COL-380 : Assignment 2

Viral Marketing

Deadline: March 10, 2023

1 Introduction

In this assignment, you will compute Trusses in a graph. The assignment can be done in groups of two. (Select your partner on Moodle.)

2 Dataset Description

Let G = (V, E) be an undirected, unweighted, and simple graph with no self-loops, where V and E are the vertex and edge set, respectively. Thus $n = |V|$ represents the number of vertices, and $m = |E|$ represents the number of edges, and we also define neighbors of a vertex v, $adj(v) = \{u \in V : (v, u) \in E\}$ and $deg(v) = |adj(v)|$ represents the number of neighbours of node v (or degree).

2.1 Input Format

The structure for each binary test-input file is provided below. The corresponding representation is little-endian-format in the binary file, e.g., **test-input.gra**:
<4-bytes-representing n> <4-bytes-representing m>
«node-info-node-0»
«node-info-node-1»
—
—
«node-info-node-(n-1)»

where
«node-info-node-i»=<4-byte-representing node-i> <4-byte-representing-deg(i)> «neighbors»
and
«neighbors» = <4-bytes-representing neighbour-1> ... <4-bytes-representing neighbour-deg(i)>

Along with each test input.gra, you will also have a meta-data file, e.g., **header.dat**. This file will contain the start offset for each node in *test-input.bat* file. The same little-endian format is used for this also, and each offset is 4-byte. Thus, the offset for vertex i starts at byte $4 \times i$ in header.dat
Note that vertices in the graph are 0-based.

3 About Truss

A truss is an assembly of members, such as beams, connected by nodes that create a rigid structure. In engineering, a truss is a structure that "consists of two-force members only, where the members are organized so that the assemblage as a whole behaves as a single object. A

truss efficiently supports and transfers loads to other supporting structural members (such as columns, walls, and/or foundations) across relatively long spans. Read more on Trusses

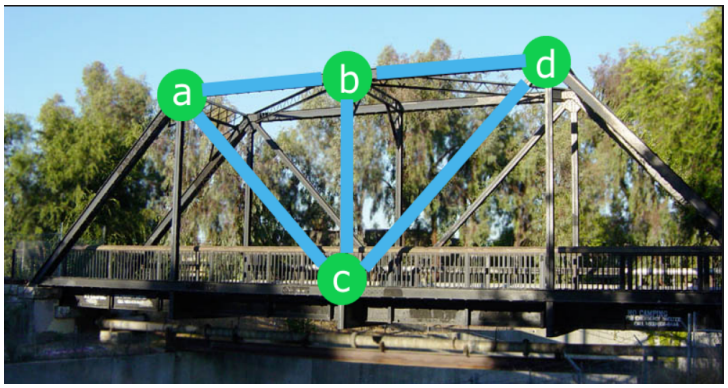

Figure 1: Truss Bridge

Above Figure1 can be represented using a graph G(V, E), where V represents the set of all junctions *a, b, c, d* as vertices and E represent the edge between the junction (aka nodes)$(a, b), (a, c), (b, c), (b, d), (c, d)$. A cycle of three vertices, e.g., b, c, d in the figure form a triangle of edges $(b, c), (c, d),$ and (d, e)

In graph theory, a *truss* is a type of subgraph that is particularly well-connected. Specifically, a k-truss of graph G is a subgraph H in which each edge is part of at least $k-2$ triangles within H. In other words, every edge in the k-truss is part of at least $k-2$ triangles made up of edged (and vertices) of that truss. We also say that each edge is supported by $k-2$ triangles or $k-2$ other vertices. (Terms nodes and vertices are used interchangeably.) Note that each triangle is formed by the given edge and edges incident on a supporting vertex. For example, in Figure 1 above, edge (b, c) is a part of two triangles and is supported by vertices a and d. Figure 2 shows a graph with a 4-truss (marked with blue vertices and edges – the size of **support** for some of its edges are also shown). Note that an edge (a, b) is supported by vertices in $adj(a) \cap adj(b)$, where $adj(v)$ is the list of vertices adjacent to v. Note also that $|adj(a) \cap adj(b)| \leq \max(deg(a), deg(b)) - 1$, where deg indicates the vertex degree.

Various applications of the truss are in **network analysis, social network analysis, and bioinformatics**

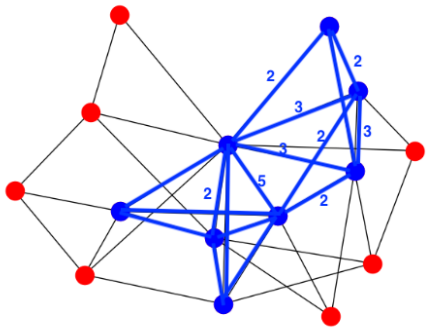

Figure 2: A sample 4-truss subgraph: The numbers indicate the number of triangles within the truss supporting the edge.

A maximal k-truss is not a subgraph of any larger k-truss. Here is a very simple algorithm to find maximal k-truss, given k and graph (V, E) –

Repeat while there is any edge to delete:
$\forall$ edge $(a, b) \in E$:
if $|adj(a) \cap adj(b)| < k - 2$:
delete (a, b)

Remaining connected components of (V, E) form k-truss. By k-truss, we will mean maximal k-truss. The algorithm above can be improved by filtering out non-promising cases first as follows. The steps are named Prefilter, Initialize, FilterEdges. (Note that orphaned vertices and edges must be removed.)

Prefilter(G, k): (Eliminate vertices with degree < k-1)

```
Deletable = {v s.t. deg(v) < k-1}
Repeat until |Deletable| > 0 {
   ∀ v ∈ Deletable:
      G = G - v
      // Queue neighbors for elimination if their degree dips below k
      ∀ u ∈ adj(v):
         if(deg(u) < k-1) Deletable += u
}
```

Initialize(G, k): (Gather edges with low support for subsequent filtering)

```
Deletable = ϕ
∀ e = (a,b) ∈ E:
   Supp(e) =  |adj(a) ∩ adj(b)|
   if(Supp(e) < k-2) Deletable += e
```

FilterEdges(G, k): (Eliminate edges with low support)

```
Repeat until |Deletable| > 0 {
   ∀ e = (a,b) ∈ Deletable:
      G = G - e
      // Removing this edge reduces support for others.
      ∀ edges f supporting e:
         Supp(f) --
         if(Supp(f) < k-2) Deletable += f
}
```

Note that one may start with a $k-1$-truss to find k-truss. Many other optimizations are possible. For examples, consult this paper and this paper.

4 Problem Statement

Insta-book is an online social networking website. It is a place where individuals exchange information with their friends. Social contagion is the phenomenon whereby individuals are influenced by the information they receive from their social neighborhoods, e.g., sharing the same posts or adopting the same political views as their friends. Social decisions made by individuals often depend on the *multiplicity of distinct social contexts* inside his/her contact neighbourhood are termed structural diversity.

You are working as a software developer in Insta-book. You are assigned to a project called *viral marketing*, and you need to solve multiple tasks based on the graph's structural diversity. You need to implement a parallel algorithm for each task using the OpenMPI Interface.

4.1 Tasks

Consider the social network of Insta-book represented as

$$G = (V, E) \ with \ n = |V| \ vertices \ and \ m = |E| \ edges$$

In Figure 3; We define that two vertices hold **friendship** relation iff there exists an edge between them.

- **Task 1: Advertisement** The client knows Social contagion and therefore requests you to publicize an advertisement on Insta-book. But the client requests you to publicize ads for only *specific groups** to maximize revenue. Each pair of friends in the group must have at least k common friends. For example, consider the edge between vertex 1 and 2 in Fig3, vertices 3, 4, and 5 are their *common friends*; 6 and 7 are not. Also, the group (1, 2, 3, 4, 5) satisfies that every pair of friends have at least two distinct friends in common. So k in the Fig3 is two, and this is the maximum value of k possible.
  Your task is to find for what values of k $\in [k_1, k_2]$, there exists a *group* of a given size k in G.

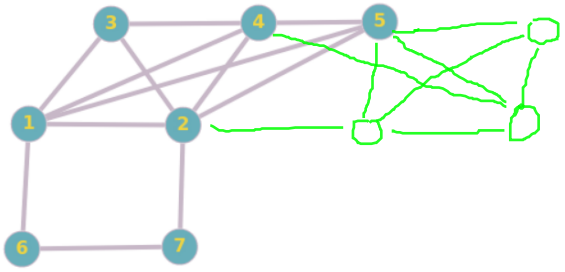

Figure 3: Task-1 Graph

- **Task 2: Ego-Network** After some research, the client identified that instead of sending an advertisement to everyone in the group, it would be better to identify a common node that connects to different social groups. We refer to these common vertices as influencers. In Figure 4(b), subgraphs H_3, H_4 and H_2 can be considered as different social groups. We refer to the friends of an influencer-vertex as its ego-network.

  In other words, the ego-network of an individual influencer-vertex v is a subgraph of G formed by all v*'s* neighbors as shown in the light gray region **(excluding vertex v)** in Figure 4(b).

  Task 2 is is required only for COV880 students. Your task is to identify all influencer vertices connected to at least p non-overlapping k-advertisement group. Two distinct influencer-vertex can share friends.

5 Deliverable

A zip archive with the filename *<FirstName>_<EntryNo>_A2.zip*. The format for the entry number should be *20XXCSXXXXX*.

On unzip, it should produce the following directory structure:

```
<FirstName>_<EntryNo>_A2/
    |-- Makefile
    |-- report.pdf
    |-- other code files
    |-- executable
```

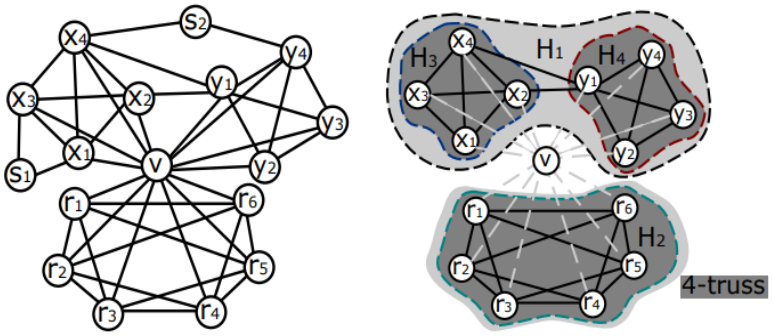

Figure 4: Graph G(left) and the Ego-Network $G_{adj(v)}$(right)

– The following command needs to be executed to run your program:
  – *make*: This should create an executable for your program named as *a2*.

A sample of the PBS script and parameter to the executable.

```
#!bin/bash
#PBS ...
mpirun -np <no-of-processes> ./a2 --taskid=<1/2>
            --inputpath=<path-to-graph-input-file>
            --headerpath=<path-to-header-file>
            --outputpath=<path-to-output-file>
            --verbose=<0/1>
            --startk=<start-of-range>
            --endk=<end-of-range>
            --p=<p>
```

Where task_id is an integer (1, 2) based on the tasks mentioned in 4.1 and defaults to 1. Arguments inputpath, headerpath, and outputpath refer to the complete path. Verbose is a binary flag that take two values <0/1>, default being 0. The output will be different based on this value. Value of p is only relevant for Task 2, and not required for Task 1.

The output file should be a normal text file. The file should be saved at the location passed to command line argument *outputpath*. The output formats for each task are described below.

Note: Based on the value of the Verbose flag the output will change for each task.

- Task 1: The output file should contain results for each k $\in [startk, endk]$, separated using *single-space*. Input to this task will contain the command line arguments *taskid, inputpath, headerpath, outputpath, verbose, startk, endk*. For each $k \in [startk, endk]$, the output file should contain a boolean value (0/1) that shows whether there exists a group of size k or not. Additionally, if the *verbose* flag is 1, and if there exists group(s) of size k, then first on the new line (after 1), output the count of the k-size group(s)(say c), and in the next c subsequent line(s), for each group output the vertices which should be space separated in any order. Also, these c groups can be in any order.

- Task 2: The output file should contain each influencer vertex separated with a *single space*. The social group is any group where each pair of friends is supported by *endk* friends. If there are no influencer vertices, then the output should contain a single integer, i.e., -1. If the *verbose* flag is 1, then addition to the influencer vertex, the next line of the file should contain the count of the social groups(say c), and in the next c, subsequent line(s) should contain the member(or vertices) of these groups which should be space separated. All the c social groups can be in any order, and within the group, the order doesn't matter.

For each task, your memory usage for a single process should be limited to 4 GB.
You must also provide a comprehensive report, including graphs and tables for speedup and efficiency. Describe in detail the approach you took for each task and describe the observations.

6 Other references

1. Truss Decomposition on Shared-Memory Parallel Systems Systems Shaden Smith, Xing Liu†
2. Parallel Triangle Counting and k-Truss Identification using Graph-centric Methods Chad Voegele, Yi-Shan Lu, Sreepathi Pait and Keshav Pingali