Lecture 3: Variants of the Chase Algorithm in Data Exchange

Angela Bonifati

University of Lyon 1
Liris – CNRS, France

March 11, 2016

Table of contents

1. The chase variants in Data Exchange
   - Roadmap of the state-of-art chase algorithms
2. The Interleaved Chase with Parallelization
   - Interleaved Chase: our contribution

Background: constraints in Data Exchange

Setting (in the relational case)[a]

[a] Fagin et al.: Data exchange: semantics and query answering. TCS 2005

S **source schema** – a finite collection of relations $R_1, \ldots, R_k$

T **target schema** – a finite collection of relations $R_1, \ldots, R_l$

Σ_{st} **s-t tgds** – $\forall \overline{x}.\ (\phi_S(\overline{x}) \rightarrow \exists \overline{y}.\ \psi_T(\overline{x}, \overline{y}))$

Σ_t **t tgds** – $\forall \overline{x}.\ (\phi_T(\overline{x}) \rightarrow \exists \overline{y}.\ \psi_T(\overline{x}, \overline{y}))$

Σ_t **t egds** – $\forall \overline{x}.\ (\psi_T(\overline{x}) \rightarrow (x_1 = x_2))$

s-t tgds in Σ_{st} and t tgds + t egds in Σ_t

- s-t tgds and t tgds: adding missing facts to the target instance J;
- t egds: equate terms in the target instance J or fail (due to clashes)

Data Exchange: Finding solutions

The Chase in general

- the Chase is a straightforward procedure of constraint enforcement

The Chase in Data Exchange – in reality a two-phase chase

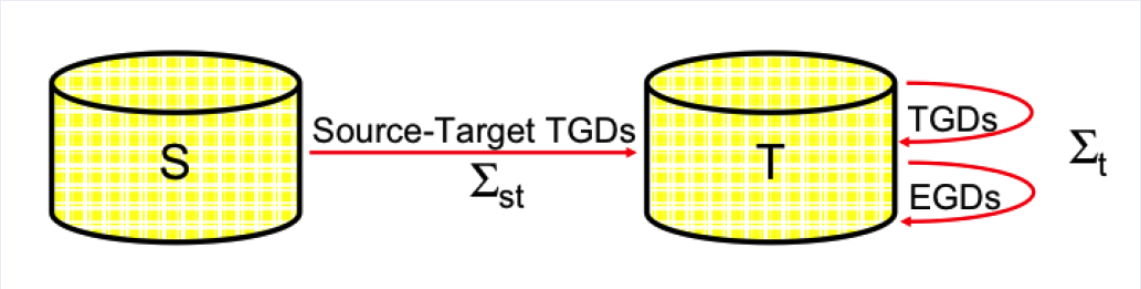


Figure: Exchanging data from source to target.

- source-to-target chase: apply Σ_{st} to I and obtain J^{pre} (also called a pre-solution);
- target chase: apply Σ_t to J^{pre} and obtain J (also called a solution);

The Stardard chase procedure for tgds and egds [Fagin et al.TCS'05].

Given a source instance I

1. Use the naïve chase to chase I with Σ_{st} and obtain a target instance J^{pre} (pre-solution).
2. Chase J^{pre} with the target tgds and the target egds in Σ_t to obtain a target instance J as follows:
   - 2.1 For target tgds introduce new facts in J as dictated by the right-hand-side of the s-t tgd and introduce new values (variables) in J each time existential quantifiers need witnesses.
   - 2.2 For target egds $\phi(\bar{x}) \rightarrow x_1 = x_2$
     - 2.2.1 If a variable is equated to a constant, replace the variable by that constant;
     - 2.2.2 If one variable is equated to another variable, replace one variable by the other variable.
     - 2.2.3 If one constant is equated to a different constant, stop and report "failure".

The naïve chase procedure in pseudocode

Input: $M = (S, T, \Sigma_{st})$,**SrcInstance** I
Output: **TgtInstance** J: a universal target solution for I w.r.t. M
$J = \emptyset$;
for $\forall \overline{x}.\ (\phi_S(\overline{x}) \rightarrow \exists \overline{y}.\ \psi_T(\overline{x}, \overline{y})) \in \Sigma_{st}$ **do**
    for all tuples of constants $\overline{a}$ such that $I \models \phi_S(\overline{a})$ **do**
        for each $y_i \in \overline{y}$, pick a fresh null value N_i for y_i
        add the facts in $\psi_T(\overline{a}, \overline{N})$ to J
    end for
end for
Return J

Recap of the relational case: definitions

Existence of solutions

Given a setting $\Omega = (S, T, \Sigma_{st}, \Sigma_t)$ and an instance I of S, decide whether there exists a solution for I under Ω, namely the existence of solutions problem

- is undecidable for schema mappings in which Σ_t are arbitrary tgds and egds;
- is in PTIME for schema mappings in which Σ_t is the union of a weakly acyclic set of tgds and a set of egds.

Query Answering

- Given a setting $\Omega = (S, T, \Sigma_{st}, \Sigma_t)$, an instance I of S, and a query Q over T, we are interested in the *certain answers of Q w.r.t. I under Ω*, denoted $certain_\Omega(Q, I)$, which are the answers that hold for all solutions i.e., the set $\bigcap\{Q(J) \mid J \in Sol_\Omega(I)\}$.
- The query answering problem consists of deciding whether a given tuple of constants belongs to $certain_\Omega(Q, I)$ or not.

Computing the certain answers

Standard Semantics for Query Answering

Theorem (FKM'05): Schema mapping M = (S, T, Σ_{st}, Σ_t) such that: Σ_{st} a set of source-to-target tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries over T. If I is a source instance and J is a universal solution for I, then

- certain(q,I) = the set of all 'null-free' tuples in q(J);
- certain(q,I) is computable in time polynomial in the size of I.

Computation of certain(q,I) in polynomial time

1. Compute a canonical universal J solution in polynomial time;
2. Evaluate q(J) and remove tuples with nulls.

Chase Termination: Weakly Acyclic Dependencies

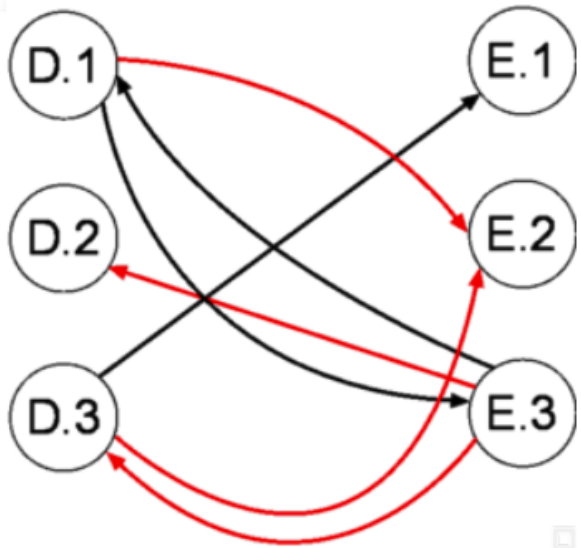
Build the dependency graph

- Σ_t is weakly acyclic iff there is no cycle trough an existential edge.
- if Σ_t is weakly acyclic, all chase sequences terminate.

Example – dependency graph

Σ_t $Departments(d, n, m) \rightarrow \exists N Employees(m, N, d)$

Σ_t $Employees(e, n, d) \rightarrow \exists D, M Departments(d, D, M)$



Chase Variants: the Naïve (or Oblivious) chase vs. the Standard Chase

The Oblivious chase

- Cali et al.[a] has introduced the Oblivious Chase by not relying on any order;
- At each iteration, all triggers are considered and not only the active ones as in the standard chase algorithm;
- Consider the tgd $\forall x, y\ R(x, y) \rightarrow \exists z\ T(x, z)$ and the instance $I = \{R(a, b), R(a, c), R(d, e), T(a, a)\}$:
  - then, $J^{obl} = \{T(a, a), T(d, X_1), T(a, X_2), T(a, X_3)\}$
  - whereas $J^{std} = \{T(a, a), T(d, X_1)\}$

[a] Andrea Calí, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *KR'08*.

Chase Variants: the Semi-oblivious chase

The Semi-Oblivious chase

- Marnette[a] has introduced the Semi-oblivious Chase, which relies on the Oblivious one;
- At each iteration, only one trigger for each equivalence class is considered and not all the triggers as in the Oblivious variant;
- Consider the tgd $\forall x, y\ R(x, y) \rightarrow \exists z\ T(x, z)$ and the instance $I = \{R(a, b), R(a, c), R(d, e), T(a, a)\}$:
  - then, $J^{obl} = \{T(a, a), T(d, X_1), T(a, X_2), T(a, X_3)\}$
  - whereas $J^{sobl} = \{T(a, a), T(d, X_1), T(a, X_2)\}$

[a] Bruno Marnette. Generalized schema-mappings: from termination to tractability. *PODS'09*.

Chase Variants: the Core chase

The Core chase

- Deutsch et al.[a] has introduced the Core Chase, which produces the core solution;
- At each iteration, all triggers are applied simultaneously and a minimization step follows to produce the core;
- It may affect termination, namely tgds that do not terminate in the Standard Chase might terminate with the Core chase.
- Consider the tgds: $m_1 :\rightarrow \exists u, v\ E(u, v), E(v, u)$
  $m_2 : E(x, y), E(y, x) \rightarrow \exists u\ E(u, u)$
  $m_3 : E(x, y) \rightarrow \exists u\ E(x, u), E(u, y)$
  - In the first step, the constraint m_1 fires to give a cycle of length 2 and the minimization step does nothing.
  - In the second step, both constraints m_2 and m_3 fire, the latter twice on the 2-cycle. Now the minimization step reduces everything to the self-loop introduced by m_2. At this point, the core chase terminates.

[a] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. *PODS'08*.

All these Chase Variants: Lessons Learned

What did we learn?

- the chase variant most implemented in Data Exchange engines is the standard chase (sometimes the Semi-oblivious one);
- the Core chase is based on the fact that s-t tgds can be evaluated in parallel, but not egds (they may modify the original instance by equating existing labeled nulls to constants and other labeled nulls);
- we also learned that the problem of termination depends on which chase variant we eventually choose and on the order in which we apply the constraints; this is the case even with tgds and egds (e.g. $R(x, y) \rightarrow \exists z R(y, z)$ and $R(x, y) \rightarrow y = b$ with $I = \{R(a, b)\}$ terminates without failure if the egd is applied early enough)

Trends of the chase implementations

Existing Implemented Chase Engines

- Clio (IBM InfoSphere) and ++Spicy (open source)[a]: they implement the standard chase (actually the naïve chase), since they mainly focus on Σ_{st} constraints (or they strive to find a rewriting into Σ_{st}); ++Spicy directly generates the core solution and exports SQL scripts.
- the Vadim Savenkov's chase engine [b](in Java) is the most faithful and complete implementation of the standard chase (as we defined it above) with a core minimization;
- the Llunatic[c] chase engine is relying on the parallel chase to evaluate complex classes of constraints (beyond data exchange constraints, and including data cleaning constraints, e.g. conditional functional dependencies).

---

[a] E.g., `http://www.db.unibas.it/projects/spicy/`

[b] Reinhard Pichler, Vadim Savenkov: Towards practical feasibility of core computation in data exchange. *TCS'10*.

[c] E.g., `http://www.db.unibas.it/projects/llunatic/`

The standard chase in SQL

Consider an example of s-t tgd:

$P(n, c) \rightarrow \exists I, S \; C(I, n, S)$

Skolemize it as follows:

$P(n, c) \rightarrow C(f_I(n), n, f_S(n))$

Translate it in SQL:

```
INSERT into C
SELECT append('f_I(',P.n,')'), P.n,
append('f_S(',P.n,')')
FROM P
```

Interplay between s-t tgds and t egds within the Chase[1]

Existing approaches that deal with t egds are best-effort

- Marnette et al. VLDB'10 presented an algorithm for handling egds in the relational case, under an additional assumption (*): a set of source constraints Σ_s is available
- The crux is to rewrite a set of s-t tgds plus t egds into a set of s-t tgds to obtain a FO implementation (and, thus, run the naïve chase)
- When Σ_s is available, and under particular assumptions, a core solution can be computed via an SQL script
- Unfortunately, both SQL-based chase engines and custom ones have exhibited limited performances in the presence of target egds.



[1] Joint work with Michele Linardi and Ioana Ileana (under submission).

A Data Exchange Scenario with Target Functional Dependencies

Moving actors information from source to target

The s-t tgds of this scenario are the following:

$\mathbf{m_1}$: $Active_Actor(n, s, a) \rightarrow Actor(n, s, a, Y_1, Y_2)$

$\mathbf{m_2}$: $Awarded_Actor(n', s', a', p', w') \rightarrow Actor(n', s', a', T, T_1) \wedge Oscar_Prize(p', w', T)$

$\mathbf{m_3}$: $Actor_Collaboration(n'', s'', n''', s''') \rightarrow Actor(n'', s'', A, E_1, E_2) \wedge Actor(n''', s''', A_1, E_3, E_2)$

The egd rules (which state the primary keys of the relations Actor and Oscar_Prize):

$\mathbf{e_1}$: $Actor(n, s, a, p, w) \wedge Actor(n, s, a, p', w') \rightarrow (p = p') \wedge (w = w')$

$\mathbf{e_2}$: $Oscar_Prize(p, w, z) \wedge Oscar_Prize(p, w, z') \rightarrow (z = z')$

The solution J'^1 without FDs contains redundant tuples

(i) Instance I

Active_Actors

name	*surname*	*age*
Leonardo	Di Caprio	40
John	Redmayne	33

Actor_Collaboration

$name_1$	$surname_1$	$name_2$	$surname_2$
Leonardo	Di Caprio	Matthew	David
Fredric	March	Miriam	Hopkins

Awarded_Actor

name	*surname*	*oscarName*	*year*
John	Redmayne	Best Actor	2014
Wallace	Beery	Best Actor	1932
Fredric	March	Best Actor	1932
Marlon	Brando Jr.	Best Actor	1954
Marlon	Brando Jr.	Best Actor	1972

(ii) pre-solution J' (does not enforce egds)
(* primary key)

Actor

*name**	*surname**	*idRewarding*	*idClub*
Leonardo	Di Caprio	NULL	NULL
Leonardo	Di Caprio	NULL	C_1
Matthew	David	NULL	C_1
John	Redmayne	NULL	NULL
John	Redmayne	N_1	NULL
Wallace	Beery	N_2	NULL
Fredric	March	N_3	NULL
Fredric	March	NULL	C_2
Miriam	Hopkins	NULL	C_2
Marlon	Brando Jr.	N_4	NULL
Marlon	Brando Jr.	N_5	NULL

Oscar_Prize

*oscarName**	*year**	*idActor*
Best Actor	2014	N_1
Best Actor	1932	N_2
Best Actor	1932	N_3
Best Actor	1954	N_4
Best Actor	1972	N_5

Figure: Exchanging actor data from source (left) to target (right).

[1] To avoid clutter, we use the same NULL instead of multiple labelled nulls, one distinct from another as it should be.

The solution J with FDs shall contain no redundancy

(iii) expected solution J
Actor

*name**	*surname**	*idRewarding*	*idClub*
Leonardo	Di Caprio	NULL	C_1
Matthew	David	NULL	C_1
John	Redmayne	N_1	NULL
Wallace	Beery	N_2	NULL
Fredric	March	N_2	C_2
Miriam	Hopkins	NULL	C_2
Marlon	Brando Jr.	N_4	NULL

Oscar_Prize

*oscarName**	*year**	*idActor*
Best Actor	2014	N_1
Best Actor	1932	N_2
Best Actor	1954	N_4
Best Actor	1972	N_4

How can we obtain this solution?

- Existing algorithms, such as Marnette et al. VLDB'10, would fail (no Σ_s).
- By applying the standard chase algorithm [Fagin et al.TCS'05], at every egd enforcement, one needs to check the entire pre-solution.

Our contribution: a new chase variant that interleaves s-t tgds and egds.

We first make the following observations:

- A direct implementation of the Classical Data Exchange Chase procedure described above, building the pre-solution before proceeding to egd application, will in general fail to achieve practical performance (the larger the pre-solution, the more expensive the final egd application will get).
- The Oblivious Chase allows for more flexibility in the choice of the chase sequences. However, which one will end up performing better than others?
- We introduce a new chase-based algorithm, which we call the Interleaved Chase. While the Interleaved Chase relies on Oblivious Chase steps, it will aim at improving performance by *playing on the chase order* and *reducing the egd application scope*. As such, it can also exploit parallelization opportunities in the application of egds.

Definition: Sets of assignments of tgds

$W_{M,I}$

Let A_{m_i} be the set of variable assignments that map variables from $\bar{x}_i, \bar{y}_i$ to constants, for each rule $m_i \in \Sigma_{st}$, we define a collection W_{m_i}, which is the set of assignments in A_{m_i} that make $\phi_i(\bar{x}_i, \bar{y}_i)$ hold in the input source instance I. That is,

$$W_{m_i} = \{a \in A_{m_i} \mid I \models a(\phi_i(\bar{x}_i, \bar{y}_i))\}$$

We denote $W_{M,I}$ the collection of all the W_{m_i}, for all m_i in Σ_{st}.

Main steps of the Algorithm

Algorithm 1 Main loop of the Interleaved Chase

Algorithm Interleaved Chase

Can raise Exception:ChaseFail

Input: $M = (S, T, \Sigma_{st}, \Sigma_t)$,**SrcInstance** I

Output **TgtInstance** J: the target solution

$J = \emptyset$

for $m \in \Sigma_{\text{st}}$ **do**

$A_m = \text{InitialAssignmentSet}(m, I)$

end for

while $\exists m \in \Sigma_{\text{st}}$ s.t. $A_m \neq \emptyset$ **do**

$seed = \text{SelectArbitraryAsg}(A_m)$

$S_{\text{sat}} = \text{BuildAndChaseSaturationSet}(seed)$

$J = J \cup \text{Materialize}(S_{\text{sat}})$

end while

return J

The BuildAndChaseSaturationSet procedure

Saturation set

A saturation set S_{sat} is a subset of facts from the final solution to be added altogether, because they share labelled nulls.

BuildAndChaseSaturationSet

The *BuildAndChaseSaturationSet* subroutine initializes the current Saturation Set with the provided seed assignment. It then iteratively expands the current Saturation Set by embedding the new assignments that have at least an overlap with assignments previously added to this Saturation Set. This is done until a fixpoint is reached.

The conflict graph

Atom overlapping data structure

In order to explicitly correlate the tgds contained in a mapping scenario M according to the overlapping which may occur among their head atoms, we introduce a specific graph-based representation on which our algorithm is grounded.

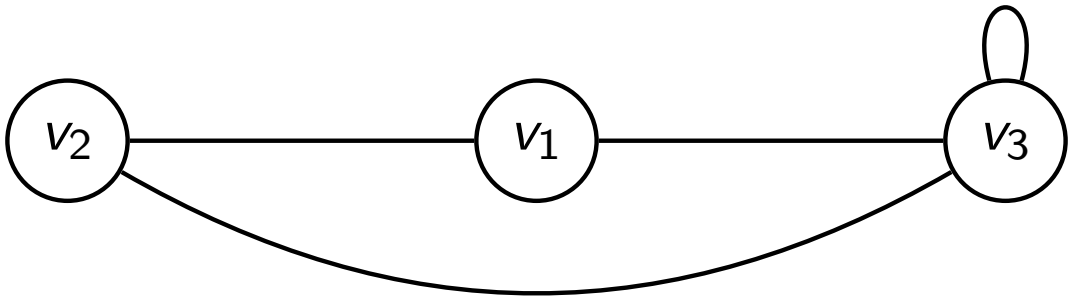
Definition

Given a data exchange scenario, its associated *conflict graph*, denoted by $G^c_{\Sigma_{st}}$, is an undirected (possibly) cyclic graph composed by an ordered pair (V, E), where V is a set of vertices, each of which represents an s-t tgd in Σ_{st}, and E is a set of edges, each edge witnessing the presence of at least one non-trivial conflict between the two s-t tgds corresponding to its incident vertices. Furthermore, every vertex in V is adorned with all the conflict areas for the corresponding tgd.

The fusion graph of our running example on actors

Example

The conflict graph for the s-t tgds of our running example is depicted below. Notice that all the edges are due to the relational atom *Actor*, and no edge is induced by the *Oscar_Prize* table.



v_1, v_2, v_3 are vertices corresponding to the tgds m_1, m_2, m_3

$$\mathbf{Areas(v_1)} = \{ca_1^1 = \langle (n, s), e_1 \rangle\}.$$
$$\mathbf{Areas(v_2)} = \{ca_2^1 = \langle (n', s'), e_1 \rangle, ca_2^2 = \langle (p', w'), e_2 \rangle\}.$$
$$\mathbf{Areas(v_3)} = \{ca_3^1 = \langle (n'', s''), e_1 \rangle, ca_3^2 = \langle (n''', s'''), e_1 \rangle\}.$$

Experimental Study

The iBench data integration benchmark@UToronto

We have used iBench, a recent data integration benchmark for creating arbitrarily large and complex mappings, schemas and schema constraints. We have focused on two primitives in iBench, namely Object Fusion (OF) and VP (Vertical Partitioning).

Experiments on iBench

We have extensively gauged the performance of our chase variant under various parameters, such as the number of s-t tgds and egds (the latter depending on the sizes of source and target schemas S and T), the size of the source instance I and the number of conjunctions in the tgds.

A comparison with state-of-the-art chase engines

We have compared the performances against ++Spicy, an all-SQL data exchange engine and Llunatic, a mapping and cleaning engine.

Excerpt from the experimental results (1/2)

Scenarios

SCENARIO	*s-t tgds*	*OF*	*OF+*	*OF++*	*# source tuples*
A	15	5 egds	10 egds	15 egds	500K
B	30	10 egds	20 egds	30 egds	1M
C	45	15 egds	30 egds	45 egds	1.5M
D	60	20 egds	40 egds	60 egds	2M
E	75	25 egds	50 egds	75 egds	2.5M
F	90	30 egds	60 egds	90 egds	3M

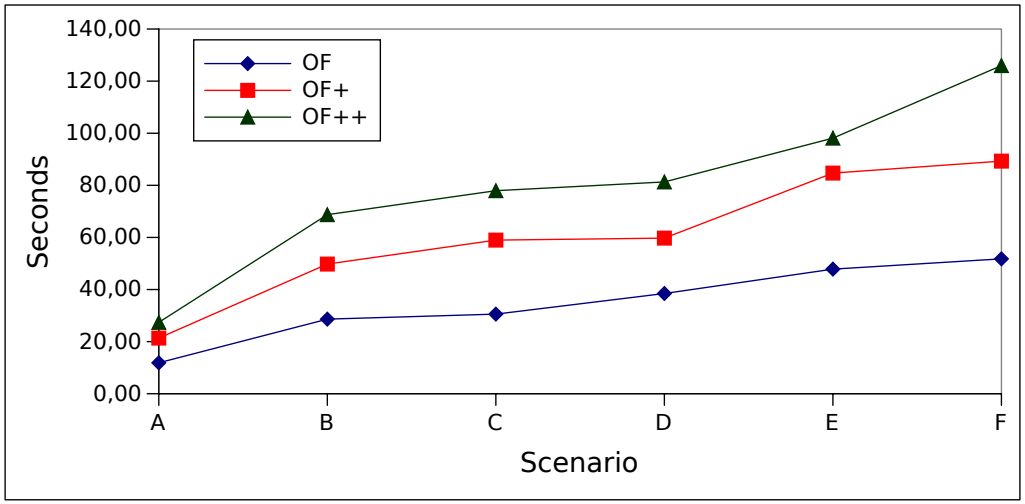
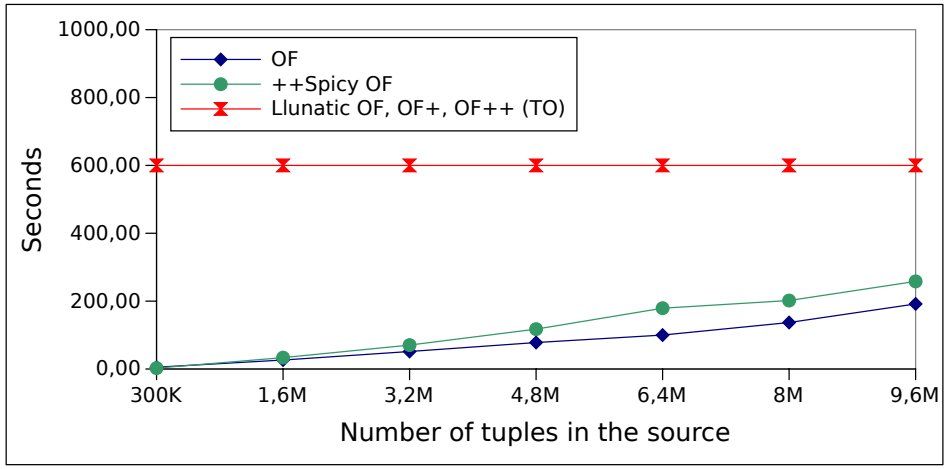


Figure: Scalability of the Interleaved Chase.

Excerpt from the experimental results (2/2)



(a) Our algorithm versus Llunatic and ++Spicy on OF scenarios.

Scenarios

Scenario	*s-t tgds*	*egds*	*# source tuples*
Employees	8	2 egds	500K
Employees	8	2 egds	1M
Workers	7	2 egds	500K

Figure: Comparison between the performances of our algorithm and state of the art engines.

Conclusion

Problem setting:

- **Data exchange** with relational source and target.
- **Target constraints (egds)** and no source constraints, as in the classical data exchange setting.

Results:

- A new chase-based algorithm **equivalent** to the Oblivious chase (proof in the paper).
- Experimental study of **its robustness** wrt. scalability.

Open questions:

- **Pure SQL implementation** based on recursive views.
- Theoretical questions about the **core** solution: minimal solution with no redundancy.