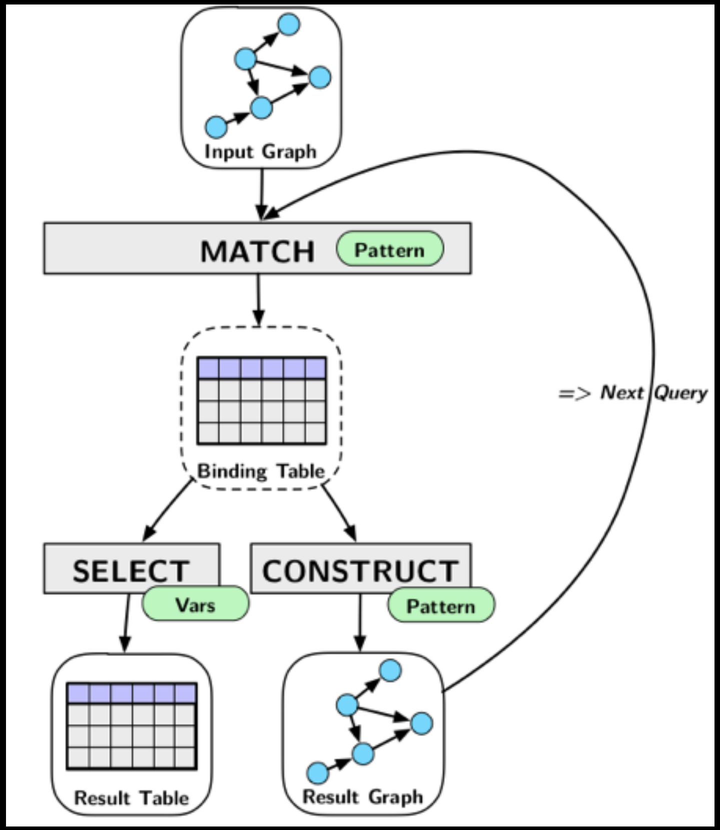
Ag: state of the art recursive queries and query planning

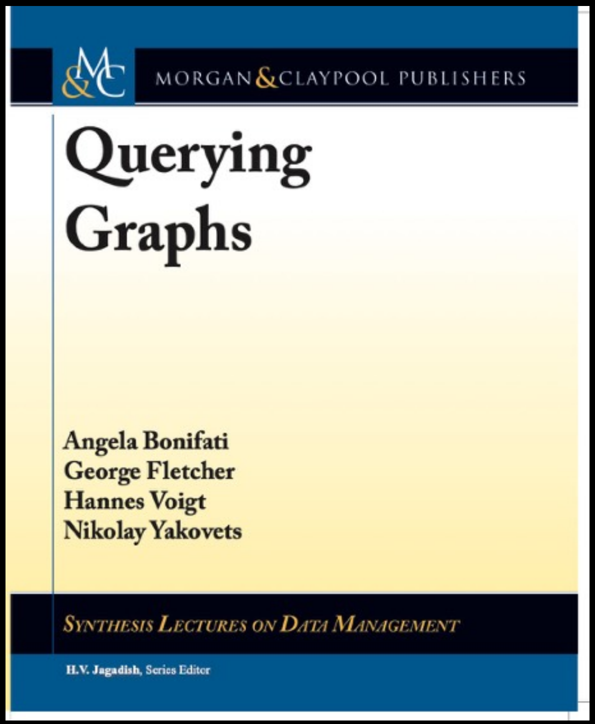
Contemporary graph QL's such as Cypher, G-CORE, PGQL, SPARQL, and the upcoming standard GQL all support advanced recursive analytics on graphs

- e.g., path navigation and paths a first-class citizens

Supporting all of these languages, Ag's internal logical language is the Regular Queries extended to the PG model

- See Bonifati, Fletcher, Voigt, Yakovets. *Querying Graphs* 2018.
- SSDBM 2019, SIGMOD 2018

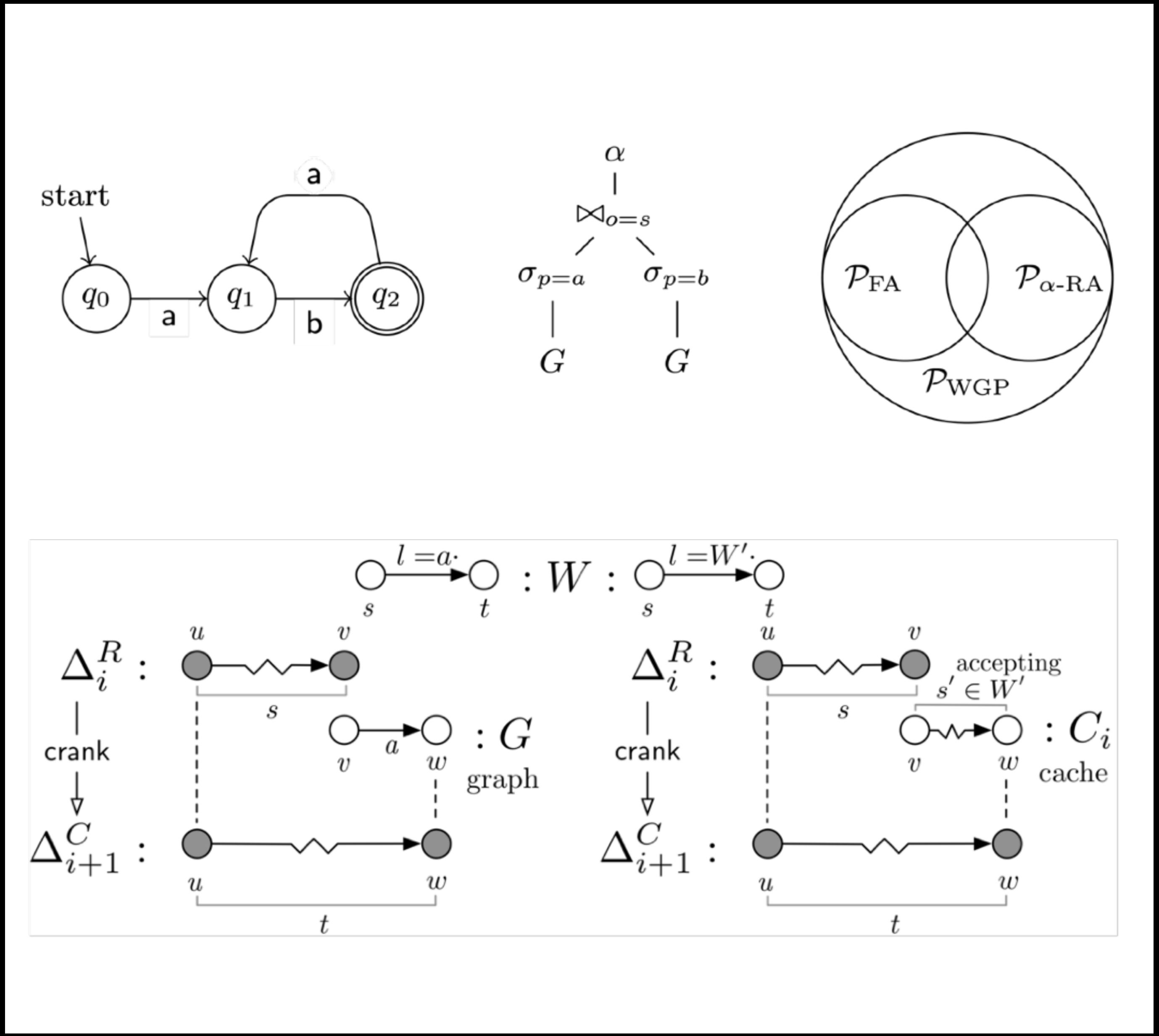





```
CONSTRUCT (n)-/@p:localPeople{distance:=c}/->(m)
MATCH (n)-/3 SHORTEST p <:knows*> COST c/->(m)
WHERE n.firstName = 'John' AND n.lastName = 'Doe'
  AND (n)-[:isLocatedIn]->()<-[:isLocatedIn]-(m)
```

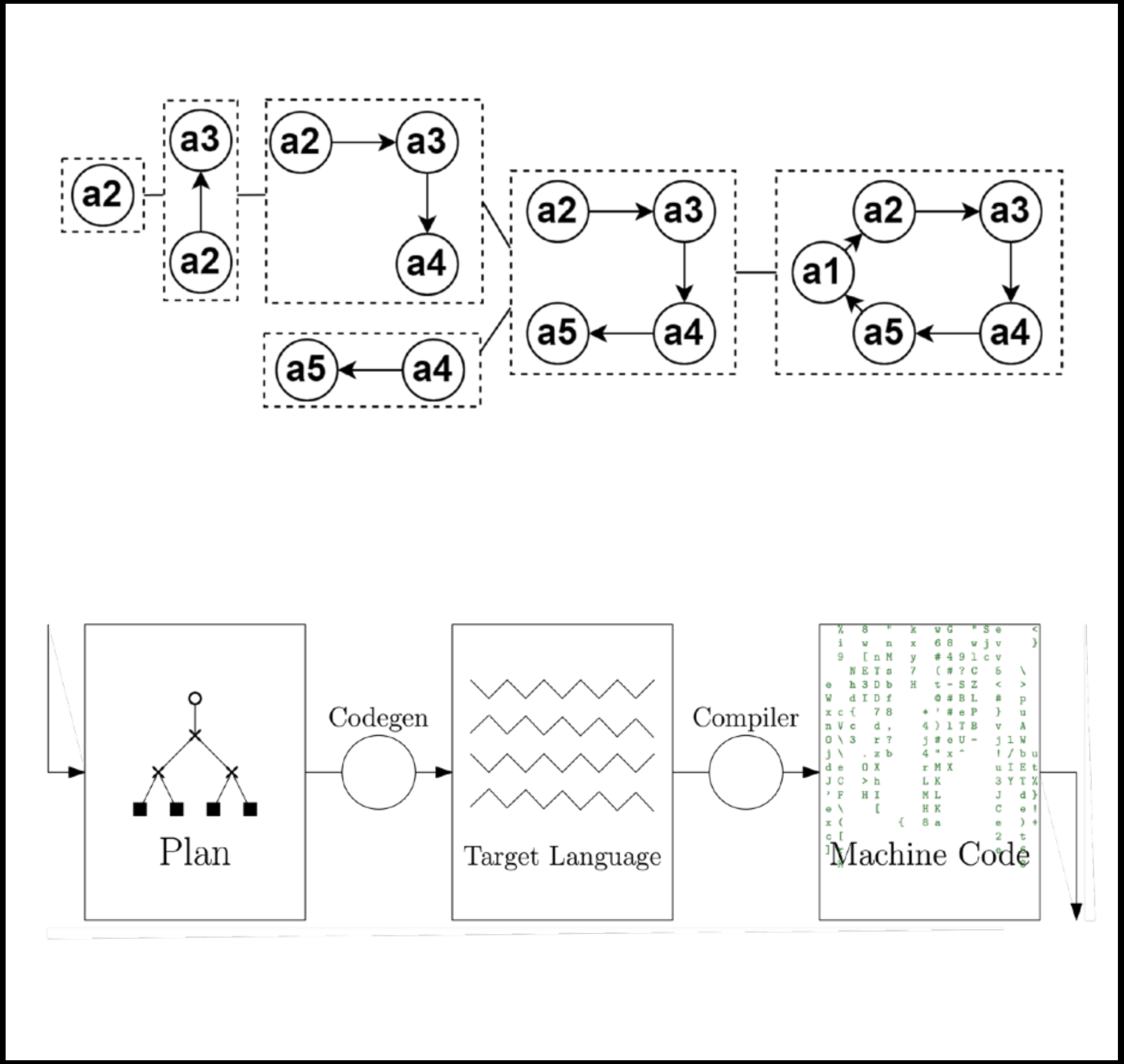
Ag: state of the art recursive queries and query planning

Ag query planner/optimizer properly extends the plan spaces of earlier RA and automata-based planners, to capture novel efficient and scalable physical execution strategies specifically for contemporary recursive graph analytics (SIGMOD 2016)



Ag: state of the art execution engine

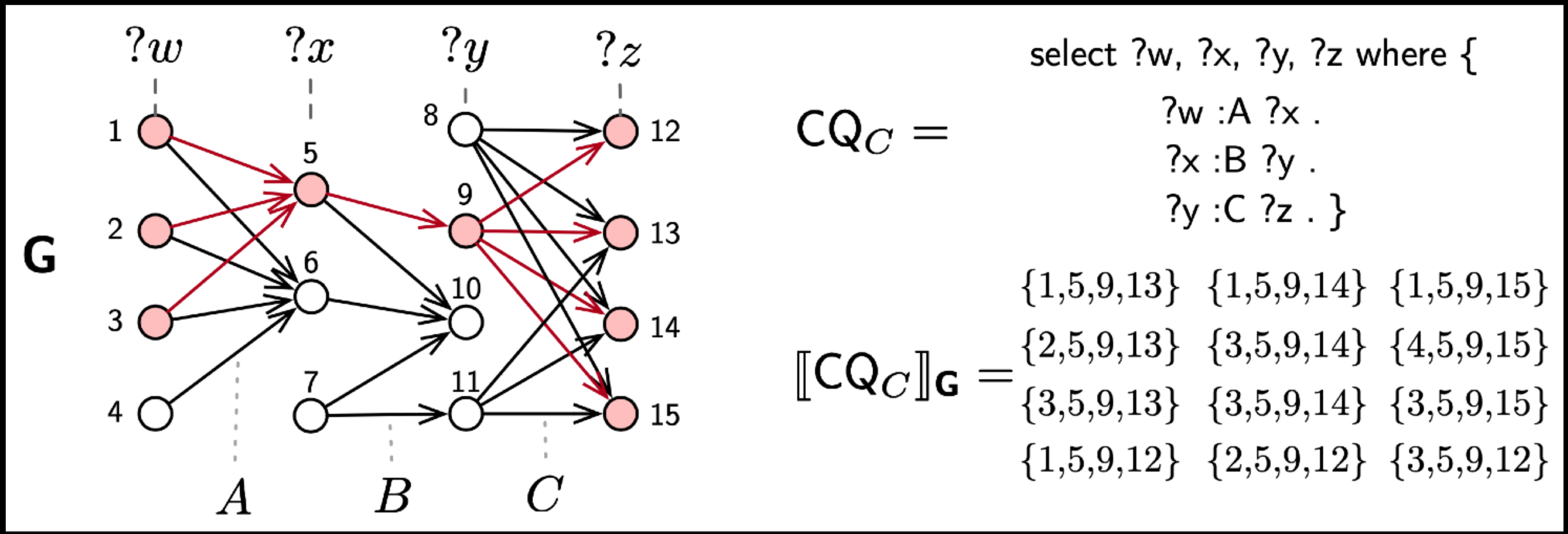
- Worst-case optimal join processing for subgraph pattern matching queries
  - de Brouwer, TU Eindhoven 2020
- Compiled and vectorised queries
  - van de Wall, TU Eindhoven 2020



WireFrame:

Factorization of intermediate results with answer graphs

- avoids explosion of intermediate results (IR) during query evaluation caused by multiplicity (AMW 2017; Clark, TU Eindhoven 2019; in submission 2020)
- use an answer graph as the representation of the IR

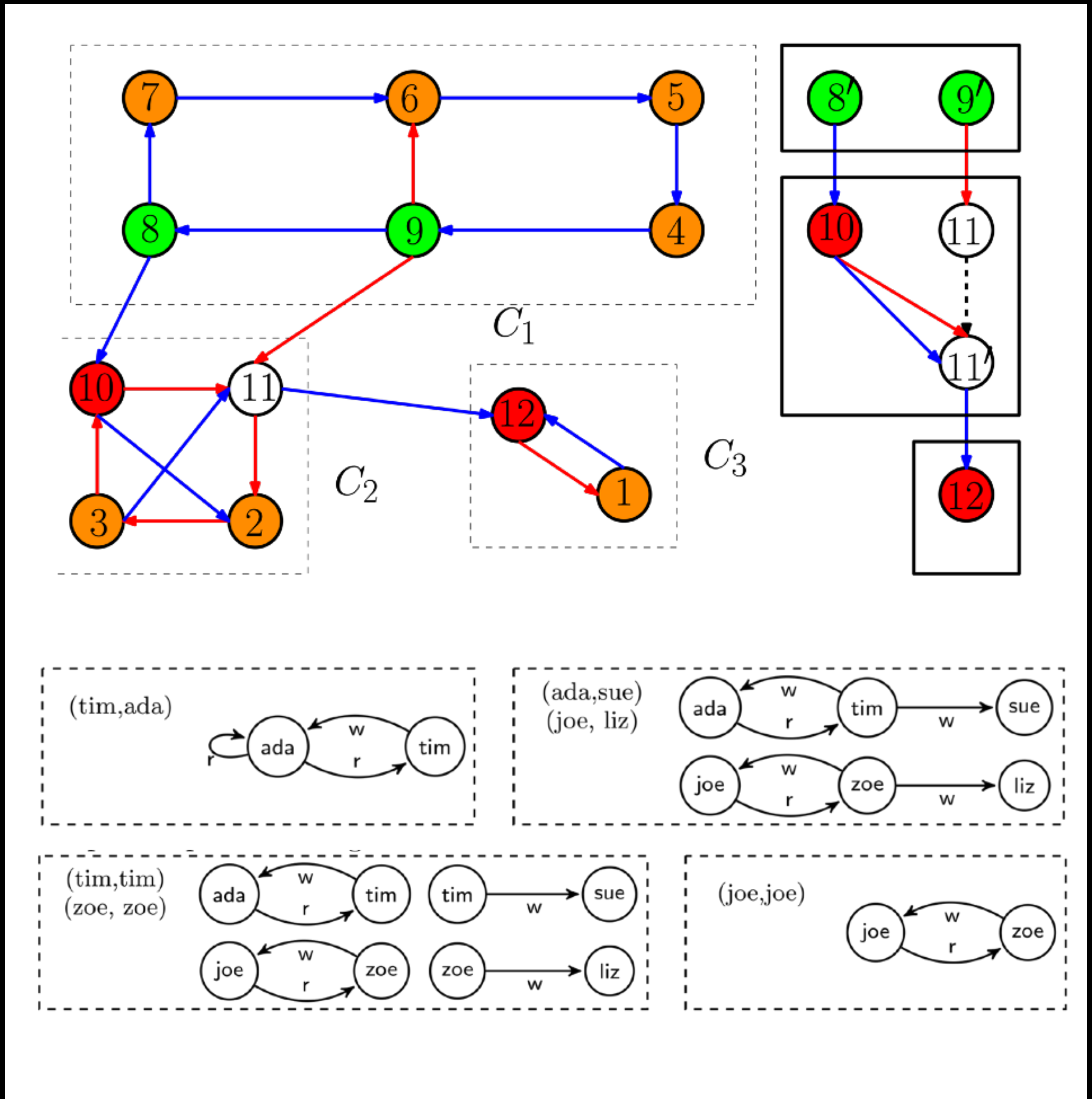


Ag: advanced reachability and structural indexing

In addition to state of the art physical graph representations, advanced indexing data structures are introduced.

Landmark indexing — for label-constrained reachability, the most common form of recursive path navigation (SIGMOD 2017)

Structural indexing — subgraph indexing for conjunctive path patterns, the core of contemporary PG languages (arXiv 2020)

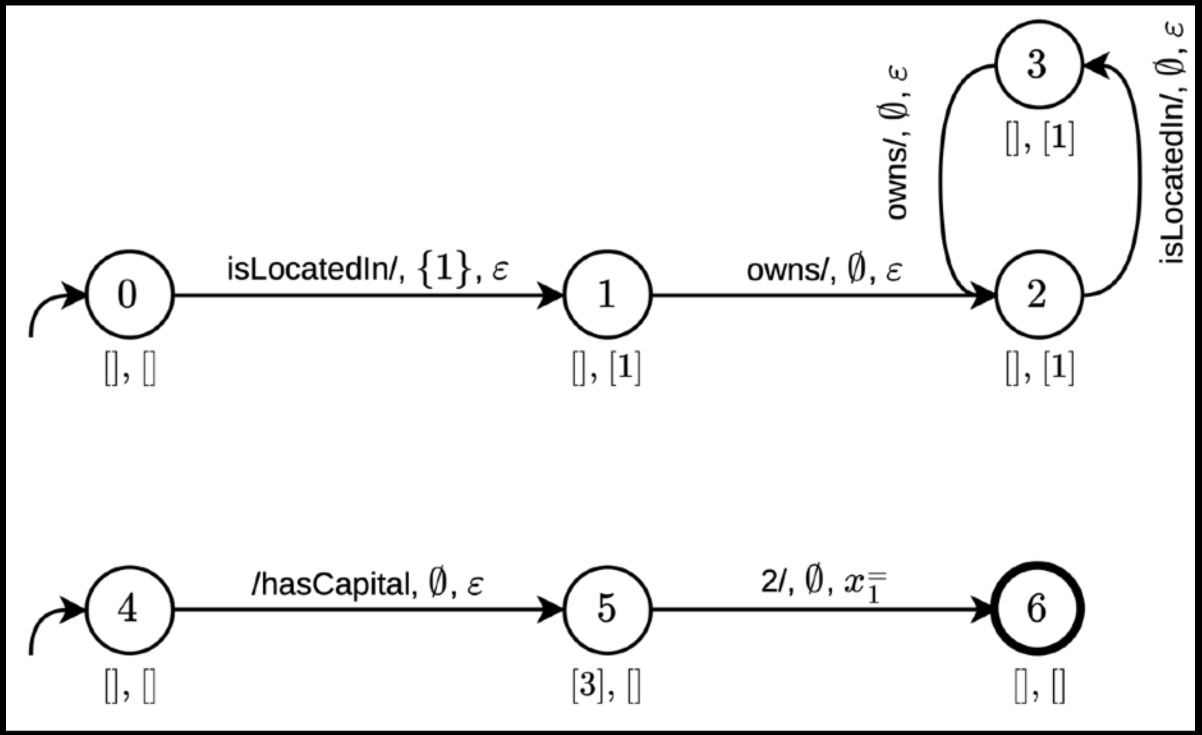


Recursive analytics with data in Ag

Nodes and edges in property graphs have local data

- e.g., `People` nodes can have a `name` and `email address`; `Follows` edges between people can have a `StartDate`

For contemporary graph language extensions for reasoning about local data in recursive analytics, we extend the planner to generate novel execution plans to leverage new data-aware optimisation opportunities (EDBT 2020)



Knowledge modelling for graphs

SHACL for RDF

- *Towards efficient validation of RDF graphs against recursive SHACL.* Collaboration with Amazon Neptune. (Lahaye, 2020).

Property Graph Schema Working Group (LDBC)

- *Modeling for graph semantics (2018-current).*
- Working closely with ISO GQL standards committee.

Open Ag research challenges

We are just at the beginning, with many exciting research challenges

- Cardinality estimation for optimising recursive analytics
- Graph aggregation: language extensions and scalable methods
- Benchmarking frameworks for knowledge graph analytics
  - State of the art frameworks such as *gMark* (IEEE TKDE 2017) support recursive analytics and flexible topological control
  - However, we need models and solutions for temporal graphs, graph aggregation, property graph data, ...
- Schema and constraints for property graphs
  - Mappings in the presence of graph schema
  - Schema discovery and conformance checking
  - Dependencies for property graph data cleaning and quality
- ....

What we talk about when we talk about graphs

What we talk about when we talk ...

Sapir-Whorf: "the structure of a language affects the ways in which its speakers conceptualize their world" (Wikipedia)

- Wilhelm von Humboldt (1767-1835): linguistics and philology
  - *The heterogeneity of language and its influence on the intellectual development of mankind* (1836)

What we talk about when we talk ...

Sapir-Whorf: "the structure of a language affects the ways in which its speakers conceptualize their world" (Wikipedia)

- Wilhelm von Humboldt (1767-1835): linguistics and philology
  - *The heterogeneity of language and its influence on the intellectual development of mankind* (1836)
- Franz Boas (1858-1942): anthropology
- Edward Sapir (1884-1939) and Benjamin Whorf (1897-1941): linguistics
  - *Language, mind, and reality* (1942)

What we talk about when we talk ...

Sapir-Whorf: "the structure of a language affects the ways in which its speakers conceptualize their world" (Wikipedia)

- Wilhelm von Humboldt (1767-1835): linguistics and philology
  - *The heterogeneity of language and its influence on the intellectual development of mankind* (1836)
- Franz Boas (1858-1942): anthropology
- Edward Sapir (1884-1939) and Benjamin Whorf (1897-1941): linguistics
  - *Language, mind, and reality* (1942)
- and in sociology, psychology, philosophy, history (e.g., Kuhn's "Structure of scientific revolutions", Wittgenstein's language games), ...
  - deep and lasting impact across the sciences

What we talk about when we talk ... about graphs

Research focus on the theory, engineering, and applications of query languages for graph/network data

Today, I will talk about one of my long-term projects

I have been investigating how graph query languages affect the way in which clients structure their world.

- i.e., how the choice of query language restricts and shapes concrete graph instances.

What we talk about when we talk ... about graphs

Research focus on the theory, engineering, and applications of query languages for graph/network data

Today, I will talk about one of my long-term projects

I have been investigating how graph query languages affect the way in which clients structure their world.

- i.e., how the choice of query language restricts and shapes concrete graph instances.

Collaborations with colleagues at Singapore, Eindhoven, Hasselt, Bloomington, Osaka, and Brussels.

Bibliographic details can be found on my homepage.

Expressive power of query languages

Notions of language expressivity

Edgar Codd (1972): invents the relational database model (i.e., SQL databases) and first algebraic and logical query languages

- How can we measure the expressive power of a database query language?

Notions of language expressivity

Edgar Codd (1972): invents the relational database model (i.e., SQL databases) and first algebraic and logical query languages

- How can we measure the expressive power of a database query language?
- *Codd's solution:* introduce notion of "relational completeness"
  - is your language as expressive as mine (i.e., the relational calculus)?

Notions of language expressivity

Edgar Codd (1972): invents the relational database model (i.e., SQL databases) and first algebraic and logical query languages

- How can we measure the expressive power of a database query language?
- *Codd's solution:* introduce notion of "relational completeness"
  - is your language as expressive as mine (i.e., the relational calculus)?
- ... rather ad hoc

What we would like is a language-independent notion of expressivity

Notions of language expressivity

Towards language-independent notions of expressivity ....

Notions of language expressivity

Towards language-independent notions of expressivity ....

Query expressivity (Aho & Ullman 1979, Chandra & Harel 1980)

- What is the expressive power of Codd's relational calculus/algebra (to formulate general functions)?

Notions of language expressivity

Towards language-independent notions of expressivity ....

Query expressivity (Aho & Ullman 1979, Chandra & Harel 1980)

- What is the expressive power of Codd's relational calculus/algebra (to formulate general functions)?
- for example,
  - *expressible:* "triangles?", "no triangles?"
  - *not expressible:* transitive closure

Notions of language expressivity

Towards language-independent notions of expressivity ....

Query expressivity (Aho & Ullman 1979, Chandra & Harel 1980)

- What is the expressive power of Codd's relational calculus/algebra (to formulate general functions)?
- for example,
  - *expressible:* "triangles?", "no triangles?"
  - *not expressible:* transitive closure

... primary focus of research community

Notions of language expressivity

Towards language-independent notions of expressivity ....

Instance expressivity (Bancilhon and Paredaens 1978)

- What is the expressive power of Codd's relational algebra (on an arbitrary fixed instance)?

Notions of language expressivity

Towards language-independent notions of expressivity ....

Instance expressivity (Bancilhon and Paredaens 1978)

- What is the expressive power of Codd's relational algebra (on an arbitrary fixed instance)?
- *fact:* T is expressible from S in Codd's algebra if and only if

$$constants(T) \subseteq constants(S)$$

and

$$automorphism(S) \subseteq automorphism(T).$$

Notions of language expressivity

Towards language-independent notions of expressivity ....

Instance expressivity (Bancilhon and Paredaens 1978)

- What is the expressive power of Codd's relational algebra (on an arbitrary fixed instance)?
- *fact:* T is expressible from S in Codd's algebra if and only if

$$constants(T) \subseteq constants(S)$$

and

$$automorphism(S) \subseteq automorphism(T).$$

i.e., characterization in terms of the structure of instance S.

Instance expressivity

On an (arbitrary) fixed instance S, characterize output space of a given language $\mathcal{L}$

Instance expressivity

On an (arbitrary) fixed instance S, characterize output space of a given language $\mathcal{L}$

> *Given a source instance S and target instance T, can S be mapped to T in $\mathcal{L}$?*

$$S \xrightarrow{\quad ? \in \mathcal{L} \quad} T$$

Instance expressivity

On an (arbitrary) fixed instance S, characterize output space of a given language $\mathcal{L}$

Given a source instance S and target instance T, can S be mapped to T in $\mathcal{L}$?

$$S \xrightarrow{\quad ? \in \mathcal{L} \quad} T$$

For two objects $o_1, o_2 \in S$, can they be distinguished by an expression $e \in \mathcal{L}$?

$$o_1 \in e(S) \qquad\qquad o_2 \notin e(S)$$

Instance expressivity

Example. Suppose S is a text document collection and $\mathcal{L}$ is keyword queries.

Then objects (i.e., documents) $o_1, o_2 \in S$ can be distinguished iff one of o_1 and o_2 has a keyword the other doesn't.

- for example, o_1 has an occurrence of the keyword "Codd" and o_2 doesn't.

Instance expressivity

Example. Suppose S is a text document collection and $\mathcal{L}$ is keyword queries.

Then objects (i.e., documents) $o_1, o_2 \in S$ can be distinguished iff one of o_1 and o_2 has a keyword the other doesn't.

- for example, o_1 has an occurrence of the keyword "Codd" and o_2 doesn't.

Example. Suppose S is an XML document and $\mathcal{L}$ is XPath restricted to parent-child navigation.

Then objects (i.e., nodes in an XML document) $o_1, o_2 \in S$ can be distinguished iff one of o_1 and o_2 has an incoming path the other doesn't.

- for example, o_1 is an "author/name/lastname" and o_2 isn't.

Instance expressivity

The BP result is for first-order logic on finite models i.e., relational calculus (= SQL) on relational databases.

Structural characterizations later discovered for query languages on nested relations, object-oriented DBs, ...

However, no significant application was made of these results towards engineering of data management systems.

Our results on instance expressivity

tree structured data

- structural characterizations and indexing for XPath fragments (*Inf Syst 2020*, *J Comput Syst Sci* 2016, *Inf Syst* 2009)

(arbitrary) graph structured data

- structural characterizations of Tarski's relation algebra on directed edge-labeled graphs (arXiv 2020, *Inf Sci* 2015, *J Logic Comput* 2015)
- structural characterizations of SPARQL fragments (DBPL 2011, ICDT 2014)
- structural indexing for accelerated SPARQL evaluation (ESWC 2012)

structured data (relational databases)

- structural characterizations and indexing for SQL (ICDT 2014, EDBT 2016)

Our results on instance expressivity

tree structured data

- structural characterizations and indexing for XPath fragments (*Inf Syst 2020*, *J Comput Syst Sci* 2016, *Inf Syst* 2009)

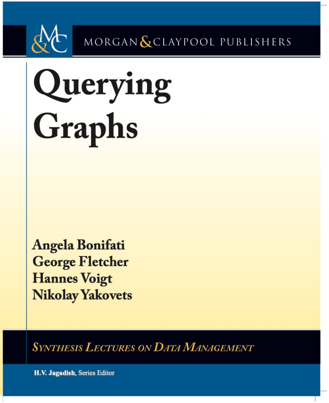
(arbitrary) graph structured data My focus today

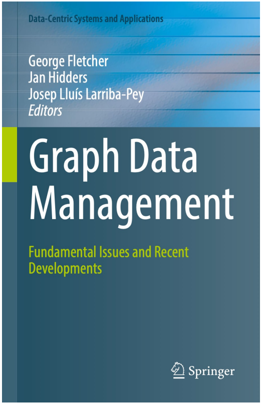
- structural characterizations of Tarski's relation algebra on directed edge-labeled graphs (arXiv 2020, *Inf Sci* 2015, *J Logic Comput* 2015)
- structural characterizations of SPARQL fragments (DBPL 2011, ICDT 2014)
- structural indexing for accelerated SPARQL evaluation (ESWC 2012)

structured data (relational databases)

- structural characterizations and indexing for SQL (ICDT 2014, EDBT 2016)

Bigger picture





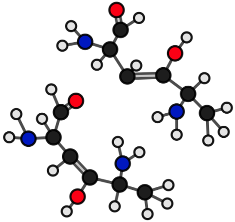
Morgan & Claypool 2018 and Springer 2018

Tarski's Relation Algebra

Why graph data?

Big graph data sets are ubiquitous

- social networks (e.g., LinkedIn, Facebook)
- scientific networks (e.g., Uniprot, PubChem)
- knowledge graphs (e.g., DBPedia, MS Academic Graph)
- ...



Focus is on "things" and their relationships

Why graph data?

Analytics on big graphs increasingly important

- role discovery in social networks
- identifying interesting patterns in biological networks
- finding important publications in a citation network
- ...

Why graph data?

Analytics on big graphs increasingly important

- role discovery in social networks
- identifying interesting patterns in biological networks
- finding important publications in a citation network
- ...

In response to these trends, we have recently witnessed an explosion of graph data management solutions, e.g.,

- Graph databases such as Neo4j and Amazon Neptune
- Graph analytics platforms such as PGX, Flink Gelly, GraphX
- Triple stores such as Virtuoso and AllegroGraph
- Datalog engines such as LogicBlox and Datomic

Paths in graphs

Relation Algebra[1] already proposed by Alfred Tarski in the 1940's as a basic query language for reasoning about paths in graphs

[1] not to be confused with Codd's *relational* algebra (circa 1970)

Graphs

We are interested in navigating over graphs whose edges are labeled by symbols from a finite label set Λ.

Graphs

We are interested in navigating over graphs whose edges are labeled by symbols from a finite label set Λ.

A graph is a relational structure G, consisting of

- a set of nodes V and,
- for every $\ell \in \Lambda$, a relation $G(\ell) \subseteq V \times V$, the set of edges with label ℓ.

Graphs

For example, suppose we have

$$V = \mathit{people} \cup \mathit{hospitals} \cup \mathit{diseases}$$

and edge labels

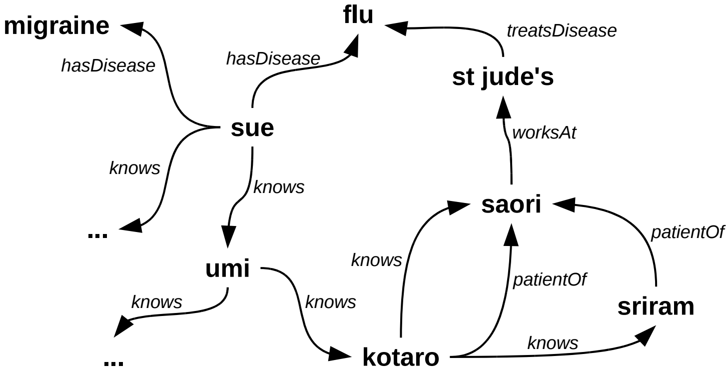
$$\Lambda = \{\text{knows}, \text{worksAt}, \text{patientOf}, \text{hasDisease}, \text{treatsDisease}\}$$

with semantics restricted as:

$$\begin{aligned}
\text{knows} &\subseteq \mathit{people} \times \mathit{people} \\
\text{worksAt} &\subseteq \mathit{people} \times \mathit{hospitals} \\
\text{patientOf} &\subseteq \mathit{people} \times \mathit{people} \\
\text{hasDisease} &\subseteq \mathit{people} \times \mathit{diseases} \\
\text{treatsDisease} &\subseteq \mathit{hospitals} \times \mathit{diseases}.
\end{aligned}$$

Graphs

A small fragment of such a graph



Basic language features

Basic conjunctive path algebra $\mathcal{T}^+$: algebra whose expressions are built up from

- the edge labels Λ,
- the primitive $\emptyset$, and
- the primitive *id*, (i.e., the identity relation)

using

- converse (e^{-1}),
- composition ($e_1 \circ e_2$), and
- intersection ($e_1 \cap e_2$).

Basic language features

Basic conjunctive path algebra $\mathcal{T}^+$: algebra whose expressions are built up from

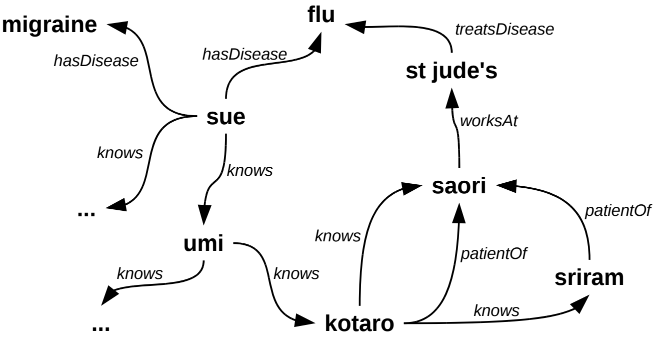
- the edge labels Λ,
- the primitive $\emptyset$, and
- the primitive *id*, (i.e., the identity relation)

using

- converse (e^{-1}),
- composition $(e_1 \circ e_2)$, and
- intersection $(e_1 \cap e_2)$.

On input graph G, each expression $e \in \mathcal{T}^+$ defines a path query $e(G)$, which evaluates to a set of paths in G

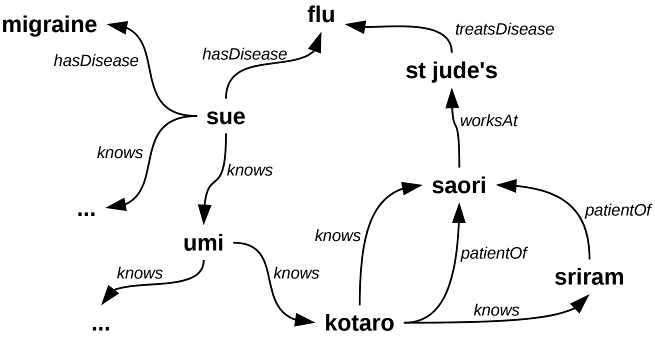
Basic language features



Example: by person, the doctors of their friends

$$\text{knows} \circ \text{patientOf}(G) = \{(\textit{umi}, \textit{saori}), (\textit{kotaro}, \textit{saori}), \ldots\}$$

Basic language features



Example: treatable diseases

$$
\begin{aligned}
[(\text{treatsDisease}^{-1} \circ \text{treatsDisease}) \cap id](G) &= \pi_2(\text{treatsDisease})(G) \\
&= \{(flu, flu), \ldots\}
\end{aligned}
$$

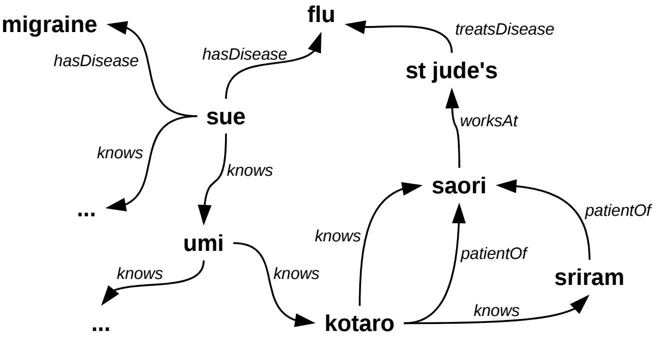
Other language features

The basic algebra is extended with the following features:

- union ($e_1 \cup e_2$),
- diversity (di), (i.e., the non-identity relation), and
- difference ($e_1 \setminus e_2$).

Tarski's algebra $\mathcal{T}$ consists of the language having all basic and nonbasic features.

Nonbasic language features



Example: people and their untreatable diseases

$$\text{hasDisease} \setminus (\text{hasDisease} \circ \pi_2(\text{treatsDisease}))(G) = \{(sue, migraine), \ldots\}$$

Tarski's algebra

Why is $\mathcal{T}$ interesting for the study of graph databases?

Tarski's algebra

Why is $\mathcal{T}$ interesting for the study of graph databases?

Tarski's algebra is to navigational graph query languages

as

Codd's algebra is to relational DB query languages.

Tarski's algebra

Why is $\mathcal{T}$ interesting for the study of graph databases?

Tarski's algebra is to navigational graph query languages
as
Codd's algebra is to relational DB query languages.

(The conjunctive path queries are to navigational graph query languages
as
The conjunctive queries are to relational DB query languages.)

Tarski's algebra

Why is $\mathcal{T}$ interesting for the study of graph databases?

Tarski's algebra is to navigational graph query languages
as
Codd's algebra is to relational DB query languages.

(The conjunctive path queries are to navigational graph query languages
as
The conjunctive queries are to relational DB query languages.)

In fact, the algebra is quite modest: $\mathcal{T}$ is equivalent to FO_2^3 on graphs, i.e., first-order logic using at most three distinct variable names, in two free variables (Tarski and Givant 1987).

- and, $\mathcal{T}^+$ is equivalent to $\exists FO_2^3$.

Instance Expressivity

Language equivalence

A marked structure $\mathbf{G}$ is a triple (G, a, b) where G is a graph, and (a, b) is an ordered pair of nodes from G.

Language equivalence

A marked structure $\mathbf{G}$ is a triple (G, a, b) where G is a graph, and (a, b) is an ordered pair of nodes from G.

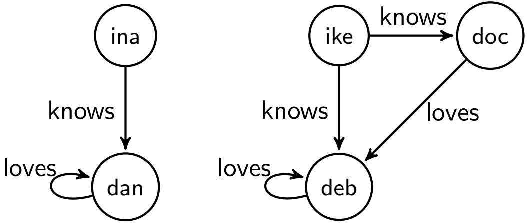
For two marked structures $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$, we write

$$\mathbf{G}_1 \equiv \mathbf{G}_2$$

if $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable in $\mathcal{T}$, i.e., for every expression e in the algebra,

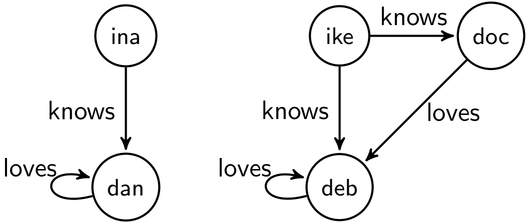
$$(a_1, b_1) \in e(G_1) \quad \Leftrightarrow \quad (a_2, b_2) \in e(G_2).$$

Language equivalence



Example. Consider graph G above.

Language equivalence



Example. Consider graph G above. Here, we have $(G, ina, dan) \not\equiv (G, ike, deb)$ since

$$\text{knows} \circ (\text{loves} \setminus id)(G) \quad = \quad \{(ike, deb)\}.$$

That is, *ina* only knows people who love themselves ...

Structural equivalence

Let G_1 and G_2 be two graphs with node sets V_1 and V_2, respectively, and $a, b \in V_1$, $c, d \in V_2$.

Furthermore, for graph G with node set V, let $\mathit{paths}(G)$ denote the set

$$\{(x, y) \mid x, y \in V \text{ and there is an undirected path from } x \text{ to } y \text{ in } G\}.$$

Structural equivalence

Marked structures (G_1, a, b) and (G_2, c, d) are bisimilar, denoted $(G_1, a, b) \approx (G_2, c, d)$, if and only if the following hold:

Structural equivalence

Marked structures (G_1, a, b) and (G_2, c, d) are bisimilar, denoted $(G_1, a, b) \approx (G_2, c, d)$, if and only if the following hold:

1.
   1.1 (forth) if $a = b$ then $c = d$;
   1.2 (back) if $c = d$ then $a = b$;

Structural equivalence

Marked structures (G_1, a, b) and (G_2, c, d) are bisimilar, denoted $(G_1, a, b) \approx (G_2, c, d)$, if and only if the following hold:

1. 1.1 (forth) if $a = b$ then $c = d$;
   1.2 (back) if $c = d$ then $a = b$;
2. for each $\ell \in \Lambda$,
   2.1 (forth) if $\ell(a, b) \in G_1$, then $\ell(c, d) \in G_2$; and, if $\ell(b, a) \in G_1$, then $\ell(d, c) \in G_2$;
   2.2 (back) if $\ell(c, d) \in G_2$, then $\ell(a, b) \in G_1$; and, if $\ell(d, c) \in G_2$, then $\ell(b, a) \in G_1$; and,

Structural equivalence

Marked structures (G_1, a, b) and (G_2, c, d) are bisimilar, denoted $(G_1, a, b) \approx (G_2, c, d)$, if and only if the following hold:

1. 1.1 (forth) if $a = b$ then $c = d$;
   1.2 (back) if $c = d$ then $a = b$;
2. for each $\ell \in \Lambda$,
   2.1 (forth) if $\ell(a, b) \in G_1$, then $\ell(c, d) \in G_2$; and, if $\ell(b, a) \in G_1$, then $\ell(d, c) \in G_2$;
   2.2 (back) if $\ell(c, d) \in G_2$, then $\ell(a, b) \in G_1$; and, if $\ell(d, c) \in G_2$, then $\ell(b, a) \in G_1$; and,
3. (forth) for each $m_1 \in V_1$, if (a, m_1) and (m_1, b) are in $paths(G_1)$, then there exists $m_2 \in V_2$ such that (c, m_2) and (m_2, d) are in $paths(G_2)$, and, furthermore, $(G_1, a, m_1) \approx (G_2, c, m_2)$ and $(G_1, m_1, b) \approx (G_2, m_2, d)$;

Structural equivalence

Marked structures (G_1, a, b) and (G_2, c, d) are bisimilar, denoted $(G_1, a, b) \approx (G_2, c, d)$, if and only if the following hold:

1. 1.1 (forth) if $a = b$ then $c = d$;
   1.2 (back) if $c = d$ then $a = b$;
2. for each $\ell \in \Lambda$,
   2.1 (forth) if $\ell(a, b) \in G_1$, then $\ell(c, d) \in G_2$; and, if $\ell(b, a) \in G_1$, then $\ell(d, c) \in G_2$;
   2.2 (back) if $\ell(c, d) \in G_2$, then $\ell(a, b) \in G_1$; and, if $\ell(d, c) \in G_2$, then $\ell(b, a) \in G_1$; and,
3. (forth) for each $m_1 \in V_1$, if (a, m_1) and (m_1, b) are in $paths(G_1)$, then there exists $m_2 \in V_2$ such that (c, m_2) and (m_2, d) are in $paths(G_2)$, and, furthermore, $(G_1, a, m_1) \approx (G_2, c, m_2)$ and $(G_1, m_1, b) \approx (G_2, m_2, d)$;
4. (back) for each $m_2 \in V_2$, if (c, m_2) and (m_2, d) are in $paths(G_2)$, then there exists $m_1 \in V_1$ such that (a, m_1) and (m_1, b) are in $paths(G_1)$, and, furthermore, $(G_2, c, m_2) \approx (G_1, a, m_1)$ and $(G_2, m_2, d) \approx (G_1, m_1, b)$.

Structural equivalence

If only the forth conditions hold, then we say (G_2, c, d) simulates (G_1, a, b), denoted by $(G_1, a, b) \preceq (G_2, c, d)$.

If $(G_1, a, b) \preceq (G_2, c, d)$ and $(G_2, c, d) \preceq (G_1, a, b)$, then we say these marked structures are similar, which we denote by $(G_1, a, b) \sim (G_2, c, d)$.

Structural equivalence

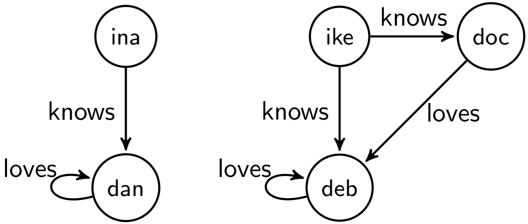
If only the forth conditions hold, then we say (G_2, c, d) simulates (G_1, a, b), denoted by $(G_1, a, b) \preceq (G_2, c, d)$.

If $(G_1, a, b) \preceq (G_2, c, d)$ and $(G_2, c, d) \preceq (G_1, a, b)$, then we say these marked structures are similar, which we denote by $(G_1, a, b) \sim (G_2, c, d)$.

Note that on a graph G, $\approx$ and $\sim$ are equivalence relations on $paths(G)$.

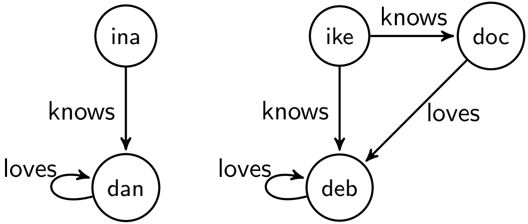
Furthermore, partitioning under $\approx$ and $\sim$ is tractable, with $O(m \log n)$ and $O(mn \log n)$ solutions, respectively, for a graph with m edges and n nodes (Paige and Tarjan 1987, Ranzato 2014).

Structural equivalence



Example. Consider again graph G above. Here, we have $(G, ina, dan) \sim (G, ike, deb)$.

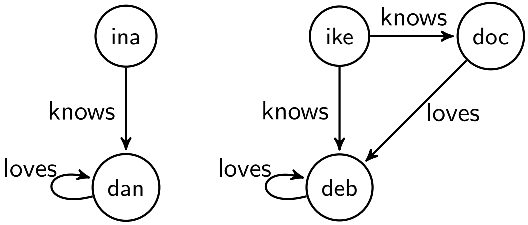
Structural equivalence



Example. Consider again graph G above. Here, we have $(G, ina, dan) \sim (G, ike, deb)$.

Example. Note, however, that $(G, ina, dan) \not\approx (G, ike, deb)$ since *ina* doesn't know someone like *doc* (i.e., someone who doesn't love themself).

Structural equivalence



Example. Consider again graph G above. Here, we have $(G, ina, dan) \sim (G, ike, deb)$.

Example. Note, however, that $(G, ina, dan) \not\approx (G, ike, deb)$ since *ina* doesn't know someone like *doc* (i.e., someone who doesn't love themself).
Recall that $(G, ina, dan) \not\equiv (G, ike, deb)$ ... this isn't a coincidence

Structural equivalence

Coupling Theorem (*J Logic Comput* 2015)

Let $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$ be marked structures. Then

$$\mathbf{G}_1 \equiv \mathbf{G}_2 \quad \Leftrightarrow \quad \mathbf{G}_1 \approx \mathbf{G}_2.$$

Structural equivalence

Coupling Theorem (*J Logic Comput* 2015)

Let $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$ be marked structures. Then

$$\mathbf{G}_1 \equiv \mathbf{G}_2 \quad \Leftrightarrow \quad \mathbf{G}_1 \approx \mathbf{G}_2.$$

We similarly obtained novel bisimulation characterizations for a wide range of fragments of the algebra.

Structural equivalence

Coupling Theorem (*J Logic Comput* 2015)

Let $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$ be marked structures. Then

$$\mathbf{G}_1 \equiv \mathbf{G}_2 \quad \Leftrightarrow \quad \mathbf{G}_1 \approx \mathbf{G}_2.$$

We similarly obtained novel bisimulation characterizations for a wide range of fragments of the algebra.

For positive algebra fragments such as $\mathcal{T}^+$, we similarly obtained new simulation characterizations.

Structural Indexing

Structural indexing

Up to this point, our investigations of Tarski's algebra have focused on the relative expressive power of the various fragments of the algebra, and their structural characterizations.

We have also obtained structural characterizations for a core fragment of SPARQL, the W3C's recommendation language for the RDF graph data model, with an eye towards "structural" index design. (DBPL 2011, ICDT 2014)

Structural indexing

Up to this point, our investigations of Tarski's algebra have focused on the relative expressive power of the various fragments of the algebra, and their structural characterizations.

We have also obtained structural characterizations for a core fragment of SPARQL, the W3C's recommendation language for the RDF graph data model, with an eye towards "structural" index design. (DBPL 2011, ICDT 2014)

The basic idea here is to group together structurally equivalent RDF triples, since the language cannot distinguish them, and build access mechanisms on top of these "blocks."

We then use this index to accelerate query processing on a reduced search space (ESWC 2012, ICDT 2014).

Empirical study

SaintDB: Implement disk-based bisimulation index atop RDF-3x open-source state-of-the-art value-based triple store.

- the first triple-based structural index for RDF
- our index is formally coupled to practical core fragment of SPARQL

Empirical analysis on community benchmark data/queries demonstrates competitiveness with RDF-3X on broad range of query scenarios, with up to multiple orders of magnitude reduction in query processing costs

Partitioning massive graphs under bisimulation

Note that this approach only works if computing bisimulation partitioning of big graphs is practical.

Partitioning massive graphs under bisimulation

Note that this approach only works if computing bisimulation partitioning of big graphs is practical.

Efficient *main memory* approaches to bisimulation partitioning have been studied since the 80's, as bisimilarity is a fundamental notion arising in a wide range of contexts (e.g., set theory, distributed computing, process modeling, social networks, ...).

However, there has been no approach to compute bisimulation on massive disk-resident graphs.

Partitioning massive graphs under bisimulation

To address this, we have developed the first I/O-efficient approaches to bisimulation partitioning of massive graphs (SIGMOD 2012, CIKM 2013)

We have also developed the first effective MapReduce and distributed solutions for this problem (BICOD 2013, SAC 2016)

Partitioning massive graphs under bisimulation

Empirical study shows that bisimulation reductions are often practical

Reductions between 10^{-1} and 10^{-4} (or better) for both number of edges and number of nodes, for many practical data sets, such as DBPedia, Linked MDB, Jamendo, DBLP, and Twitter (CIKM 2013, SAC 2016)

Partitioning massive graphs under bisimulation

Empirical study shows that bisimulation reductions are often practical

Reductions between 10^{-1} and 10^{-4} (or better) for both number of edges and number of nodes, for many practical data sets, such as DBPedia, Linked MDB, Jamendo, DBLP, and Twitter (CIKM 2013, SAC 2016)

of course, for some data, there is no structure to compress, and the "reductions" are too fine

Ongoing and open research directions

Ongoing and open research directions

(1) Further engineering studies into structural indexing for efficient path query processing.

Current focus: the conjunctive fragment of Tarski's Algebra, $\mathcal{T}^+$, in analogy to the conjunctive FO queries for efficient SQL evaluation.

- core of industrial graph query languages such as Cypher (Neo4j), PGQL (Oracle), and our standards proposal G-CORE (SIGMOD 2018)

Ongoing and open research directions: $\mathcal{T}^+$

Consider the following query languages:

- $\exists FO_2^3$, the fragment of the positive existential first order queries on graphs consisting of all those queries having one or two free variables, constructed using at most three distinct variables and having a connected join graph.

Ongoing and open research directions: $\mathcal{T}^+$

Consider the following query languages:

- $\exists FO_2^3$, the fragment of the positive existential first order queries on graphs consisting of all those queries having one or two free variables, constructed using at most three distinct variables and having a connected join graph.
- *SPII*, the family of all those graph patterns expressible as source-to-target directed edge-labeled graphs recursively constructed by a finite sequence of series and parallel combinations of nodes, forward edges, and inverse edges.

Ongoing and open research directions: $\mathcal{T}^+$

Consider the following query languages:

- $\exists FO_2^3$, the fragment of the positive existential first order queries on graphs consisting of all those queries having one or two free variables, constructed using at most three distinct variables and having a connected join graph.
- *SPII*, the family of all those graph patterns expressible as source-to-target directed edge-labeled graphs recursively constructed by a finite sequence of series and parallel combinations of nodes, forward edges, and inverse edges.
- *TarskiLog*, the language of positive non-recursive Datalog programs over graphs where the body of each rule uses at most three distinct variables and has a connected join graph; and, each rule has a distinct binary head predicate.

Ongoing and open research directions: $\mathcal{T}^+$

Example. Consider the query "doctors and their known patients."

Ongoing and open research directions: $\mathcal{T}^+$

Example. Consider the query "doctors and their known patients." This is expressed in $\exists FO_2^3$ as

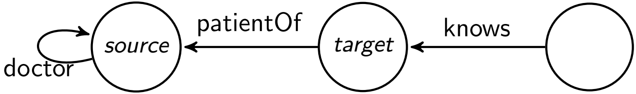
$$\{(x, y) \mid \text{doctor}(x, x) \wedge \text{patientOf}(y, x) \wedge \exists x(\text{knows}(x, y))\}$$

Ongoing and open research directions: $\mathcal{T}^+$

Example. Consider the query "doctors and their known patients."
This is expressed in $\exists FO_2^3$ as

$$\{(x, y) \mid \text{doctor}(x, x) \wedge \text{patientOf}(y, x) \wedge \exists x(\text{knows}(x, y))\}$$

in *SPII* as

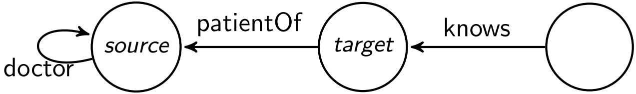


Ongoing and open research directions: $\mathcal{T}^+$

Example. Consider the query "doctors and their known patients."
This is expressed in $\exists FO_2^3$ as

$$\{(x, y) \mid \mathrm{doctor}(x, x) \wedge \mathrm{patientOf}(y, x) \wedge \exists x(\mathrm{knows}(x, y))\}$$

in *SPII* as



and in *TarskiLog* as

$$\begin{array}{lcl} known(X, X) & :- & knows(Y, X) \\ result(X, Y) & :- & doctor(X, X), patientOf(Y, X), known(Y, Y). \end{array}$$

Ongoing and open research directions: $\mathcal{T}^+$

Example "doctors and their known patients", cont.

In $\mathcal{T}^+$, we can express this as

$$(\mathsf{doctor} \cap id) \circ \mathsf{patientOf}^{-1} \circ \pi_2(\mathsf{knows}).$$

Ongoing and open research directions: $\mathcal{T}^+$

In general, we can establish that:

$\mathcal{T}^+$, $\exists FO^3_2$, SPII, and TarskiLog are equivalent in expressive power.

Ongoing and open research directions: $\mathcal{T}^+$

In general, we can establish that:

> *$\mathcal{T}^+$, $\exists FO^3_2$, SPII, and TarskiLog are equivalent in expressive power.*

Hence we have four natural alternative syntaxes (algebraic, declarative, graphical, and rule-based) for the conjunctive path queries.

Ongoing and open research directions: $\mathcal{T}^+$

In general, we can establish that:

> *$\mathcal{T}^+$, $\exists FO_2^3$, SPII, and TarskiLog are equivalent in expressive power.*

Hence we have four natural alternative syntaxes (algebraic, declarative, graphical, and rule-based) for the conjunctive path queries.

Leveraging the Coupling Theorem, we have been developing structural indexes and query evaluation methods for $\mathcal{T}^+$. We are able to demonstrate across a wide range of scenarios up to 3 orders of magnitude speed-up over the state of the art, while being maintainable and without increasing index size (arXiv 2020).

Ongoing and open research directions

In addition to the affordances of structural indexes, $\mathcal{T}^+$ has many other nice properties.

For example, it is known that all queries expressible in conjunctive finite variable logics have bounded treewidth (Kolaitis and Vardi 2000)

- in the case of $\mathcal{T}^+$, treewidth 2.

Hence, reasoning about $\mathcal{T}^+$ (i.e., query evaluation, static analysis, query minimization) is practical.

Ongoing and open research directions

(2) Study other basic issues in graphs, such as uncertain/dirty data, reasoning about time, and distributed query processing

- path queries on uncertain temporal knowledge graphs

Ongoing and open research directions

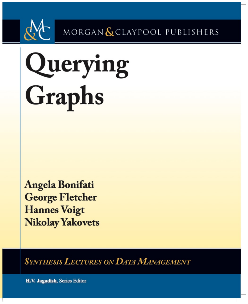
(2) Study other basic issues in graphs, such as uncertain/dirty data, reasoning about time, and distributed query processing

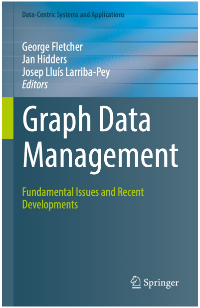
- path queries on uncertain temporal knowledge graphs

(3) Study other basic applications of structural characterizations of query languages, e.g.,

- query language design in social network analysis (cf. Marx and Masuch, *Social Networks* 25(1), 2003; Fan ICDT 2012)
- structure-sensitive privacy and security mechanisms
- dynamic structure (e.g., schema) discovery, via language-distinguishability
- visualizing language-induced structures (e.g., interplay of "schema" knowledge)

What we talk about when we talk about graphs





Thanks very much! Questions?

George Fletcher
Database Group
Eindhoven University of Technology