EmptyHeaded: A Relational Engine for Graph Processing

Jason Priest
2018-02-28
6.886
Key Contributions

- Join optimization based on *generalized hypertree decomposition*
- Data and algorithm optimization based on local graph skew
Traditional Relational Query

How many github organizations (groups) have a C++ developer who registered this year?

```sql
select count(*)
from group
join user on group.users contains user.id
join project on project.owner is user.id
where user.joined after "2018-01-01"
and project.lang is "C++";
```
Join Order Matters

Group ~100
User ~5,000
Project ~10,000
Cyclic Queries

\[ Q_\Delta = R(A, B) \Join S(B, C) \Join T(A, C). \]

- Joins usually implemented pairwise, or between two sets at a time
- For cyclic queries such as above, this leads to suboptimal behavior
Pairwise Joins Insufficient

\[
R = \{a_0\} \times \{b_0, \ldots, b_m\} \cup \{a_0, \ldots, a_m\} \times \{b_0\}
\]

\[
S = \{b_0\} \times \{c_0, \ldots, c_m\} \cup \{b_0, \ldots, b_m\} \times \{c_0\}
\]

\[
T = \{a_0\} \times \{c_0, \ldots, c_m\} \cup \{a_0, \ldots, a_m\} \times \{c_0\}
\]

Figure 2: Counter-example for join-project only plans for the triangles (left) and an illustration for \(m = 4\) (right). The pairs connected by the red/green/blue edges form the tuples in the relations \(R/S/T\) respectively. Note that the in this case each relation has \(N = 2m + 1 = 9\) tuples and there are \(3m + 1 = 13\) output tuples in \(Q_\Delta\). Any pair-wise join however has size \(m^2 + m = 20\).

Worst-Case Optimal Join Algorithm
Graph Covers

\[ |Q| = |\bigwedge_{F \in \mathcal{E}} R_F| \leq \prod_{F \in \mathcal{E}} |R_F|^{x_F}. \quad (6) \]
Graph Decomposition

https://en.wikipedia.org/wiki/Tree_decomposition
Worst-Case Optimal Join Algorithm

- Brute force all decompositions of relations
- Find minimum width decomposition
- Use decomposition to inform much order of joins, order of comparing fields, etc
- Resulting plan for joins is optimal intermediate output sizes

```
ALGORITHM 2: Enumerating all GENDs via Brute Force Search
1 // Input: Hypergraph H = (V,E) and a set of parent edges P.
2 // Output: A list of GENDs (in the form of (GEND node, subtrees) pairs).
3 GEND Enumeration(\(V,E,P\))
4 GENDs = [ ]
5 // Iterate over all subset combinations of edges.
6 foreach [C | C \subseteq E] do
7 // The remaining edges, not in C.
8 \(E = E\setminus C\)
9 // If the running intersection property is broken, the GEND is
10 // not valid. The check makes sure that all attributes in the
11 // parent and subtrees of a specified GEND node also appear
12 // within the specified GEND node. Here we use U as a set
13 // of edges to indicate the union of their attributes.
14 if not (\(U = U\cup \{E\} \subseteq U\)) then continue
15 // Consider each subgraph of the remaining edges. For each
16 // subgraph, recursively enumerate all possible GENDs.
17 PartitionChildren = [ ]
18 foreach Pr in Partition(I) do
19 PartitionChildren = \(\text{GENDenumeration}(\{P,Pr\})\)
20 // Consider all possible combinations of subtrees by calling the
21 // method below, for each, construct a GEND with C as the root.
22 foreach CB in Subtrees\(\cup\)Combination(PartitionChildren) do
23 GENDs = [GCB, CB]
24 return GENDs
25 // Input: A list of lists of GENDs; for each partition of a hypergraph
26 // (the outer list), all possible decompositions for that partition
27 // (the inner lists).
28 // Output: A list where each member of this list is a list that
29 // contains one subtree from each partition.
30 Subtrees\(\cup\)Combination(PartitionChildren)
31 ChildrenCombinations = [ ]
32 if [PartitionChildren\(\subseteq\) \(\subseteq\) 0] then
33 if [PartitionChildren\(\subseteq\) \(\subseteq\) 1] then
34 // If there is only one partition, for each of the possible GENDs
35 // of this partition, add a combination with just this GEND.
36 foreach CB in PartitionChildren do
37 ChildrenCombinations = [CB]
38 else
39 // Recursively generate combinations for the partitions after
40 // the first one.
41 RemainingCombinations = Subtrees\(\cup\)Combination(PartitionChildren)[]
42 // If there is more than one partition, each subtree is the
43 // first partition is combined with each list of subtrees in
44 // the recursively generated combinations for the remaining
45 // partitions.
46 foreach CB in RemainingChildren do
47 foreach C in RemainingChildren do
48 FinalCombination = [CB, C]
49 ChildrenCombinations = [FinalCombination]
50 return ChildrenCombinations
```
Skew

- Density skew
  - some values are much more common
  - some relations are much more selective

- Cardinality skew
  - some tables are much larger
  - some nodes have much greater degree
Set Layout

- bitsets: bitvectors with offsets to first element
- pshort: nearby values may have similar prefix, thus store repeated high 16 bits
- varint: difference encoding with continue bits
- uint: sorted array with binary search
Set Intersection Algorithm

- uint & uint
  - 5 SIMD techniques, chosen based on density skew and cardinality
- bitset & bitset
  - Just SIMD AND comparison
Evaluation
<table>
<thead>
<tr>
<th>Name</th>
<th>Query Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangle</td>
<td>Triangle(x,y,z):-R(x,y),S(y,z),T(x,z).</td>
</tr>
<tr>
<td>4-Clique</td>
<td>4Clique(x,y,z,w):-R(x,y),S(y,z),T(x,z),U(x,w),V(y,w),Q(z,w).</td>
</tr>
<tr>
<td>Lollipop</td>
<td>Lollipop(x,y,z,w):-R(x,y),S(y,z),T(x,z),U(x,w).</td>
</tr>
<tr>
<td>Barbell</td>
<td>Barbell(x,y,z,x',y',z'):-R(x,y),S(y,z),T(x,z),U(x,x'),R'(x',y'),S'(y',z'),T'(x',z').</td>
</tr>
<tr>
<td>Count Triangle</td>
<td>CntTriangle(;w:long):-R(x,y),S(x,z),T(x,z); w=&lt;COUNT(*)&gt;&gt;.</td>
</tr>
<tr>
<td>4-Clique-Selection</td>
<td>S4Clique(x,y,z,w):-R(x,y),S(y,z),T(x,z),U(x,w),V(y,w),Q(z,w),P(x,<code>node</code>).</td>
</tr>
<tr>
<td>Barbell-Selection</td>
<td>SBarbell(x,y,z,x',y',z'):-R(x,y),S(y,z),T(x,z),U(x,<code>node'),V(</code>node',x'),R'(x',y'),S'(y',z'),T'(x',z').</td>
</tr>
<tr>
<td>PageRank</td>
<td>N(;w:int):-Edge(x,y); w=&lt;COUNT(x)&gt;.</td>
</tr>
<tr>
<td></td>
<td>PageRank(x;y:float):-Edge(x,z); y= 1/N.</td>
</tr>
<tr>
<td></td>
<td>PageRank(x;y:float)<em>[i=5]:-Edge(x,z),PageRank(z),InvDeg(z); y=0.15+0.85</em>&lt;SUM(z)&gt;&gt;.</td>
</tr>
<tr>
<td>SSSP</td>
<td>SSSP(x;y:float):-Edge(`start',x) y=1.</td>
</tr>
<tr>
<td></td>
<td>SSSP(x;y:float)*:-Edge(w,x),SSSP(w); y=&lt;MIN(w)&gt;&gt;+1.</td>
</tr>
</tbody>
</table>
## Performance

Table 9. Triangle Counting Runtime (in Seconds) for EmptyHeaded and Relative Slowdown for Other Engines Including PowerGraph, a Commercial Graph Tool (CGT-X), Snap-Ringo, SocialLite, and LogicBlon

<table>
<thead>
<tr>
<th>Dataset</th>
<th>EmptyHeaded</th>
<th>Low-Level</th>
<th>High-Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PowerGraph</td>
<td>CGT-X</td>
</tr>
<tr>
<td>Google+</td>
<td>0.31</td>
<td>8.40×</td>
<td>62.19×</td>
</tr>
<tr>
<td>Higgs</td>
<td>0.15</td>
<td>3.25×</td>
<td>57.96×</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>0.48</td>
<td>5.17×</td>
<td>3.85×</td>
</tr>
<tr>
<td>Orkut</td>
<td>2.36</td>
<td>2.94×</td>
<td>-</td>
</tr>
<tr>
<td>Patents</td>
<td>0.14</td>
<td>10.20×</td>
<td>7.45×</td>
</tr>
<tr>
<td>Twitter</td>
<td>56.81</td>
<td>4.40×</td>
<td>-</td>
</tr>
</tbody>
</table>

48 threads used for all engines. “-” indicates the engine does not process over 70 million edges. “t/o” indicates the engine ran for over 30 minutes.
### Performance

Table 11. SSSP Runtime (in Seconds) Using 48 Threads for All Engines

<table>
<thead>
<tr>
<th>Dataset</th>
<th>EmptyHeaded</th>
<th>Low-Level</th>
<th></th>
<th></th>
<th>High-Level</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Galois</td>
<td>PowerGraph</td>
<td>CGT-X</td>
<td>Socialite</td>
<td>LogicBlox</td>
<td></td>
</tr>
<tr>
<td>Google+</td>
<td>0.024</td>
<td>0.008</td>
<td>0.22</td>
<td>0.51</td>
<td>0.27</td>
<td>41.81</td>
<td></td>
</tr>
<tr>
<td>Higgs</td>
<td>0.035</td>
<td>0.017</td>
<td>0.34</td>
<td>0.91</td>
<td>0.85</td>
<td>58.68</td>
<td></td>
</tr>
<tr>
<td>LiveJournal</td>
<td>0.19</td>
<td>0.062</td>
<td>1.80</td>
<td>-</td>
<td>3.40</td>
<td>102.83</td>
<td></td>
</tr>
<tr>
<td>Orkut</td>
<td>0.24</td>
<td>0.079</td>
<td>2.30</td>
<td>-</td>
<td>7.33</td>
<td>215.25</td>
<td></td>
</tr>
<tr>
<td>Patents</td>
<td>0.15</td>
<td>0.054</td>
<td>1.40</td>
<td>4.70</td>
<td>3.97</td>
<td>159.12</td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>7.87</td>
<td>2.52</td>
<td>36.90</td>
<td>-</td>
<td>x</td>
<td>379.16</td>
<td></td>
</tr>
</tbody>
</table>

"-" indicates the engine does not process over 70 million edges. The other engines include Galois, PowerGraph, a commercial graph tool (CGT-X), Socialite, and LogicBlox. "x" indicates the engine did not compute the query properly.