Interactive Inference of Join Queries

Angela Bonifati Radu Ciucanu Sławek Staworko

University of Lille, France
INRIA Lille – Nord Europe
Links Project

EDBT'14
March 27, 2014

Table of contents

Context

Large amount of **data**

- Coming from disparate data sources
- Shaped as denormalized tables
- Carrying little knowledge of metadata

Example: a denormalized table result of an integration scenario

Non-expert **user**

- Willing to query the data
- Unfamiliar with language formalisms

We investigate the problem of **query inference** from **user interactions**.

Example

A set of tuples (no knowledge of integrity constraints)

Flight			*Hotel*	
From	*To*	*Airline*	*City*	*Discount*
Paris	Lille	AF	NYC	AA
Paris	Lille	AF	Paris	None
Paris	Lille	AF	Lille	AF
Lille	NYC	AA	NYC	AA
Lille	NYC	AA	Paris	None
Lille	NYC	AA	Lille	AF
NYC	Paris	AA	NYC	AA
NYC	Paris	AA	Paris	None
NYC	Paris	AA	Lille	AF
Paris	NYC	AF	NYC	AA
Paris	NYC	AF	Paris	None
Paris	NYC	AF	Lille	AF

Many possible n-ary predicates

$From = City$
$To = City$
$Airline = Discount$
$To = City \wedge Airline = Discount$
.
.
.

Example

A set of tuples (no knowledge of integrity constraints)

Flight			*Hotel*	
From	*To*	*Airline*	*City*	*Discount*
Paris	Lille	AF	NYC	AA
Paris	Lille	AF	Paris	None
Paris	Lille	AF	Lille	AF
Lille	NYC	AA	NYC	AA
Lille	NYC	AA	Paris	None
Lille	NYC	AA	Lille	AF
NYC	Paris	AA	NYC	AA
NYC	Paris	AA	Paris	None
NYC	Paris	AA	Lille	AF
Paris	NYC	AF	NYC	AA
Paris	NYC	AF	Paris	None
Paris	NYC	AF	Lille	AF

Many possible n-ary predicates

$From = City$
$To = City$
$Airline = Discount$
$To = City \wedge Airline = Discount$
.
.
.

Interactive scenario

- We present tuples to the user
- The user labels the tuples as **positive** or **negative examples**

Query inference via simple tuple labeling

	From	*To*	*Airline*	*City*	*Discount*	
−	Paris	Lille	AF	NYC	AA	(1)
−	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)

Consistent predicates:

$Airline = Discount$

$To = City$

$To = City \wedge Airline = Discount$

Query inference via simple tuple labeling

	From	*To*	*Airline*	*City*	*Discount*	
−	Paris	Lille	AF	NYC	AA	(1)
−	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)
+	Lille	NYC	AA	NYC	AA	(4)
−	Lille	NYC	AA	Paris	None	(5)
−	Lille	NYC	AA	Lille	AF	(6)
−	NYC	Paris	AA	NYC	AA	(7)

Consistent predicates:

$Airline = Discount$

$To = City$

$To = City \wedge Airline = Discount$

~~$Airline = Discount$~~

Query inference via simple tuple labeling

	From	*To*	*Airline*	*City*	*Discount*	
−	Paris	Lille	AF	NYC	AA	(1)
−	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)
+	Lille	NYC	AA	NYC	AA	(4)
−	Lille	NYC	AA	Paris	None	(5)
−	Lille	NYC	AA	Lille	AF	(6)
−	NYC	Paris	AA	NYC	AA	(7)
−	NYC	Paris	AA	Paris	None	(8)

Consistent predicates:

$Airline = Discount$

$To = City$

$To = City \wedge Airline = Discount$

~~$Airline = Discount$~~

~~$To = City$~~

Query inference via simple tuple labeling

	From	*To*	*Airline*	*City*	*Discount*	
−	Paris	Lille	AF	NYC	AA	(1)
−	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)
+	Lille	NYC	AA	NYC	AA	(4)
−	Lille	NYC	AA	Paris	None	(5)
−	Lille	NYC	AA	Lille	AF	(6)
−	NYC	Paris	AA	NYC	AA	(7)
−	NYC	Paris	AA	Paris	None	(8)
−	NYC	Paris	AA	Lille	AF	(9)
−	Paris	NYC	AF	NYC	AA	(10)
−	Paris	NYC	AF	Paris	None	(11)
−	Paris	NYC	AF	Lille	AF	(12)

Consistent predicates:

$Airline = Discount$

$To = City$

$To = City \wedge Airline = Discount$

~~$Airline = Discount$~~

~~$To = City$~~

Output:

$To = City \wedge Airline = Discount$

Query inference via simple tuple labeling

	From	*To*	*Airline*	*City*	*Discount*	
−	Paris	Lille	AF	NYC	AA	(1)
−	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)
+	Lille	NYC	AA	NYC	AA	(4)
−	Lille	NYC	AA	Paris	None	(5)
−	Lille	NYC	AA	Lille	AF	(6)
−	NYC	Paris	AA	NYC	AA	(7)
−	NYC	Paris	AA	Paris	None	(8)
−	NYC	Paris	AA	Lille	AF	(9)
−	Paris	NYC	AF	NYC	AA	(10)
−	Paris	NYC	AF	Paris	None	(11)
−	Paris	NYC	AF	Lille	AF	(12)

Consistent predicates:

~~*Airline = Discount*~~

~~*To = City*~~

$To = City \wedge Airline = Discount$

Only some tuples contribute to the inference process

- The **order** of presenting tuples to the user is important
- We want to **minimize** the number of user interactions

Problem statement

Problem statement

Setting

- Two relations, no knowledge of the integrity constraints
- User input via simple tuple labeling ($+$ and $-$ examples)

Objective

- Infer an arbitrary n-ary join predicate across the two relations
- Minimize the number of interactions with the user

Problem statement

Setting

- Two relations, no knowledge of the integrity constraints
- User input via simple tuple labeling (+ and – examples)

Objective

- Infer an arbitrary n-ary join predicate across the two relations
- Minimize the number of interactions with the user

Contributions

- Equijoins
  - Polynomial characterization of uninformative tuples
  - Practical strategies of presenting tuples to the user
  - Experimental validation on TPC-H and synthetic datasets
- Semijoins
  - Intractability of the consistency checking

Inference of equijoins

Equijoins

Two relations

$R =$

From	*To*	*Airline*
Paris	Lille	AF
Lille	NYC	AA
NYC	Paris	AA
Paris	NYC	AF

$P =$

City	*Discount*
NYC	AA
Paris	None
Lille	AF

Equijoin predicate

$$\theta = \{(\textit{To}, \textit{City}), (\textit{Airline}, \textit{Discount})\}$$

Semantics

$R \bowtie_\theta P =$

From	*To*	*Airline*	*City*	*Discount*
Paris	Lille	AF	Lille	AF
Lille	NYC	AA	NYC	AA

Consistency checking

Problem

- An **example** is a pair $(t, +)$ or $(t, -)$, where $t \in R \times P$
- A **sample** is a set of positive examples S_+ and negative examples S_-
- Given a sample, there exists a **consistent** equijoin predicate?

Characterization

- Let $T(t) = \{(A_i, B_j) \mid t[A_i] = t[B_j]\}$
  - the most specific predicate selecting a tuple $t \in R \times P$
- Let $T(S_+) = \bigcap_{t \in S_+} T(t)$
  - the most specific predicate selecting all positive examples
- The sample is consistent iff $T(S_+)$ selects no tuple in S_-
- There may exist many consistent predicates

Deciding "uninformativeness"

Uninformative tuples (w.r.t. the sample and the goal query)

- A tuple is **uninformative** if knowing its label does not eliminate any consistent join predicate
- Notion impossible to use without knowledge of the goal query

Certain tuples (w.r.t. the sample only)

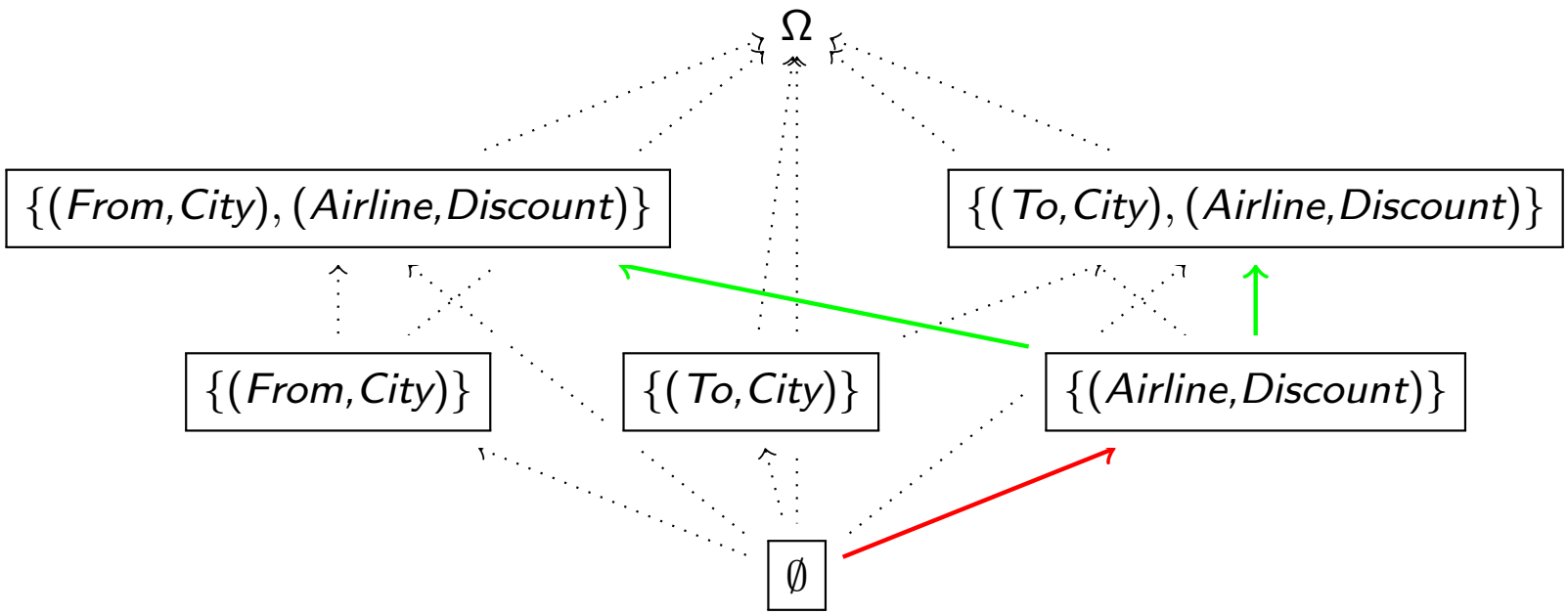
- A tuple is **certain** if either
  1. It is selected by all consistent join predicates or
  2. It is rejected by all consistent join predicate
- Inspired by possible world semantics and certain answers

Theorem

1. Uninformative = Certain
2. Deciding whether a tuple is certain is in PTIME

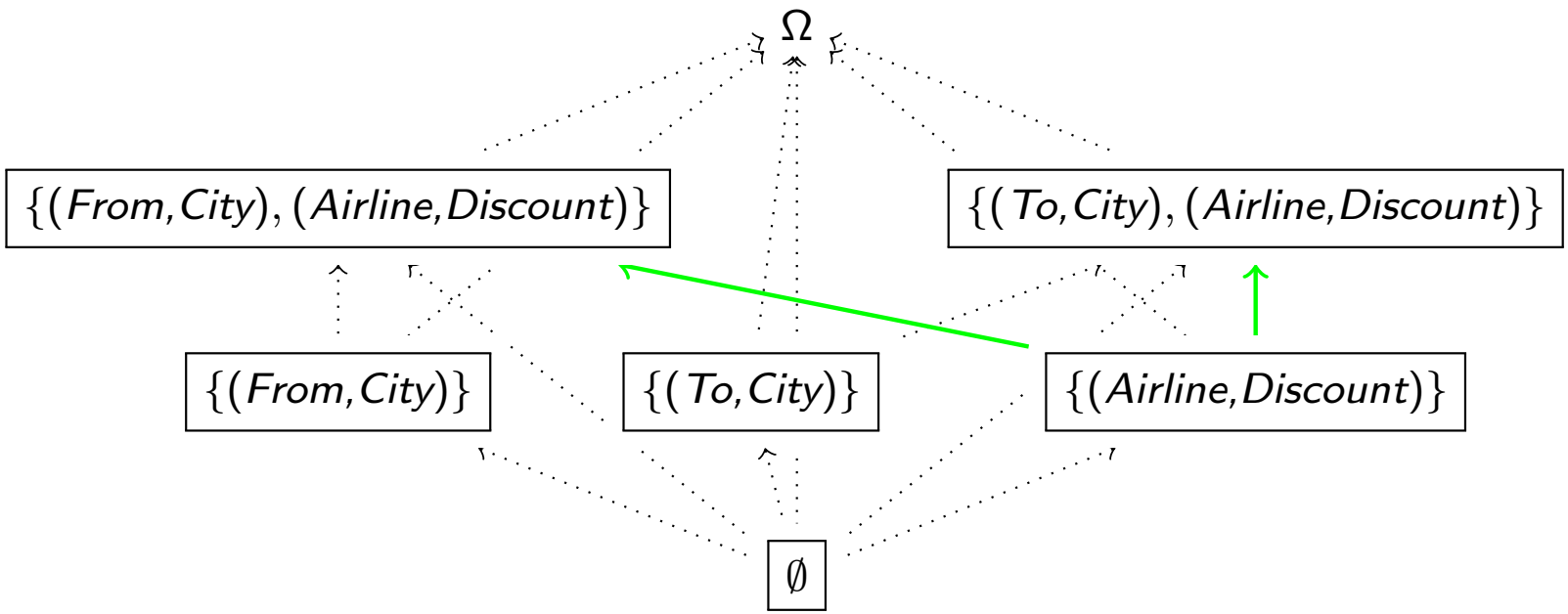
Lattice of join predicates for deciding uninformativeness

From	*To*	*Airline*	*City*	*Discount*	$T(t)$
Paris	Lille	AF	NYC	AA	$\emptyset$
Paris	Lille	AF	Paris	None	$\{(From, City)\}$
Paris	Lille	AF	Lille	AF	$\{(To, City), (Airline, Discount)\}$
Lille	NYC	AA	NYC	AA	$\{(To, City), (Airline, Discount)\}$
Lille	NYC	AA	Paris	None	$\emptyset$
Lille	NYC	AA	Lille	AF	$\{(From, City)\}$
NYC	Paris	AA	NYC	AA	$\{(From, City), (Airline, Discount)\}$
NYC	Paris	AA	Paris	None	$\{(To, City)\}$
NYC	Paris	AA	Lille	AF	$\emptyset$
Paris	NYC	AF	NYC	AA	$\{(To, City)\}$
Paris	NYC	AF	Paris	None	$\{(From, City)\}$
Paris	NYC	AF	Lille	AF	$\{(Airline, Discount)\}$



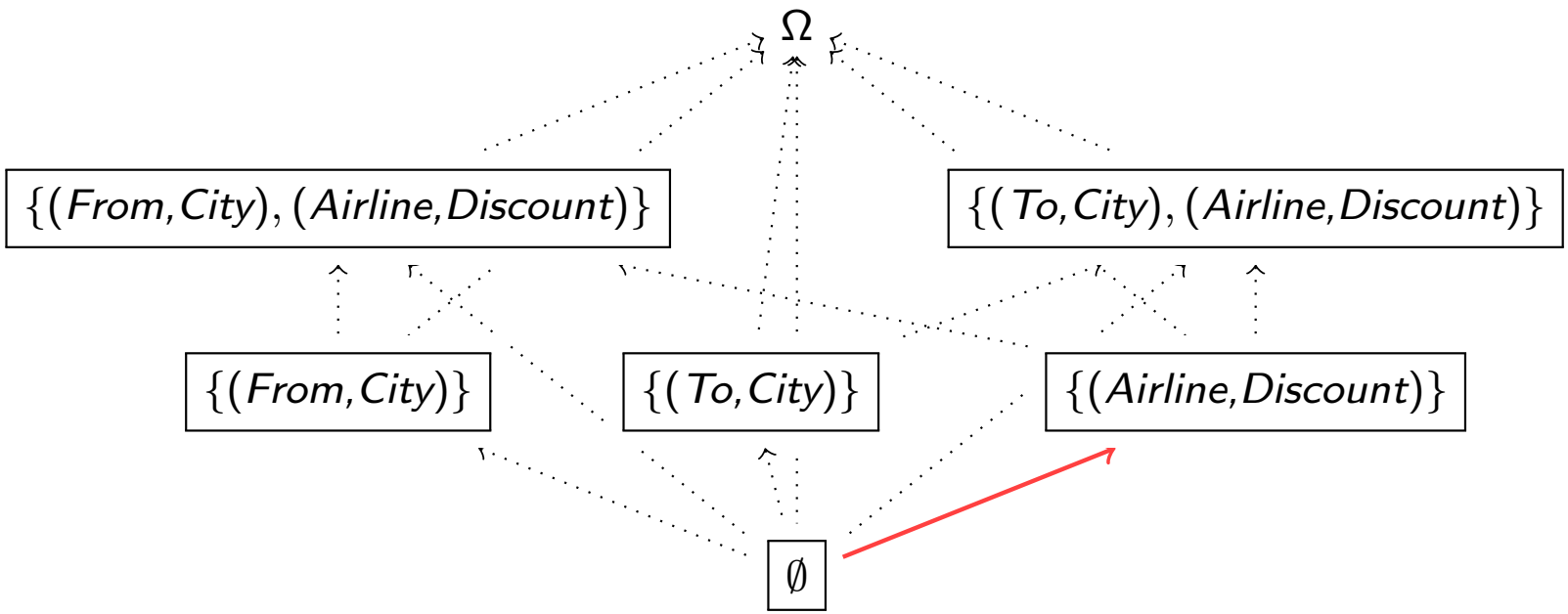
Lattice of join predicates for deciding uninformativeness

From	*To*	*Airline*	*City*	*Discount*	$T(t)$
Paris	Lille	AF	NYC	AA	$\emptyset$
Paris	Lille	AF	Paris	None	$\{(From, City)\}$
Paris	Lille	AF	Lille	AF	$\{(To, City), (Airline, Discount)\}$
Lille	NYC	AA	NYC	AA	$\{(To, City), (Airline, Discount)\}$
Lille	NYC	AA	Paris	None	$\emptyset$
Lille	NYC	AA	Lille	AF	$\{(From, City)\}$
NYC	Paris	AA	NYC	AA	$\{(From, City), (Airline, Discount)\}$
NYC	Paris	AA	Paris	None	$\{(To, City)\}$
NYC	Paris	AA	Lille	AF	$\emptyset$
Paris	NYC	AF	NYC	AA	$\{(To, City)\}$
Paris	NYC	AF	Paris	None	$\{(From, City)\}$
Paris	NYC	AF	Lille	AF	$\{(Airline, Discount)\}$



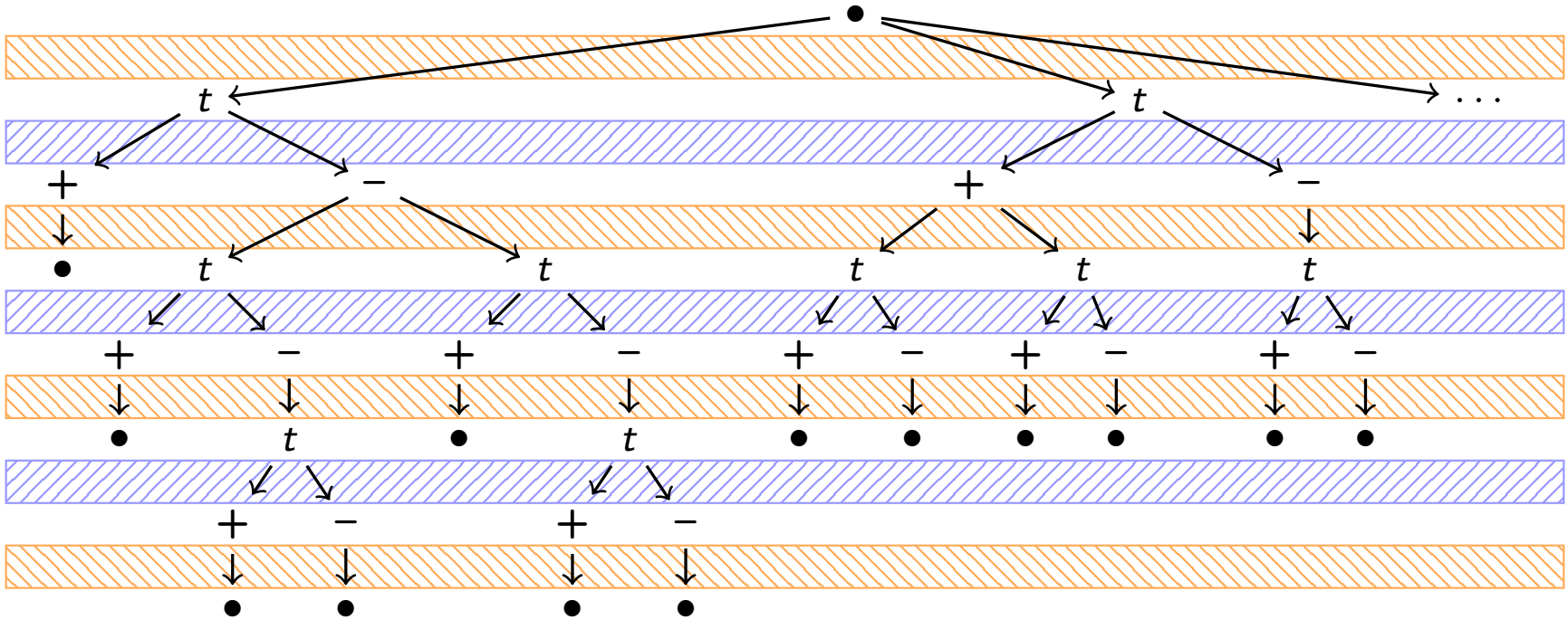
Lattice of join predicates for deciding uninformativeness

From	*To*	*Airline*	*City*	*Discount*	$T(t)$
Paris	Lille	AF	NYC	AA	$\emptyset$
Paris	Lille	AF	Paris	None	$\{(From, City)\}$
Paris	Lille	AF	Lille	AF	$\{(To, City), (Airline, Discount)\}$
Lille	NYC	AA	NYC	AA	$\{(To, City), (Airline, Discount)\}$
Lille	NYC	AA	Paris	None	$\emptyset$
Lille	NYC	AA	Lille	AF	$\{(From, City)\}$
NYC	Paris	AA	NYC	AA	$\{(From, City), (Airline, Discount)\}$
NYC	Paris	AA	Paris	None	$\{(To, City)\}$
NYC	Paris	AA	Lille	AF	$\emptyset$
Paris	NYC	AF	NYC	AA	$\{(To, City)\}$
Paris	NYC	AF	Paris	None	$\{(From, City)\}$
Paris	NYC	AF	Lille	AF	$\{(Airline, Discount)\}$



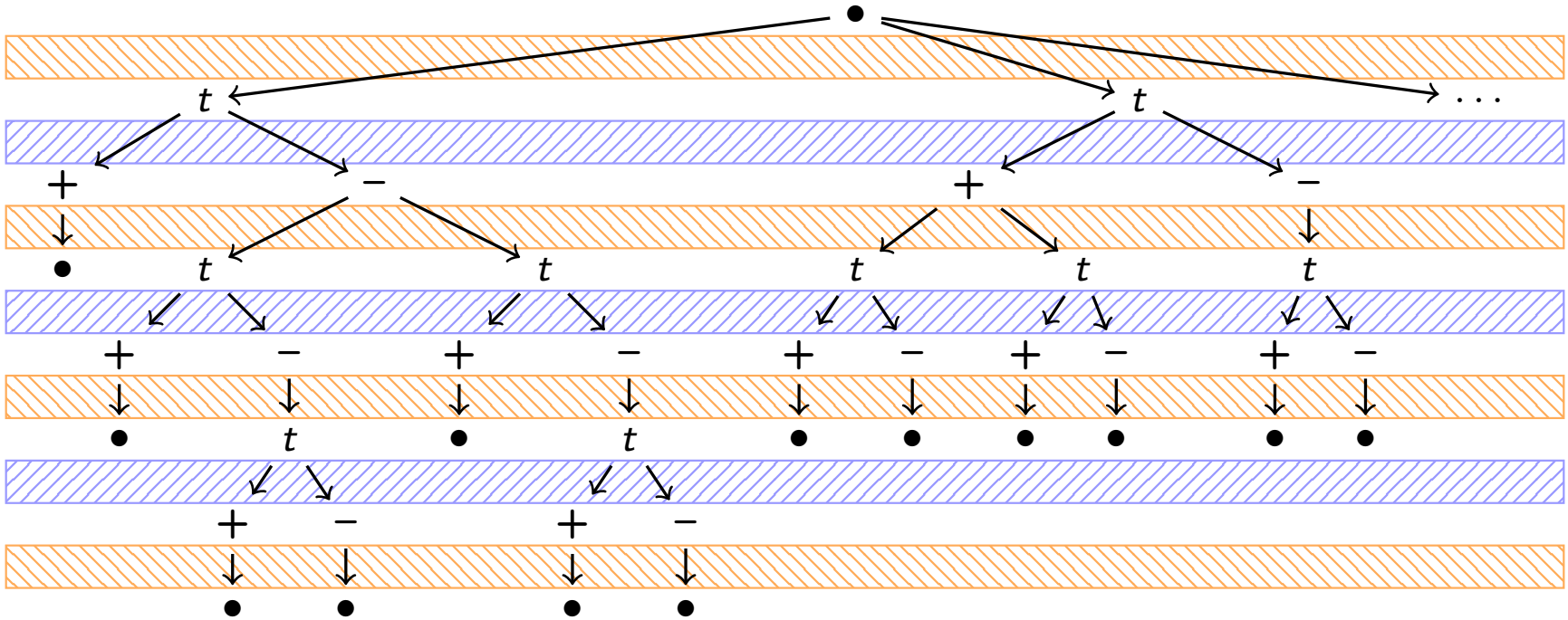
Strategies

Optimal strategy – minimax algorithm



Strategies

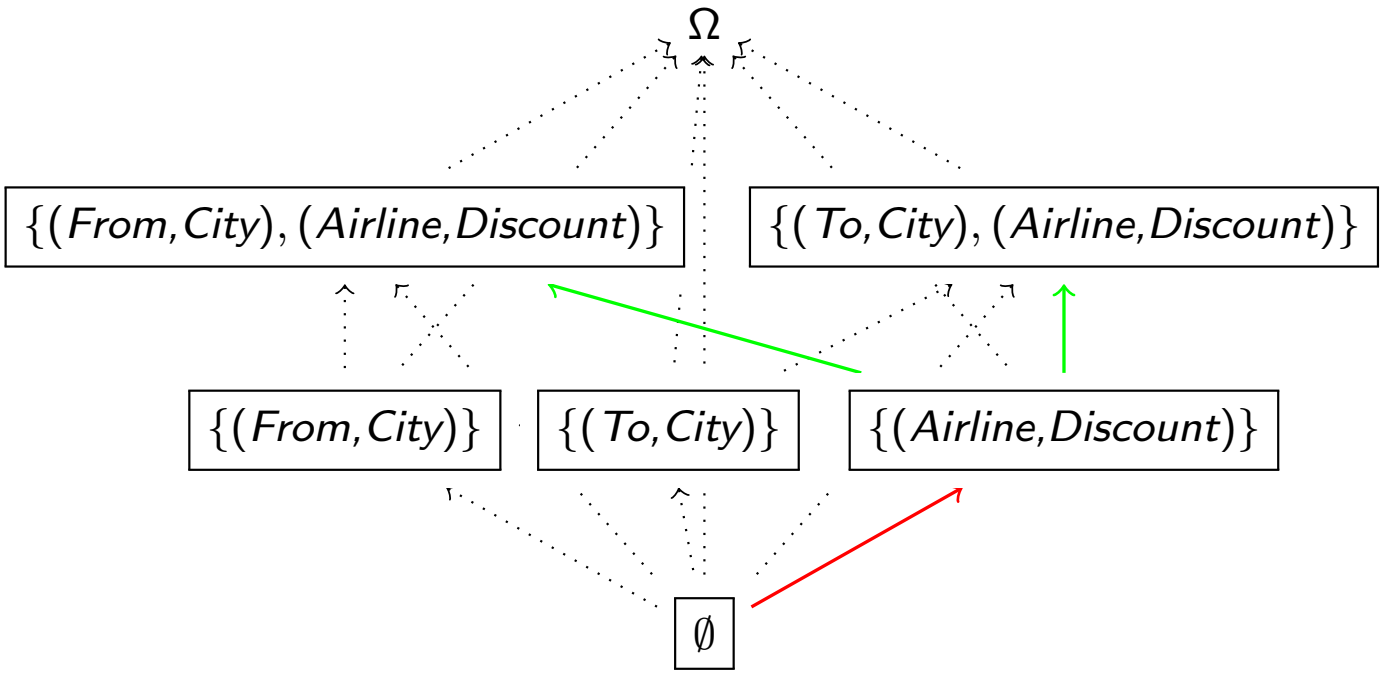
Optimal strategy – minimax algorithm



- A straightforward implementation requires exponential time
- The exact complexity remains an open question

Practical strategies

- Local strategies – follow a simple order on the lattice
  - Bottom-up
  - Top-down



- Lookahead strategies – take into account the entropy of a tuple
  - *k*-step lookahead (optimal for "big" k)

Experiments

We measured the number of interactions and inference time for:

start with an empty sample
while there are informative tuples left
    choose a tuple according to a **strategy**
    ask the user to **label** the chosen tuple

Experiments

We measured the number of interactions and inference time for:

start with an empty sample
while there are informative tuples left
    choose a tuple according to a **strategy**
    ask the user to **label** the chosen tuple

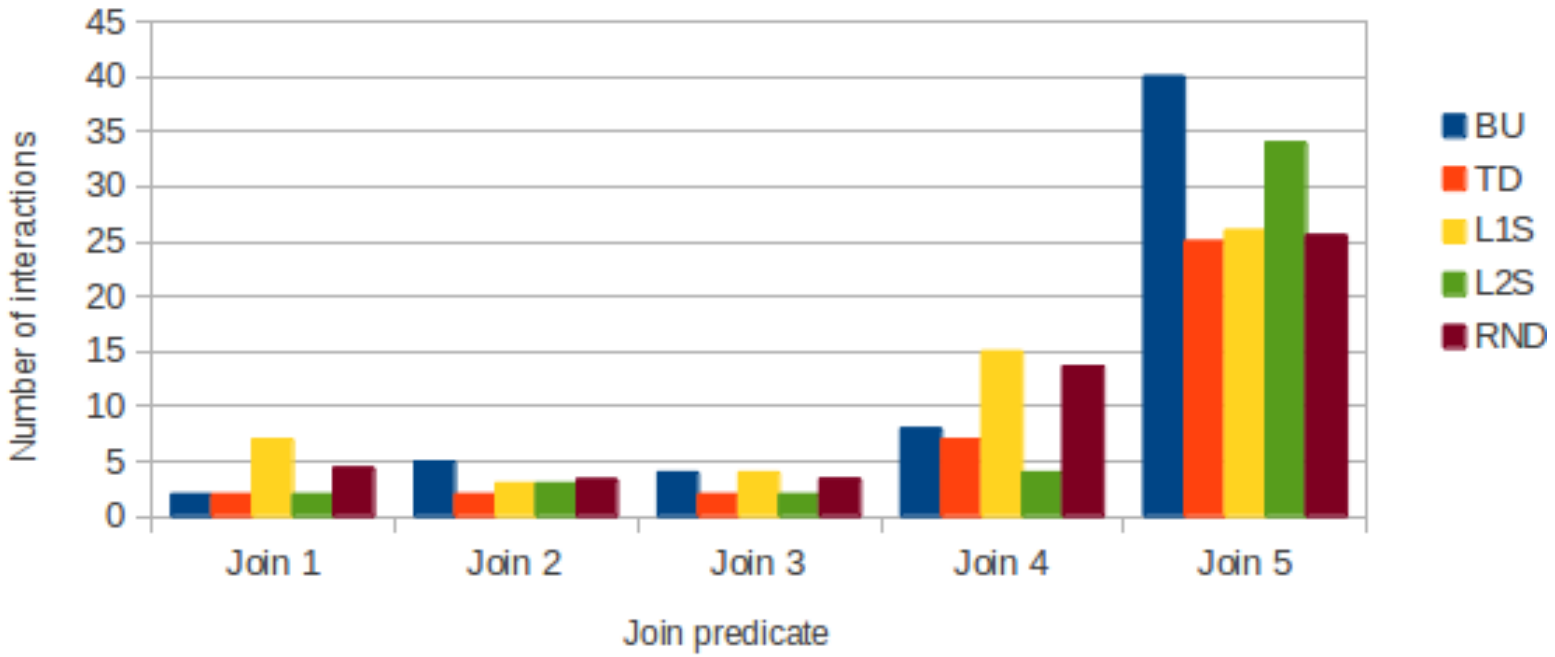
Datasets

- TPC-H benchmark – joins of size 1,2 (key-foreign key relationships)
  - **Difficulty**: a value "15" may refer to a key, size, price, quantity, etc.
  - Scaling factors from 1 to 100000
- Synthetic datasets – joins of size $0, 1, \ldots, 4$

Experimental results

- The performances are influenced by the **instance** and the **join**'s size
- Two-step lookahead is a good practical compromise

TPC-H experiments

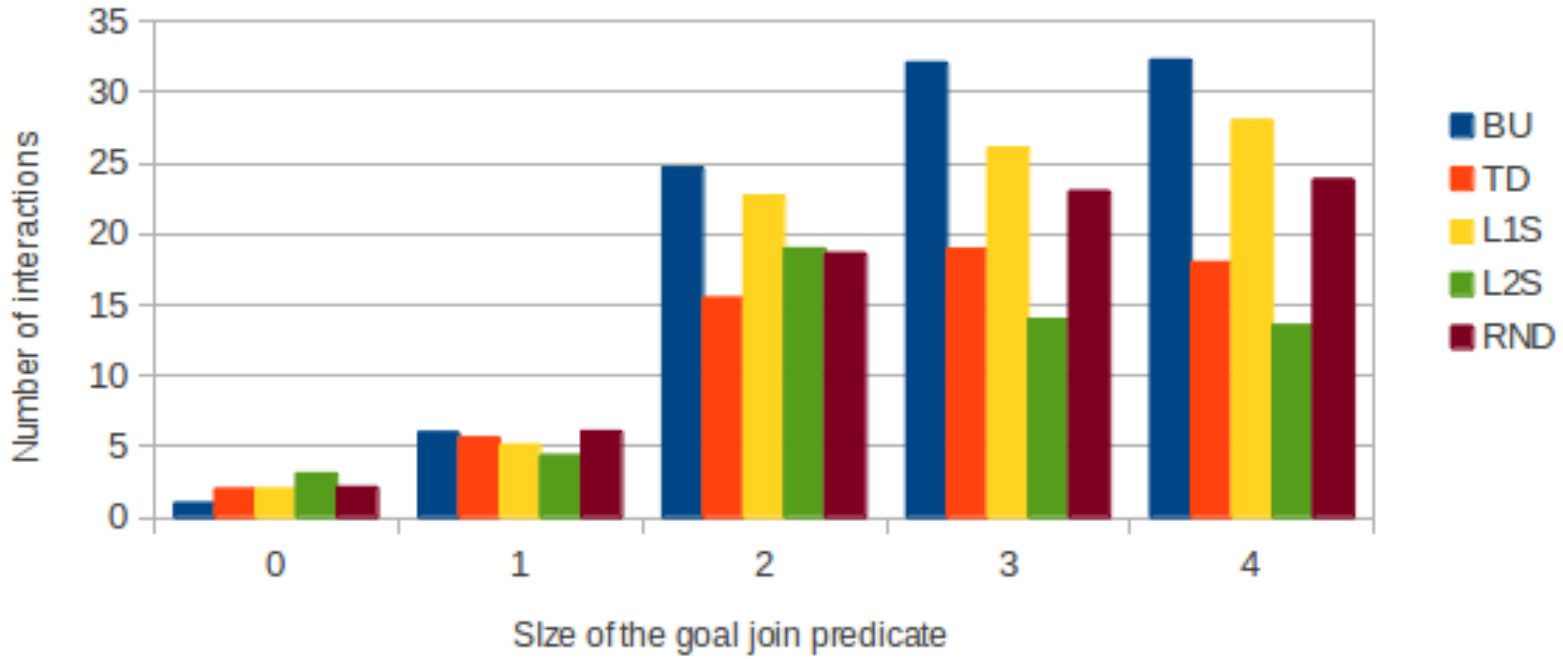


Results

- Joins of bigger size require more interactions
- The instance for Join 4 has a "dense" lattice

Synthetic experiments

2 tables $\times$ 3 attributes $\times$ 100 lines $\times$ 100 possible values.



Results

- Joins in the "intermediate" part of the lattice are particularly difficult
- Two-step lookahead is a good practical compromise

Conclusions and future work

Conclusions

We studied the problem of inferring join queries from user interactions.

- Inference of equijoins
  - Polynomial characterization of uninformative tuples
  - Practical strategies of presenting tuples to the user
  - Experimental validation on TPC-H and synthetic datasets
- Intractability of semijoins
  - Consistency checking is NP-complete

Related work

1. Inference of relational queries:
   - Reverse engineering complex join queries [Zhang et al., SIGMOD'13]
   - Query by output [Tran et al., SIGMOD'09]
   - Synthesizing view definitions from data [Das Sarma et al., ICDT'10]
2. Inference of integrity constraints:
   - Discovering conditional functional dependencies [Fan et al., TKDE'11]
3. Learning with membership queries [Angluin, Machine Learning'88]:
   - Learning quantified queries [Abouzied et al., PODS'13]

Differences

- 1. and 2. – more expressive concepts, but no interactions
  - We focus on a simpler class of queries, by exploiting user interactions
- 1. and 3. – (partial) knowledge of the integrity constraints
  - We assume no knowledge of integrity constraints

Applications

- Interactively infer integrity constraints when missing
  - Then apply existing algorithms for inferring complex join queries
- Joining datasets using crowdsourcing [Marcus et al., VLDB'12]
  - Minimizing user interactions $\rightarrow$ minimizing the financial cost
  - We handle arbitrary n-ary join predicates
- Schema mapping inference
  - Our queries can be seen as simple GAV mappings
  - We assume a less expert user than [Alexe et al., SIGMOD'11]

Future work

- Extensions of our approach:
  - Other algebraic operators and join paths
  - Interactive inference of integrity constraints
  - Interactive inference of graph queries
- Applications of our algorithms:
  - Demo where human users provide the examples