CS F364 ASSIGNMENT 2

Submitted To

Prof.Apurba Das

Design and Analysis of Algorithms : CS F364

BITS Pilani

Submitted By **Group 38** :

Name	BITS ID
Kakade Rohan Bhaskar	2022A7PS0216H
Rithvik Raajha	2022A7PS1361H
Vatsal Tyagi	2022A7PS1388H
Nishit Nilay	2022A7PS0230H
Aditya Narwania	2022A7PS1540H

Birla Institute of Technology and Science, Hyderabad Campus

Introduction:

This algorithm is used to find the densest subgraphs in the most

efficient manner. For a graph Density can be defined as the ratio of the number of

edges and the number of vertices.

$$density\ of\ a\ graph\ G(v, E) = \frac{number\ of\ Edges}{number\ of\ vertices}$$

This problem in graphs has wide application in many fields

including networks , biology , graph databases and system

optimization.

The problem has two approaches :

1. Edge-density based DSD

2. h-clique based DSD

The challenges to solve this problem are enormous with the

already existing algorithms (These algorithms currently use

flow), because these algorithms (edge-density based DSD) are

very slow for large graphs and (h-clique based DSD) is even

more computationally expensive. To handle this issue the

authors of this paper have come up with a solution using k-core decomposition, where k core is a maximal subgraph where each vertex has a degree at least k. (k, Ψ)-core generalizes k-core for h-cliques. Crucial for efficiently solving DSD based on h-clique-density or pattern-density.

Datasets

Datasets	Vertices	Edges
as20000102	6474	12572
CA-HepTh	9577	25998
Netscience	1589	2742
as-Caida	26475	106762

Algorithm 1 : Exact Algorithm

Algorithm 1: The algorithm: `Exact`.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The CDS $D(V_D, E_D)$;
initialize $l \leftarrow 0$, $u \leftarrow \max_{v \in V} deg_G(v, \Psi)$;
initialize $\Lambda \leftarrow$all the instances of $(h$–$1)$-clique in G, $D \leftarrow \emptyset$;
while $u - l \geq \frac{1}{n(n-1)}$ **do**
  $\alpha \leftarrow \frac{l+u}{2}$;
  $V_\mathcal{F} \leftarrow \{s\} \cup V \cup \Lambda \cup \{t\}$; `// build a flow network`
  for *each vertex* $v \in V$ **do**
    add an edge $s \rightarrow v$ with capacity $deg_G(v, \Psi)$;
    add an edge $v \rightarrow t$ with capacity $\alpha |V_\Psi|$;
  for *each (h–1)-clique* $\psi \in \Lambda$ **do**
    for *each vertex* $v \in \psi$ **do**
      add an edge $\psi \rightarrow v$ with capacity $+\infty$;
  for *each (h–1)-clique* $\psi \in \Lambda$ **do**
    for *each vertex* $v \in V$ **do**
      if ψ *and* v *form an h-clique* **then**
        add an edge $v \rightarrow \psi$ with capacity 1;
  find minimum st-cut $(\mathcal{S}, \mathcal{T})$ from the flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;
  if $\mathcal{S}=\{s\}$ **then** $u \leftarrow \alpha$;
  else $l \leftarrow \alpha$, $D \leftarrow$ the subgraph induced by $\mathcal{S} \backslash \{s\}$;
return D;

In graph theory, a clique is a subset of vertices in a graph where every two distinct vertices are adjacent. The problem of detecting an h-clique, where each clique contains exactly hhh vertices, has significant applications in network analysis, social networks, and bioinformatics.

The Exact algorithm aims to efficiently identify h-cliques in a graph **G=(V,E)** using a flow network approach. The algorithm proceeds through the following key steps:

Exact Algorithm Overview

1. **Input Parameters:** The algorithm operates on a graph **G=(V,E)** with vertices V and edges E.
   A set of potential h-cliques ψ, and a degree function **deg(v,ψ)** that determines the number of vertices a node connects to within the clique.

2. **Initialization:** A binary search is set up for the maximum degree within the graph. The lower bound l=0 and the upper bound **u=max deg(v,ψ)** for all vertices v.
   A flow network is initialized to store the identified h-cliques.

3. **Building the Flow Network:** A source node is connected to each vertex vvv with a capacity equal to its degree.
   Vertices are connected to a sink with a capacity scaled by a factor α\alphaα, depending on the graph's properties.
   For each identified h-clique (denoted Δ), edges are created to

represent the connections between the vertices forming the clique.

4. **Flow Calculations:** The algorithm calculates maximum flow iteratively, using binary search on the edge capacities. This helps identify a subgraph with a density that satisfies the conditions for an h-clique.

5. **Subgraph Extraction:** If a minimum cut is found in the flow network, and the nodes connected to the sink are reachable under the set capacity, the subgraph is extracted as a valid h-clique.

6. **Termination**: The binary search converges, and the final set of h-cliques or the densest subgraph is returned, based on the flow calculations and capacities.

Algorithm 2 : CodeExact Algorithm

Algorithm 4: The algorithm: `CoreExact`.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The CDS $D(V_D, E_D)$;
perform core decomposition using Algorithm 3;
locate the (k'', Ψ)-core using pruning criteria;
initialize $\mathcal{C} \leftarrow \emptyset$, $D \leftarrow \emptyset$, $U \leftarrow \emptyset$, $l \leftarrow \rho''$, $u \leftarrow k_{\max}$;
put all the connected components of (k'', Ψ)-core into $\mathcal{C}$;
for *each connected component* $C(V_C, E_C) \in \mathcal{C}$ **do**
  if $l > k''$ **then** $C(V_C, E_C) \leftarrow C \cap (\lceil l \rceil, \Psi)$-core;
  build a flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$ by lines 5-15 of Algorithm 1;
  find minimum st-cut $(\mathcal{S}, \mathcal{T})$ from $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;
  if $\mathcal{S} = \emptyset$ **then** continue;
  while $u - l \geq \frac{1}{|V_C|(|V_C|-1)}$ **do**
    $\alpha \leftarrow \frac{l+u}{2}$;
    build $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$ by lines 5-15 of Algorithm 1;
    find minimum st-cut $(\mathcal{S}, \mathcal{T})$ from $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;
    if $\mathcal{S} = \{s\}$ **then**
      $u \leftarrow \alpha$;
    else
      if $\alpha > \lceil l \rceil$ **then** remove some vertices from C;
      $l \leftarrow \alpha$;
      $U \leftarrow \mathcal{S} \backslash \{s\}$;
  if $\rho(G[U], \Psi) > \rho(D, \Psi)$ **then** $D \leftarrow G[U]$;
return D;

CoreExact is an advanced algorithm used for core decomposition in graph theory, designed to efficiently identify the most connected subgraphs (the "core") within a graph **G=(V,E).** It builds on the concept of k-core decomposition, where vertices with degree less than kkk are removed, leaving behind a k-core — a subgraph where each vertex has at least k neighbors.

This algorithm is widely applied in fields such as social network analysis, bioinformatics, and complex systems, where understanding the structure of a graph and identifying its densest, most strongly connected subcomponents is crucial.

CoreExact Algorithm Overview

The CoreExact algorithm processes a graph G through the following steps:

- **Input and Initialization:** The algorithm begins with a graph G consisting of vertices V and edges E, along with a set Ψ of potential core decompositions.
  The goal is to identify the core decomposition $D(V_D, E_D);$, where is the set of core vertices and EDE_DED is the set of edges connecting them.

- **Core Decomposition:** The algorithm starts by performing core decomposition (using Algorithm 3), identifying k-core subgraphs within the graph.
  It initializes the sets for the k-core, vertices, and other parameters like thresholds α and kmax.

- **Identifying Connected Components:** The algorithm iterates over the connected components within the k-core. For each component C, if its size exceeds k’, it proceeds to construct a flow network to calculate the minimum s-cut.

- **Flow Network Construction**: A flow network $\mathcal{F}(V_{\mathcal{F}}, E_{\mathcal{F}});$ is built using the maximum flow algorithm. This network helps identify the strongest parts of the graph by calculating the flow and determining the minimum cut between source SSS and sink T.

- **Refining the Core**: After processing the components and calculating the flow cut, the algorithm updates the core by removing vertices that don’t meet the k-core threshold. This refinement continues until the algorithm converges.

- **Termination**: The process concludes when the most stable and connected subgraphs are identified, providing the core decomposition of the graph.

Results :

1. Exact

Datasets	H=2	H=3	H=4
as20000102	8.865	35.910	85.130
CA-HepTh	15.4	155	1123.25
Netscience	9.2	56.5	242.4
as-Caida	17.34	114.44	405.213

2. CoreExact

Datasets	H=2	H=3	H=4
as20000102	8.865	35.910	85.130
CA-HepTh	15.3	154.5	1123.10
Netscience	9.2	56.5	242.35
as-Caida	17.34	114.44	405.213