Multi-query optimization for entity-to-entity path discovery

Jean Law Ioana Manolescu

November 27, 2022

1 Outline

1.1 Our problem

Our goal is to optimize, together, a set of (acyclic) **path queries**, each of which connects two sets of nodes. Each of these have text labels which contain some given kinds of **named entities**. If we assume given a set $\mathcal{E}$ of named entity types, e.g., $\mathcal{E} = \{\text{Person, Organization, Location, email}\}$, then our problem can be, for instance: optimize together all the paths that lead from nodes containing Person occurrences to nodes containing Location occurrences. The paths $\mathcal{P}$ are given as input.

Note that a path may also contain *inversions*, as illustrated in the following example. We may want the paths that go:

- From a Person node identified in a node n_1
- To a company n_2 that n_1 worked for: $n_1 \xrightarrow{\text{worksFor}} n_2$
- To another person n_3 that worked in the same company: $n_2 \xleftarrow{\text{worksFor}} n_3, n_1 \neq n_3$
- To the place where n_3 was born: $n_3 \xrightarrow{\text{bornIn}} n_4$
- Such that a Location node has been extracted from n_4.

The above is a path with one inversion, that goes from a Person entity to a Location entity. Note the inequality predicate $n_1 \neq n_3$ that comes with the inversion.

1.2 General observations about MQO

All MQO problems are defined by an input set of queries and an output set of queries, and the goal is to find what to evaluate so that some subqueries from the input are evaluated only once (as opposed to every time they are needed), thus reducing the overall evaluation cost.

Each MQO algorithm makes some choices:

- What is the "sharing unit"? The relational MQO paper shares **tasks** which are operations in the relational query plans. In contrast, the SPARQL MQO paper shares **subqueries** which are subgraphs of the original queries.
- What are we allowed to evaluate? The relational MQO paper only evaluates tasks that exist in at least one of the input queries. The SPARQL paper can evaluate queries that did not exist in the input.
- How to enumerate sharing units (sharing opportunities)? The relational MQO paper uses an A* version. The SPARQL paper uses a set of heuristics (including clustering and Maximal Common Edge subgraphs) plus their own techniques.
- How to estimate what is the most cost-efficient sharing? We may have several sharing opportunities, each of which reduces the evaluation cost in a certain way. The relational paper assumes given a cost for each task, and then introduces a specialization of the A* cost function for their problem. The MQO paper relies on some selectivity and cost estimation approach from prior work ([33]).

2 Preliminary notions, discussion etc.

2.1 Query languages and query dialects

A **query language** is characterized by (i) a **syntax** (what are the legal queries that one can write, i.e., what should the query language parser accept) and (ii) a **semantics**, that is: what does a given query mean on a given database on which it is asked, or, equivalently, what does the query return if evaluated on that database. The standard (definition) of a language typically defines both the syntax and the semantics. Here are a few examples:

- **SQL** is a query language for relational databases. Its syntax is standardized by the ISO committee (`https://www.iso.org/standard/63555.html`).
- **SPARQL** is a query language for RDF databases, standardized by the W3C (`https://www.w3.org/TR/rdf-sparql-query/`).
- The so-called **conjunctive queries (CQ)** for relational databases is not an industry standard, but it is closely derived from logic, and widely used in research, as it provides a simple and practical syntax.
  - A conjunctive query consists of a *head* and a *body*.
  - The body is a conjunction of *atoms*. Each atom refers to a table from the underlying database; the atom has as many positions as there are attributes in the table. Within the atom, some positions may be taken by variables, and some positions may be taken by constants.
  - The head is a subset of the variables of the body; this is what the query returns.

  For instance, the conjunctive query $q(x,y) \text{ :- } R(x,y)$ returns all the (x,y) tuples from relation R. The CQ $q(x) \text{ :- } R(x,5)$ returns all the values of the first attribute in R from all the tuple where the second attribute is 5. The CQ $q(x,y) \text{ :- } R(2,x), S(x,y)$ returns all combinations of x and y such that $(2,x) \in R$ and $(x,y) \in S$.

  CQs can only express *conjunctions*, thus the name: there has to be a tuple in R *and* a tuple in S such that...

  The semantics of CQ is usually defined considering that a relation is a *set* of tuples (set semantics, no duplicates).
- **CQs over graphs** CQs can be easily adapted to other data models than relational, by describing such a model as a set of relations. For instance, we can model ConnectionLens graphs using two relations: $nodes(id, type, label)$, $edge(idsource, idtarget, label)$. Then, the query

  $$q(x) \text{ :- } \quad node(z, XML\_NODE, 'book'), edge(z,u,\epsilon), node(u, XML\_NODE, 'title'), \\ edge(u,v,\epsilon), node(v, XML\_VALUE, x)$$

  returns all the book titles found in an XML document, i.e., if the database contains the nodes and edges obtained from:

  ⟨book⟩
  ⟨title⟩"Database theory and practice"⟨/title⟩
  ⟨/book⟩

  the query result is a single tuple whose single attribute is: "Database theory and practice".

  The above graph model, based on $nodes(id, type, label)$, $edge(idsource, idtarget, label)$, assumes that nodes have two attributes other than IDs, specifically, a type, and a label; this is the case in ConnectionLens graphs. In RDF, nodes have no IDs other than their labels, thus it suffices to use $node(label), edge(lsource, ltarget, label)$, which can be further simplified into just $edge(source, target, label)$, or $triple(subject, property, object)$ (if we move the edge label from the third to the second position).

A **query dialect** is a subset of a query language, typically excluding some features of the full query language.

For instance, the **conjunctive dialect of SQL** contains unnested SQL queries, without aggregation, without group-by, and using only the keyword `and` in the `where` clause (not using the keywords `or`, `not` etc.)

If we consider that relations do not contain duplicates, and that every SQL query starts with `select distinct...`*, then CQs can express exactly what the conjunctive dialect of SQL can express.* This, and the fact that CQs are very concise (they do not require us to write the `select, from, where` keywords, not even `and`!) is a reason why *CQs are often a preferred formalisms for problemq related to queries.* As stated above, any result obtained for CQs immediately transfers back to conjunctive SQL.

2.2 Query semantics

The semantics of a query is typically defined using *embeddings*. We illustrate this below for the relational model.

Let $R_1, R_2, \ldots, R_n$ be a set of relations. Let $q(\bar{x}) \text{:-} a_1, a_2, \ldots, a_m$ be a relational CQ such that each a_i, $1 \leq i \leq m$ is an atom referring to a relation R_j, $1 \leq j, \leq n$. $\bar{x}$ is the tuple of variables in the head of the query. For instance, if $n = 3$ and the database consists of R_1, R_2, R_3, here are three simple queries:

query	$\bar{x}$	m	a_1	a_2	a_3
$q_1(x) \text{:-} R_1(5, x)$	(x)	1	$R_1(5, x)$		
$q_2(x) \text{:-} R_1(5, x), R_3(x, 7)$	(x)	2	$R_1(5, x)$	$R_3(x, 7)$	
$q_3(x, y) \text{:-} R_1(x, z), R_1(x, w), R_2(z, w, y)$	(x, y)	3	$R_1(x, z)$	$R_1(x, w)$	$R_2(z, w, y)$

Definition 1 (Embedding) *Given a database instance I (that is, for each relation $R_1, \ldots, R_n$, a set of tuples having the schema of that relation) and a CQ q, an* embedding *of q in I is a function mapping each atom $a_i \in q$, where a_i refers to the relation R_j, into an R_j tuple $\phi(a_i)$ present in the instance, such that*

1. *For each a_i, and each position in a_i, if a_i has a constant in that position, then $\phi(a_i)$ has the same constant in that position;*
2. *For each variable v that appears in the q atoms $\{a_v^1, \ldots, a_v^k\}$, for some $1 \leq k \leq m$, in positions $i_1, i_2, \ldots, i_k$, respectively, i_k, we have*

$$\phi(a_v^1)[i_1] = \phi(a_v^2)[i_2] = \ldots = \phi(a_v^k)[i_k]$$

where $\phi(a_v^1)[i_1]$ denotes the value of the tuple $\phi(a_v^1)$ at position i_1, and similarly for the others.

Example 1 *Consider $n = 3$ as above and the following instance, where we also assigned some identifiers t_i^j to tuples, to help refer to them in the text:*

I:

R_1	
t_1^1	$(5, 2)$
t_1^2	$(5, 3)$
t_1^3	$(3, 6)$

R_2	
t_2^1	$(3, 2, 9)$
t_2^2	$(1, 2, 3)$
t_2^3	$(6, 6, 2)$

R_3	
t_3^1	$(2, 7)$

- *Embeddings of q_1: there are two ways to embed q_1's single atom into a tuple in R_1 (two distinct embeddings), denoted ϕ_1^1 and ϕ_1^2:*
  - $\phi_1^1(a_1) = t_1^1$
  - $\phi_1^2(a_1) = t_1^2$
- *Embeddings of q_2: we need to embed a_1 in R_1 and a_2 in R_3, so that the first attribute of R_1 is 5, and the second attribute in the embedding of a_1 is equal to the first in the embedding of R_3. There is only one way:*
  - $\phi_2^1(a_1) = t_1^1$, $\phi_2^1(a_2) = t_3^1$
- *Embeddings of q_3: by a similar reasoning, we obtain:*
  - $\phi_3^1(a_1) = t_1^2$, $\phi_3^1(a_2) = t_1^1$, $\phi_3^1(a_3) = t_2^1$
  - $\phi_3^2(a_1) = t_1^3$, $\phi_3^2(a_2) = t_1^3$, $\phi_3^2(a_3) = t_2^1$

Definition 2 (Query result (semantics)) *Let I be an instance of a database and q be a query whose head is $\bar{x}$. The result of q on I, denoted $q(I)$, is the set:*

$$\{\phi(\bar{x}) \mid \phi \text{ is an embedding of } q \text{ into } I\}$$

where $\phi(\bar{x})$ is the tuple of values from I, assigned by the embedding ϕ to the variables in the head of q.

Example 2 *Continuing with the sample queries q_1, q_2, q_3 and instance I above, we get:*

- $q_1(I) = \{(2), (3)\}$. *The first tuple is because of ϕ_1^1, while the second is because of ϕ_1^2.*
- $q_2(I) = \{(2)\}$, *due to the single embedding ϕ_2^2*
- $q_3(I) = \{(5, 9), (3, 2)\}$. *The first tuple is because of ϕ_3^1 (note that $\phi_3^1(a_1)[1] = \phi_3^1(a_2)[1] = \phi_3^1(x) = 5$, and $\phi_3^1(y) = 9$), while the second is because of ϕ_3^2 ($\phi_3^2(x) = 3, \phi_3^2(y) = 2$).*

2.3 Relational algebra vs. conjunctive queries

It is easy to see that relational algebra operators apply quite naturally on a semantic query, and produce another semantic query, as follows.

Definition 3 (Projection of a CQ) *Given a CQ q whose head is $\bar{x}$, and a projection π_{col} such that each variable in col appears in $\bar{x}$, the projection $\pi_{col}(q)$ is a CQ having the same body as q, and as head, only those variables in $\bar{x}$ that also appear in col.*

Definition 4 (Predicate and condition) *A predicate is an expression of the form $v = c$ where v is a variable and c is a constant, or of the form $v_1 = v_2$, where v_1, v_2 are variables. A condition is a conjunction of predicates. For simplicity, we assume that a variable appearing in a predicate of the form $v_1 = v_2$ does not appear in a predicate of the form $v_1 = c$. (It is easy to modify a condition that does not respect this constraint, into a condition that does respect it. For instance, $(v_1 = 5) \wedge (v_1 = v_2) \wedge (v_3 = v_4)$ can be equivalently rewritten as $(v_1 = 5) \wedge (v_2 = 5) \wedge (v_3 = v_4)$).*

Definition 5 (Selection of a CQ) *Given a CQ q and a condition cond over the variables of q, the selection $\sigma_{cond}(q)$ is CQ obtained from q modified as follows:*

- *For each predicate of the form $v_1 = v_2$ in cond (where v_1, v_2 are distinct variables), replace all occurrences of v_2 by v_1 (v_1, v_2 are both variables, and they play symmetric roles)*
- *For each predicate of the form $v = c$ in cond, replace all occurrences of v by c.*

Definition 6 (Join of two CQs) *Given two CQs q_1, q_2 whose heads are $\bar{x_1}, \bar{x_2}$, and such that some variables appear both in the body of q_1 and in the body of q_2, the natural join $q_1 \bowtie q_2$ is a CQ whose body contains all the atoms of q_1 and all the atoms of q_2, and whose head contains all the variables in $\bar{x_1}$, followed by all the variables in $\bar{x_2} \setminus \bar{x_1}$.*

Example 3 *Consider the query q_3 from Example 1.*

- $\pi_y(q_3)$ *is the CQ:* $q_3'(y) \text{ :- } R_1(x, z), R_1(x, w), R_2(z, w, y)$
- $\sigma_{(x=4)\wedge(y=z)}(q_3)$ *is the CQ:* $q_3''(4, y) \text{ :- } R_1(4, z), R_1(4, w), R_2(y, w, y)$
- *Now, consider also the CQ* $q_4(y, t) \text{ :- } R_2(y, t, 5)$. *Note that y is the common variable between q_3 and q_4. Then, $q_3 \bowtie q_4$ is the CQ:*

$$q_{3,4}(x, y, t) \text{ :- } R_1(x, z), R_1(x, w), R_2(z, w, y), R_2(y, t, 5)$$

2.4 Query containment and equivalence

Definition 7 (Query Containment) *We say query q is contained in q', denoted $q \subseteq q'$, iff for any database instance D, $q(D) \subseteq q'(D)$.*

- "find all the red boats" is contained in "find all the boats" (and also in: "find all the red boats").

Definition 8 (Query Equivalence) *We say q is equivalent to q', denoted $q \equiv q'$, if for any database instance D, $q(D) = q'(D)$ (or, equivalently, $q \subseteq q'$ and $q' \subseteq q$.*

Definition 9 (Query homomorphism) [1] *A homomorphism ϕ is a function that maps the atoms[2] from query q_1 into the atoms of query q_2 such that all the predicates that hold on some variables in q_1 also hold on their images in the variables of q_2.*

For instance, consider the queries:

- q_2: `select * from R r, S s where r.a=s.b and r.a=5;`
- q_1: `select * from R x1, S y1 where y1.b=5 and x1.a=y1.b and y1.c=7;`

In this case, there exists a homomorphism from q_2 to q_1: $\phi(r) = x_1$, $\phi(s) = y_1$. There does not exist a homomorphism in the other direction, because the condition y1.c=7 cannot be mapped into q_2.

Query equivalence vs. query homomorphism (Chandra and Merlin, 1977)
The first result of Chandra and Merlin is that:

> Given conjunctive queries q_1, q_2, we have $q_1 \subseteq q_2$ if and only if there exists an homomorphism from q_2 to q_1.

As a consequence (also in Chandra and Merlin):

> Given conjunctive queries q_1, q_2, we have $q_1 \equiv q_2$ if and only if there exists an homomorphism from q_2 to q_1 and one in the opposite direction, which is equivalent to saying that there exists an isomorphism between q_1 and q_2.

2.5 Selectivity

Selectivity in relational databases In the simplest, relational database setting, the selectivity of a selection is the ratio between the size of the selection output and the size of the selection input. Specifically, let e be a relational algebra expression and $\sigma_c(e)$ be a selection on top of e. Then, the selectivity of σ_c is: $|\sigma_c(e)|/|e|$.

For instance, if R is a relation of 1000 tuples, and $\sigma_{R.a=5}(R)$ returns only 1 tuple, then the selectivity of $\sigma_{R.a=5}$ is 0.001.

Selectivity also generalizes to *joins*. Recall that a join is a selection over a cartesian product: $R \bowtie_c S \equiv \sigma_c(R \times S)$. In such a context, the selectivity of $\bowtie_c$ is defined as: $|R \bowtie_c S|/|R \times S|$.

Selectivity in graph databases In this context, the definitions are not as clear-cut, but we can still provide good intuitions on what this could mean.

- In an RDF database, edges may have different labels, and some edge labels may be more popular than others. The selectivity of a triple pattern of the form $(?x, p, ?y)$ (all the edges labeled p) can be defined for instance as: the fraction of all the graph edges that are labeled p.

[1] Definition details depend on query language

[2] What *atom* means depends on the query language. In the relational examples below, an atom is an occurrence of a relation, for instance, both q_1 and q_2 have two atoms, one referring to the R table, one referring to the S table.

- More generally, we can say a graph pattern query is selective if it has "few results" (depending on what we compare).
- Further, if we consider the two queries $q_1 = (?x, p, ?y)$ and $q_2(?x, p, ?y), (?y, q, ?z)$:
  - q_1 may have less results than q_2, if few targets of p edges have q outgoing edges; for instance, suppose only 1 out of 10 nodes that are targets of p are also sources of q, and each of these has only one outgoing edge labeled q.
  - q_1 may have more results that q_2, if (i) a good part of the targets of p are sources of q, and/or (ii) each of these has many q edges.

3 Interesting subqueries of ConnectionLens path queries

We model ConnectionLens graphs as two relations, storing respectively nodes and edges. Specifically, we use $n(\underline{id}, \tau, l, eq)$ for nodes, where τ is the type of a node, l is its label (possibly empty , denoted ϵ), and eq identifies an equivalence class to which the node belongs (we can think of eq as an integer). The edge relation is $e(s, t, l)$, where s and t are foreign keys in the nodes (n) relation, thus they are node IDs, and l is the label of the edge.

Definition 10 (ConnectionLens path query) *A ConnectionLens path query (or just path query, or pquery, in short) is a conjunctive query over the relations n, e of the form:*

$$q(\bar{x}) \text{ :- } an_1, ae_1, an_2, ae_2, \ldots, an_{k-1}, ae_{k-1}, an_k$$

where k is a non-zero integer; each an_i, for $1 \leq i \leq k$, is an atom referring to the $\underline{n}$ relation, with a constant in the equivalence class position; each ae_j, for $1 \leq j < k$, is an atom referring to the $\underline{e}$ relation; and $\bar{x}$ is a subset of the variables in the body of the query.

Example 4 *The following is a sample query for $k = 3$:*

$$\begin{aligned} pq_1(x) \text{ :- } \quad & n(z, XML\_NODE, 'book', \underline{1}), e(z, u, \epsilon), n(u, XML\_NODE, 'title', \underline{2}), \\ & e(u, v, \epsilon), n(v, XML\_VALUE, x, \underline{3}) \end{aligned}$$

For readability, we will always associate underlined constants, e.g., $\underline{1}$ etc., to IDs of equivalence classes.

Definition 11 (Subquery) *Given a pquery pq whose head variables are $\bar{x}$, a* subquery *of pq is a CQ sq_1 such that there exists another pquery sq_2, and a selection condition σ_{sel} such that:*

$$pq \equiv \pi_{\bar{x}}(\sigma_{sel}(sq_1 \bowtie sq_2))$$

where $sq_1 \bowtie sq_2$ denotes the natural join of sq_1 and sq_2 on all their shared variables (recall Section 2.3).

As particular cases, in the above: $\pi_{\bar{x}}$ may not be needed (if its input returns already the right attributes); σ_{sel} may not be needed (if there is no extra condition); sq_2 may be empty, in which case $sq_1 \bowtie sq_2$ becomes just sq_1.

Example 5 *Given the pquery pq_1 from Example 4, a possible proper subquery sq_1, corresponding sq_2, and equivalent rewriting of pq_1 based on sq_1 and sq_2 are:*

$$\begin{aligned} sq_1(z, u, v, x) \text{ :- } \quad & n(z, XML\_NODE, l, \underline{1}), n(u, XML\_NODE, 'title', \underline{2}), n(v, XML\_VALUE, x, \underline{3}) \\ sq_2(z, u, v) \text{ :- } \quad & e(z, u, \epsilon), e(u, v, \epsilon) \\ pq_1(x) \text{ :- } \quad & \pi_x(\sigma_{l='book'}(sq_1 \bowtie sq_2)) \end{aligned}$$

Note that each sq_1 is a cartesian product of three instances of the n relation. This shows that even if a pquery is connected (each atom shares at least one variable with other atoms), its subqueries are not necessarily connected. Below, we generalize this observation to derive the set of all subqueries of a given pquery.

Definition 12 (Canonical form of a pquery) *Let $q(\bar{x})$:- $an_1, ae_1, an_2, ae_2, \ldots, an_{k-1}, ae_{k-1}, an_k$ be a pquery. We call the* canonical form of *q, denoted $q^c(\bar{x})$, the expression $\sigma_c(q_0(\bar{x}))$, where*

- $q_0(\bar{x})$:- $n(v_1^1, v_1^2, v_1^3, \underline{c_1}), e(u_1^1, u_1^2, u_1^3), n(v_2^1, v_2^2, v_2^3, \underline{c_2}), \ldots, e(u_{k-1}^1, u_{k-1}^2, u_{k-1}^3), n(v_k^1, v_k^2, v_k^3, \underline{c_k})$ *is a CQ*
  - *whose body contains k atoms over the n table, and $k-1$ atoms over the e table;*
  - *in each of these atoms, the first three attributes are variables, while the last attribute is the constant equivalence class, appearing in the corresponding position in q;*
  - *no variable appears more than once, that is, all the atoms participate in a cartesian product.*
- *σ_c is a selection with all the necessary conditions so that $\sigma_c(q_0(\bar{x}))$ is exactly q.*

Example 6 (Canonical form of a pquery) *Considering the pquery pq_1 from Example 4, we obtain:*

- *$pq_0(x)$:- $n(v_1^1, v_1^2, l, \underline{1}), e(u_1^2, u_1^2, u_1^3), n(v_2^1, v_2^2, v_2^3, \underline{2}), e(u_2^1, u_2^2, u_2^3), n(v_3^1, v_3^2, x, \underline{3})$ (for readability, we left unchanged the names of each variable that appears only once in pq_1)*
- *σ_c is: $(v_1^1 = u_1^2) \wedge (v_1^2 = XML\_NODE) \wedge (l =' book') \wedge (u_2^1 = v_1^2) \wedge (u_1^3 = \epsilon) \wedge (v_2^2 = XML\_NODE) \wedge (v_2^3 =' title') \wedge (v_2^1 = u_2^1) \wedge (u_2^3 = \epsilon) \wedge (u_2^2 = v_3^1) \wedge (v_3^2 = XML\_VALUE)$ (the condition is a conjunction of 11 predicates).*

Notation We say that a pquery q is **defined by** $\langle \bar{x}, \sigma_c, q_0 \rangle$, where $\bar{x}$ denotes the head variables of q, while σ_c and q_0 are components of the canonical form of q.

Property 1 (Subqueries of a pquery) *Let q be a pquery such that q_0 and σ_c compose the canonical form of q.*

1. *Let σ_b be a subset of the predicates in σ_c. Then, the CQ sq_1 defined as $\sigma_b(q_0)$ is a subquery (Definition 11) of q.*
2. *Let b be a subset of the atoms in the body of q_0, σ_b be a subset of the predicates in σ_c that apply only on variables present in b, and $\bar{y}$ be the set of q variables that includes $\bar{x}$ and any other variable v such that σ_c contains a predicate of the form $v = x$ where $x \in \bar{x}$ and $v \notin \bar{x}$.*
   Let q_b be the query whose head is $\bar{y}$ and body consists exactly of the atoms in b.
   Let sq_1 be the CQ whose canonical form is as $\pi_{\bar{x}}(\sigma_b(q_b))$. Then, sq_1 is a subquery of q.

Proof

1. Let σ_d be the conjunction of all predicates in σ_c that are not in σ_b. It is easy to see that $q \equiv \sigma_d(sq_1)$.
2. Let d be the set of atoms in the body of q that are not in b, and σ_d be all the σ_c predicates that are not in σ_b. Let $\bar{z}$ be the set of all variables appearing in d and/or σ_d. Let q_d^1 be the CQ having the head $\bar{z}$ and the body consisting of exactly the atoms of d; let $q_d^2 = \sigma_d(q_d^1)$. It is easy to see that $q \equiv \pi_{\bar{x}}(sq_1 \bowtie sq_2)$.

Property 1 provides **a way to enumerate subqueries of a query** q_0: select σ_b, then b and $\bar{x}$ as prescribed, and obtain a subquery q_1.

From Property 1, it follows that a the number of subqueries of q_0 is at least $O(2^{|\sigma_c|}) + O(2^{|q|})$, where $|\sigma_c|$ denotes the number of predicates in σ_c and $|q|$ denotes the number of atoms in the body of q. Some of these subqueries may be isomorphic to each other (and, thus, logically, should count as one). For instance, applying Property 1 point 2. to the sample query pq_1 and its canonical form shown in Example 6, the subquery sq_1' obtained with $\sigma_b = \emptyset$ and the first e atom, respectively, sq_1'' obtained with $\sigma_b = \emptyset$ and the second e atom are isomorphic. However, even taking this into account, the number of subqueries of a pquery remains quite high.

At the same time, *some subqueries may not be connected*, i.e., include cartesian products. This is the case, for instance, of sq_1 and sq_2 in Example 5. We are not interested in such subqueries, since their evaluation can hardly be made efficient, and (especially) their result size is likely to be huge. Therefore, evaluating such a subquery is unlikely to help our performance. Instead, we focus on *connected subqueries*, as follows:

Definition 13 (Connected subquery) *Given a pquery q whose canonical form is defined by q_0 and σ_c, and having the head variables $\bar{x}$. A connected subquery s of q is a CQ having the following canonical form:*

- *The body of s, denoted b, is a* subsequence *of the atoms in the body of q_0, starting and ending with an n atom;*
- *Let σ_s be a subset of the predicates in σ_c that only refer to the variables in b, which, if $|b| > 1$, ensures that each atom in b is connected (shares a variable with) at least another atom in b;*
- *Let $\bar{y}$ be the variables appearing in b that either (i) appear in $\bar{x}$, or (ii) appear in some predicate in $\sigma_c \setminus \sigma_s$.*
- *Then, s is the CQ defined by the canonical form $\pi_{\bar{y}}(\sigma_s(b))$.*

Example 7 *We continue to to rely on the sample query pq_1 and its canonical form shown in Example 6.*

- *Let b consist of the last three atoms $n(v_2^1, v_2^2, v_2^3, \underline{2}), e(u_2^1, u_2^2, u_2^3), n(v_3^1, v_3^2, x, \underline{3})$ from the body of pq_0.*
- *The predicates in σ_c that only refer to these variables are:*

$$(v_2^2 = XML\_NODE) \wedge\ (v_2^3 =' title') \wedge (v_2^1 = u_2^1) \wedge (u_2^3 = \epsilon) \wedge (u_2^2 = v_3^1) \wedge (v_3^2 = XML\_VALUE)$$

- *We can, for instance, take this whole set as σ_s. It contains two predicates $(v_2^1 = u_2^1)$ and $(u_2^2 = v_3^1)$ which ensure that the atoms in b are connected.*
- *Then, $\bar{y}$ becomes: (x, u_2^1). We added u_2^1 because $\sigma_c \setminus \sigma_s$ contains the predicate $(u_2^1 = v_1^2)$.*
- *The resulting subquery s has the canonical form: $\pi_{\bar{y}}(\sigma_s(b)$, which corresponds to:*

$$s(x, u_2^1) \text{ :- } n(v_2^1, XML\_NODE, 'title'), e(u_2^1, u_2^2, \epsilon), n(u_2^2, XML\_VALUE, x)$$

Compared with the original pquery (Example 4), it corresponds to: taking the last three atoms; renaming some variables; and adding to the head, the variable that s shares with the atoms in pq_1's body, that are not in the body of s.

Property 2 *Any connected subquery (Definition 13) is a pquery (Definition 10).*

Proof We show that a connected subquery s (of the pquery q defined by the canonical form q_0, σ_c) is a conjunctive query over the relations n, e of the form:

$$\text{q}(\bar{x}) \text{ :- } an_1, ae_1, an_2, ae_2, \ldots, an_{k-1}, ae_{k-1}, an_k$$

where k is a non-zero integer; each an_i, for $1 \leq i \leq k$, is an atom referring to the n relation; each ae_j, for $1 \leq j < k$, is an atom referring to the e relation; and $\bar{x}$ is a subset of the variables in the body of the query.

s, by definition, is a conjunctive query. It is a connected subquery, so there exists a subsequence of q_0 that starts and ends with an n atom that defines its body that we denote by b. Additionally, σ_s ensures that each atom in b is connected with another atom in b. This proves that s has the aforementioned form.

Property 3 *Let pq be a pquery (Definition 10) defined by $\langle \bar{x}, q_0, \sigma_c \rangle$. Any connected subquery s (Definition 13) of pq is also a subquery (in the sense of Definition 11) of pq.*

Proof We show that s is a CQ sq_1 such that there exists another pquery sq_2, and a selection condition σ_{sel} such that:

$$pq \equiv \pi_{\bar{x}}(\sigma_{sel}(sq_1 \bowtie sq_2))$$

where $sq_1 \bowtie sq_2$ denotes the natural join of sq_1 and sq_2 on all their shared variables.

Property 2 ensures that q_1 is a pquery. Assume it is determined by $\langle \bar{y_1}, b_1, \sigma_{c_1} \rangle$. b_1 is a subsequence of q_0 hence there exists b_2 such that $b_1 \cup b_2 = q_0$. We define the canonical form of sq_2 as b_2 and $\sigma_{c_2} = \emptyset$. We have, for this definition of sq_2 ad $\sigma_{sel} = \sigma_c \backslash \sigma_{c1}$, that the above equivalence is true.

Connected queries only Unless otherwise specified, all queries and subqueries mentioned from this point onwards are connected.

A first possible definition of candidate subqueries we are interested in, considers very broadly any subquery of at least one query:

Definition 14 (Candidate subquery) *Given a set Q of pqueries $q_1, \ldots, q_n$, a query s is a candidate subquery of Q if there exists $q \in Q$ such that s is a subquery of q (Definition 11).*

However, this definition leads to considering too many subqueries. In particular, if s is a subquery of only one query $q \in Q$, materializing s will reduce the cost of evaluating q, but not the other queries in Q. Thus, we are more interested in subqueries shared by several queries in Q. Further, among such shared subqueries, we are interested in those that apply as many computation steps as possible. This is formalized as follows:

Definition 15 (Most specific subquery of a set of queries) *Let Q be a set of queries.*

1. The subqueries of Q, *denoted S_Q, are subqueries shared by all the queries in Q:*

$$\{s \mid \forall q \in Q, s \text{ is a subquery of } q\}$$

2. *A* most specific subquery (msq, in short) *of Q is a subquery $s_1 \in S_Q$ such that s_1 is a subquery of no other query in S_Q.*

Note that a set of queries may have several msqs. Further, one may think that an msq has the largest number of atoms among all queries in S_Q. But this is not always true, as shown by the following example.

Example 8 (MSQ) *Let $Q = \{q_1, q_2\}$ such that q_1 is:*

$$\begin{aligned} q_1(\bar{u}) \text{ :- } \quad & n(u_1, u_2, u_3, \underline{1}), e(u_1, A, u_4), n(u_4, u_5, u_6, \underline{2}), e(u_4, B, u_5), n(u_5, u_6, u_7, \underline{3}) \\ & e(u_5, C, u_8), n(u_8, u_9, u_10, \underline{4}), e(u_8, X, u_{11}), n(u_{11}, u_{12}, u_{13}, \underline{5}), \\ & e(u_{11}, I, u_{12}), n(u_{12}, u_{13}, u_{14}, \underline{6}), e(u_{12}, J, u_{15}), n(u_{15}, u_{16}, u_{17}, \underline{7}) \end{aligned}$$

*for some $\bar{u}$ not detailed. For readability, we introduce the **shorthand** below to denote q_1:*

$$\underline{1} - A - \underline{2} - B - \underline{3} - C - \underline{4} - X - \underline{5} - I - \underline{6} - J - \underline{7}$$

Observe that the shorthand notation hides some information about the query, such as: the node labels (if they are known; in the above, we assumed they are variables, but if some of them had been constants, the shorthand would not have reflected them), the exact head variables, etc. That is fine (we know that an actual pquery needs to specify these details). Further, consider a query q_2, shown directly in the shorthand notation:

$$\underline{1} - A - \underline{2} - B - \underline{3} - C - \underline{4} - Y - \underline{5} - I - \underline{6} - J - \underline{7}$$

such that the subqueries between $\underline{1}$ and $\underline{4}$ are identical, and similarly those between $\underline{5}$ and $\underline{7}$. The underlined numbers, as usual, are node equivalence Then, these subqueries are two msq's, none of which is a subquery of the other. They do not have the same number of atoms, they are not over the same atoms, and their predicate sets are not comparable, since they carry over different atom sets.
One could try to say $\underline{1} - A - \underline{2} - B - \underline{3} - C - \underline{4}$ is better, since it is over 3 node atoms. But depending on the data cardinalities, $\underline{5} - I - \underline{6} - J - \underline{7}$ may be more beneficial.

4 Multi-query optimization for a set of pqueries

We are given as input a set Q of pqueries $\{q_1, ..., q_n\}$.

4.1 Identifying common subqueries

We start by introducing an alignment among sequences of atoms, through the node equivalence classes:

Equivalence class sequence For a query q, let the equivalence class sequence of q, denoted $ecs(q)$, be the sequence of equivalence classes of its node atoms.

Example 9 (Equivalence class sequence) *Consider again the query q_1 from Example 8. Its ecs is:*

$$\underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{6}, \underline{7}$$

Algorithm 1 computes the msqs of two queries $Q = \{q_1 = (q_{01}, \sigma_{c1}), q_2 = (q_{02}, \sigma_{c2})\}$. Further, it computes, for each msq s of q_1, q_2, the rewriting of q_1 based on s, and the rewriting of q_2 based on s.

1. Let M_{q_1,q_2} be the set of maximal common subsequences of $ecs(q_1)$ and $ecs(q_2)$. Each element in M_{q_1,q_2} is a sequence of node equivalence class values appearing consecutively both in q_1 and in q_2; further, no element in M_{q_1,q_2} is a subsequence of another.

2. Each element $e \in M_{q_1,q_2}$ naturally determines a subquery s_1^e of q_1, and a subquery s_2^e of q_2, consisting of the respective node atoms, the edge atoms connecting them, all the predicates on the variables of these node and edge atoms, and corresponding head variables.
   The queries s_1^e, s_2^e have the same number of node atoms and, thus, of edge atoms. By construction, the node equivalence class values in node atoms appearing at the same position (index) in s_1^e and s_2^e are equal. This allows us to establish a one-to-one correspondence between node atoms, thus also between edge atoms, and from these, a **one-to-one correspondence between attributes (variables or constants)** in s_1^e, s_2^e.

3. For each $e \in M_{q_1,q_2}$, based on s_1^e, s_2^e and the correspondence between their attributes, create a pquery s^e, together with the rewritings r_1, r_2 of s_1^e, s_2^e as follows:

   (a) Initially, r_1 and r_2 are T (true), i.e., they just copy s^e.

   (b) s^e is a pquery mirroring the node and edge atoms of s_1^e, s_2^e. Initially, s^e has a different (fresh) variable in each attribute, except for the node equivalence classes (where it has the values that s_1^e, s_2^e share).

   (c) From the first s^e atom to the last, and for each attribute in that atom, starting from the first:

   - If s_1^e, s_2^e both have a constant in that position, and it is the same constant, replace all occurrences of the corresponding s^e variable with that constant.
   - Otherwise, do not modify s^e (leave a variable there). If s_1^e (respectively, s_2^e) had a constant in this position, add a selection predicate enforcing this predicate to r_1 (respectively, to r_2).

   (d) After traversing all attributes in this way, r_1, r_2 show how to rewrite s_1^e, s_2^e through selections on s^e. The rewritings of q_1, q_2 are obtained by joining r_1, r_2 with all the atoms in q_1 (respectively, q_2) that were not in e.

Extended shorthand We extend the shorthand notation of connected pqueries to also *show the node label, when known* (that is, when it is given as a constant in the query). Instead of specifying, for each node atom, just its equivalence class ID, e.g., $\underline{1}$, $\underline{2}$, etc., we use a lowercase letter from the beginning of the alphabet, with the equivalence class ID as an index, e.g., $\underline{a_1}$ designates a node atom whose label is specified to be the constant a, and whose equivalence class ID is 1.

Illustration of Algorithm 1

We will apply algorithm 1 on the following pqueries that are inspired by those in Example 8, namely:

$$q_1 : \underline{a_1} - A - \underline{b_2} - B - \underline{c_3} - C - \underline{l_4} - X - \underline{e_5} - I - \underline{f_6} - J - \underline{g_7}$$
$$q_2 : \underline{a_1} - A - \underline{b_2} - B - \underline{c_3} - \alpha - \underline{d_4} - \mathrm{Y} - \underline{k_8} - I - \underline{f_6} - J - \underline{g_7}$$

A constant label in a node is represented by a lower-case latin alphabet, whereas a constant label in an edge is represented by an upper-case latin alphabet. The equivalence class of each node is indicated as a subscript (in the form of an arabic numeral). Variables are represented by the Greek alphabet (e.g. α).

We use u_j^i and v_j^i to denote the variables in the canonical body q_0 of both queries, written in the following form:

$$q_0(x) :\text{-}\, n(u_1, u_2, u_3, \underline{1}), e(u_1, u_4, u_5), n(u_5, u_6, u_7, \underline{2}), ..., e(u^{20}, u_{21}, u_{22}), n(u_{22}, u_{23}, u_{24}, \underline{7})$$

Step 0: Define $ecs(q_1)$ and $ecs(q_2)$

$$ecs(q_1) : \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{6}, \underline{7}$$
$$ecs(q_2) : \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{8}, \underline{6}, \underline{7}$$

Step 1: Calculate the set of maximal common subsequences M_{q_1,q_2} There are two maximal common subsequences:

$$e_1 : \underline{1}, \underline{2}, \underline{3}, \underline{4}$$
$$e_2 : \underline{6}, \underline{7}$$

From Step 2 onwards, we only consider e_1 for illustration purposes (in reality, we need to apply the same steps to all elements in M_{q_1,q_2}).

Step 2: There is a one-to-one correspondence between attributes in $s_1^{e_1}$ and $s_2^{e_1}$

$$s_1^{e_1} : \underline{u_{1_1}} - A - \underline{b_2} - B - \underline{c_3} - C - \underline{l_4}$$
$$s_2^{e_1} : \underline{a_1} - A - \underline{b_2} - B - \underline{c_3} - \alpha - \underline{d_4}$$

Step 3: Construction of s_{e_1} and rewritings $r_{1_{e_1}}$ and $r_{2_{e_1}}$ of $s_1^{e_1}$ and $s_2^{e_1}$

3.1: Initialization of s_{e_1}, $r_{1_{e_1}}$ and $r_{2_{e_1}}$:

$$s_{e_1} : \underline{\imath} - y_1 - \underline{x_2} - y_2 - \underline{x_3} - y_3 - \underline{x_4}$$

where a, b, c, d, x, y and z are variables.

$$r_{1_{e_1}} = r_{2_{e_1}} = True$$

3.2.1: Iteration for each attribute on the first s_{e_1} atom: $s_1^{e_1}$ and $s_2^{e_1}$ have the same constant label in their first atom, hence we update the label of the first atom of s_{e_1}:

$$s_{e_1} : \underline{1_i} - x - \underline{b_i} - y - \underline{c_i} - z - \underline{d_i}$$

In this illustration, the nodes do not have the attribute type - but it also needs to be iterated on in the algorithm.

3.2.2: Iteration on the second s_{e_1} atom: Again, $s_1^{e_1}$ and $s_2^{e_1}$ have the same constant label in their second atom (first edge), hence we update the label of the second atom of s_{e_1}:

$$s_{e_1} : \underline{1_i} - A - \underline{b_i} - y - \underline{c_i} - z - \underline{d_i}$$

3.2.3 to 3.2.5 proceed in the same vein, so we skip to 3.2.6.

3.2.6: Iteration on sixth atom: $s_1^{e_1}$ has a constant label in its sixth atom whereas and $s_2^{e_1}$ does not, hence we leave the variable z unchanged:

$$s_{e_1} : \underline{1_i} - A - \underline{2_i} - B - \underline{3_i} - z - \underline{d_i}$$

Since $s_1^{e_1}$ has a constant label in its sixth atom, add a selection predicate enforcing the label to $r_{1_{e_1}}$:

$$r_{1_{e_1}} : u_6^3 = C$$
$$r_{2_{e_1}} : True$$

Final output of 3.2:

$$s_{e_1} : \underline{1_i} - A - \underline{2_i} - B - \underline{3_i} - z - \underline{d_i}$$
$$r_{1_{e_1}} : (u_6^3 = C) \wedge (v_7^3 = 9)$$
$$r_{2_{e_1}} : (v_7^3 = 4)$$

Rewriting q_1, q_2:

$$Originalq_1 : \underline{1_i} - A - \underline{2_i} - B - \underline{3_i} - C - \underline{9_i} - X - \underline{5_{ii}} - I - \underline{6_{ii}} - J - \underline{7_{ii}}$$
$$Rewrittenq_1 : \underline{1_i} - A - \underline{2_i} - B - \underline{3_i} - z - \underline{d_i} - X - \underline{5_{ii}} - I - \underline{6_{ii}} - J - \underline{7_{ii}} withr_{1_{e_1}}$$
$$Originalq_2 : \underline{1_i} - A - \underline{2_i} - B - \underline{3_i} - \alpha - \underline{4_i} - Y - \underline{8_{iii}} - I - \underline{6_{ii}} - J - \underline{7_{ii}}$$
$$Rewrittenq_2 : \underline{1_i} - A - \underline{2_i} - B - \underline{3_i} - z - \underline{d_i} - Y - \underline{8_{iii}} - I - \underline{6_{ii}} - J - \underline{7_{ii}} withr_{2_{e_1}}$$

4.2 Quantitative measure of selectivity or cost

[SSB+08] discussed the selectivity estimation for the conjunctive Basic Graph Pattern, in which they assume that for t = (s p o), the subject, predicate and object are statistically independent. In this case, the selectivity of t $sel(t) = sel(s) \times sel(p) \times sel(o)$ (refer to 2.5 for the definition of the selectivity of an edge).

Definition 16 (Cost model) *We define the cost of evaluating a pquery $q(\bar{x}) :\!\!- an_1, ae_1, an_2, ae_2, \ldots, an_{k-1}, ae_{k-1}, an_k$ as $cost(q) = min(sel(o))$, where o ranges over all the node and edge atoms. The cost of an empty pquery is infinite.*

Definition 17 (Cost of computing q based on the results of s) *Since s is a subquery of q, there exists a unique subquery t such that $q = \sigma(t \bowtie s)$. Given that q and s are connected, one of the following two holds:*

- *t is a connected pquery, or*
- *t is not connected, but there exist two connected pqueries t_1, t_2 such that $t = t_1 \times t_2$, and $q = \sigma(t_1 \bowtie s \bowtie t_2)$*

The cost of computing q, based on the results of s, is:

- *In the first case above: the cost of t;*

- *In the second case above: the sum of the costs of t_1 and t_2.*

Given a set S of subqueries of a given query q, we could evaluate q using one subquery from S, or two, or more, etc. For simplicity, **we will consider using at most one subquery for each query**[3].

Definition 18 (Best evaluation method of q) *Given a query q and assuming a set of subqueries S, the **best evaluation method of** q is:*

- *If S contains no subquery of q, then evaluating q directly on the graph.*
- *If S contains some subqueries of q:*
  - *Let min_{rew} be the rewriting of q of the form $q = \sigma(s \bowtie t)$ which has the least cost across all the subqueries s of q.*
  - *If min_{rew} < the cost of evaluating q directly, then the best evaluation method of q is $\sigma(s \bowtie t)$. In this case, we say that the **cost saving** of s is (cost of q - min_{rew})*
  - *Else, the best evaluation method is directly on the graph.*

[3]This is because for each subquery s, we know how to rewrite q as "s join something". But we have not yet the knowledge of "how to rewrite q with s_1 **and** s_2".