Erschienen in: 2014 Proceedings of the sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX) : January 5, 2014, Portland, Oregon, USA / Catherine C. McGeoch ... (eds.). - Philadelphia : Society for Industrial and Applied Mathematics, 2014. - S. 1-8. - ISBN 978-1-61197-319-8

Triangle Listing Algorithms: Back from the Diversion*

Mark Ortmann† Ulrik Brandes†

Abstract

We show that most algorithms from the literature on listing the triangles of a graph have a common abstraction. Our unifying framework highlights that these seemingly different algorithms are in fact instantiations of a single generic procedure, and even suggests some additional variants. More importantly, it yields parsimonious implementations that are in general more efficient than those described in the original works. In addition, we show that the running time of nearly every triangle listing variant is in $\mathcal{O}(a(G)m)$, where $a(G)$ is the arboricity of the graph and m the number of edges. So far this bound has been proven only for Chiba and Nishizeki's (*SIAM J. Computing*, 1985) triangle listing algorithm. Finally, algorithmic experimentation reveals that an improved implementation of this algorithm outperforms all subsequently proposed algorithms.

1 Introduction

Interest in analyzing the triangles of a graph has increased considerably because of several important concepts in network science, the most prominent examples being the *clustering coefficient* [24] and the *triad census* [11]. The latter is an essential ingredient in statistical network modeling [19, 23], where it needs to be determined repeatedly. We focus on algorithms for listing all triangles; although there are fast matrix methods for counting triangles [1], using such methods for the triad census does not appear to be beneficial [17].

One of the first efficient algorithms for listing all triangles of a graph G was proposed by Chiba and Nishizeki [6] and runs in $\mathcal{O}(a(G)m)$ time, where m is the number of edges and $a(G)$ the arboricity of the graph. Several other algorithms have been proposed (e.g., [13, 14, 20, 21]) and proclaimed to be more efficient. However, these claims have never been substantiated convincingly.

The theoretical contribution of this paper is the description of a unifying framework for triangle listing algorithms. This makes it easy to spot the differences between various instances, despite their largely differing original presentation. As a byproduct, the framework yields even more variant algorithms, and provides simple proofs that almost all of the known algorithms actually have a worst-case running time bound of $\mathcal{O}(a(G)m)$. The practical contribution is an experimental analysis showing that our variant implementation of Chiba and Nishizeki's algorithm is by far the fastest in-memory algorithm to list all triangles of a graph.

2 Triangle Listing Framework

We consider finite simple undirected graphs $G = (V, E)$ and denote the number of vertices by $n = n(G) = |V|$ and the number of edges by $m = m(G) = |E|$. The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{w : \{v, w\} \in E\}$ of all adjacent vertices, its cardinality $\deg(v) = |N(v)|$ is called the *degree* of v, and $\Delta(G) = \max_{v \in V}\{\deg(v)\}$ denotes the maximum degree of G.

For finite simple directed graphs $G = (V, E)$ we denote the *outgoing neighborhood* of a vertex $v \in V$ by $N^+(v) = \{w : (v, w) \in E\}$, the *outdegree* of vertex v by $\deg^+(v) = |N^+(v)|$ and the *maximum outdegree* by $\Delta^+(G) = \max_{v \in V}\{\deg^+(v)\}$. The *incoming neighborhood* $N^-(v)$, *indegree* $\deg^-(v)$ and *maximum indegree* $\Delta^-(G)$ are defined analogously.

A *triad* is an induced subgraph on three vertices, and a *triangle* is a triad, where each pair of vertices is connected.

2.1 Algorithm of Chiba and Nishizeki In 1985, Chiba and Nishizeki proposed an algorithm to list all triangles of a graph by intersecting the neighborhoods of adjacent vertices. We will refer to this algorithm as K3. For efficiency, the intersections are performed in a certain order which ensures that for each intersection only the neighborhood of the vertex with smaller degree needs to be scanned. This is made precise in Alg. 2.1 and is motivated by the following theorem.

*We gratefully acknowledge finacial support from Deutsche Forschungsgemeinschaft under grant Br 2158/6-1

†Department of Computer & Information Science, University of Konstanz

ALGORITHM 2.1. **K3** (Chiba and Nishizeki [6])

```
1.  sort vertices such that deg(v_1) ≥ ... ≥ deg(v_n);
2.  for u = v_1, ..., v_{n-2} do {
3.      for each v ∈ N(u) do mark v;
4.      for each v ∈ N(u) do {
5.          for each w ∈ N(v) do {
6.              if w is marked then {
7.                  output triangle {u, v, w};
8.              }
9.          }
10.         unmark v;
11.     }
12.     G ← G − u;
13. }
```

THEOREM 2.1. ([6]) *For a graph $G = (V, E)$,*

$$\sum_{\{u,v\}\in E} \min\{\deg(u), \deg(v)\} \;\leq\; 2a(G)m \;\in\; \mathcal{O}(m^{3/2}) \;,$$

where $a(G)$ is the arboricity of G, i.e., the minimum number of forests needed to cover E.

Since arboricity is related to edge density via $a(G) = \max_{H\subseteq G}\left\{\frac{m(H)}{n(H)-1}\right\}$ [18], it is rather small in sparse graphs which in turn are typical for empirical network analysis [9].

After intersecting their neighborhood with those of their neighbors, vertices are deleted from the graph in the last line of the algorithm to avoid intersecting the same pairs of neighborhoods again by scanning the larger neighborhood. Chiba and Nishizeki propose to represent the graph with doubly-linked adjacency lists and mutual references between the two stubs of an edge to ensure constant time deletion of edges. Since the number of triangles in a graph is bounded by $a(G)m$, K3 is worst-case optimal.

While Chiba and Nishzeki were most likely interested only in proving that the asymptotic running time of Alg. 2.1 is in $\mathcal{O}(a(G)m)$, it seems that the rather involved data structures have hampered adoption of the algorithm [20]. We show below that substantial improvements are possible and that double linkage and vertex deletion can be avoided altogether.

2.2 Other Algorithms Subsequently proposed algorithms for triangle listing can be assigned to one of two categories. The first category consists of those that intersect, like K3, the neighborhoods of adjacent vertices. Algorithms in the other category turn this view around: for each pair of incident edges, the other two vertices are tested for adjacency.

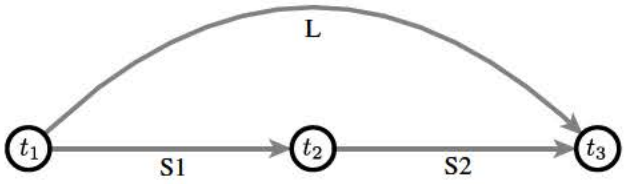


Figure 1: In a transitive triad, each vertex and edge has a unambiguous role. Vertices come first, second, or third, and edges serve either as the long edge, or as the first or second short edge

Neighborhood intersection. Algorithms in this category include edge-iterator [20], forward [21], and compact-forward [14]. They iterate over all edges and intersect the neighborhoods of the two adjacent vertices using a procedure called *sorted-merge-join* [5]. While this requires $\mathcal{O}(\deg(u) + \deg(v))$ per edge $\{u, v\}$, and thus more time than K3, it does not require additional space for vertex marks. The two variants edge-iterator-hashed and forward-hashed [20] utilize $\mathcal{O}(m)$ extra space to represent neighborhoods in hash sets and thus carry out the intersection in $\mathcal{O}(\min\{\deg(u), \deg(v)\})$ time. A combination of K3 and edge-iterator has been termed new-listing [14].

Adjacency testing. The complementary approach is to iterate over all vertices and, for each pair of incident edges, test whether the two neighbors are also adjacent to each other. Algorithms of this kind have been termed node-iterator and node-iterator-core in [20]. Both use hash sets to test pairs of vertices for adjacency in constant time, and thus also require $\mathcal{O}(m)$ extra space. Since adjacency lists are ordered in these algorithms, binary search can be used to trade space for running time. A related technique called tree-lister [13] determines a spanning forest, tests whether non-tree edges complete a triangle with two incident tree edges, removes the forest, and iterates.

Asymptotic running times of these algorithms have not been analyzed in detail. We give straightforward bounds derived from their description in Tab. 1. These bounds depend on the enumeration order of vertices and edges. By choosing a suitable such order we can make them match the bound of K3; see Tab. 2 below.

2.3 Unifying Framework It turns out that many triangle listing algorithms are best described by aligning the execution with an acyclic orientation of the input graph. Let us fix a vertex ordering $\pi : V \to \{1, \ldots, n\}$ and orient the edges from the lower-numbered vertex to the higher-numbered vertex. Let $G[\pi] = (V, E[\pi])$

strategy	extra space	variant	running time	related algorithm
intersection	$\mathcal{O}(1)$	S1+1	$\mathcal{O}\left(\sum_{v\in V}\binom{\deg^+(v)}{2} + \deg^-(v)\deg^+(v)\right)$	
		S2+1	$\mathcal{O}\left(\sum_{v\in V}\binom{\deg^-(v)}{2} + \deg^-(v)\deg^+(v)\right)$	(compact-)forward
		L+1 L'+1	$\mathcal{O}\left(\sum_{v\in V}\binom{\deg^+(v)}{2} + \binom{\deg^-(v)}{2}\right)$	edge-iterator
	$\mathcal{O}(n)$	S1+n	$\mathcal{O}\left(m + \sum_{v\in V}\deg^+(v)\deg^-(v)\right)$	K3
		S2+n		
		L+n	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^+(v)}{2}\right)$	
		L'+n	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^-(v)}{2}\right)$	K3
	$\mathcal{O}(m)$	S1+m	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^+(v)}{2}\right)$	
		S2+m	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^-(v)}{2}\right)$	forward-hashed
		L+m L'+m	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^{+/-}(v)}{2}\right)$	edge-iterator-hashed
testing	$\mathcal{O}(1)$	T1+1	$\mathcal{O}\left(\log\Delta^{+/-}(G)\sum_{v\in V}\binom{\deg^+(v)}{2}\right)$	
		T2+1	$\mathcal{O}\left(\log\Delta^{+/-}(G)\sum_{v\in V}\deg^-(v)\deg^+(v)\right)$	
		T3+1	$\mathcal{O}\left(\log\Delta^{+/-}(G)\sum_{v\in V}\binom{\deg^-(v)}{2}\right)$	
	$\mathcal{O}(m)$	T1+m	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^+(v)}{2}\right)$	node-iterator-core
		T2+m	$\mathcal{O}\left(m + \sum_{v\in V}\deg^-(v)\deg^+(v)\right)$	node-iterator, tree-lister
		T3+m	$\mathcal{O}\left(m + \sum_{v\in V}\binom{\deg^-(v)}{2}\right)$	tree-lister

Table 1: Algorithms and running times by operation count

denote the resulting DAG.

Then, each triangle of G yields a *transitive triad* in $G[\pi]$ relative to which vertices and edges assume unique roles. Shorthand names for these roles are assigned as shown in Fig. 1. We refer to an algorithm by the basic element from which triangles are supposed to be constructed (edge or vertex with a given role) with the amount of extra space used by the algorithm (constant or linear in n or m).

To list each triangle exactly once, intersection-based algorithms may iterate over all edges and intersect incoming or outgoing neighborhoods of its vertices based on the role (L, $S1$, $S2$) that the edge is assumed to play in the triad. The resulting algorithms are listed and put in relation to previous algorithms in Tab. 1. In variant L'+n, for example, an edge (u, v) is considered to be the transitive (long) edge, i.e. (t_1, t_3), and utilized to identify triads $\{t_1, t_2, t_3\}$ by fixing $N^+(u)$ and scanning $N^-(v)$ for vertices $t_2 \in N^+(u) \cap N^-(v)$. Indeed, variant S1+n also fixes $N^+(u)$, but (u, v) is considered to be edge $S1$ so that $N^+(v)$ is processed instead. With an appropriate ordering, the combination of these two variants corresponds to K3, although each variant is sufficient by itself to list all triangles.

Adjacency-testing algorithms, on the other hand, iterate over all vertices and examine incident pairs of edges based on the role (t_1, t_2, t_3) that the vertex is assumed to play in the triad.

2.4 Orderings As summarized in Tab. 1, the running time of each variant algorithm hinges on the vertex ordering π. The ordering employed in K3 is determined by non-increasing vertex degrees in the input graph. For ease of exposition, we consider the reverse of this order and refer to it as degree ordering.

The rationale of this ordering was to reduce the number of neighbors tested for membership in the intersection. Since the degree in the input graph is only an upper bound on the remaining degree after several vertices have been processed and deleted, a potential improvement is to dynamically select the next vertex based on the remaining degree.

Orderings determined by iteratively removing vertices of minimum induced degree in the remaining subgraph are called smallest-first ordering [4, 16]. The maximum value encountered when eliminating all vertices is known as the *degeneracy* [15], *width* [10], or *core number* [22], *core*(G), of a graph. Since this number equals arboricity up to a constant factor [25], we obtain easy bounds on the running time of all instantiations of the above framework (see Tab. 2). Note that there exists no vertex ordering π with $\Delta^+(G[\pi])$ strictly less than *core*(G) [8].

2.5 Running Times From the explanations given in Sect. 2.3, the running times presented in Tab. 1 can be derived as follows.

variant	suffix	ordering	running time	previous results
S1, S2	+1	smallest-first	$\mathcal{O}(a(G)\,m)$	S2+1 ≈ (compact)-forward: $\mathcal{O}(m^{3/2})$
	+n	degree	$\mathcal{O}(a(G)m)$	S1+n ≈ K3: $\mathcal{O}(a(G)\,m)$
	+m	any	$\mathcal{O}(a(G)\,m)$	S2+m ≈ forward-hashed: $\mathcal{O}(\sum_{(u,v)\in E}\min\{\deg^-(u),\deg^-(v)\})$
L, L'	+1	smallest-first	$\mathcal{O}(m\,(a(G)+\Delta^-(G)))$	edge-iterator: $\mathcal{O}(\Delta(G)m)$
	+n	degree	$\mathcal{O}(a(G)\,m)$	L'+n ≈ K3: $\mathcal{O}(a(G)m)$
	+m	any	$\mathcal{O}(a(G)\,m)$	edge-iterator-hashed: $\mathcal{O}\left(\sum_{(u,v)\in E}\min\{\deg(u),\deg(v)\}\right)$
T1, T2, T3	+1	smallest-first	$\mathcal{O}(\log(a(G))\,a(G)\,m)$	
	+m	smallest-first	$\mathcal{O}(a(G)\,m)$	T1+m ≈ node-iterator-core: $\mathcal{O}(core(G)m)$ T2+m ≈ node-iterator: $\mathcal{O}(\Delta(G)^2 n)$

Table 2: Asymptotic running times relative to vertex ordering. These running times derive directly from Tab. 1, Thm. 2.1 and the characteristics of the smallest-first ordering. Note that smallest-first always applies and in some cases the ordering has to be reversed, e.g. T3+1 and T3+m

The algorithms based on adjacency testing strategy generate all pairs of outgoing, incoming, or mixed neighbors of a given vertex v. Consequently, the operation count of both T1 variants is in $\mathcal{O}(\sum_{v\in V}\binom{\deg^+(v)}{2})$, where each operation consists of an adjacency test. Given that adjacency testing can be done in constant time using hash sets, this is also the total running time. If the extra space is to be avoided, however, binary search can be used instead at an additional cost of $\mathcal{O}(\log\Delta^{+/-}(G))$ time per operation. Running times of the T2 and T3 variants can be derived analogously.

For the algorithms based on the intersection strategy, we only give the idea of the proof for the example of L+n. Recall that this variant marks all $w \in N^-(v)$, where v is the currently processed vertex, and computes intersections with the outgoing neighborhood of each w. From the construction of $G[\pi]$ it is known that $\{u \in N^+(w) | \pi(u) \geq \pi(v)\} \cap N^-(v) = \emptyset$. Since the adjacency lists are ordered, these entries can be omitted from $N^+(w)$. Therefore, between two consecutive intersections with $N^+(w)$, the number of relevant entries differs exactly by one, resulting in the presented running time. Keeping in mind that some entries can be omitted, the running times of the other variants are obtained.

The transition from Tab. 1 to Tab. 2 for smallest-first ordering is obtained by replacing $\deg^{+/-}(v)$ with $\Delta^{+/-}(G)$ and using the inequality $\sum_{v\in V}\binom{\deg^{+/-}(v)}{2} \leq \sum_{v\in V}\deg^{+/-}(v)^2$. Since $\Delta^{+/-}(G) \in \mathcal{O}(a(G))$ for smallest-first ordering the presented running times directly derive. For the algorithms running in $\mathcal{O}(a(G)m)$ on an ordering other than smallest-first ordering this bound is the result of an armortized running time analysis based on Thm. 2.1. Recall that the time to intersect two hash sets H_1 and H_2 is in $\mathcal{O}(\min\{|H_1|,|H_2|\})$.

3 Experimental Study

We have seen that asymptotic analysis does not discriminate between the different instantiations of our algorithmic framework. Algorithmic experimentation is thus needed to shed more light on practical and relative performance. All comparable algorithms from above were implemented and tested on both collected and generated data. Instead of reporting repetitive details, we focus on the main findings and the evidence supporting them.

3.1 Setup All framework instantiations and original versions of all previous algorithms listed in Tab. 1 except tree-lister have been implemented by the same person in C++ using the *Standard Template Library* and the g++ version 4.6.3 compiler set to the highest optimization level.

For the algorithms using $\mathcal{O}(m)$ additional space, we used the hash set implementation provided in the C++ *Technical Report 1* library extension. We implemented counting sort to sort vertices by degree in $\mathcal{O}(n)$ time, and used the `std::sort` routine to sort adjacency lists. For smallest-first ordering and related orderings, we used a slight variant of the linear-time algorithm of [4].

The code was executed on a 64-bit machine with a quad-core 3.40 GHz Intel Core i7-2600K CPU and 16 GB RAM, running Ubuntu 12.04.1 LTS, in a single thread on a single CPU. Elapsed CPU time is measured using the `gettimeofday` command with a precision of 10^{-6} seconds.

Generated Data. The experimental region is defined by two graph generators with two parameter selection schemes. The generators can be controlled for the expected number of triangles they contain and differ strongly in the degree sequences produced.

network	n	m	#triangles	time (sec)
ca-AstroPh	18,772	198,050	1,351,441	0.011
ca-CondMat	23,133	93,439	173,361	0.004
ca-GrQc	5,242	14,484	48,260	0.001
ca-HepPh	12,0008	118,489	3,358,499	0.008
ca-HepTh	9,877	25,973	28,339	0.001
cit-HepPh	34,546	420,877	1,276,868	0.025
cit-HepTh	27,770	352,285	1,478,735	0.021
cit-Patents	3,774,768	16,518,947	7,515,023	2.425
com-LiveJournal	3,997,962	34,681,189	177,820,130	5.691
com-Orkut	3,072,441	117,185,083	633,319,568	32.433
com-Youtube	1,134,890	2,987,624	3,056,386	0.285
com-DBLP	317,080	1,049,866	2,224,385	0.083
com-Amazon	334,863	925,872	667,129	0.080
email-Enron	36,692	183,832	727,044	0.001
email-EuAll	265,214	364,481	267,313	0.025
loc-Brightkite	58,228	214,079	494,728	0.011
loc-Gowalla	196,591	950,329	2,273,138	0.066
soc-Epinions1	75,879	405,740	1,624,481	0.027
soc-LiveJournal1	4,847,571	42,851,237	285,730,264	7.469
soc-Slashdot0811	77,360	469,180	551,724	0.030
soc-Slashdot0922	82,168	504,230	602,592	0.032
wiki-Talk	2,394,385	4,659,565	9,203,519	0.407
wiki-Vote	7,115	100,762	608,389	0.006

Table 3: Running times of L+n with degree ordering on data from the Stanford Large Network Dataset Collection, `http://snap.stanford.edu/data/`

Small worlds [24]: Given parameters n, $0 < r \ll n$, and $0 \leq p \leq 1$, we first create a $2r$-regular graph $(\{1,\ldots,n\}, \{\{v,w\} : |v-w| \leq r\})$ and then add random noise by flipping each dyad independently with probability p. This process yields graphs in which the expected degree and number of triangles can be controlled via r and the degrees are concentrated around $2r$.

Preferential attachment with triadic closure [12]: Given parameters n, $0 < r \ll n$, $0 \leq p \leq 1$, we create an n-vertex graph one vertex at a time. Each new vertex v is made adjacent with r existing vertices, each of which selected either preferentially according to its degree or randomly from $\bigcup_{u \in N(v)} N(u)$. The second case is applied with probability p. While the number of triangles is controlled via p, the degree sequence follows a power law.

For both classes sampling is performed using adaptations of the algorithms in [2]. After sampling, vertices and adjacency lists are permuted randomly via Fisher-Yates shuffle to prevent potential systematic biases.

We generated graphs from both models using two families of parameters that are motivated by the empirical data described below (see also Fig. 2). In the first family, the number of vertices is fixed to $n = 250,000$ and r is varied to obtain graphs with an average degree of $6, 12, \ldots, 66$. In the second family, the number of vertices and the average degree grow simultaneously from $n = 500,000$ and $\deg(G) = 6$ to $n = 5,000,000$ and $\deg(G) = 36$ in proportional increments of $900,000$ vertices and 6 degrees. Further variance is introduced

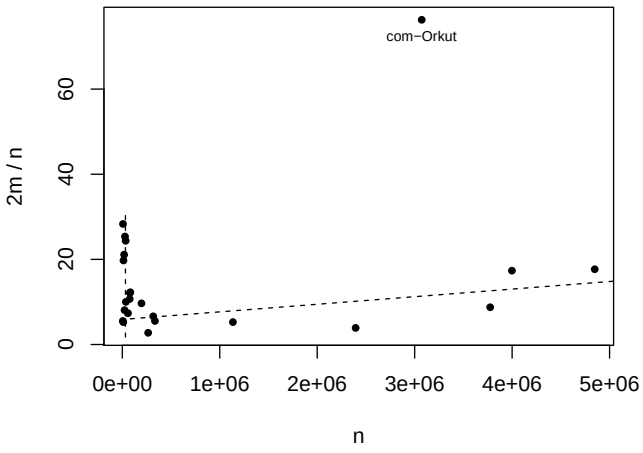
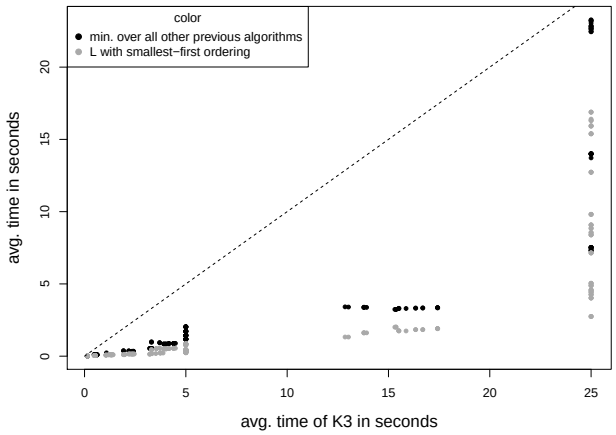
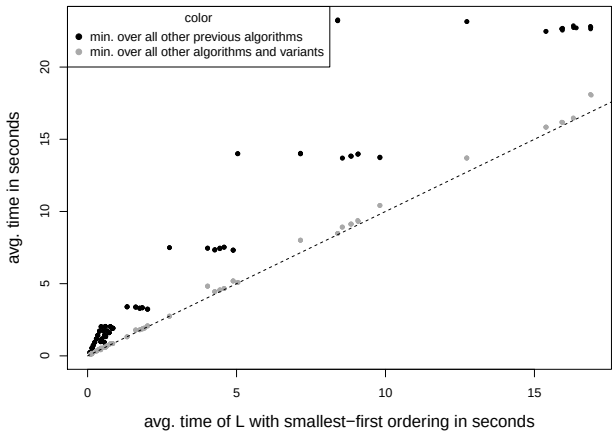


Figure 2: Average degree for graphs of Tab. 3. With the exception of instance com-Orkut, two clusters are apparent. One with fixed order and increasing edge density, the other with correlated increases. Dashed lines indicate these clusters



(a) avg. time of K3 vs. minimum avg. time of any of the other algorithms from the literature and L with smallest-first ordering



(b) avg. time of L with smallest-first ordering vs. minimum avg. time of other variants and algorithms

Figure 3: Paired comparisons of running times on generated graphs (in seconds, excl. vertex ordering). Each dot represents one of four graphs sampled for each of the 17 parameter combinations and two generators; coordinates are determined from five runs each for two algorithms on the same graph. Dots below (above) the diagonal indicate that the algorithm on the x-axis is slower (faster)

by choosing p such that the ratio of the number of edges in the regular graph and the expected number of flips is 0.5, 1, and 2, and the percentage of attachments that yield a triangle edge is 25%, 50%, and 75%. For each of the 17 parameter combinations we sampled 4 graphs; reported running times are averages over five repetitions for each generated graph.

Collected Data. We used data downloaded from the *Stanford Large Network Dataset Collection*[1] which includes mostly bibliometric, email, and online social networks. The networks described in Tab. 3 are not representative of anything, but provide realistic examples of large network for which clustering coefficients or triad census may be of interest.

3.2 Findings We present the most interesting conclusions from our experiments together with tailored summaries. Note that in the remainder L denotes L+n.

The first finding relates K3, the implementation proposed by Chiba and Nishizeki [6], to subsequently proposed algorithms.

FINDING 3.1. *K3 is outperformed by subsequently proposed algorithms.*

Figure 3(a) compares the running time (excl. time needed for the orderings) of K3 with the best running times observed for other algorithms from the literature.[2] It thus shows that an approximately 5-fold improvement has indeed been achieved. In fact, algorithms forward [21] and compact-forward [14] are fastest on every instance which is consistent with [14, 20]. We note, however, that K3 is not dominated this clearly by the other algorithms.

As it turns out, the reasons for the relative inefficiency of K3 are not in the design of the algorithm, but in the proposed implementation with doubly-linked lists and vertex and edge deletions.

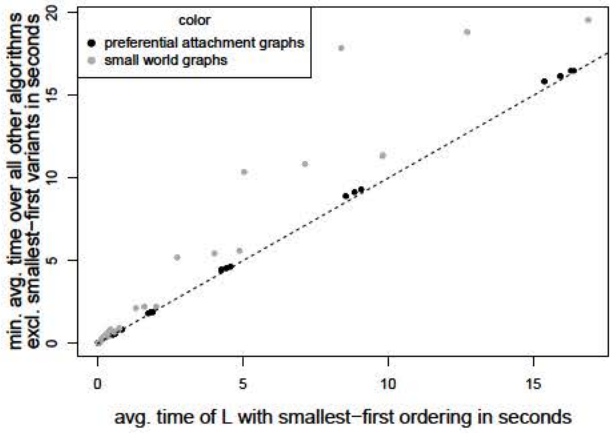
FINDING 3.2. *L with smallest-first ordering, our variant implementation of K3, outperforms all other algorithms and framework instantiations.*

Figure 3(b) shows that running times are reduced substantially by our streamlined implementation of the K3 approach. In fact, L with smallest-first ordering consistently outperforms all previous algorithms and nearly always all other instantiations of our framework, and is roughly 7× faster than the original, cf. Fig.3(a). On com-Orkut we observe a speedup of about 28×.

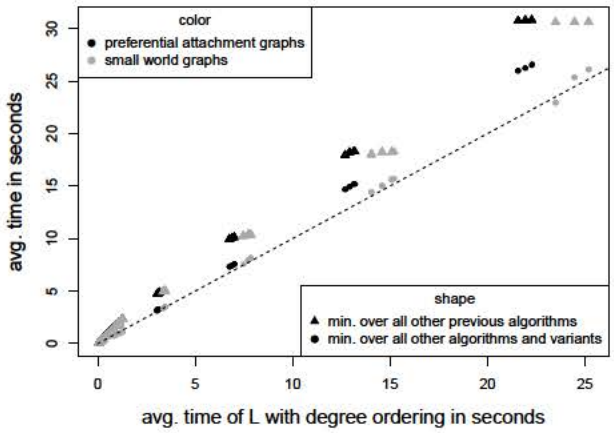
FINDING 3.3. *Running times are affected substantially by the input vertex ordering. Yet, the extra running time to determine smallest-first ordering as compared to degree ordering does not pay off. As a consequence L with degree ordering outperforms L with smallest-first ordering.*

[1] `http://snap.stanford.edu/data/`

[2] For the graph family with static (changing) n the algorithms have been stopped after 5 (25) seconds



(a) avg. time of L with smallest-first ordering vs. minimum avg. time of non smallest-first ordering variants (excl. time for orderings)



(b) avg. time of L with degree ordering vs. minimum avg. time of other variants and algorithms

Figure 4: Running times on generated graphs (in seconds)

Overall, the theoretical argument that the smallest-first ordering is the superior ordering because of its lower outcome for $\sum_{v \in V} \binom{\deg^+(v)}{2}$ and therefore for algorithm L, cf. Tab. 1, is confirmed[3]. Yet the advantage is insufficient to compensate for the additional efforts during vertex ordering. In Fig. 4(a) it can also be seen that while the smallest-first ordering is strongly beneficial for regular graphs, such as small worlds, the gain rapidly drops with increasing random noise and is rather negligible for a skewed degree distribution. As a result, L with degree ordering essentially outperforms all other algorithms and instantiations of our framework, as illustrated in Fig. 4(b).

FINDING 3.4. *The previous findings are reinforced on collected data, and the dominant combination of L with degree ordering is practical even for large graphs.*

All experiments were repeated on collected data, but did not provide any additional insight. As can be seen in Tab. 3, concrete running times are negligible except for the largest and densest networks.

4 Conclusion

We have presented a generic framework for triangle listing algorithms which makes the strategies of known algorithms comparable and introduces several other possible variants. From this framework, running time bounds for previous algorithms are straightforward to obtain, and several superfluous steps can be avoided in implementations.

The most important findings are that, given an appropriate ordering, nearly all algorithms have a worst-case running time bound of $\mathcal{O}(a(G)m)$, and that our improved implementation of one of the oldest algorithms for triangle listing, K3 [6], is actually the fastest.

In our experiments, we also observed that the number of elementary operations is not always indicative of the actual running time, mostly due to cache misses when switching between adjacency lists of different vertices. Future work on the most practical variant will have to study the consequences of these effects in more detail.

We finally note that any of these triangle listing algorithms, when combined with the *Tricode* routine in [3] and the system of linear equations from Eppstein et al. [7],[4] can be used to compute the triad census. As the running time of the resulting algorithms is dominated by the time to list all triangles, the full triad census can also be determined in $\mathcal{O}(a(G)m)$ time. This is an improvement on the $\mathcal{O}(\Delta(G)m)$ running time of what appears to be the most common approach to date [3].

References

[1] N. ALON, R. YUSTER, AND U. ZWICK, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[3] Our experiments expose that removing the vertex with the highest degree in the remaining subgraph yields always the best results for $\sum_{v \in V} \deg^+(v) \deg^-(v)$ and thus is beneficial for variants S1, S2, and T2 in many cases

[4] Note that the equations for n_6 and n_7 contain typos

[2] V. Batagelj and U. Brandes, *Efficient generation of large random networks*, Physical Review E, 71 (2005).

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

[4] V. Batagelj and M. Zaveršnik, *Fast algorithms for determining (generalized) core groups in social networks*, Advances in Data Analysis and Classification, 5 (2011), pp. 129–145.

[5] M. W. Blasgen and K. P. Eswaran, *Storage and access in relational data bases*, IBM Systems Journal, 16 (1977), pp. 362–377.

[6] N. Chiba and T. Nishizeki, *Arboricity and subgraph listing algorithms*, SIAM J. Computing, 14 (1985), pp. 210–223.

[7] D. Eppstein, M. T. Goodrich, D. Strash, and L. Trott, *Extended dynamic subgraph statistics using h-index parameterized data structures*, in Proceedings of the 4th international conference on Combinatorial optimization and applications - Volume Part I, COCOA'10, Berlin, Heidelberg, 2010, Springer-Verlag, pp. 128–141.

[8] D. Eppstein, M. Löffler, and D. Strash, *Listing all maximal cliques in sparse graphs in near-optimal time*, in ISAAC (1), 2010, pp. 403–414.

[9] D. Eppstein and E. S. Spiro, *The h-index of a graph and its application to dynamic subgraph statistics*, J. Graph Algorithms and Applications, 16 (2012), pp. 543–567.

[10] E. C. Freuder, *A sufficient condition for backtrack-bounded search*, J. ACM, 32 (1985), pp. 755–761.

[11] P. W. Holland and L. Samuel, *Local structure in social networks*, Sociological Methodology, (1976), pp. 1–45.

[12] P. Holme and B. J. Kim, *Growing scale-free networks with tunable clustering*, Phys. Rev. E, 65 (2002), p. 026107.

[13] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, SIAM J. Computing, (1978), pp. 413–423.

[14] M. Latapy, *Main-memory triangle computations for very large (sparse (power-law)) graphs*, Theor. Comput. Sci., 407 (2008), pp. 458–473.

[15] D. R. Lick and A. T. White, *k-degenerate graphs*, Canadian Journal of Mathematics, 12 (1970), pp. 1082–1096.

[16] D. W. Matula and L. L. Beck, *Smallest-last ordering and clustering and graph coloring algorithms*, J. ACM, 30 (1983), pp. 417–427.

[17] J. Moody, *Matrix methods for calculating the triad census*, Social Networks, 20 (1998), pp. 291–299.

[18] C. S. J. A. Nash-Williams, *Edge-disjoint spanning trees of finite graphs*, Journal of London Mathemathical Society, 36 (1961), pp. 445–450.

[19] G. Robins, P. Pattison, Y. Kalish, and D. Lusher, *An introduction to exponential random graph (p *) models for social networks*, Social Networks, 29 (2007), pp. 173–191.

[20] T. Schank, *Algorithmic Aspects of Triangle-Based Network Analysis*, PhD thesis, Universität Fridericiana zu Karlsruhe (TH), 2007.

[21] T. Schank and D. Wagner, *Finding, counting and listing all triangles in large graphs, an experimental study*, in Workshop on Experimental and Efficient Algorithms (WEA), 2005, pp. 606–609.

[22] S. B. Seidman, *Network structure and minimum degree*, Social Networks, 5 (1983), pp. 269–287.

[23] T. A. B. Snijders, *The statistical evaluation of social network dynamics*, Sociological Methodology, 31 (2001), pp. 361–395.

[24] D. J. Watts and S. H. Strogatz, *Collective dynamics of "small-world" networks.*, Nature, 393 (1998), pp. 440–442.

[25] X. Zhou and T. Nishizeki, *Edge-coloring and f-coloring for various classes of graphs*, Journal of Graph Algorithms and Applications, 3 (1999), pp. 199–207.