# Implementation and Improvement of K-truss Decomposition Algorithm

## Introduction

Graph is commonly used to modeling relationships between entities. In the mining of graph, it is important to find dense subgraphs. Dense subgraph indicates that the nodes inside share common characters and have closer connections. Interesting result may be found by focusing on these areas. Additionally, the dense subgraph is the fundamental build blocks for other complex problems. Since the input graph is large, efficient algorithms are demanded to find and update the dens subgraphs[1].

There exist several definitions on dense subgraphs, such as k-clique, k-truss and k-core. In the application in real network, k-truss is a good choice, since k-clique has a more tight requirement while k-core is more loose. Compare to them, k-truss is more common and meaningful in social network analysis. In this work, we focus on the implementation and improvement of k-truss related algorithms.

## K-truss Definition

In the work of Cohen[2], the k-truss is defined as “a one-component subgraph such that each edge is reinforced by at least k-2 pairs of edges making a triangle with that edge”. An example of k-truss and k-truss decomposition is shown in the following figure. In the work of Cohen, they also proves that a k-clique is contained in a k-truss (notice the reverse is not ture).

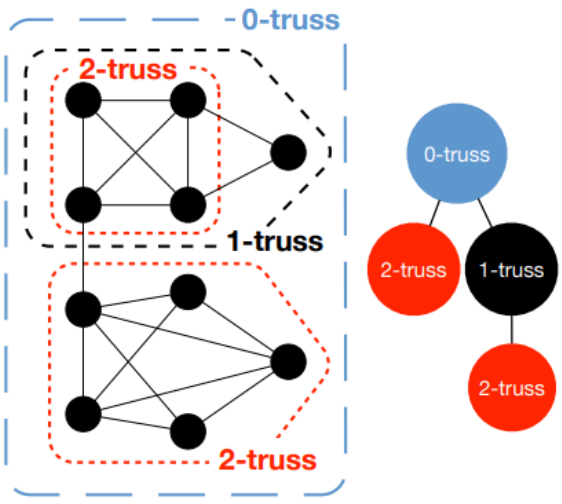


*Figure 1: K-truss and k-truss decomposition[2]*

# Peeling Algorithm

Most k-core and k-truss related algorithms follows a patten call the peeling process. Before the peeling process, counting for vertices (degree value) or edge (truss value) are prepared. When running the peeling algorithm, we remove the vertices/edges with the smalles degree/truss value iteratively untile no vertex/edge left. This is like peeling from outside to inside. Besides, there also exist streaming algorithms that can update the k-core value incrementally[3].

Due to the character of the peeling algorithm, it is hard to modify it totaly. Most of the improvements were done inside the peeling process, like accelerate the query and update when removing a vertex or an edge. Some methods have been proposed by using additional data to memorize triangles or motifs inside the graph[4, 5], and the inside process can be parallized.

# Progresses

## Implementation

In the first step, one fundamental algorithm proposed by Cohen[2] was studied and implemented. The original algorithm first reduces the graph to k-core subgraphs, then reduces the k-core subgraphs to k-truss subgraphs. However, the k-truss reduction process can also run without k-core reduction. In our implementation, only k-truss reduction process is used.

This is an early algoritm (algorithm A) to decompose k-truss subgraphs, and many optimizations were proposed by later researchers. Here we provide a comparation with the method proposed by Sariyuce et al. [4] (algorithm B). The data used here is “ego-Facebook” with 4,039 nodes and 88,234 edges.

| Algorithm | Time(s) |
|---|---|
| A | 109 |
| B | 0.6 |

As can be seen from the table, the improvemen of algorithm B is significant. In algorithm A, many times were spended on searching truss for the edge. This is because each time we remove an edge, many triangles may be broken, and we need to update all edges’ truss value in these triangles. Because the algorithm only saves vertex’s neighbors, we will have a nested loop to find influenced triangles. Assume the average number of vertex neighbor is $n$ , then each time we check the neighbor to find triangle and edge will take $O(n^2)$ to search and update edge truss value. Besides, the search space is not tight since many edge many not form another triangle. Another problem

with the algorithm is that, once an edge is deleted, we need to remove the edge and vertices in many places explicitly to avoid later duplicate decreasing of truss value. This also cost lots of time. Therefore, to improve the algorithm efficiency, additional imformations, like triangle count, will be introduced.

## Improvements in Literature

Many improvements are available for the k-truss decomposition. Here we take the work by Pearce and Sanders[5] as an example. In their work, they first assign an unique number to every vertex based on their degree. Then an Degree Ordered Directed Graph (DODG) is constructed. In each triangle, the vertices (p, q, r) are sorted that dp < dq < dr. The vertex p is the peak in the wedge, and edge {q, r} is a closure of the wedge to form a triangle.

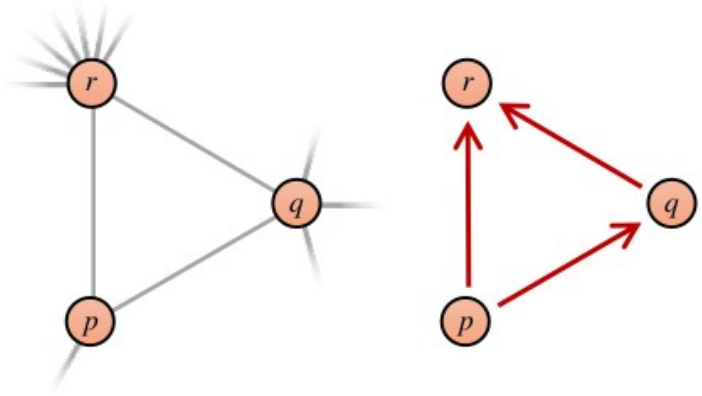


*Figure 2: From undirected graph to DODG[5]*

They saves not only the triangle count TC, but also the query edge count qrC and reference count pRefC. When removing an edge, they check if the edge is a query edge or a reference edge for other triangles, and reduce the count fore each edge respectively. By using these relations, they reduced the time to verify the existance of triangles, and only need to reduce the count for the edge rather than remove vertices/edges from the graph. More detaild removing process can be found from the original article.

In the work of  Saríyüce et al.[4] k-core and k-truss are generated as (1, 2) nuclei and (2, 3) nuclei. Fast and general algorithms were proposed for nuclei decomposition. Besides, they also proposed methods to generate the hierarchy structure of decomposition at the same time. (add more details later)

## Potential Improvements

As we have seen from above analysis, the bottle neck of the algorithm is how to check

the existance of triangle when we remove an edge. The check of triangle can also be considered as finding the intersection of neighbor vertices for two edge vertices.

In the work of  Eppstein et al.[6], method using cuckoo hash table to find the intersection of two set is proposed. Their method has potential to be introduced into our algorithm to acclerate the query process. In the following steps, the related 2-3 cuckoo hash-filter will be studied.

[1] Tsourakakis, Charalampos E. et al. “Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees.” KDD (2013).
[2] Cohen, Jonathan. "Trusses: Cohesive subgraphs for social network analysis." National security agency technical report 16 (2008): 3-1.
[3] Saríyüce, Ahmet Erdem, et al. "Streaming algorithms for k-core decomposition." Proceedings of the VLDB Endowment 6.6 (2013): 433-444.
[4] Sariyüce, Ahmet Erdem, and Ali Pinar. "Fast hierarchy construction for dense subgraphs." Proceedings of the VLDB Endowment 10.3 (2016): 97-108.
[5] Pearce, Roger, and Geoffrey Sanders. "K-truss decomposition for Scale-Free Graphs at Scale in Distributed Memory." 2018 IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.
[6] Eppstein, David et al. “2-3 Cuckoo Filters for Faster Triangle Listing and Set Intersection.” PODS (2017).