DualSim: Parallel Subgraph Enumeration in a Massive Graph on a Single Machine

Hyeonji Kim, Juneyoung Lee, Sourav S. Bhowmick, Wook-Shin Han, JeongHoon Lee, Seongyun Ko, Moath H.A. Jarrah

Presented by: Bishesh Khadka
MIT 6.886 - Graph Analytics
Motivation

- Subgraph enumeration
  - Network motif discovery
  - Graphlet kernel computation
  - Subgraph frequency
Existing Work

- MapReduce & Distributed Graph Engines
  - Redundant work-- duplicated edges
  - Multiway join memory intensive
- Shao et al.
  - PSGL
    - No multiway join
- All approaches have serious performance issues
Problem Statement

- Can subgraph enumeration be done disk-based, on a single machine in a way that is scalable and efficient?

Motivating Principles
- Existing methods fail due to exponential partial solutions
- Disk access one of costliest bottlenecks
- CPU stall also notable bottleneck
Key Contributions

- DualSim does not maintain explosive partials
- Dual Approach
  - V-group sequence
  - V-group forest
  - Further optimizations
- Red Black Ivory query graph transformation
- Evaluation vs. state-of-the-art subgraph enumeration techniques
Dual Approach

- Given query graph q, data graph g, page graph p
(a) Data graph $g$, query graph $q$, and page graph $p_g$. 

Partial order: $u_2 < u_1$ 
Buffer frames
Dual Approach

- Given query graph $q$, data graph $g$, page graph $p$
- Enumerate all possible query sequences
- Full-order query sequences
  - Each matches an ordered data seq. $\Rightarrow$ fixes data seq.
(b) Full-order query sequences and \( v \)-group sequences.
Dual Approach

- Given query graph $q$, data graph $g$, page graph $p$
- Enumerate all possible query sequences
- Full-order query sequences
  - Each matches an ordered data seq. $\Rightarrow$ fixes data seq.
- Prune FOQS with given partial orders
- Group FOQS into v-group sequences based on topology
(b) Full-order query sequences and $v$-group sequences.
Dual Approach

- Given query graph q, data graph g, page graph p
- Enumerate all possible query sequences
- Full-order query sequences
  - Each matches an ordered data seq. \(\Rightarrow\) fixes data seq.
- Prune FOQS with given partial orders
- Group FOQS into v-group sequences based on topology
- Traverse graph stored in pages based on v-group sequences
(a) Data graph $g$, query graph $q$, and page graph $p_g$.

(b) Matching v-group sequences:

<table>
<thead>
<tr>
<th>Page sequences</th>
<th>Matching v-group sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $(p_0, p_1, p_1)$</td>
<td>$vgs_2$</td>
</tr>
<tr>
<td>2. $(p_0, p_1, p_2)$</td>
<td>$vgs_1, vgs_2$</td>
</tr>
<tr>
<td>3. $(p_0, p_1, p_3)$</td>
<td>$vgs_1$</td>
</tr>
<tr>
<td>4. $(p_0, p_2, p_2)$</td>
<td>$vgs_2$</td>
</tr>
<tr>
<td>5. $(p_0, p_2, p_3)$</td>
<td>$vgs_1$</td>
</tr>
<tr>
<td>6. $(p_1, p_2, p_2)$</td>
<td>$vgs_2$</td>
</tr>
<tr>
<td>7. $(p_1, p_2, p_3)$</td>
<td>$vgs_1, vgs_2$</td>
</tr>
<tr>
<td>8. $(p_1, p_3, p_3)$</td>
<td>$vgs_2$</td>
</tr>
<tr>
<td>9. $(p_2, p_3, p_3)$</td>
<td>$vgs_1, vgs_2$</td>
</tr>
<tr>
<td>10. $(p_3, p_3, p_3)$</td>
<td>$vgs_2$</td>
</tr>
</tbody>
</table>

(c) A page mapping example.
Dual Approach cont’d

- For each valid mapping
  - enumerate all FOQS in v-group sequence
  - Generate data mappings
    - Vertex level match: (v1, v3, v5)
    - For Vgs1: only 1 FOQS: (u3, u2, u1)
      - Solution: {(u3, v1), (u2, v3), (u1, v5)}
      - (v2, v3, v5) also valid match
(d) A vertex mapping example for the page sequence \((p_0, p_1, p_2)\).
RBI Graph

- Idea: Disk reads minimized if we use minimum number of query vertices during graph traversal
- Colored vertices:
  - Red: Data must retrieve adj. List
  - Ivory:
    - Is adj to $m > 1$ reds
    - $m$-way intersection
  - Black:
    - Is adj to $m = 1$ red
    - Scan red's adj list
- MCVC is colored red
  - Reachability
  - NP hard but $|Vq|$ small enough for approx.
DualSim Algorithm

Algorithm 1. DUALSIM

Input: A data graph $g$, A query graph $q$

1. Preparation step

   /\ 1.
   
   $PO \leftarrow$ FINDPARTIALORDERS($q$);
   
   $(q_{RB1}, q_R) \leftarrow$ GENERATERB1QUERYGRAPH($q, PO$);
   
   $\{vgs_i\} \leftarrow$ FINDVGROUPSEQUENCES($q_R, PO$);
   
   $mo_g \leftarrow$ FINDGLOBALMATCHINGORDER($\{vgs_i\}$);
   
   $\{vgf_i\} \leftarrow$ BUILDVGROUPFORESTS($\{vgs_i\}, mo_g$);

2. Execution step

   /\ 2.
   
   INITIALIZECANDIDATESEQUENCES(\text{root nodes in } \{vgf_i\});
   
   foreach (merged vertex/page window $(mvw_1, mpw_1)$ from $\{vgf_i[1]\}$) do
   
   foreach (page id $pid \in mpw_1$) do
   
   
   | AsyncRead($pid$, COMPUTECANDIDATESEQUENCES($pid$, \n   
   | \{cvw_{1,1}\}, all child nodes of $\{vgf_i[1]\}$));
   
   end
   
   wait until COMPUTECANDIDATESEQUENCES executions are finished
   
   level $\leftarrow$ 2;
   
   DELEGATEEXTERNALSUBGRAPHENUMERATION($level, q_{RB1}$, \n   
   \{vgs_i\}, $\{vgf_i\}$, $\{mvw_j\}$, $\{mpw_j\}$);
   
   INTERNALSUBGRAPHENUMERATION($mvw_1$, $mpw_1$);

   UNPINPAGES($mpw_1$);

   CLEARCANDIDATESEQUENCES(\text{the children of } $\{vgf_i[1]\}$);

end
Evaluation - Single Machine

(a) Query $q_1$.  
(b) Query $q_4$.

Figure 10: Varying datasets in a single machine.

(c) Query $q_3$.  
(d) Query $q_4$.

Figure 12: Varying graph size in a single machine.

(a) Query $q_1$.  
(b) Query $q_2$.  
(c) Query $q_3$.  
(d) Query $q_4$.  
(e) Query $q_5$.

Figure 11: Varying queries in a single machine.
Evaluation - Cluster

Figure 13: Varying datasets in a cluster.

Figure 14: Varying queries in a cluster.

Figure 15: Varying graph size in a cluster.
Conclusion

- Significant CPU processing reduction due to dual approach’s traversal
- Disk I/O reduction
- DualSim outperforms existing solutions in both single machine and distributed environment for subgraph enumeration
References

- Hyeonji Kim, Juneyoung Lee, Sourav S. Bhowmick, Wook-Shin Han, JeongHoon Lee, Seongyun Ko, Moath H.A. Jarrah
  DualSim: Parallel Subgraph Enumeration in a Massive Graph on a Single Machine