N-way multi-join

Medha Atre

Aug 8, 2016

Input to the following algorithm is a list of *triple patterns*, i.e., the edges in the pattern query sorted in the ascending order of number of triples (graph edges) that match that edge pattern *after* doing the semi-joins based pruning as we saw in the lecture slides.

In practice we use additional trick (not included in the following algorithm for simplicity and clarity) – we first sort all the triple patterns in the ascending order of number of triples matching them. We pick the first triple pattern from this list, and from the second triple pattern onwards, we recursively choose the triple pattern that shares a common *wildcard node*, i.e., a join variable, e.g., *?friend* with at least one of the previously chosen triple patterns. Come to think of it, this is similar to the way we choose a *spanning tree* over a graph – every time a node is chosen such that it is connected to at least one node of the previously chosen nodes as a part of the spanning tree.

We always find such a spanning tree for our query graph, because we assume that the query graph is always *connected*, i.e., there do not exist any Cartesian products in the query. Cartesian products are very rare (and do not hold much significance) in the context of graph shaped data.

Algorithm 1: `multi-way-join`

input : vmap, stps, visited
output: all the results of the query

```
1  if visited.size == stps.size then
2  |  output (vmap);                                   // generate a single result
3  |  return;
4  if visited is empty then
5  |  tp_1 = first TP from stps;
6  |  visited.add(tp_1);
7  |  for each triple t ∈ BM_{tp_1} do
8  |  |  generate bindings for vars(tp_1) from t, store in vmap;
9  |  |  multi-way-join(vmap, stps, visited, nulreqd);
10 else
11 |  for each tp_i in stps do
12 |  |  if tp_i ∈ visited then
13 |  |  |  continue;
14 |  |  get bindings for vars(tp_i) from vmap;
15 |  |  if no bindings found then
16 |  |  |  continue;
17 |  |  atleast-one-triple = false;
18 |  |  for each triple t ∈ BM_{tp_i} with same bindings do
19 |  |  |  atleast-one-triple = true;
20 |  |  |  store vars(tp_i) bindings from t in vmap;
21 |  |  |  visited.add(tp_i);
22 |  |  |  multi-way-join(vmap, stps, visited);
23 |  |  |  visited.remove(tp_i);
24 |  |  if (atleast-one-triple == false) then
25 |  |  |  return;
```