Query Decompositions

survey

Nicola Onose

January 19, 2007
Outline

1. Speeding Up Query Evaluation
Outline

1. Speeding Up Query Evaluation
2. Graphs & Queries
Outline

1. Speeding Up Query Evaluation
2. Graphs & Queries
3. Back to Hypergraphs
1. Speeding Up Query Evaluation

2. Graphs & Queries

3. Back to Hypergraphs

4. Computing Decompositions and Widths

5. Related Concepts & Conclusions
Query Evaluation

- The eternal problem: given Q and DB, compute Q(DB).
- Need to run Q over DB as efficiently as possible.
- Classical DB approach: build a query plan
- ... and optimize it
- How far can we get?
Evaluating a conjunctive query is NP-complete in the combined complexity [Chandra & Merlin, 1977]

In general, evaluating a FO query is PSPACE in the combined complexity

However, certain (acyclic) conj. queries can be evaluated in PTIME [Yannakakis, 1981]
Evaluating a conjunctive query is NP-complete in the combined complexity [Chandra & Merlin, 1977]

In general, evaluating a FO query is PSPACE in the combined complexity

However, certain (acyclic) conj. queries can be evaluated in PTIME [Yannakakis, 1981]

Can this result be generalized to other classes of CQ?
Example

Schema Example

- `teaches(Professor, Course)`
- `enrolled(Student, Course, GradeOption)`
- `parent(Person1, Person2)`

Query $Q_1$

∃ a professor who has a child enrolled in some course?

\[ ans() \leftarrow teaches(P, C), \text{enrolled}(S, C', G), \text{parent}(P, S) \]
Example 1

Figure: Evaluation Plan for $Q_1$
Example 1

Now, we keep only a table with 1 attribute after the 1st join.
The size of the left branch of the 2nd join decreases.

Figure: A Better Plan for $Q_1$
Example 2

Query $Q_2$

∃ a student enrolled in a course taught by his parent?

$ans(\) \leftarrow enrolled(S, C, G), \ teaches(P, C), \ parent(P, S)$
Example 2

**Query Q₂**

\[ \exists \text{ a student enrolled in a course taught by his parent?} \]

\[ \text{ans()} \leftarrow \text{enrolled}(S,C,G), \text{teaches}(P,C), \text{parent}(P,S) \]

**Figure:** Evaluation Plan for Q₂

Need to keep 2 attributes after the 1st join.
1. Speeding Up Query Evaluation
2. Graphs & Queries
3. Back to Hypergraphs
4. Computing Decompositions and Widths
5. Related Concepts & Conclusions
The Hypergraph of a Query

- Vertices are the const. & vars. of the query.
- There is a hyperedge between the terms from each atom.

Figure: Hypergraph of $Q_1$
Acyclic Queries

- Elimination Tree: order of removing edge $e_1$ in favor of $e_2$, where $e_1$ does not intersect any other edge but $e_2$.
- A query having an elimination tree is called **acyclic**.

**Figure:** Elimination tree for $Q_1$
A query decomposition [Chekuri & Rajaraman, 1997] of \( Q \) is a tree \((I, F)\) plus a map \( X : I \rightarrow \{ \text{variables & subgoals of } Q \} \) such that:

- any subgoal appears in some \( X(i) \)
- for any subgoal \( s \), \( \{ i \mid s \in X(i) \} \) is connected
- for any var. \( V \), \( \{ i \mid V \in X(i) \} \cup \{ i \mid V \text{ appears in } s \text{ and } s \in X(i) \} \) is connected
The *width* of a decomposition is $\max |X(i)|$.

The *query width* of $Q$ is the min. width over all decompositions of $Q$.

A query is acyclic iff its query width is 1.

**Figure:** Query decomposition for $Q_1$
Evaluating Queries of Bounded Querywidth

- For an acyclic query, the decomposition gives a join tree (one atom / node)
- In general, $|X(i)| \geq 1$
- Efficient evaluation (Chekuri&Rajaraman):
  - create a new relation for each node
  - bottom-up: semijoin it with the children
For an acyclic query, the decomposition gives a join tree (one atom / node)

In general, $|X(i)| \geq 1$

Efficient evaluation (Chekuri & Rajaraman):
- create a new relation for each node
- bottom-up: semijoin it with the children

For a fixed query width $k$ and a given query decomposition, query evaluation is polynomial in the combined complexity and exponential in $k$.

Query containment can be decided similarly.
Problems with Query Width

- Deciding whether the query width of a CQ is $\leq 4$ is NP-complete [Gottlob, Leone, Scarcello, 1999].
- Contrast with Yannakakis's linear time algorithm for deciding if a query is acyclic.
- Results from graph theory proved useful for defining alternative decompositions.
**Treewidth**

- *Treewidth* of a graph: previous applications in CSP, networks etc.

- *Tree decomposition* of $G = (V, E)$: tree $T = (I, F)$ and $X : I \rightarrow 2^V$ s.t.
  - $\forall$ vertex $v \in V$, $\exists i \in I$, $v \in X(i)$
  - $\forall$ edge $e \in E$, $\exists i \in I$, $e \subseteq X(i)$
  - The subset of $I$ where a vertex $v \in V$ appears is connected

- The width of a tree decomposition is $\max_i |X(i)| - 1$

- The *treewidth* of $G$ is the min. width over all tree decompositions of $G$. 
**Incidence Graph**

- *Incidence graph* of a query [Chekuri & Rajaraman, 1997]
- The treewidth of a query is the treewidth of its incidence graph.

\[ \text{incidence graph} \]

\[ \text{enrolled}(S,C,G) \]

\[ \text{teaches}(P,C) \]

\[ \text{parent}(P,S) \]

**Figure:** Incidence graph of \( Q_2 \)
Tree Decomposition

Figure: A tree decomposition for $Q_2$
Treewidth: Properties

- $tw.$ is an approximation of $qw.$:
  \[ tw(Q)/a \leq qw(Q) \leq tw(Q) + 1 \]
  where $a$ is the max. arity of a predicate of $Q$

- There are PTIME algorithms that decide if $tw(G) \leq k$ and output a corresponding tree decomposition

- Evaluating queries of bounded treewidth: a tree decomposition is also a query decomposition, so same results apply.
Unfortunately, treewidth does not tell us anything about the acyclicity of the query.
Unfortunately, treewidth does not tell us anything about the acyclicity of the query.

Example

$R_n(x_1, x_2, \ldots x_n)$ is acyclic, but has treewidth $= n - 1$. 
Unfortunately, treewidth does not tell us anything about the acyclicity of the query.

**Example**

\( R_n(x_1, x_2, \ldots, x_n) \) is acyclic, but has treewidth \( = n - 1 \).

**Example**

\( Q_2 \) is cyclic, but has treewidth only 2.
1. Speeding Up Query Evaluation

2. Graphs & Queries

3. Back to Hypergraphs

4. Computing Decompositions and Widths

5. Related Concepts & Conclusions
Hypertrees

- Main problem: treewidth is a property of graphs, while acyclicity is a property of the hypergraph.
- Solution: instead of trees, look for hypertrees [Gottlob, Leone, Scarcello]
- Hypertree: $\langle T, X, L \rangle$
  - $T = (N, E)$ is a tree
  - $p \in N$, $X(p) \subseteq \text{var}(Q)$
  - $p \in N$, $L(p) \subseteq \text{atoms}(Q)$
Hypertree decomposition

- **Hypertree decomposition**: \(< T = (N, E), X, L >\) such that
  - \(\forall\) atom \(A\), \(\exists\) vertex \(p\) s.t. \(\text{var}(A) \subseteq X(p)\)
  - The subset of \(N\) where a variable of \(Q\) appears is connected
  - \(\forall p \in N, X(p) \subseteq \text{var}(L(p))\)
  - \(\forall p \in N, X(p) \supseteq \text{var}(L(p)) \cap X(\text{subtree rooted at } p)\)
Hypertree decomposition

- **Hypertree decomposition**: \(< T = (N, E), X, L >\) such that
  - \(\forall\) atom \(A\), \(\exists\) vertex \(p\) s.t. \(\text{var}(A) \subseteq X(p)\)
  - The subset of \(N\) where a variable of \(Q\) appears is connected
  - \(\forall p \in N, X(p) \subseteq \text{var}(L(p))\)
  - \(\forall p \in N, X(p) \supseteq \text{var}(L(p)) \cap X(\text{subtree rooted at } p)\)

- **Hypertree-width**
  - The width of a hypertree decomposition is \(\max_p |L(p)|\)
  - The *hypertree-width* of \(Q\) is the min. width over all hypertree decompositions of \(Q\).
Hypertree-Width

Query $Q_2$

$Q_2$ has a hypertree decomposition of width 2.

\[
\begin{align*}
\{C, P, S\} \{&parent(P,S), \text{teaches}(P,C)\} \\
\{S, C, G\} \{&enrolled(S,C,G)\} \\
\{P, C\} \{&teaches(P,C)\} 
\end{align*}
\]
Another decomposition for $Q_2$:

\[
\{C, P, S\} \quad \{\text{parent}(P,S), \text{teaches}(P,C)\}
\]

\[
\{S, C, G\} \quad \{\text{enrolled}(S,C,G)\}
\]
Hypertree-Width: Properties

- $\forall$ CQ $Q$, $hw(Q) \leq qw(Q)$
- In particular, $Q$ is acyclic iff $hw(Q) = 1$
- $\exists Q$ s.t. $hw(Q) < qw(Q)$
- For a fixed $k$, deciding $hw(Q) \leq k$ and computing the decomposition is PTIME
- Evaluating a CQ of bounded $hw$ is PTIME (combined complexity)
- The tight bounds are in fact better than PTIME (LOGCFL-completeness)
Hypertree-Width vs. Query Width

\[ Q_4 : \text{ans} \leftarrow a(S, X, X', C, F) \land b(S, Y, Y', C', F') \land c(C, C', Z) \land d(X, Z) \land e(Y, Z) \land f(F, F', Z) \land g(X', Z') \land h(Y', Z') \land j(J, X, Y, X', Y') \]

[Gottlob, Leone, Scarcello, 1999] show that \( qw(Q_4) = 3 \) and \( hw(Q_4) = 2 \)
Evaluating Queries of Bounded Hypertree-Width

- Build a join tree, where each node joins its local atoms
- Evaluate the tree bottom-up, by upward semijoins (à la Yannakakis).
1. Speeding Up Query Evaluation

2. Graphs & Queries

3. Back to Hypergraphs

4. Computing Decompositions and Widths

5. Related Concepts & Conclusions
Algorithms for Computing Treewidth

- computing treewidth efficiently: algorithms from graph theory (see especially works by Arnborg and by Bodlaender)
- preprocessing techniques can be very useful:
  - reduction rules [Arnborg, Prokurowski]
  - splitting [Bodlaender, Koster]
- approximation algorithms: contracting edges, min-max etc
exact algorithms:

- reduction is enough to test \( k \leq 3 \)
- algorithms based on dynamic programming (such as Arnborg’s triangulation algo. applied by [Shoikhet, Geiger] to treewidth)
- [Bodlaender, Kloks, 1991]: linear time algorithm, which given a decomp. of width \( l \), det. if \( \exists \) one of width \( k \) and computes it.
- [Bodlaender, 1993]: linear time algorithm for finding tree decomp. (sometimes attacked for being slow in practice)
- other methods: elimination ordering, branch and bound etc
[Gottlob, Leone, Scarcello] give two versions of an algorithm for computing $hw$ and a corresponding decomp.:

- a LOGCFL parallel version
- a deterministic polynomial version

Important result: there exists a normal form for a hypertree decomp. that preserves the hypertree-width.

The normal form allows to characterize each child of a node by a tree component and to break the problem into smaller subproblems.

The optimal decomp. is computed by assigning weights and building a tree in dynamic prog. style.
1 Speeding Up Query Evaluation
2 Graphs & Queries
3 Back to Hypergraphs
4 Computing Decompositions and Widths
5 Related Concepts & Conclusions
Extensions of treewidth to other classes of queries 
[Flum, Frick, Grohe, 2002]

- MSO queries
- nonrecursive stratified Datalog
- CQ with negation
- tractable fragments of FOL
Other decomposition methods (for CQ & CSP):

- biconnected components [Freuder]
- cycle cutset [Dechter]
- tree clustering [Dechter, Pearl]
- queries of bounded degree of cyclicity [Gyssens, Paredaens]
## Conclusions

<table>
<thead>
<tr>
<th></th>
<th>$qw$</th>
<th>$tw$</th>
<th>$hw$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characterizes acyclicity</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Computable in PTIME</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Efficiently computable on</td>
<td></td>
<td>machines</td>
<td>no</td>
</tr>
<tr>
<td>Applications other than CQ</td>
<td>no</td>
<td>yes</td>
<td>yes (with mods.)</td>
</tr>
</tbody>
</table>