GC300- Graph Compression using Triad 300
Developer Harshavardhan Achrekar
Course: - Massive Graph taught by Professor Grinstein

GC300 is a graph compression technique that works for massive real world graphs that obey power law and small world properties and which cannot be read into the main memory all at once.

Inspiration of this project
One Day while having a lunch buffet with my friend “Tarun Makhija”, I took a few things which I would say starters in my plate and then made multiple trips to fetch rest of the lunch until the point I was full. My Friend took all the food at once and then never went to fetch any till he was full too. In the compression world I follow model 1 multiple fetch over model 2(used by my friend) all at once. My idea is compress the graph as you move along. My graph is massive in terms of the number of nodes as well as edges. I compress the network at several points in time and create snapshot of the network over time.

Asynchronous Call to Twitter Streaming API
Twitter provides asynchronous calling to its streaming API to fetch samples of data once you are authorized and authenticated.
stream_url="http://stream.twitter.com/1/statuses/sample.json";
Out of the many dataset that I could choose from I preferred a network that obeyed power law and was scale free network. The scale free model much like Barabasi model has a probability of making connections to other nodes in a network. This could be possible and much like obtainable in a real world network. Twitter users interactions in real time provide us with scenario suitable to perform my set of experiments.
Code Provided :- streamming_code.txt

GraphML Network Generation
Twitter stream has fields which provides us the status id of the tweet, the user who tweeted, Timestamp and the tweet was in response to an earlier tweet by a user or a new tweet altogether. If the “reply_to_status_id” is null then I simply create a node in the network and give the userid as its label. If the “reply_to_status_id ” is not null then I create 2 nodes one for a user who wrote the previous tweet and one for the user who is writing this tweet. I do not duplicate node creation ad only unique nodes can exist in a network. Multiple edge do exists as part of to and fro communication between two users on same or different topic.
Edge creation is an outcome of the same process with an edge created to node “reply_to_user_id” from “user_id” field of the json-parsed format. All this has been implemented as part of database schema for faster retrieval. The outcome of this
program is an xml generated file which can be converted into a GRAPHML for importing it into socknetV or Pajek that uses .net format as input.

Code Provided :- GC300_NETWORK_GRAPHML.txt
File :- Pajek.graphml
Format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
  http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="directed">
    <node id="76227604">
      <data key="name">76227604</data>
    </node>
    <node id="86020378">
      <data key="name">86020378</data>
    </node>
    <node id="90161364">
      <data key="name">90161364</data>
    </node>
    ....
    <edge id="e12391" directed="true" source="75861739" target="56744055"/>
    <edge id="e12392" directed="true" source="87518104" target="75861739"/>
    <edge id="e12393" directed="true" source="75861739" target="87518104"/>
  </graph>
</graphml>
```

**Connected Component**

As suggested by professor Grinstein the techniques of compression does have overhead and does not scale well for a sparse graph hence I decided to identify the connected components in the network and try and compress them. I could identify the connected components in my network using option within Pajek Net→Component→weak. I verified it using R as well. I took data visualization I course last semester under from Grinstein.

**Triad**

If we select three vertices and the lines among them in the directed network we always get one of the 16 combinations. The subgraph on three vertices is called a *triad*. 
Technique
Traid 300 is a fully connected clique and is building block of my network compression technique. I try and match this traid 300 over my network and extract fragments from within my network that resembles size 3 cliques. Then for each connected component/subgraph I reduce all the nodes in that subgraph to a single supernode identified as a representative of all the other nodes. This node is chosen
as the one with highest degree. In case of conflict in the degree centrality within that
subgraph, I randomly choose one of the many possible candidate nodes which can
be supernodes.
Once the choice of supernodes is made all nodes in each subgraph other than
supernodes are shrink/remove/replaced by those shrink nodes along with all edges
imbibed within them.
This technique is applied recursively because in a network like ours there is an
increase in the number of triangles been detected over shrinking the nodes to
supernodes as it exhibits the small world property of a dense network.
We call it compression because at every stage we compress the nodes as well as
edges till our network fits into the memory. Our scale for comparing our
compression is number of nodes reduces as well as number of edges removed from
network. One more performance parameter being “number of triads found”. Thus it
is trial that our compression has better results when number of edges in a network
grows more rapidly as compared to number of nodes creating a high interaction
among nodes within a network.
We rely on the property that there exists a large number of edges between friends
within a network than number of friends between other networks.
Also Clique identification is a time consuming technique and does not scale well for
large scale networks/massive graphs.

Experimental setup:

Pajek.net is my original network with 6459 vertices and 12393 edges.
Click.net is my triad 300 graph with 3 vertices and 6 edges.

Clique.net triad 300.
On using the $n^2$ approximation triad detection algorithm we could search for 7 fragments from a symmetrized pajek.net network.

---

Searching fragments

---

Working...

7 fragments found.
The fragment which form the triads as seen as extracted from the pajek.net is shown below. It has two subgraphs. So we have two supernodes.

Subgraph 1 contains
12367722 → 1630
14570033 → 1599............supernode 1....highest degree
84702700 → 1598
6232242 → 383
15969179 → 371
24027953 → 131
27396708 → 223
Subgraph 2 contains
60536939 → 348
24769795 → 349
107442865 → 382............supernode 2...highest degree
84473231 → 3993
So we shrink 11 nodes to two nodes and 11 edges.

5. Subnetwork induced by Ind fragments like 4 in 2 (11)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Line</th>
<th>Value</th>
<th>Line-Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3-6</td>
<td>1.00000</td>
<td>60536939-107442865</td>
</tr>
<tr>
<td>2</td>
<td>6-11</td>
<td>1.00000</td>
<td>107442865-84473231</td>
</tr>
<tr>
<td>3</td>
<td>1-8</td>
<td>1.00000</td>
<td>24027953-8470270</td>
</tr>
<tr>
<td>4</td>
<td>1-9</td>
<td>1.00000</td>
<td>24027953-14570033</td>
</tr>
<tr>
<td>5</td>
<td>2-9</td>
<td>1.00000</td>
<td>27396708-14570033</td>
</tr>
</tbody>
</table>
Pajek_1.net
After replacing the nodes in the original network of pajek.net to supernodes and removing all interconnected edges in the extracted subgraph the left network is given a new name “pajek_1” which implies it is the network reduced once using triad technique. This new network has 463 triads as shown by triad census.

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of Triads (nl)</th>
<th>Expected (el)</th>
<th>(nl-el)/el</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>102</td>
<td>23545.06</td>
<td>2380.33</td>
<td>Balan</td>
</tr>
<tr>
<td>16</td>
<td>300</td>
<td>463</td>
<td>1920214220657.76</td>
<td>Balan</td>
</tr>
<tr>
<td>1</td>
<td>003</td>
<td>44833090566</td>
<td>44776707373.79</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>021D</td>
<td>23545.06</td>
<td>-1.00</td>
<td>Rank</td>
</tr>
<tr>
<td>5</td>
<td>021U</td>
<td>23545.06</td>
<td>-1.00</td>
<td>Rank</td>
</tr>
<tr>
<td>9</td>
<td>030T</td>
<td>19.71</td>
<td>-1.00</td>
<td>Rank</td>
</tr>
<tr>
<td>12</td>
<td>120D</td>
<td>0.00</td>
<td>-1.00</td>
<td>Rank</td>
</tr>
<tr>
<td>13</td>
<td>120U</td>
<td>0.00</td>
<td>-1.00</td>
<td>Rank</td>
</tr>
<tr>
<td>2</td>
<td>012</td>
<td>112477744.17</td>
<td>-1.00</td>
<td>Trans</td>
</tr>
<tr>
<td>14</td>
<td>120C</td>
<td>0.01</td>
<td>-1.00</td>
<td>Hier</td>
</tr>
<tr>
<td>15</td>
<td>210</td>
<td>0.00</td>
<td>-1.00</td>
<td>Hier</td>
</tr>
<tr>
<td>6</td>
<td>021C</td>
<td>47090.12</td>
<td>-1.00</td>
<td>Forb</td>
</tr>
<tr>
<td>7</td>
<td>111D</td>
<td>19.71</td>
<td>-1.00</td>
<td>Forb</td>
</tr>
<tr>
<td>8</td>
<td>111U</td>
<td>19.71</td>
<td>-1.00</td>
<td>Forb</td>
</tr>
<tr>
<td>10</td>
<td>030C</td>
<td>6.57</td>
<td>-1.00</td>
<td>Forb</td>
</tr>
<tr>
<td>11</td>
<td>201</td>
<td>143427</td>
<td>34753994.08</td>
<td>Forb</td>
</tr>
</tbody>
</table>

Chi-Square: 894177362901323.1480***
6 cells (37.50%) have expected frequencies less than 5.
The minimum expected cell frequency is 0.00.

Search fragments

Working...
100 fragments found.
200 fragments found.
300 fragments found.
400 fragments found.
463 fragments found.

Pajek_463_extract.net
This time we reduce 213 vertices with 656 edges to a single vertex as it is a single connected component.
Subgraph 1 contains 213 vertices out of which we choose one node as supernode. Based on its highest degree.

\[ 29657072 \rightarrow 113 \ldots \ldots \text{supernode 3...highest degree} \]
pajek_2.net
After replacing supernode 3 the new network left out from pajek_1.net is pajek_2.net with 447 triads as reported by triad census.

---

Searching fragments

Working...
100 fragments found.
200 fragments found.
300 fragments found.
400 fragments found.
447 fragments found.

Pajek_447_extract.net
So now we have extracted triad graph with 389 vertices and 835 edges which will be reduced to a single supernode.
The extracted network looks like shown below.
Results

Attached in a separate excel sheet gc300.xlsx

<table>
<thead>
<tr>
<th>Original network</th>
<th>extracted subgraph</th>
<th>original n/w nodes</th>
<th>original n/w Edges</th>
<th>no of Triads</th>
<th>extracted n/w nodes</th>
<th>extracted n/w edges</th>
<th>supernode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pajek.net</td>
<td>pajek_7_extract</td>
<td>6450</td>
<td>12393</td>
<td>7</td>
<td>11</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Pajek_1.net</td>
<td>pajek_463_extract</td>
<td>6450</td>
<td>12377</td>
<td>463</td>
<td>213</td>
<td>656</td>
<td>1</td>
</tr>
<tr>
<td>Pajek_2.net</td>
<td>pajek_447_extract</td>
<td>6238</td>
<td>11721</td>
<td>447</td>
<td>389</td>
<td>835</td>
<td>1</td>
</tr>
</tbody>
</table>

Future work:
1. Using Hash Table to search a node within a supernode.
2. Decompressing the graph

Conclusion:
9.428% node reduction and 12.16 % edge reduction in 4 node compression