Triad Enumeration at Trillion-Scale using a Single Commodity Machine

Yudi Santoso, Venkatesh Srinivasan, Alex Thomo
University of Victoria
Canada
{santoso,srinivas,thomo}@uvic.ca

Sean Chester
Norwegian University of Science and Technology
Norway
sean.chester@ntnu.no

ABSTRACT

Triad enumeration yields more detailed information than triangle enumeration. However, triad enumeration is more complex as it has to list the edges as well as the nodes of the triads. Furthermore, it is challenging to do on large graphs because of two reasons: how to deal with large amounts of data using limited memory, and how to do the computation in a reasonable amount of time. While distributed computing can take care of both problems, it requires large investment and high operating cost, as well as a distributed algorithm design which is not always possible. In this paper we show that triad enumeration of very large graphs at the web-scale can actually be done on a single commodity machine. Memory space limitation can be overcome by using data compression and partial loading. Performance can be greatly improved through optimized preprocessing and parallelization.

1 INTRODUCTION

Triangles play an important role in network analysis. For example, the presence of triangles is an indicator of communities in the network [13]. Triangles are also central to computing the connectivity of a graph [2], the clustering coefficient [18], and the transitivity [11]. There are many practical applications of these, for example, detecting fake users in social networks [19] and uncovering hidden thematic layers in the Web [9].

Most real world networks have directed relationships, and therefore we should consider directed graphs for better representations of those networks. A triad is a subgraph of three nodes in a directed graph [1, 8]. When each pair of the nodes is connected we have a closely connected triad. Since we are only interested in closely connected triads we will simply call them triads in this paper. There are seven types of triads, called by some authors as seven types of triangles [14, 16]. They are shown in Fig. 1. Note, however, that we define our own numbering here.

Enumerating triads means listing the edges as well as the nodes inside every triad. Triad enumeration would reveal a more detailed picture of the network, and hence opening up more possible applications. For example, transitivity can be more accurately analyzed by using triads [2], and directed clustering coefficient can be used as a measure of systemic risk in complex banking networks [15]. Also, triad enumeration is an important element in social network analysis [17].

Nonetheless, triad enumeration is more complex than triangle enumeration. The general belief is that it is considerably more difficult [14] and that it would take much longer running time. We found that this is not necessarily the case. Although there are some challenges, it is possible to devise an efficient algorithm

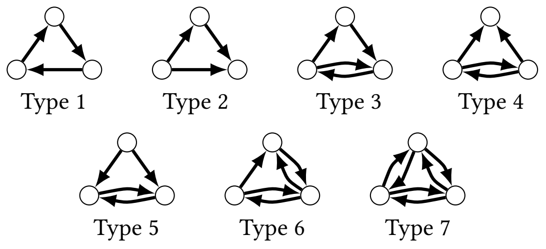

Figure 1: Seven types of triads.

which, combined with a compression framework such as WebGraph, is able to enumerate triads on a graph with a billion nodes and billions of edges using a single commodity machine.

1.1 Related Work

Pajek is a well known graph analysis software for triad enumeration. The triad census algorithm by Batagelj and Mrvar [1], which is employed in Pajek, has been known as the standard algorithm to enumerate triads for several years. However, this program is not suitable for very large graphs with millions of nodes and edges.

Chin et al. [7] proposed a compact data structure where both outgoing and incoming edges are listed in the same adjacency list, and the edge direction is encoded using the 2 lowest bits out of the 32 bits (i.e., using `int`) in each entry. This data structure is suitable for parallel computation using shared memory architectures.

This idea was further refined by Parimalarangan et al. [12], who proposed two types of algorithms, intersection based (AI) and marking based (AM), as the most efficient algorithms to enumerate triads on shared memory platforms.

While those ideas significantly improve the running time, there is a cost on the scalability. With two bits used for edge direction, the order of the graphs that can be processed is reduced. This becomes a problem when we want to analyse a graph of a billion nodes. Theoretically, we can switch to 64-bit integers and the problem should be moot. Practically, however, this would at least double the memory requirement. With limited memory budget, space is already a problem for analysing very large graphs.

1.2 Contributions

1. We implemented Parimalarangan et. al. AI algorithm in Java. We chose AI over AM because it uses less memory. AM is unable to work in a consumer-grade machine of 32GB memory even for graphs of moderate size. Our code is designed for execution on a single machine with parallel threads. Together with optimized preprocessing on the input graphs, we were able to run triad enumeration faster than the ones reported in Parimalarangan's paper, hence raising the known record.

2. For enormous graphs, such as ClueWeb12, AI is not able to work in a consumer-grade machine of 32GB memory. In order to address the case of such graphs we propose another algorithm, which has better scalability. It uses both the graph and transpose graph as input, and computes the node connections on the fly.
3. We are able to employ WebGraph compression which archives a more than 7-fold compression ratio and thus allows loading big graphs (or significant parts of them) in main memory. This enables our new algorithm to complete triad enumeration on ClueWeb12 in the aforementioned machine.

2 TRIAD ENUMERATION

The Batagelj and Mrvar triad census algorithm [1] assigns a code to each pair of nodes to represent the directed edges between them. For each triple of nodes it then uses a table to find the triad types based on the combined codes. Although this algorithm can do triad enumeration in subquadratic time, it is not fast enough for very large graphs with millions of nodes and edges.

Chin et al. [7] developed a compact data structure which makes it easier to parallelize the computation. They combined the adjacency list to contain both outgoing and incoming edges. The edge information or the link is coded using 2-bits: 01 (forward), 10 (backward), and 11 (both), embedded in the neighbour node labels inside the list. Suppose the nodes were labeled by using 32-bit integers. The bits are shifted to the left by two, and the two lowest bits are then used for the edge direction. Thus, only 30 bits can actually be used to label the nodes.

Parimalarangan et al. [12] took on this idea and combined it with the most efficient algorithms known for triangle enumeration on single machines [10]. They came up with two algorithms, the AI algorithm which is intersection based, and the AM algorithm which is marking based. Although AM can in some cases be faster it requires more memory than AI. Therefore, we chose AI and implemented it in Java for our base comparison. We use parallel streams introduced in Java 8 to make use of the multiple threads in our machine.

In our experiments, we use directed networks and their transpose in compressed WebGraph format. For our implementation of AI, we first build the compact data structure (of edges and their direction 2-bit-encoding) from these datasets and save it in plain text format, which is then used as input to the AI program.

The WebGraph framework [5, 6] provides compression schemes suitable for graph adjacency lists. The compression factor can be more than seven-fold, significant in saving disk space. The framework also provides some tools to work with files in WebGraph format. One of them is the `loadMapped`, which allows partial loading of the dataset using memory-mapped files. As we will see later, this is one of the key tools that enable us to process very large graph such as ClueWeb12. Note, however, that using WebGraph comes with the cost of decompressing the dataset.

To improve the performance, we did some preprocessing on the graphs (before building the compact data structure). First, note that each triad should be iterated only once. To avoid multiple counting, we assert condition $u < v < w$ for a triad (u, v, w). Consequently, we only need to consider bigger neighbours. Also, permutation on the labels should not change the number of triads in a graph. Thus, we first sort the nodes according to degrees from lowest to highest, relabel them, and then cut out smaller neighbours from the adjacency list. That is, suppose N_u is the set of neighbours of u after relabelling, then after the cut the adjacency list only contains $N_u \setminus \{v | v < u\}$. This is done simultaneously for the graph and its transpose as the relabelling must be the same for both. After this preprocessing, a node which originally has the highest degree would have zero/no neighbours. The degree distribution in the new adjacency list would have a hill shape with highest (effective) degrees gathered in the middle. When we parallelize on the first node iteration, this distribution can lead to imbalanced workload among the threads. To alleviate this, we do a further preprocessing which redistributes the nodes in steps of 10,000. That is, we pick the nodes and rearrange them in order (0, 10000, 20000, …, 1, 10001, 20001, …), and then relabel them as (0, 1, 2, …).

Algorithm 1 Four Pointers Triad Enumeration

```
Input: A directed graph $G = (V, E)$ and its transpose $G^T$
Output: The number of each type of triads in $G$, $\Delta_i$.
1: $\Delta_1 \leftarrow 0, \ldots, \Delta_7 \leftarrow 0$
2: for all $u \in V$ do                                  ▷ Parallelize
3:     while there is next do
4:         Find next neighbour in $N^+(u)$ and/or $N^-(u)$: $v$.
5:         Code the link $uv$ as $e1$: either 01, 10 or 11
6:         while there is next do
7:             Find next common neighbour of $u$ and $v$: $w$, in $(N^+(u), N^-(u))$ and $(N^+(v), N^-(v))$.
8:             Code the links $vw$ as $e2$, and $wu$ as $e3$.
9:             Look up triad type $i$ using $e1, e2, e3$.
10:            enum(u,v,w,e1,e2,e3)
11:            $\Delta_i \leftarrow \Delta_i + 1$
12: return $\Delta_1, \ldots, \Delta_7$
```

The drawback of the compact data structure solution is that it leads to a reduced scalability. In Java, the 32-bit integer data type can be used to label up to 2^{31} nodes (because we can have only signed integers), but with 2 bits used for edge information, it can label only up to 2^{29} (or about 1/2 billion) nodes. This becomes problematic when we want to analyze a graph such as ClueWeb12 which has almost a billion nodes and about forty two billion edges. Theoretically, we can switch to 64-bit long data type and be able to do ClueWeb12. However, we still need to overcome the memory limitation problem. With ClueWeb12, forty two billion edges translates to more than 300GB RAM if we use 8 bytes for each, which is way beyond the typical amount of RAM in current commodity machines.

In order to process ClueWeb12, we tried several modifications of the AI algorithm, separating the edge direction information from the node labels in the adjacency list. For instance, we put it in separate list of bytes, where each byte encoding a link. We also tried to use BitSet to more compactly encode the edge and save the space, and WebGraph for the combined adjacency list. However, none of our attempts succeeded in running on ClueWeb12 using 32GB RAM.

To this end, we develop a new algorithm which computes the type of connections between each pair of nodes on the fly, using both the graph and its transpose as input. Using this algorithm, and partial loading method of WebGraph , we were able to process ClueWeb12 on our machine with a budget of 32GB RAM. The algorithm, in a simplified version, is shown in Algorithm 1. We call this Four Pointers Triad Enumeration algorithm due to the fact that we use four pointers: one on each of $N^+(u)$, $N^-(u)$, $N^+(v)$, and $N^-(v)$. Here, $N^+(u)$ is the set of out-neighbours of u

in G, while $N^{-}(u)$ is the set of out-neighbours of u in G^T. This algorithm is an expansion of the 2-pointer algorithm commonly used for set intersections.

Algorithm 1 iterates over the first node u. This iteration can easily be parallelized. For each, it checks both $N^{+}(u)$ and $N^{-}(u)$ to find the neighbours of u and their respective links. For each neighbour of u, v, it finds their common neighbours using four pointers. For each, w, it looks up the triad type based on the links among the three nodes (u, v, w). In line 10, `enum()` is a space holder for an enumeration or listing function.

3 EXPERIMENTS AND DISCUSSIONS

3.1 The Setup

We ran our experiments on a machine with dual Intel Xeon E5620 CPUs and 64GB RAM. Its price is less than $3K, qualifying it as a commodity machine. However, to make a better comparison with other papers, we allowed only 32GB of RAM to be used by the Java virtual machine. The Xeon CPU has a clock speed of 2.40 GHz and 8 threads (16 threads total for the dual). The OS is Linux Ubuntu 14.04.5. We used Java 8 and the WebGraph 3.6.1.

We have five programs to run, listed in Table 1. Both AI and 4P are parallelized on the first node iteration.

Name	Description
SC	Sort and cut preprocessing
RD	Node redistribution
CDt	Build compact dataset
AI	Our AI implementation
4P	Four pointers enumeration

Table 1: The programs used in this experiment.

3.2 The Datasets

We applied our algorithms on five networks in compressed WebGraph format. The datasets were downloaded from the WebGraph website [4] (http://law.di.unimi.it/datasets.php). For each, we downloaded both the graph and the transpose graph.

Here are our selection of networks:

1. **cnr-2000**: a small crawl from the Italian CNR (Consiglio Nazionale delle Ricerche).
2. **ljournal-2008** (abbreviated as **ljournal**): a snapshot from LiveJournal (https://www.livejournal.com/) in 2008.
3. **arabic-2005** (abbreviated as **arabic**): a of websites that contain arabic, performed by UbiCrawler [3] in 2005.
4. **uk-2005**: a shallow crawl of .uk domain, performed by UbiCrawler [3] in 2005.
5. **clueweb12** (abbreviated as **clueweb**): a crawl of English webpages, created by the Lemur Project (http://www.lemurproject.org/clueweb12/index.php), with outlink nodes removed.

The statistics of these datasets are listed in Table 2. Notice that G and G^T can have different sizes because the results of the compression can be different. The smallest dataset, `cnr-2000`, has 3.2M edges, while the largest dataset, `clueweb`, has more than 42B edges. The degree statistics are listed in Table 3. The statistics of the resulting graphs after preprocessing are listed in Table 4. Note that redistribution will not change these statistics.

			Size of	
Name	$\|V\|$	$\|E\|$	G	G^T
cnr-2000	325,557	3,216,152	1.2M	920K
ljournal	5,363,260	79,023,142	105M	105M
arabic	22,744,080	639,999,458	141M	96M
uk-2005	39,459,925	936,364,282	201M	140M
clueweb	978,408,098	42,574,107,469	12G	7.0G

Table 2: Dataset statistics of the directed graphs. The sizes are of the compressed WebGraph files, in bytes.

Name	$d^{+}_{\max}$	$d^{-}_{\max}$	d_{avg}	$d^{-}_{\max}/d^{+}_{\max}$
cnr-2000	2,716	18,235	9.9	6.7
ljournal	2,469	19,409	14.7	7.9
arabic	9,905	575,618	28.1	58.1
uk-2005	5,213	1,776,852	23.7	340.9
clueweb	7,447	75,611,690	43.5	10,153.3

Table 3: Degrees statistics in the original datasets.

Name	$\|E^{\text{eff}}\|$	$\|E^{T,\text{eff}}\|$	$d^{+,\text{eff}}_{\max}$	$d^{-,\text{eff}}_{\max}$
cnr-2000	2,580,192	548,518	1,336	81
ljournal	42,947,594	35,043,920	1,257	397
arabic	534,631,498	96,522,171	6,646	3,126
uk-2005	759,564,189	161,780,889	5,213	584
clueweb	36,605,200,001	5,968,907,468	5,873	4,242

Table 4: Dataset statistics of the effective directed graphs.

3.3 Results

We ran the AI, and 4P triad enumeration programs on the chosen datasets. We verified that we got the same numbers from both programs, when run with or without pre-processing. The numbers of triads for each types are listed in Table 5.

In Table 6 we list the running times. The graphs had been preprocessed by the SC program (sort and cut) before being used as input, but without node redistribution. The running times on graphs without preprocessing are not shown here. However, note that this preprocessing is important in keeping the running time low. The CDt program takes the graph and its transpose in the compressed WebGraph format and produces a compact dataset written into a plain text file which is then used as input by the AI program. The 4P program takes the WebGraph files directly as input. Therefore, we compare 4P with AI+CDt. Except for `cnr-2000`, AI+CDt is faster than 4P. This is due to repeat decompression cost in 4P. However, AI+CDt is unable to process `clueweb` for the reason that is explained in the previous section (inability to represent 1 billion nodes using 29 bits for each), while 4P can. Notice that our AI implementation can process `arabic` in a shorter time than the one reported in Parimalarangan's paper [12] (In that paper they did not report the time to build the compact data set, CDt, and excluded the loading time. Our numbers here for AI include the loading time.).

Next, we checked whether redistribution can improve the performance. In Table 7 we list the running times on the graphs that had been preprocessed and which nodes had been redistributed. The cost of redistributing the nodes is listed as RD. We see that this RD cost can be quite significant. Taking this cost into consideration, it is not always worth it to do redistribution. For example, on `ljournal`, the 4P time without RD is 81 seconds, while RD +

Name	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7
cnr-2000	10,342	9,899,367	85,969	2,433,041	6,736,504	419,472	1,392,934
ljournal	530,051	86,777,707	10,421,919	69,748,792	44,608,271	80,177,727	118,890,977
arabic	2,668,704	6,906,765,421	30,427,662	1,571,745,235	11,765,868,185	384,594,679	16,233,290,956
uk-2005	5,335,890	5,198,533,331	48,779,535	1,773,901,843	9,499,139,863	411,396,906	4,842,278,688
clueweb	281,444,867	517,684,665,693	2,261,300,705	153,674,084,413	790,291,640,762	28,556,769,295	502,545,385,030

Table 5: The counts of triads of each types on the selected networks.

Name	SC	AI	CDt	AI+CDt	4P
cnr-2000	2.75	1.25	2.06	3.31	3.0
ljournal	74	27	28	55	81
arabic	200	429	206	635	2961
uk2005	311	354	319	673	796
clueweb	12,870	-	-	-	115,960

Table 6: The running time (in seconds) of triad enumeration using Four Pointer algorithm (4P), and AI algorithm (AI). Also listed are the preprocessing time (SC), and the time to build the compact dataset (CDt). Since, 4P does not need compact dataset, we compare 4P with AI+CDt.

$4P^{RD}$ is 86 + 30.5 = 116.5 seconds. For `arabic`, though, the RD is helpful in bringing the overall cost down.

Name	RD	AI^{RD}	CDt^{RD}	$AI+CDt^{RD}$	$4P^{RD}$
cnr-2000	3.7	0.98	1.49	2.5	3.2
ljournal	86	18.4	22.8	41.2	30.5
arabic	453	215	154	369	483
uk2005	632	312	248	560	366
clueweb	30,413	-	-	-	-*

Table 7: The running time (in seconds) of triad enumeration using Four Pointer algorithm (4P), and AI algorithm (AI) on redistributed graphs. *The run on `clueweb` cannot be done in a reasonable amount of time.

Notice that with RD our 4P becomes competitive to AI+CDt. This can be understood from the fact that workload imbalanced affects 4P more than AI since 4P has to do decompression on the adjacency list, and hence RD benefits 4P more as it reduces the imbalance. It is faster on `ljournal` and `uk2005`, and just a bit slower on `cnr-2000` and `arabic`.

The redistribution, however, has an unwanted effect on WebGraph compression. The compression works best if the distances among the neighbour nodes are not large. With redistribution, there are large distances within a neighbour set, which in turn lower the overall compression ratio. As such, we do not advise performing redistribution on very large graphs such as `clueweb`. For such graphs sort-and-cut preprocessing is all what is needed for our algorithm 4P to complete in reasonable time.

4 CONCLUSIONS

We have shown that through optimized preprocessing and parallelization we are able to run triad enumeration on very large graphs using a single commodity machine in reasonable time. We also designed an algorithm, 4P, with better scalability than the state-of-the-art, which, with some trade-off on the performance, can run on graphs of a billion nodes and billions of edges, counting trillions of triads. In our solution, the WebGraph framework plays an important role in alleviating the memory problem. In conclusion, our results show that triad enumeration can be done on a commodity machine even for very large graphs such as ClueWeb12.

REFERENCES

[1] Vladimir Batagelj and Andrej Mrvar. 2001. A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social networks* 23, 3 (2001), 237–243.

[2] Vladimir Batagelj and Matjaž Zaveršnik. 2007. Short cycle connectivity. *Discrete Mathematics* 307, 3-5 (2007), 310–318.

[3] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. UbiCrawler: A scalable fully distributed web crawler. *Software: Practice & Experience* 34, 8 (2004), 711–726.

[4] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiResolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web*, Srinivasan Sadagopan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM Press, 587–596.

[5] Paolo Boldi and Sebastiano Vigna. 2004. The Webgraph Framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*. ACM, New York, NY, USA, 595–602. https://doi.org/10.1145/988672.988752

[6] Paolo Boldi and Sebastiano Vigna. 2004. The Webgraph Framework II: Codes for the world-wide web. In *Data Compression Conference, 2004. Proceedings. DCC 2004*. IEEE, 528.

[7] George Chin Jr, Andres Marquez, Sutanay Choudhury, and John Feo. 2012. Scalable triadic analysis of large-scale graphs: Multi-core vs. multi-processor vs. multi-threaded shared memory architectures. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*. IEEE, 163–170.

[8] James A Davis and Samuel Leinhardt. 1972. The structure of positive interpersonal relations in small groups. *Sociological Theories in Progress* 2 (1972), 218–251.

[9] Jean-Pierre Eckmann and Elisha Moses. 2002. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the National Academy of Sciences* 99, 9 (2002), 5825–5829.

[10] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science* 407, 1-3 (2008), 458–473.

[11] Mark EJ Newman, Duncan J Watts, and Steven H Strogatz. 2002. Random graph models of social networks. *Proceedings of the National Academy of Sciences* 99, suppl 1 (2002), 2566–2572.

[12] Sindhuja Parimalarangan, George M Slota, and Kamesh Madduri. 2017. Fast parallel graph triad census and triangle counting on shared-memory platforms. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 1500–1509.

[13] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. 2004. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences* 101, 9 (2004), 2658–2663.

[14] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. 2013. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, Vol. 4. 5.

[15] Benjamin M Tabak, Marcelo Takami, Jadson MC Rocha, Daniel O Cajueiro, and Sergio RS Souza. 2014. Directed clustering coefficient as a measure of systemic risk in complex banking networks. *Physica A: Statistical Mechanics and its Applications* 394 (2014), 211–216.

[16] Pinghui Wang, Yiyan Qi, Yu Sun, Xiangliang Zhang, Jing Tao, and Xiaohong Guan. 2017. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. *Proceedings of the VLDB Endowment* 11, 2 (2017), 162–175.

[17] Stanley Wasserman and Katherine Faust. 1994. *Social network analysis: Methods and applications*. Vol. 8. Cambridge University Press.

[18] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (1998), 440.

[19] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. 2014. Uncovering social network sybils in the wild. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 1 (2014), 2.