Name-bharat gupta
Group no-58
Total group members-1
Id- f2022A7PS1374H

Algorithm 1

Algorithm 1 is designed to find the densest subgraph within a given network. It sets up a flow network where the goal is to determine how strongly nodes are connected at different density thresholds. By running a minimum s-t cut algorithm, it divides the graph into two disjoint sets: one containing the source node and the other containing the sink node. The minimum cut identifies the smallest capacity of edges needed to separate these sets, revealing whether a densely connected group of nodes exists. If a meaningful subgraph is found (i.e., S contains more than just the source), the algorithm increases the density guess; otherwise, it lowers it. Through this binary search over density values, the algorithm efficiently zooms in on the densest subgraph in the network.

What does finding disjoint sets (S, T) mean here?

- After setting up the **flow network** (adding all edges and capacities),
- The algorithm runs a **minimum s-t cut** algorithm.

This **divides** the nodes into two **disjoint sets**:

- **S**: the set containing the source node **s**.
- **T**: the set containing the sink node **t**.

No node is in both S and T — they are disjoint. And **the total capacity** of edges going from **S to T** is **minimized**.

Why do we compute the minimum cut?

- The minimum cut tells **how much "flow" is needed to separate s from t**.
- If the minimum cut is **small**, it suggests a **dense cluster** of nodes in S (excluding s).
- The nodes in **S (excluding s)** are candidates for being the **densest subgraph**.

- If **S only contains {s}**, it means there is **no dense enough subgraph** at the current density threshold α, so the algorithm adjusts the search.

After finding (S, T):

- If **S = {s} only** → then no dense subgraph found at current guess α → **decrease upper bound u = α**.
- Else → we **found a dense subgraph** → **increase lower bound l = α** and **set D to the subgraph induced by S{s}**.

Intuition:

- The algorithm **searches** for the highest α (density) where a nontrivial cut exists.
- The **minimum s-t cut** helps find groups of vertices that can "support" that density.

In simple terms:

Step	Meaning
Build flow network	Encode density constraints into flow edges.
Find min s-t cut (S, T)	Separate graph into two parts minimizing cut capacity.
Analyze S	If S is non-trivial (more than {s}), it represents a **candidate densest subgraph**. Otherwise, adjust α.

Algorithm 2

What the (k, Ψ)-Core Decomposition algorithm does:

- It **computes the clique-core number** for every vertex in a graph.
- The **(k, Ψ)-core** is the largest subgraph where **each vertex participates in at least k instances** of a specific pattern Ψ (often Ψ is a clique, but it could be other patterns like stars or loops).

How it works:

1. **Compute the clique-degree** of each vertex (number of cliques it participates in).
2. **Sort vertices** in **increasing order** of their clique-degree.
3. **Iteratively remove** the vertex with the **smallest** clique-degree:
    - When a vertex is removed, **decrease** the clique-degree of its neighbors that share clique instances with it.
    - **Resort** vertices efficiently using **bin-sort**.
4. Record the **clique-core number** for each vertex when it is removed.
5. Continue until all vertices are removed.
6. Finally, return the array containing the **clique-core numbers**.

Advantages of (k, Ψ)-Core Decomposition:

- **Efficient**: The algorithm is designed to run in **O(n·d^(h−1))** time and **O(m)** space, where:
    - n = number of vertices,
    - d = maximum degree,

- h = size of the cliques.

- **Scalable**: Using bin-sort allows **fast re-sorting** after every removal.
- **Generalizable**: Can handle **any pattern Ψ** (not just cliques), making it flexible.
- **Helpful for Core-Based Algorithms**:
    - Reduces the size of the graph for further computations (e.g., finding densest subgraphs).
    - Provides **tighter bounds** and **smaller search spaces** in optimization problems like dense subgraph discovery.

Dataset used:-

Name
Yeast
Netscience
As-733
Ca-HepTh
As-Caida

Results

2. Netscience Dataset

Exact (Algo-1) Performance

Nodes	Edges	Density	Execution Time	Clique Number
1589	2742	9.000023	3.771 s	2
1589	2742	58.9992	0.208 s	3
1589	2742	242.7	2.680 s	4

CoreExact (Algo-4) Performance

Nodes	Edges	Density	Execution Time	Clique Number
1589	2742	9.5273	18.5757 s	2
1589	2742	56.628	23.8944 s	3
1589	2742	242.0000	923.63 s	4

3. As20000102 Dataset

Exact (Algo-1) Performance

Nodes	Edges	Density	Execution Time	Clique Number
6474	13233	8.8	13.2441 s	2
6474	13233	36.01	7.944 s	3
6474	13233	85.5	14.713 s	4

CoreExact (Algo-4) Performance

Nodes	Edges	Density	Execution Time	Clique Number
6474	13233	8.87	230.17 s	2
6474	13233	36.88	223.2 s	3
6474	13233	84.9999	1043.8 s	4

4. CA-HepTh Dataset

Exact (Algo-1) Performance

Nodes	Edges	Density	Execution Time	Clique Number
9877	51971	15.5234	4.6s	2
9877	51971	155.002	4.465 s	3
9877	51971	1123.752	19.2812 s	4

CoreExact (Algo-4) Performance

Nodes	Edges	Density	Execution Time	Clique Number
9877	51971	15.4	512.2 s	2
9877	51971	154.8	1022.1 s	3
9877	51971	1123.75	2309.9 s	4

5. AS-Caida Dataset

Exact (Algo-1) Performance

Nodes	Edges	Density	Execution Time	Clique Number
26475	106762	17.5341	4.2764 s	2
26475	106762	114.847	86.2427 s	3
26475	106762	405.333	211.743 s	4

CoreExact (Algo-4) Performance

Nodes	Edges	Density	Execution Time	Clique Number
26475	106762	17.535	1.32 hrs	2
26475	106762	114.85	3.59 hrs	3
26475	106762	405.333	6.77hrs	4