Advanced Data Management

Medha Atre

Office: KD-219
atrem@cse.iitk.ac.in

Oct 20, 2016

Keyword Searches over Relational DB

- Unlike plain text, the underlying data has inherent structure in it, which indirectly defines the relationship between the "data nodes" that contain those keywords.
- The underlying structure needs to be taken into consideration while determining the answers to the keyword searches.
- Hence the problem is no longer confined to just creating an inverted word to document ID index as is done in the IR approaches.
- Tuples are viewed as vertices in the "data-graph".
- Connections between the tuples are primary-foreign key constraints.
- Results to the keyword searches are *subgraphs* of this data-graph.

Schema-Based Keyword Search

- Two graphs considered – graph of database relations, based on the schema (schema-graph G_S), and graph of the tuples based on the schema (data-graph G_D).
- Basic SQL queries are used to locate all the tuples that contain given keywords (or subsets of the given keywords).
- A **Minimal Total** Joining Network of Tuples (MTJNT) is such that – it is a subgraph of the data-graph, where two tuples are connected to each other if they have a primary-foreign key dependency, and they contain a subset of the query keywords. Together, all the tuples in a given subgraph covers all the given keywords.

Schema-Based Keyword Search

- Size of this subgraph is controlled with T_{max} parameter to avoid arbitrarily large subgraphs. T_{max} defines the maximum distance between the two tuples in the given subgraph.
- Additionally a scoring function is defined (domain specific) to avoid generating too many results, especially for frequently occuring keywords.

Schema-based Keyword Search

- **Candidate Network Generation**: A set of candidate networks (schema-subgraphs) are generated over the given database schema graph. These set of CNs will be *complete* and *duplication free*. Algorithms like DISCOVER [Hritidis2008] S-KWS [Markowetz2007] propose to propose a good set of CNs in order to avoid evaluation of a large number of them.
- **Candidate network evaluation**: After identifying CNs, they are translated into proper SQL queries in order to get the set of candidate tuple-subgraphs, i.e., to get *all* MTJNT for the each of the CNs.

Schema-based Keyword Search

- **Candidate network evaluation**: two main challenges:
  - CNs share common subexpressions, so we want to identify and evaluate them only once to improve performance.
  - Optimizing each of the SQL queries, and especially making use of these common subexpressions in the optimization plans.

Schema-based Keyword Search

- Without complete CN evaluation:
  - Distinct root semantics: Define a distinct root, and identify all the tuples that are reachable within certain distance (D_{max}) from the root tuple – this is more like a star graph than connected trees.
  - Distinct core semantics: Instead of just one distinct root, define a community of roots, multi-centers that are connected to each other in the data-graph. Find tuples within D_{max} distance of these multi-centers, over a path following certain *path tuples*.

Graph-based Keyword Search

- Does not consider DB schema, but considers tuples and their primary-foreign key dependencies as the connections.
- No use of structured queries like SQL.
- Tree-based or Subgraph-based semantics used to decide the structure of the tuple subgraphs to be returned.
- Tree-based semantics: (1) Steiner tree based semantics, and (2) Distinct root based semantics.

Graph-based Keyword Search

- Finding optimal steiner trees is an NP-complete problem.
- But since the size of distinct keywords in the query and hence the size of the tuple subgraphs (constrained by the top-k scoring or weight function) is small, we can indeed find the optimal Steiner tree.
- BANKS-I [Bhalotia2002] uses *backward search*.
- Dynamic-Programming Best First (DPBF) [Ding2007] uses dynamic programming.

Graph-based Keyword Search

- BANKS-II proposes bidirectional search instead of just backward search.
- Bi-level indexing (BLINKS [He2007]) uses indexes to speed up BANKS-II.
- Data-graph summaries are created using graph of *SuperNodes* and *SuperEdges*. This graph can fit in memory and can be used to prune unwanted components of the data-graph to limit the search space and improve performance.

Keyword Search over Native Graphs

- Ideas remain the same, but the data representation and interpretation changes.
- Graphs often don't have an associated schema, hence native schema-based approaches are not useful.
- Graphs like RDF have edge-labels which define the relationship and can be part of the keyword searches.
- Concept of distance can be more well-defined in terms of the edge-weights in the graphs.

r-Cliques [Karger2011]

- Does not assume underlying schema (schema-less).
- Instead of tree-based substructures, it assumes arbitrary subgraphs as the answers.
- Filtering criterion is that the distance between any two pair of nodes within the given substructure is at most "r".
- For outputting top-k results, it generates all the qualifying r-cliques and then does relative ranking among them to output top-k.
- Finding optimal r-cliques is NP-hard, hence they propose a branch and bound kind algorithm, which approximates r-cliques to a factor of 2, i.e., the distance between the pair of nodes in the candidate subgraph can be at most $2r$.

r-Cliques

- Branch and Bound:
  - For each keyword in the set of keywords $\{k_1, k_2 ... k_l\}$, find all the graph nodes that contain that keyword – use pre-built inverted index.
  - Initialize *rList* to contain all the nodes for a keyword say k_1.
  - For each $k_i, 2 \leqslant i \leqslant l$, find all the nodes that contain k_i and that are within r distance from the nodes in the *rList*. Add all such qualifying nodes to the respective *rList*.

r-Cliques

- Branch and Bound is quite slow due to having to consider all the candidate nodes in a pairwise manner, hence authors propose Polynomial Delay Algorithm.
  - For each keyword in the set of keywords $\{k_1, k_2 ... k_l\}$, find the respective graph nodes that contain the particular keyword $\{C_1, C_2 ... C_l\}$.
  - Now consider the search space $C_1 \times C_2 \times ... \times C_l$, and from this find *one* top answer.
  - This is done by by iteratively choosing the shortest distance (less than r) node from every node in C_i to every other set $C_j, i \neq j$.

r-Cliques

- After outputting the top answer from this space the space is divided as follows:
- If the top answer from the original search space was $\{v_1, v_2, V-3, v_4\}$, the space is divided into following subspaces:
  - $\{C_1 - v_1\} \times C_2 \times C_3 \times C_4$
  - $C_1 \times \{C_2 - v_2\} \times C_3 \times C_4$
  - $C_1 \times C_2 \times \{C_3 - v_3\} \times C_4$
  - $C_1 \times C_2 \times C_3 \times \{C_4 - v_4\}$
- The procedure is repeated on these subspaces, until we have top-k answers, or until we can no longer produce an answer that satisfies the r distance criterion.

Top-k Keyword Queries over RDF graphs [Tran2009]

- Take an RDF graph and create a summary over it.
- Create an inverted index on the RDF data graph, and also consider IR techniques like stemming, synonyms etc.
- From a given set of keywords, first match the nodes in the summary graph, augmented with the nodes matching from the data graph.
- Form top-k SPARQL basic graph pattern queries based on various scoring parameters like path-lengths in the queries, popular score of the keywords, and keyword matching score.
- Evaluate the chosen top-k SPARQL queries over the original RDF graph to output results.
- Note that here query results can be larger than k because it is the SPARQL query candidates that are bounded by k!

Key Points

- Other approaches more or less follow the same concepts.
- Key points to note:
  - Consider graph summaries for fast pruning of search space.
  - Inverted index for fast locating the candidate data and summary graph nodes.
  - Come up with SPARQL pattern (or SQL join) queries and evaluate them to get the candidate results, filter them based on scoring function and threshold criterion.
  - Use more native approaches like Steiner Trees, Distinct Root trees, Distinct Core, r-Cliques to get the top-k answers.