# Efficient Algorithms for Densest Subgraph Discovery

**SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS OF**

**CS F364:**

**DESIGN AND ANALYSIS OF ALGORITHMS**


| CONTRIBUTORS | ROLL NUMBERS | CONTRIBUTIONS |
|---|---|---|
| Anshul Gopal | 2022A7PS0009H | Algorithm 4 and Readme file |
| Mohammad Jambughodawala | 2022A7PS0044H | Algorithm 1 and Website |
| GB Hemanth Karthikeya | 2022A7PS0053H | Algorithm 4 and Website |
| Vaishnav Devaguptapu | 2022A7PS0085H | Algorithm 4 and Project Report |
| Harsh Vikram Jajodia | 2022A7PS0171H | Algorithm 1 and Project Report |

# 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.

## ABOUT THE DATASETS:

| Datasets | Type Of Graph | No. Of Vertices | No. Of Edges |
|---|---|---|---|
| as20000102 | Real small graphs | 6,474 | 12,572 |
| as-Caida | Real small graphs | 26,475 | 106,762 |
| Netscience | Real small graphs | 1,589 | 2,742 |
| CA-HepTh | Real small graph | 9,877 | 25,998 |

# ALGORITHM 1 (EXACT):

This algorithm is called exact and it computes the Connected Dominating Set (CDS) for a graph based on h-clique.

The following steps in this algorithm are:

1. Initialize: set l = 0, u = maximum degree of a vertex, Set Λ to all the (h−1)-cliques in the graph. Initialize the dominating set D as empty.
2. Binary Search Loop (repeat while u - l ≥ $\frac{1}{n(n-1)}$¿:

   Set α = $u + \left(\frac{l-u}{2}\right)$ . Then build a flow network.

3. Find the minimum s-t cut (S,T) in the flow network.
4. If only the source s is in S (i.e., s ε {S}) then update u to α
5. Otherwise update l to α, then update the dominating set D to a subgraph induced by S / {s}
6. After the loop ends return D.

The algorithm uses a **binary search combined with flow network minimum cuts** to **find a minimum-weight Connected Dominating Set (CDS)** based on the structure of **cliques** in the graph.

The algorithm smartly shrinks the search space for the optimal dominating set using binary search, and verifies candidate solutions by modeling the problem as a network flow, using clique structures to enforce connectivity and domination.

## ALGORITHM 4 (CORE EXACT) :

Use flow networks and core-decompositions, on G(V,E) with vertex set V and Edge Set E to compute Connected Dominated Set(CDS).

1. **Core Decomposition**

   Perform a core decomposition of the graph using another algorithm (Algorithm 3).
   (Basically, break the graph into smaller "core" structures based on how strongly nodes are connected.)

2. **Locate Important Core:**
   Find the (k'',Ψ)-core of the graph using pruning rules.
   (This is a highly connected subset of the graph.)

3. **Initialize Variables:**

   Set up some empty sets and variables:

   - C, D, U = empty sets
   - l, ρ'' = 0
   - u = kmax(the maximum core number)

4. **Group Connected Components:**

   Find all connected components of the (k'', Ψ)-core and add them into a set C. (Basically, split the core into groups where each group is internally connected.)

5. **Process Each Connected Component:**

For each component C (VC, EC) in C:

- If l>k'':
  Update the component to an even tighter core.
- Build a Flow Network:
  Build a flow network using the technique from lines 5–15 of Algorithm 1.
- Find the Minimum s-t Cut:
  Find a minimum cut separating the source s from the sink t.
- If the result is empty, skip this component.

**6. Binary Search for Best Cut:**

- While u- is still big enough (≥ a threshold):
- Set α = the middle value between l and u.
- Build another Flow Network using α\alphaα.
- Find the minimum cut again.
- If the cut separates only source s from everything else:
- u=α
- Else:
- If α>[l](some threshold):
  1. Remove some vertices from C.
  2. Update l=α.
  3. Update U as the remaining vertices after removing s.

7. **Final Step for Component**

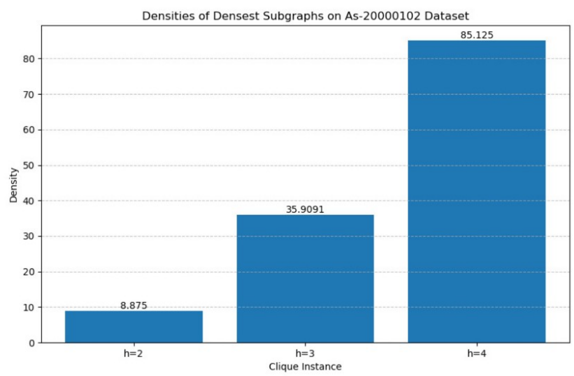
After binary search is done: If the density of G[U]is better than the previous density, update include G[U].

8. After all components are processed, **return D as the CDS.**

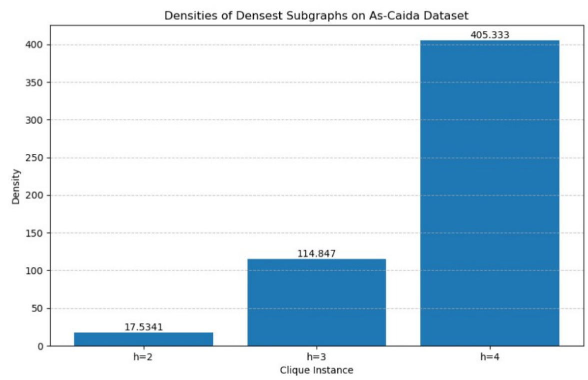
# RESULTS:

| DataSet | h = 2 (h-1 clique Density) | h = 3 (h-1 clique Density) | h = 4 (h-1 clique Density) |
|---|---|---|---|
| as20000102 | 35.9091 | 8.875 | 85.125 |
| as-Caida | 17.5341 | 114.847 | 405.333 |
| Netscience | 9.5 | 57 | 242.25 |
| CA-HepTh | 15.5 | 155 | 1123.75 |

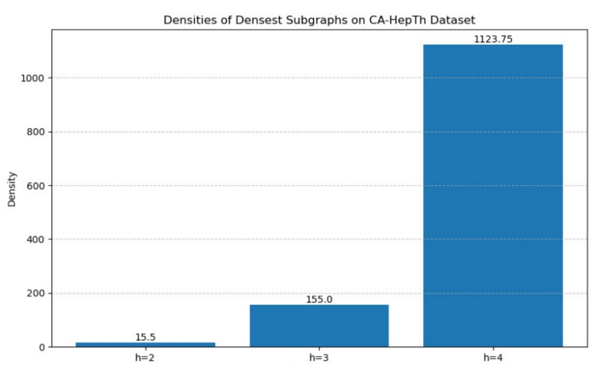
# PLOTS:



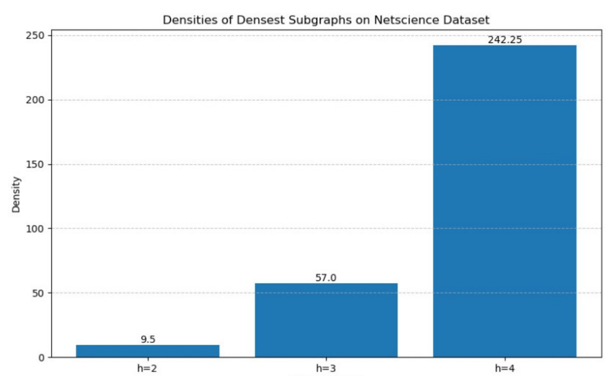

**as20000102**




**as-Caida**




**CA-HepTh**




**Netscience**

*THANK YOU*