Sublinear-Space Bounded-Delay Enumeration for Massive Network Analytics: Maximal Cliques

Alessio Conte, Roberto Grossi
Andrea Marino, Luca Versari

conte@di.unipi.it - luca.versari@sns.it

University of Pisa
Scuola Normale Superiore
Graph enumeration: does and donts

Given a graph list all the wanted solutions

- **Brute force**: test all subsets of vertices and check if they form a solution

- **Better way**: generate only the wanted solutions, possibly in a “smart” way
Why cliques?

Fundamental in **Network Analysis**

- It generalizes **triangles** (that show how clustered or “**social**” a network is)
- Used to define communities
Notation

**G**: an undirected graph with \( n \) nodes and \( m \) edges.

**Clique**: a set of nodes of \( G \) all pairwise connected.

**Maximal**: if *not* properly contained in another clique.
Input:
"Smart" enumeration:

Bounded-Delay:
Maximum time between two outputs is bounded by \(\text{poly}(m,n)\)

Sublinear-Space:
Use \(o(|E|)\) (small o) memory in addition to the input graph
Bounded-Delay:
Maximum time between two outputs is bounded by $\text{poly}(m,n)$

Sublinear-Space:
Use $o(|E|)$ (small o) memory in addition to the input graph
Bounded-delay/output-sensitive: why?

- **Bounded-delay** gives us **output-sensitive** algorithms (cost $\propto$ output)

- **Output-sensitive** algorithms are great when the number of solutions is variable
Some output-sensitive results:

**Listing Maximal Acyclic Subgraphs:**

- **Schwikowski et al.** On enumerating all minimal solutions of feedback problems. *DAM 2002*

**Listing Trees:**

- **Shioura et al.** An Optimal Algorithm for Scanning All Spanning Trees of undirected Graphs. *SIAM J COMPUT 1997*

**Listing Paths/Cycles:**

- **Eppstein et al.** Finding the K Shortest Paths. *SIAM J COMPUT 1998*
- **Birmelé et al.** Optimal Listing of Cycles and st-Paths in Undirected Graphs. *SODA 2012*
- **Johnson** Finding All the Elementary Circuits of a Directed Graph. *SIAM J COMPUT 1975*

**Listing Pseudo Cliques:**

- **Uno** An efficient algorithm for solving pseudo clique enumeration problem. *ALGORITHMICA 2010*

**Listing Triangles:**

- **Bjørklund et al.** Listing Triangles. *ICALP'14*
- **Kopelowitz et al.** Higher Lower Bounds from the 3SUM Conjecture. *SODA’16*
Delay: a stronger constraint

ITERATOR
getNextSolution()
Delay: a stronger constraint

The complexity of an iterator step is the *delay* of the algorithm
Bounded-Delay:
Maximum time between two outputs is bounded by $\text{poly}(m,n)$

Sublinear-Space:
Use $o(|E|)$ (small o) memory in addition to the input graph
Advantages in modern multicore environments:

- Input graph is read-only in the shared memory
- Each algorithm execution has small footprint (read/write) that fits in the private L1/L2 cache

Many instances can run in parallel without write conflicts and cache coherence issues
Cache coherence:

“Working” memory

Input graph
Cache coherence:

“Working” memory

Input graph

NO PROPAGATION NEEDED!
Back to maximal cliques: state of the art
State of the art:

Algorithms can be classified as

- Based on Bron-Kerbosch (BK)
- Based on Reverse-Search
State of the art: BK-based

[Eppstein et al] Listing All Maximal Cliques in Large Sparse Real-World Graphs  JEA 2013

- Total running time: $O(nd^{3(d/3)})$ (d is the degeneracy) which can be optimal on sparse graphs

- Additional space is $O(n + d\Delta)$

- Delay is $\Omega(3^{n/6})$
Degeneracy ordering:

- Each vertex has at most $d$ neighbors to its right
- E.G., Recursively remove smallest-degree node

Example with $d = 3$
Eppstein et al:

Main idea

- Process the graph in a *degeneracy* ordering
- This creates $n$ subgraphs with $d$ nodes to visit
Eppstein et al: exponential delay

Degeneracy Ordering: \(\{1,2,3,4, x \ldots\}\)
Degeneracy Ordering: \{1,2,3,4, x \ldots\} \hspace{1cm} (x \text{ is a node of the Moon-Moser graph})

When \(x\) is processed, NO clique is found!
Eppstein et al: exponential delay

Processing \( x \): recursive calls make a complete ternary tree of height \( k-1 \), i.e. there are \( \Theta(3^k) = \Theta(3^{n/6}) \) calls.
State of the art: reverse-search

[Tsukiyama et al] A new algorithm for generating all the maximal independent sets

SIAM J COMPUT 1977

- Can be used for cliques (use the complement graph)
  - Builds a solution tree which can be traversed
    - Additional space is $O(m)$
    - Delay is $O(mn)$
State of the art: reverse-search

- [Chiba-Nishizeki] $O(md)$ $O(m)$
- [Makino-Uno] $O(\Delta^4)$ $O(m)$
- [Chang et al] $O(\Delta h^3)$ $O(n^2)$
- [Comin-Rizzi] $O(n^{2.09})$ $O(n^{4.27})$
- See paper for more...
State of the art: reverse-search

- [Chiba-Nishizeki] \(O(md)\) \(\Theta(m)\)
- [Makino-Uno] \(O(\Delta^4)\) \(\Theta(m)\)
- [Chang et al] \(O(\Delta h^3)\) \(O(n^2)\)
- [Comin-Rizzi] \(O(n^{2.09})\) \(O(n^{4.27})\)
- See paper for more...
- [This work] \(\tilde{O}(qd(\Delta+qd))\) \(O(q)\)
- [This work] \(\tilde{O}(\min\{qd\Delta,md\})\) \(O(d)\)

Note: \(q < d < h < \Delta < n\) sublinear!
## Performance:

<table>
<thead>
<tr>
<th>ALGO.</th>
<th>TIME (sec)</th>
<th>DELAY (ms)</th>
<th>MEM (MiB)</th>
<th>TIME (sec)</th>
<th>DELAY (ms)</th>
<th>MEM (MiB)</th>
<th>TIME (sec)</th>
<th>DELAY (ms)</th>
<th>MEM (MiB)</th>
<th>TIME (sec)</th>
<th>DELAY (ms)</th>
<th>MEM (MiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>&gt;2h</td>
<td>475</td>
<td>97.56</td>
<td>&gt;2h</td>
<td>636</td>
<td>78.26</td>
<td>&gt;2h</td>
<td>5285</td>
<td>258.21</td>
<td>&gt;2h</td>
<td>4077</td>
<td>164.54</td>
</tr>
<tr>
<td>MU</td>
<td>39.5</td>
<td>18</td>
<td>3.01</td>
<td>16.9</td>
<td>22</td>
<td>3.01</td>
<td>6102.7</td>
<td>3691</td>
<td>52.62</td>
<td>&gt;2h</td>
<td>11395</td>
<td>30.52</td>
</tr>
<tr>
<td>CXQ</td>
<td>21.2</td>
<td>4</td>
<td>0.11</td>
<td>60.7</td>
<td>12</td>
<td>0.25</td>
<td>&gt;2h</td>
<td>6004</td>
<td>1.02</td>
<td>&gt;2h</td>
<td>621</td>
<td>3.15</td>
</tr>
<tr>
<td>RALG2</td>
<td>1.8</td>
<td>0.6</td>
<td>1.57</td>
<td>3.1</td>
<td>0.8</td>
<td>0.81</td>
<td>66.6</td>
<td>401</td>
<td>6.46</td>
<td>237.4</td>
<td>245</td>
<td>3.16</td>
</tr>
<tr>
<td>ALG2</td>
<td>2.3</td>
<td>15</td>
<td>0.01</td>
<td>4.4</td>
<td>0.9</td>
<td>0.01</td>
<td>100.7</td>
<td>410</td>
<td>0.03</td>
<td>363.8</td>
<td>277</td>
<td>0.02</td>
</tr>
</tbody>
</table>

**CN:** Chiba-Nishizeki [11]  
**MU:** Sparse graph Makino-Uno [25]  
**CXQ:** Chang et al. [9]  
**RALG2:** Recursive Algorithm 2  
**ALG2:** Algorithm 2
## Comparison:

<table>
<thead>
<tr>
<th>ALGO.</th>
<th>DBLP-2008</th>
<th>Amazon-0505</th>
<th>IN-2004</th>
<th>EU-2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>511163</td>
<td>410236</td>
<td>1353703</td>
<td>862664</td>
</tr>
<tr>
<td>m</td>
<td>1871070</td>
<td>2439436</td>
<td>13126172</td>
<td>16138468</td>
</tr>
<tr>
<td>α</td>
<td>447563</td>
<td>1034135</td>
<td>3384922</td>
<td>5727256</td>
</tr>
<tr>
<td>Δ</td>
<td>576</td>
<td>2760</td>
<td>21869</td>
<td>68963</td>
</tr>
<tr>
<td>d</td>
<td>114</td>
<td>10</td>
<td>488</td>
<td>388</td>
</tr>
<tr>
<td>q</td>
<td>115</td>
<td>11</td>
<td>489</td>
<td>387</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALGO.</th>
<th>TIME sec</th>
<th>DELAY ms</th>
<th>MEM MiB</th>
<th>TIME sec</th>
<th>DELAY ms</th>
<th>MEM MiB</th>
<th>TIME sec</th>
<th>DELAY ms</th>
<th>MEM MiB</th>
<th>TIME sec</th>
<th>DELAY ms</th>
<th>MEM MiB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>&gt;2h</td>
<td>475</td>
<td>97.56</td>
<td>&gt;2h</td>
<td>636</td>
<td>78.26</td>
<td>&gt;2h</td>
<td>5285</td>
<td>258.21</td>
<td>&gt;2h</td>
<td>4077</td>
<td>164.54</td>
</tr>
<tr>
<td>MU</td>
<td>39.5</td>
<td>18</td>
<td>3.01</td>
<td>16.9</td>
<td>22</td>
<td>3.01</td>
<td>6102.7</td>
<td>3691</td>
<td>52.62</td>
<td>&gt;2h</td>
<td>11395</td>
<td>30.52</td>
</tr>
<tr>
<td>CXQ</td>
<td>21.2</td>
<td>4</td>
<td>0.11</td>
<td>60.7</td>
<td>12</td>
<td>0.25</td>
<td>&gt;2h</td>
<td>6004</td>
<td>1.02</td>
<td>&gt;2h</td>
<td>621</td>
<td>3.15</td>
</tr>
<tr>
<td>RALG2</td>
<td>1.8</td>
<td>0.6</td>
<td>1.57</td>
<td>3.1</td>
<td>0.8</td>
<td>0.81</td>
<td>66.6</td>
<td>401</td>
<td>6.46</td>
<td>237.4</td>
<td>245</td>
<td>3.16</td>
</tr>
<tr>
<td>ALG2</td>
<td>2.3</td>
<td>15</td>
<td>0.01</td>
<td>4.4</td>
<td>0.9</td>
<td>0.01</td>
<td>100.7</td>
<td>410</td>
<td>0.03</td>
<td>363.8</td>
<td>277</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Our algorithm
Child clique:

\[ D = \text{COMPLETE}(K' \cup \{V\}) \]
Complete:

\[ \text{COMPLETE}({2}) \]
Complete:

$\text{COMPLETE}({\{2\}})$
Complete:

\text{COMPLETE}\left(\{2\}\right)
Complete:

\text{COMPLETE}\{\{2\}\}

MAXIMAL!
Child clique:
Solution graph:
Parent clique:
Implicit solution tree:

State during the implicit tree traversal: current node + current edge. Going up from K ⇔ parent(K) and PI(K).
Checking a child [Makino-Uno]:

Check if an edge should be used (the straight way):

\[
PARENT(D) = K \text{ and } PI(D) = v
\]

**THIS IS THE COMPUTATIONAL BOTTLENECK!!**
Improved Check:

From:

\[ \text{PARENT}(D) = K \text{ and } \text{PI}(D) = v \]

To (our way):

\[ \text{N}(K'_v) \text{ is disjoint from } N_{<}(v) \text{ and } B_K \]

The children of a clique can now be found in

\[ O(qd\Delta) \]
What is $B_K$?

$v$ is in $B_K$ if it satisfies the following property:

For each $u \in K$ smaller than $v$, $u$ is a neighbor of $v$. 
Improved delay:

Compute $B_k$ every time we change clique and store it:

$O(d)$ space

$O(\min(qd\Delta, md))$ time
$O(q)$ space:

**Iterate** over $B_k$ in increasing order whenever we need it:

$O(q)$ space

$O(qd(\Delta+qd))$ time
Conclusions:

- Output-sensitive algorithms can be **efficient** both in theory and in practice.
- Stateless search can heavily reduce **space usage**.
- We showed these principles applied to **MCE**.
The end.
Thank you for your attention.
Any questions?