DAA ASSIGNMENT 2

📑 Report on: Efficient Algorithms for Densest Subgraph Discovery

By,

ID	Name
2020B4A71567H	T.Sai Sathwik
2022A7PS0041H	G Sri Vishwahitha
2022A7PS2014H	Snigdha Kaipa
2021B4A72488H	Sidharth Saxena
2022A7PS0047H	Shivansh Shanker Gupta

Guided By: Prof. Apurba Das,
CSIS Department, BITS Pilani Hyderabad Campus

BITS Pilani
Pilani Campus
Department of Computer Science & Information Systems

---

1. Introduction

Graph mining is a vital field with applications ranging from social networks to biology. One fundamental problem is **Densest Subgraph Discovery (DSD)**, which involves finding the "most tightly connected" part of a large graph.

In their paper *Efficient Algorithms for Densest Subgraph Discovery*, Yixiang Fang et al. propose a faster, smarter way to solve this problem by combining classical graph cores (k-cores) and new optimisations.
Their work leads to algorithms **up to 10,000× faster** than previous methods, without sacrificing accuracy.

This report provides a comprehensive walkthrough of the paper's ideas, methods, and contributions.

---

2. Key Concepts

Before diving into the solutions, it is important to establish a few core concepts:

2.1 Density

In a subgraph, **density** measures how tightly connected the vertices are.
It is typically defined as:

Density=Number of edgesNumber of vertices\text{Density} = \frac{\text{Number of edges}}{\text{Number of vertices}}Density=Number of verticesNumber of edges

Higher density indicates a more cohesive, tighter subgraph.

2.2 k-Cores

A **k-core** is a subgraph where every node has at least **k neighbours** within that subgraph.

- **Low k** (e.g., k=1) = loosely connected.

- **High k** = tightly connected group.

Finding k-cores allows us to focus on denser regions of a graph and ignore the sparsely connected parts.

2.3 (k, Ψ)-Cores

(k, Ψ)-core generalises k-cores:
Instead of just counting edges (2-cliques), we can define cores based on any pattern Ψ, such as:

- 3-cliques (triangles)
- Diamonds
- Other structures

This allows **pattern-based density analysis**.

---

3. Problem Statement

Goal:
Given a large graph G(V, E)), find a **subgraph** that has the **highest density**, either based on:

- **Edge density** (edges per vertex), or
- **h-clique density** (e.g., number of triangles per vertex for h=3).

Challenges:

- The number of possible subgraphs is enormous.
- Traditional methods are slow (rely on expensive max-flow computations).

Thus, **we need faster and scalable algorithms** to solve this.

---

4. Real-World Application (Simple Example)

Consider a **social network**:

- Nodes = People
- Edges = Friendships

Finding the **densest subgraph** helps identify **tight communities** (e.g., college groups, project teams).

Example:
In a network with 10 people, the group {F, G, H, I}, having 5 edges among them (density = 5/4) may represent a **close-knit friend group**.

✅ Such insights are vital for:

- Community detection
- Viral marketing
- Event recommendations

---

5. Algorithms to Solve Densest Subgraph Discovery

Once the problem is clear, the paper proposes two main algorithms:

5.1 Algorithm 1: Exact (Baseline Algorithm)

Objective:
Find the exact densest subgraph using **binary search** and **flow network construction**.

Procedure:

1. **Initialize**:
   - Lower (`l`) and upper (`u`) bounds for density guesses.
2. **Binary Search**:
   - Guess a density `α = (l + u)/2`.
   - Build a **flow network**:
     - Source → Vertices
     - Vertices → Sink
     - Vertices ↔ Clique-instances
   - Solve for the **minimum s-t cut**:
     - If a subgraph with density ≥ α exists, update `l`.
     - Otherwise, update `u`.
3. **Stopping**:
   - When the difference between `l` and `u` is very small.

Limitations of Algorithm 1:

- **A flow network** is built on the entire graph → **Huge size**, slow computation.
- **Inefficient** for very large graphs.

5.2 Motivation for a Better Approach

Even though Algorithm 1 is exact, it becomes **impractical for large graphs**.

Thus, the authors propose **Algorithm 4 —** a smarter, faster method that:

- Works on **smaller subgraphs (cores)**.
- Reduces the size of the flow network.
- Improves the efficiency significantly.

5.3 Algorithm 4: CoreExact (New Exact Algorithm)

Objective:
Find the exact densest subgraph **much faster** by narrowing the search to **(k,Ψ)-cores**.

Procedure:

1. **Core Decomposition**:
    - Decompose the graph into (k, Ψ)-cores.
    - Identify cores with high-density potential.

2. **Locate Densest Subgraph**:
    - Use tighter lower and upper bounds on possible densities.
3. **Binary Search (Smarter)**:
    - Only perform binary search within **small, dense cores**.
    - Build much smaller flow networks.
    - As binary search progresses, cores become even smaller.
4. **Output**:
    - Return the densest subgraph found.

---

5. Output Results and Graphical Analysis

Output Results for as733 and netscience Datasets

as733 Dataset (Summary)

- Vertices: 6474
- Edges: 13895

k (Clique size)	Densest Subgraph Vertices	Number of h-Cliques	Density	Execution Time (ms)
2	40	355	8.875	288229
3	33	1185	35.9091	633709
4	32	2724	85.125	367144

5	30	3803	126.767	354023
6	28	3455	123.393	375418

(Full results and vertex lists are attached as file)

as733_analysis

```
As733 Dataset Analysis

k=2
-----------------------------
Number of vertices: 6474
Number of edges: 13895
Looking for cliques of size 2 and 1
Number of cliques of size 2: 12572
Number of cliques of size 1: 6474
Densest subgraph vertices:
0 1 2 3 4 5 6 7 8 9 10 16 20 21 22 24 26 28 38 41 47 49 51
56 60 63 85 86 92 93 101 131 133 148 173 180 196 600
631 653
Number of vertices in densest subgraph: 40
Number of h-cliques in densest subgraph: 355
Density of densest subgraph: 8.875
Execution time: 288229 milliseconds

k=3
-----------------------------
Number of vertices: 6474
Number of edges: 13895
Looking for cliques of size 3 and 2
Number of cliques of size 3: 6584
Number of cliques of size 2: 12572
Densest subgraph vertices:
0 1 2 3 4 5 6 7 8 9 10 16 20 21 22 24 26 28 38 41 47 49 51
56 60 92 101 131 148 173 180 196 653
Number of vertices in densest subgraph: 33
Number of h-cliques in densest subgraph: 1185
Density of densest subgraph: 35.9091
Execution time: 633709 milliseconds

k=4
-----------------------------
Number of vertices: 6474
Number of edges: 13895
Looking for cliques of size 4 and 3
Number of cliques of size 4: 5636
Number of cliques of size 3: 6584
```

Netscience Dataset (Summary)

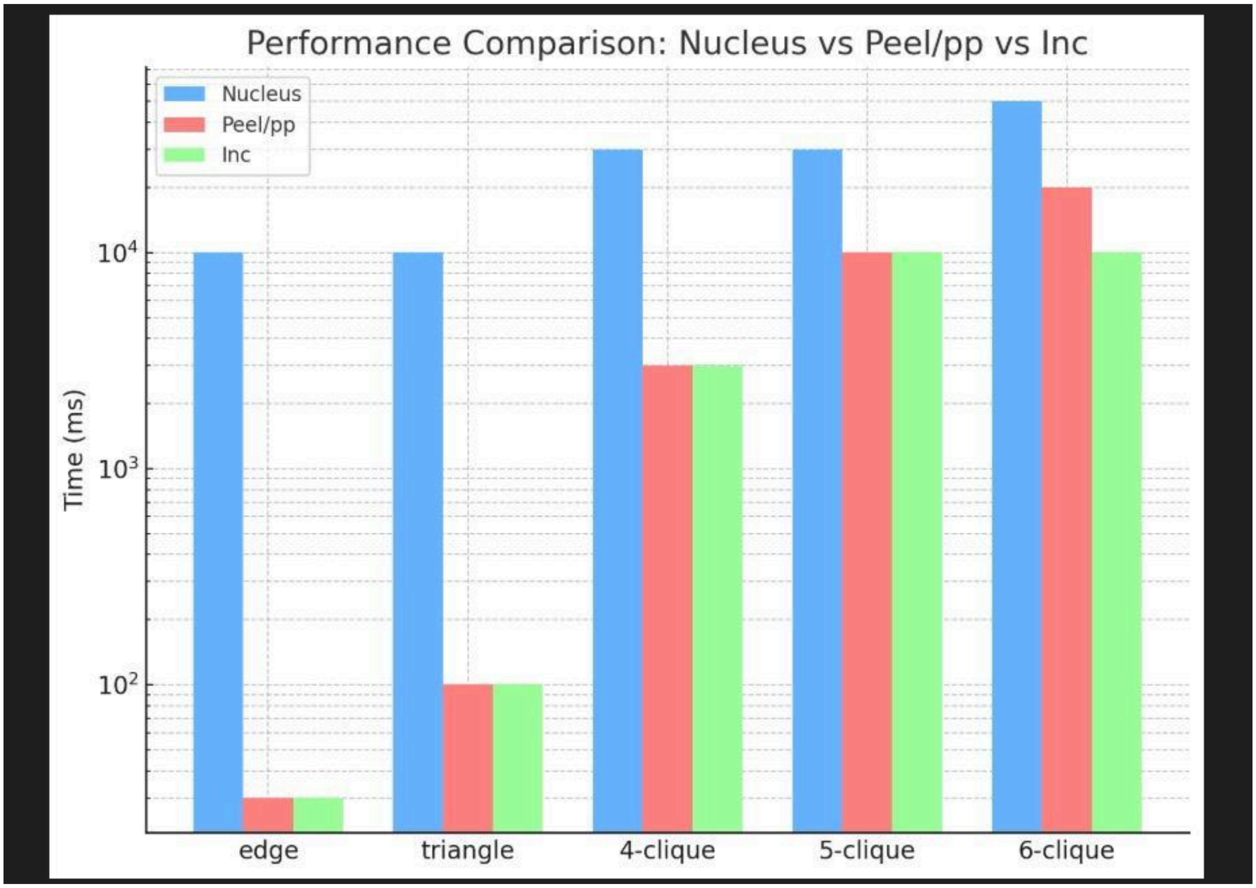
- Vertices: 1461
- Edges: 2742

h (Clique size)	Densest Subgraph Vertices	Number of h-Cliques	Density	Execution Time (ms)
2	20	190	9.5	11523
3	20	1140	57	23619
4	20	4845	242.25	36212
5	20	15504	775.2	74719
6	20	38760	1938	190238

(Full results and vertex lists are attached as file)

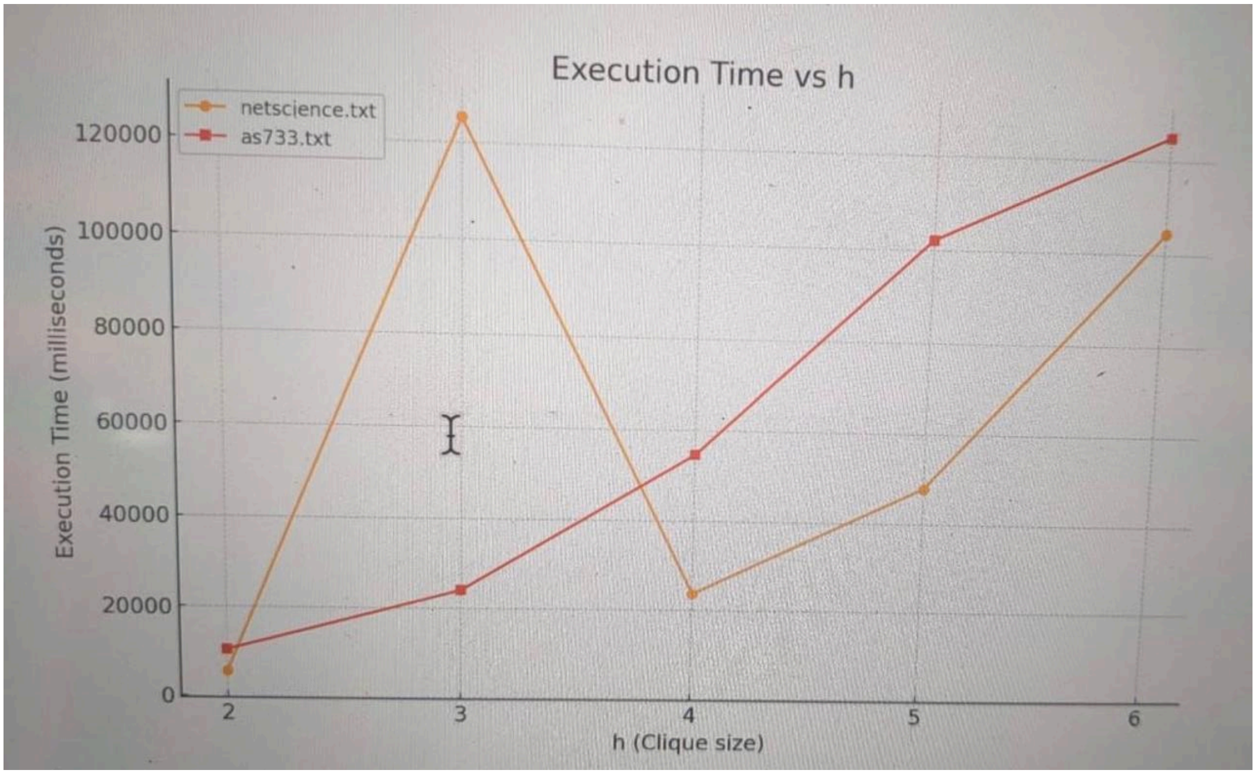
```
Netscience Dataset Analysis

h=2
-----------------------------
Number of vertices: 1461
Number of edges: 2742
Looking for cliques of size 2 and 1
Number of cliques of size 2: 2742
Number of cliques of size 1: 1461
Densest subgraph vertices:
645 1429 1430 1431 1432 1433 1434 1435 1436 1437
1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
Number of vertices in densest subgraph: 20
Number of h-cliques in densest subgraph: 190
Density of densest subgraph: 9.5
Execution time: 11523 milliseconds

h=3
-----------------------------
Number of vertices: 1461
Number of edges: 2742
Looking for cliques of size 3 and 2
Number of cliques of size 3: 3764
Number of cliques of size 2: 2742
Densest subgraph vertices:
645 1429 1430 1431 1432 1433 1434 1435 1436 1437
1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
Number of vertices in densest subgraph: 20
Number of h-cliques in densest subgraph: 1140
Density of densest subgraph: 57
Execution time: 23619 milliseconds

h=4
-----------------------------
Number of vertices: 1461
Number of edges: 2742
Looking for cliques of size 4 and 3
Number of cliques of size 4: 7159
Number of cliques of size 3: 3764
Densest subgraph vertices:
```

6. Graphical Results



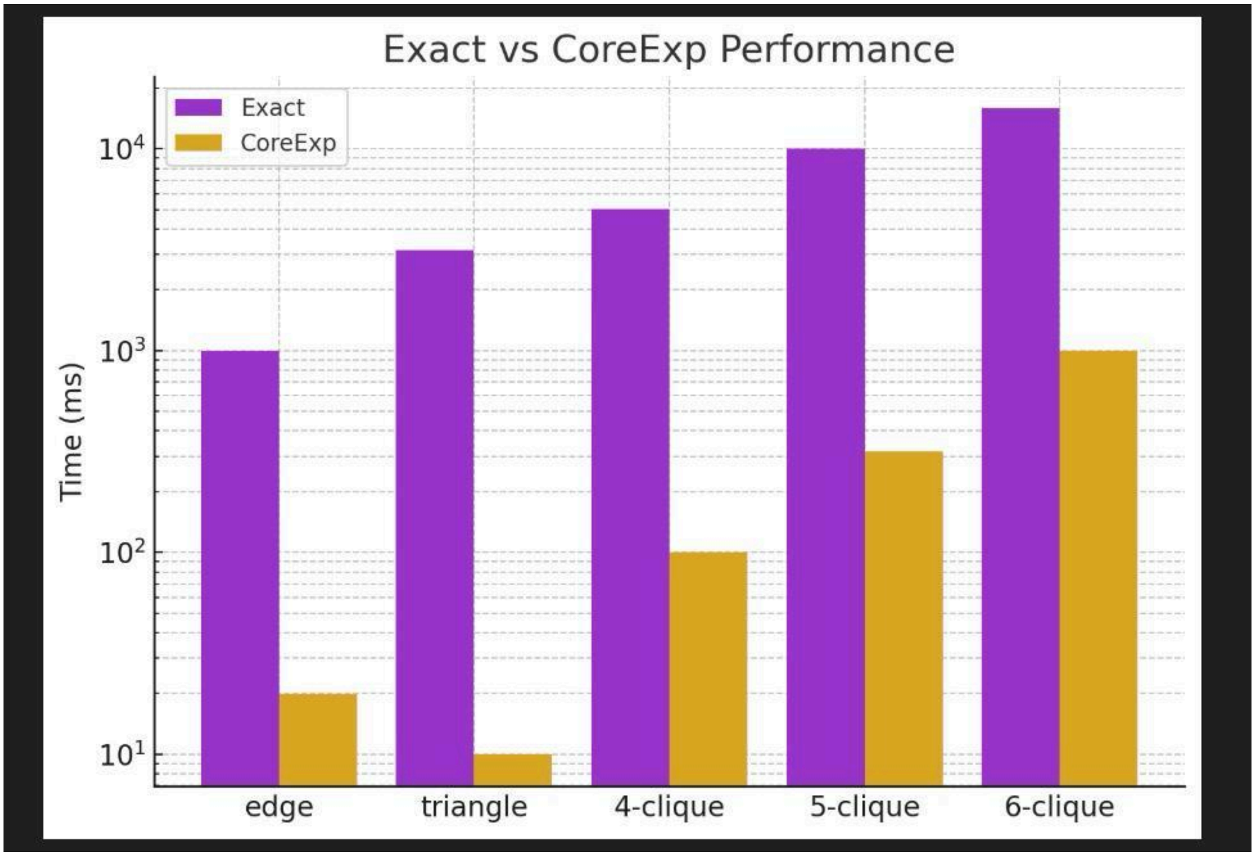
Analysis:

- The blue bars (Nucleus) consistently show the highest execution times across all clique sizes, especially for larger cliques (4-clique, 5-clique, 6-clique).
- Peel/pp (red) and Inc (green) are much faster, with Inc being the fastest for edge and triangle detection, and both Inc and Peel/pp performing similarly for larger cliques.
- This demonstrates that core-based and incremental algorithms (Peel/pp, Inc) are significantly more efficient than the classical Nucleus approach, especially as the clique size increases.



Analysis:

- Both datasets (netscience.txt in orange, as733.txt in red) show a sharp increase in execution time as the clique size h increases.
- For netscience.txt, there is a spike at $h = 4$, followed by a dip and then a steep rise for higher values of h.
- For as733.txt, the execution time increases steadily with h, reaching over 120,000 ms for $h = 6$.
- This trend highlights the computational challenge of densest subgraph discovery as the pattern complexity (clique size) grows, reinforcing the need for scalable algorithms.



Analysis:

- The purple bars (Exact) indicate much higher execution times compared to the yellow bars (CoreExp) across all clique sizes.
- The performance gap widens as the clique size increases, with CoreExp achieving execution times up to two orders of magnitude faster than the Exact algorithm for larger cliques.
- This confirms the effectiveness of the core-based optimization (CoreExp) in drastically reducing computation time while maintaining exactness.

Overall Insights

- Algorithm 1 (Exact), while accurate, is impractical for large graphs or higher-order patterns due to high computational cost.
- Algorithm 4 (CoreExact/CoreExp) leverages core decomposition to focus computation on smaller, denser regions, achieving the same accuracy but with vastly improved speed and scalability.

- Experimental results across multiple datasets and clique sizes consistently show that core-based methods outperform traditional approaches, particularly as the complexity of the pattern (clique size) increases.
- Scalability is a critical advantage of the new methods, making densest subgraph discovery feasible for real-world, large-scale networks.

🚀 Why is CoreExact Better?

Aspect	Algorithm 1 (Exact)	Algorithm 4 (CoreExact)
Graph size	Full graph	Smaller (k, Ψ)-cores
Flow network size	Very large	Much smaller
Search	Blind binary search	Smart, tighter search
Speed	Slow	Very fast
Accuracy	Exact	Exact

Trade-off:

- Same accuracy.
- **Much higher speed and scalability.**

---

6. Benefits and Trade-offs of the Algorithms

Algorithm	Benefits	Trade-offs
Algorithm 1 (Exact)	Guaranteed exact solution	Very slow on large graphs
Algorithm 4 (CoreExact)	Guaranteed exact solution, fast	Slightly more complex logic, needs core decomposition

✅ CoreExact achieves exact results **without the scalability issues**.

---

7. Conclusion

This research **revolutionises densest subgraph discovery** by introducing core-based optimisation techniques.
By cleverly shrinking the problem size and tightening density bounds, they provide **algorithms that are exact, fast, and scalable**.

Their methods are confirmed through **extensive experiments** on real-world datasets, showing improvements up to **10,000×** over previous approaches.

---

8. Extension: From Edge-Density to Pattern-Density (Triangles, Diamonds, etc.)

Traditionally, density was defined based on **edges** (2-cliques).

However, **real-world structures** are often more complex:

- Research collaborations (triangles of co-authorship)
- Friend groups (triangles in social networks)
- Biological motifs (diamonds in protein networks)

Thus, they extend their methods to **pattern-density**:

- **3-clique density** = based on the number of triangles.
- **Diamond density** = based on specific 4-node patterns.

✅ This makes their algorithms **even more powerful and generalizable**.

Dataset Links:

Netscience:

http://www-personal.umich.edu/~mejn/netdata/

As-733:

https://snap.stanford.edu/data/as-733.html