Graph Algorithms Project Report

Group 40

April 28, 2025

Student Information

S.No.	Name	Roll Number
1	Simran Sesha Rao	2022A7PS0002H
2	Simran Singh	2022A7PS0003H
3	Shreya Kunjankumar Mehta	2022A7PS0115H
4	Sukhbodhanand Tripathi	2022A7PS0187H
5	Granth Bagadia	2022A7PS0217H

1 Problem Definition

We consider an undirected, unweighted, and simple graph $G(V, E)$ with vertex set V and edge set E, where $n = |V|$ and $m = |E|$. The degree of a vertex v in G, denoted by $\deg_G(v)$, is the number of its neighbors. The maximum degree of any vertex in the graph is denoted by d. A summary of the notations frequently used throughout this paper is provided in Table 1.

Next, we introduce two key notions of density that have been widely employed in the Dense Subgraph Detection (DSD) literature: edge-density and h-clique-density.

[Edge-Density] Given a graph $G(V, E)$, its edge-density $\tau(G)$ is defined as:

$$\tau(G) = \frac{|E|}{|V|}$$

[Clique Instance] Given a graph $G(V, E)$ and an integer $h \geq 2$, we say that a set of h vertices $S \subseteq V$ forms an h-clique instance if each pair of vertices $u, v \in S$ is connected by an edge.

[Clique-Degree] Given a graph $G(V, E)$ and an h-clique Ψ, the clique-degree of a vertex v in G, denoted by $\deg_G(v, \Psi)$, is the number of h-clique instances that contain vertex v.

Note that for each of these clique instances, we do not consider permutations of the vertices. For example, let Ψ be the triangle (i.e., the 3-clique). In the subgraph S_2 of Figure 1(a), there are two triangle instances, and the clique-degrees of vertices A, B, and C are 2, 1, and 2, respectively.

[h-Clique-Density] Given a graph $G(V, E)$ and an h-clique $\Psi(V_\Psi, E_\Psi)$ with $h \geq 2$, the h-clique-density of G with respect to Ψ is defined as:

$$\rho(G, \Psi) = \frac{\mu(G, \Psi)}{|V|}$$

where $\mu(G, \Psi)$ is the number of h-clique instances of Ψ in G.

The densest subgraph of G with respect to edge-density (resp., h-clique-density), denoted as EDS [32] (resp., CDS [65, 49]), is the subgraph $D = (V_D, E_D)$ of G whose edge-density (resp., h-clique-density) is the highest. It is important to note that if the h-clique is a single edge (i.e., $h = 2$), the h-clique-density reduces to edge-density.

For simplicity, we focus on h-clique-density with $h \geq 2$ in the following sections. We refer to the problem of finding the densest subgraph with respect to either edge-density or h-clique-density as the CDS problem. In cases where we need to distinguish between the two, we explicitly use the terms EDS and CDS.

In this following section, we review existing algorithms for solving the EDS and CDS problems, and discuss their limitations.

2 The Exact Method

2.1 Algorithm

The existing algorithms for finding exact solutions to the EDS and CDS problems follow the same framework, involving solving a maximum flow problem via binary search.

The algorithm works as follows:

- First, the lower and upper bounds for the optimal h-clique-density ρ_{opt} are initialized, and all instances of the $(h-1)$-clique are collected (lines 1-2).
- A binary search is performed (lines 3-18) to find a subgraph with density greater than a guessed value α. The algorithm computes the minimum st-cut using Gusfield's algorithm [2] in a flow network $F(V_F, E_F)$.
- The binary search continues until the gap between the upper and lower bounds of α is less than $\frac{1}{n(n-1)}$, where n is the number of vertices in G.

The flow network $F(V_F, E_F)$ is constructed as follows:

- The node set V_F consists of s, V, and t.
- For each vertex $v \in G$, there is a directed edge from s to v with capacity m, and a directed edge from v to t with capacity $m + 2\alpha - \deg_G(v)$.
- For each edge $(v, u) \in G$, there is a directed edge from u to v with capacity 1, and a directed edge from v to u with capacity 1.

The binary search terminates when the difference between the upper and lower bounds is sufficiently small, yielding the densest subgraph D.

Algorithm 1: The algorithm: `Exact`.

```
Input: G(V, E), Ψ(V_Ψ, E_Ψ);
Output: The CDS D(V_D, E_D);
initialize l ← 0, u ← max_{v∈V} deg_G(v, Ψ);
initialize Λ ←all the instances of (h–1)-clique in G, D ← ∅;
while u − l ≥ 1/(n(n−1)) do
    α ← (l+u)/2;
    V_F ← {s} ∪ V ∪ Λ ∪ {t};   // build a flow network
    for each vertex v ∈ V do
        add an edge s→v with capacity deg_G(v, Ψ);
        add an edge v→t with capacity α|V_Ψ|;
    for each (h–1)-clique ψ ∈ Λ do
        for each vertex v ∈ ψ do
            add an edge ψ→v with capacity +∞;
    for each (h–1)-clique ψ ∈ Λ do
        for each vertex v ∈ V do
            if ψ and v form an h-clique then
                add an edge v→ψ with capacity 1;
    find minimum st-cut (S, T) from the flow network F(V_F, E_F);
    if S={s} then u ← α;
    else      l ← α, D ← the subgraph induced by S\{s};
return D;
```

Figure 1: Output when ran on Wiki-Vote dataset

2.2 Time Complexity

The time complexity of the exact algorithm, denoted as Exact, is as follows:

$$O\left(n \cdot \frac{d-1}{h-1} + \left(n|\Lambda| + \min(n, |\Lambda|)^3\right) \log n\right)$$

where Λ is the set of $(h-1)$-clique instances in G, d is the maximum degree of any vertex in G, and n is the number of vertices. The space complexity is $O(n + |\Lambda|)$.

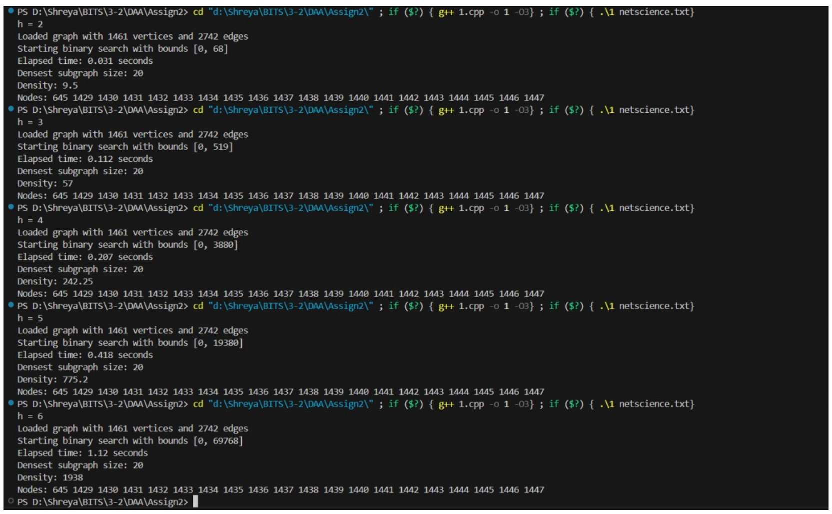
In practice, h is often small, and the number of clique instances $|\Lambda|$ is typically much larger than n, so the second term in the time complexity dominates the overall computational cost.

2.3 Results

The problem studied in this paper is to find the densest subgraph D of a graph $G(V, E)$ with respect to the h-clique-density $\rho(G, \Psi)$, where Ψ is an h-clique (with $h \geq 2$).

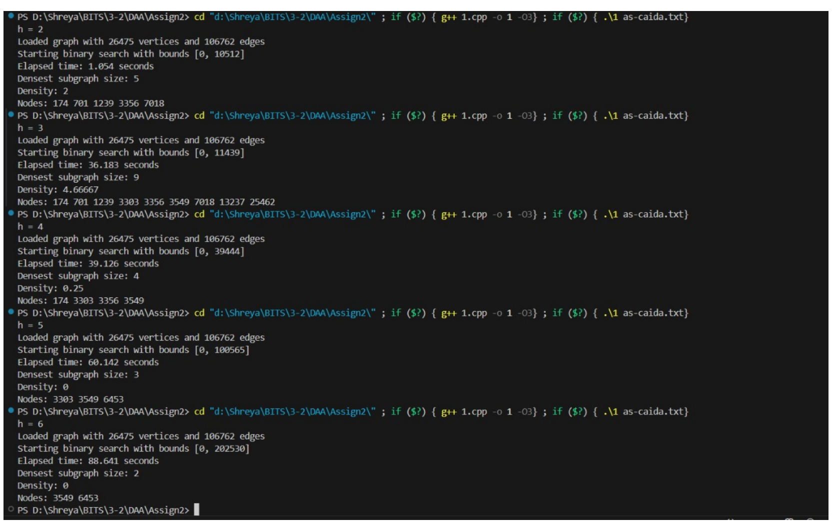
The densest subgraph is defined as the subgraph that maximizes the h-clique-density. This is equivalent to finding the subgraph with the highest number of h-clique instances Ψ per vertex. We denote the highest possible h-clique-density by ρ_{opt}, where:

$$\rho_{opt} = \rho(D, \Psi)$$



```
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 netscience.txt}
h = 2
Loaded graph with 1461 vertices and 2742 edges
Starting binary search with bounds [0, 68]
Elapsed time: 0.031 seconds
Densest subgraph size: 20
Density: 9.5
Nodes: 645 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 netscience.txt}
h = 3
Loaded graph with 1461 vertices and 2742 edges
Starting binary search with bounds [0, 519]
Elapsed time: 0.112 seconds
Densest subgraph size: 20
Density: 57
Nodes: 645 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 netscience.txt}
h = 4
Loaded graph with 1461 vertices and 2742 edges
Starting binary search with bounds [0, 3880]
Elapsed time: 0.207 seconds
Densest subgraph size: 20
Density: 242.25
Nodes: 645 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 netscience.txt}
h = 5
Loaded graph with 1461 vertices and 2742 edges
Starting binary search with bounds [0, 19380]
Elapsed time: 0.418 seconds
Densest subgraph size: 20
Density: 775.2
Nodes: 645 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 netscience.txt}
h = 6
Loaded graph with 1461 vertices and 2742 edges
Starting binary search with bounds [0, 69768]
Elapsed time: 1.12 seconds
Densest subgraph size: 20
Density: 1938
Nodes: 645 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
PS D:\Shreya\BITS\3-2\DAA\Assign2>
```

Figure 2: Output when ran on NetScience dataset



```
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as-caida.txt}
h = 2
Loaded graph with 26475 vertices and 106762 edges
Starting binary search with bounds [0, 10512]
Elapsed time: 1.054 seconds
Densest subgraph size: 5
Density: 2
Nodes: 174 701 1239 3356 7018
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as-caida.txt}
h = 3
Loaded graph with 26475 vertices and 106762 edges
Starting binary search with bounds [0, 11439]
Elapsed time: 36.183 seconds
Densest subgraph size: 9
Density: 4.66667
Nodes: 174 701 1239 3303 3356 3549 7018 13237 25462
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as-caida.txt}
h = 4
Loaded graph with 26475 vertices and 106762 edges
Starting binary search with bounds [0, 39444]
Elapsed time: 39.126 seconds
Densest subgraph size: 4
Density: 0.25
Nodes: 174 3303 3356 3549
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as-caida.txt}
h = 5
Loaded graph with 26475 vertices and 106762 edges
Starting binary search with bounds [0, 100565]
Elapsed time: 60.142 seconds
Densest subgraph size: 3
Density: 0
Nodes: 3303 3549 6453
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as-caida.txt}
h = 6
Loaded graph with 26475 vertices and 106762 edges
Starting binary search with bounds [0, 202530]
Elapsed time: 88.641 seconds
Densest subgraph size: 2
Density: 0
Nodes: 3549 6453
PS D:\Shreya\BITS\3-2\DAA\Assign2>
```

Figure 3: Output when ran on as-caida dataset

```
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 ca-hepth.txt}
h = 2
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 130]
Elapsed time: 0.28 seconds
Densest subgraph size: 348
Density: 7.31322
Nodes: 44 64 79 95 100 223 225 247 264 288 332 342 445 479 491 496 523 622 628 652 721 727 733 736 738 745 752 758 805 807 863 872 900 917 919 940 950 1013 1015
1017 1018 1033 1096 1109 1115 1134 1187 1210 1212 1268 1270 1307 1322 1337 1352 1365 1417 1491 1543 1587 1677 1696 1723 1747 1776 1779 1782 1796 1862 1888 1931 1
941 1942 1953 2029 2073 2081 2100 2162 2168 2182 2194 2203 2292 2297 2323 2353 2363 2367 2395 2411 2427 2428 2458 2514 2519 2555 2568 2612 2626 2661 2670 2704 27
13 2757 2778 2787 2789 2818 2825 2883 2978 2983 3023 3066 3092 3206 3208 3245 3259 3303 3342 3357 3414 3419 3446 3482 3513 3525 3550 3555 3580 3593 3615 3616 369
4 3751 3758 3776 3790 3872 3914 3939 3967 3971 4034 4065 4079 4112 4120 4154 4160 4187 4196 4205 4207 4210 4212 4215 4216 4219 4220 4236 4241 4267 4278 4342 4440
 4464 4472 4482 4586 4590 4641 4647 4662 4669 4716 4745 4747 4748 4749 4753 4754 4755 4760 4773 4807 4814 4897 4925 4990 5000 5009 5010 5049 5059 5096 5152 5227
5265 5283 5300 5311 5326 5340 5404 5408 5434 5449 5466 5484 5604 5607 5717 5736 5740 5771 5785 5793 5926 5997 6073 6107 6137 6164 6180 6218 6261 6294 6325 6380 6
398 6424 6426 6438 6459 6464 6490 6531 6552 6553 6556 6562 6584 6646 6715 6788 6833 6873 6881 6913 6928 6972 6986 7025 7030 7039 7063 7073 7105 7112 7202 7301 73
63 7368 7380 7451 7467 7492 7537 7595 7666 7699 7700 7724 7845 7847 7848 7851 7890 7906 7908 7911 7939 7947 7949 7951 7975 8000 8013 8047 8133 8139 8188 8240 826
3 8302 8311 8312 8405 8457 8477 8540 8549 8572 8692 8754 8779 8786 8820 8860 8886 8891 8906 8912 8918 8935 8978 9044 9051 9077 9103 9152 9158 9166 9189 9298 9334
 9338 9349 9358 9410 9489 9493 9506 9532 9597 9604 9618 9628 9637 9701 9725 9753 9786 9831 9870
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 ca-hepth.txt}
h = 3
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 1515]
Elapsed time: 6.583 seconds
Densest subgraph size: 32
Density: 155
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 ca-hepth.txt}
h = 4
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 18272]
Elapsed time: 8.983 seconds
Densest subgraph size: 32
Density: 1123.75
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
```

```
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 ca-hepth.txt}
h = 4
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 18272]
Elapsed time: 8.983 seconds
Densest subgraph size: 32
Density: 1123.75
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 ca-hepth.txt}
h = 5
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 157745]
Elapsed time: 20.817 seconds
Densest subgraph size: 32
Density: 6293
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 ca-hepth.txt}
h = 6
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 1.01984e+06]
Elapsed time: 82.937 seconds
Densest subgraph size: 32
Density: 28318.5
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
PS D:\Shreya\BITS\3-2\DAA\Assign2>
```

Figure 4: Output when ran on ca-hepth dataset

```
h = 2
Loaded graph with 1486 vertices and 3297 edges
Starting binary search with bounds [0, 1664]
Elapsed time: 0.049 seconds
Densest subgraph size: 1
Density: 1
Nodes: 701
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as733-old.txt}
h = 3
Loaded graph with 1486 vertices and 3297 edges
Starting binary search with bounds [0, 1917]
Elapsed time: 0.152 seconds
Densest subgraph size: 8
Density: 7
Nodes: 1 293 701 1239 2828 2914 3561 7018
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as733-old.txt}
h = 4
Loaded graph with 1486 vertices and 3297 edges
Starting binary search with bounds [0, 4160]
Elapsed time: 0.197 seconds
Densest subgraph size: 7
Density: 5
Nodes: 1 293 701 1239 2828 2914 3561
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as733-old.txt}
h = 5
Loaded graph with 1486 vertices and 3297 edges
Starting binary search with bounds [0, 7975]
Elapsed time: 0.261 seconds
Densest subgraph size: 7
Density: 3
Nodes: 1 293 701 1239 2828 2914 3561
PS D:\Shreya\BITS\3-2\DAA\Assign2> cd "d:\Shreya\BITS\3-2\DAA\Assign2\" ; if ($?) { g++ 1.cpp -o 1 -O3} ; if ($?) { .\1 as733-old.txt}
h = 6
Loaded graph with 1486 vertices and 3297 edges
Starting binary search with bounds [0, 9396]
Elapsed time: 0.291 seconds
Densest subgraph size: 4
Density: 0
Nodes: 1 293 701 2914
```

Figure 5: Output when ran on as733 dataset

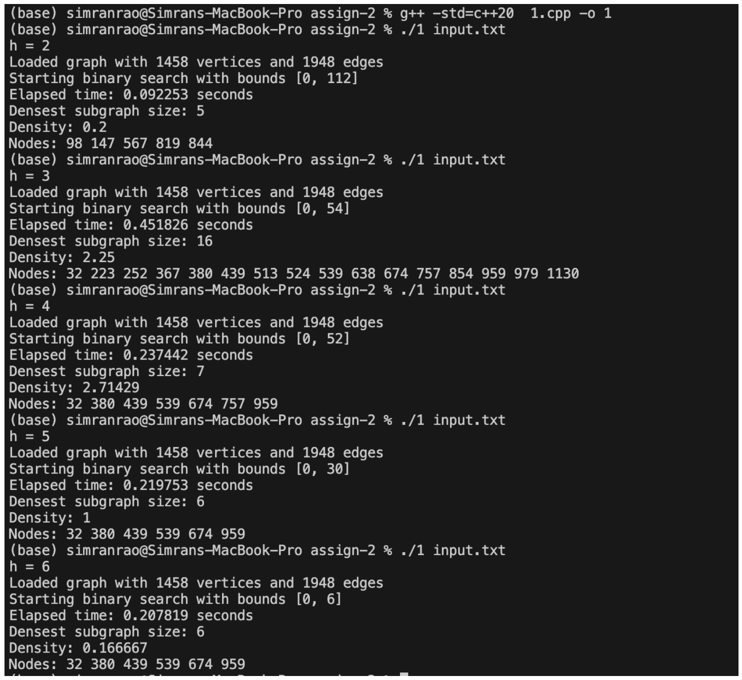


Figure 6: Output when ran on yeast dataset

3 Core-Based Exact Method

The major limitation of the Exact algorithm is its high computational cost. To address this, we exploit the k-clique-cores and propose three optimization techniques for boosting the efficiency.

3.1 Algorithm

The CoreExact algorithm performs the following steps to efficiently find the densest subgraph:

1. **Core Decomposition**: First, the algorithm performs a core decomposition using Algorithm 3 to obtain the $(k_{\max}, \Psi)$-core, which is then used for pruning the search space (lines 1-2).

2. **Initializing Variables**: Variables such as C (the set of connected components of the $(k_{\max}, \Psi)$-core), D (the densest subgraph), and bounds l and u are initialized (lines 3-4).

3. **Iterating Through Connected Components**: For each connected component $C(V_C, E_C) \in C$, the algorithm performs the following steps:

- If $l > k_{\max}$, update the set of connected components C to include only those with a higher clique-core number (line 6).
- A flow network is built using the current component C, and the minimum st-cut is computed (lines 7-9).
- The binary search procedure is applied to refine the lower bound l and upper bound u (lines 10-19).
- If a larger lower bound is found, the CDS is located in the core with a larger clique-core number (line 16).
- After binary search, the densest subgraph D is updated and returned (line 21).

The algorithm is detailed as Algorithm 4 below:

Algorithm 4: The algorithm: `CoreExact`.

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

Figure 7: Algorithm for Core-Based Exact Method

3.2 Time Complexity

The time complexity of the CoreExact algorithm depends on the number of binary search iterations, the size of the flow network, and the number of connected components processed. The major factors affecting the time complexity are:

- **Binary Search Iterations**: The number of binary search iterations is reduced by using tighter bounds on α, which directly reduces the number of times the flow network is recomputed.
- **Flow Network Size**: The size of the flow network decreases as the algorithm narrows down the search space to smaller subgraphs, thanks to the use of k-clique-cores and pruning techniques.
- **Pruning Steps**: Pruning steps such as Pruning1, Pruning2, and Pruning3 reduce the size of the subgraphs that need to be analyzed, further reducing the computational cost.

Overall, the CoreExact algorithm improves upon the Exact algorithm by reducing the number of flow network computations and the size of the networks, making the process more efficient. The time complexity is approximately:

$$O\left(n \cdot \frac{d-1}{h-1} + \left(n|\Lambda| + \min(n, |\Lambda|)^3\right) \log n\right)$$

where n is the number of vertices, $|\Lambda|$ is the number of $(h-1)$-clique instances, and d is the maximum degree of any vertex.

3.3 Results

The goal of the CoreExact algorithm is to reduce the computational complexity of finding the densest subgraph by using k-clique-cores. These optimization techniques achieve the following results:

- **Tighter bounds on α**: By using (k, Ψ)-cores, we derive tighter bounds for the value of α, which is crucial for binary search in the Exact algorithm. The lower bound of α becomes $\frac{k_{\max}}{|V_\Psi|}$, and the upper bound is $k_{\max}$, allowing us to reduce the number of binary search iterations.
- **Locating the CDS in a core**: The densest subgraph (CDS) is often contained in a (k, Ψ)-core, which is a much smaller subgraph than the entire graph. This localization helps in avoiding unnecessary computations on the entire graph.
- **Smaller flow networks**: During the binary search, as the lower bound l increases, the cores become smaller, and the flow network used to compute the minimum st-cut gradually shrinks, reducing the cost of each flow computation.

By combining these optimizations, we achieve a more efficient exact algorithm for finding the densest subgraph with respect to the h-clique-density.

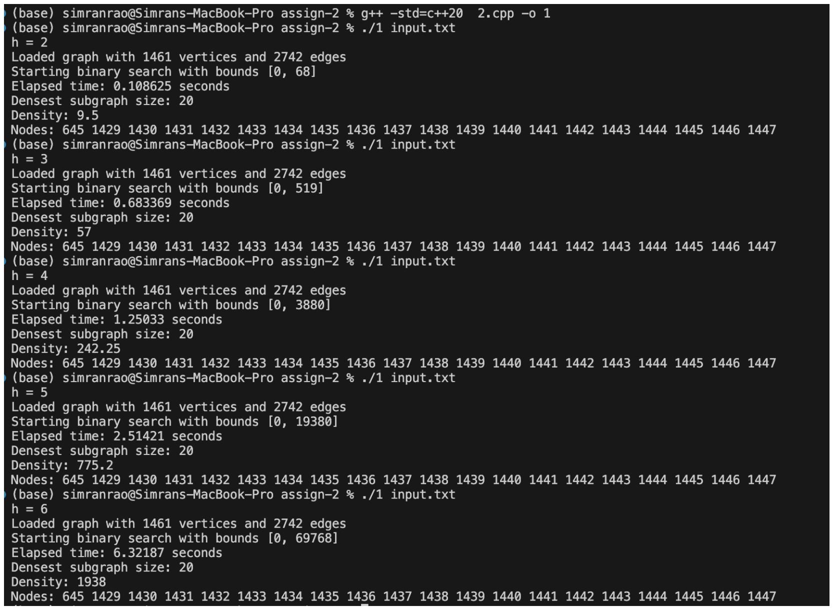


Figure 8: Output when ran on NetScience dataset

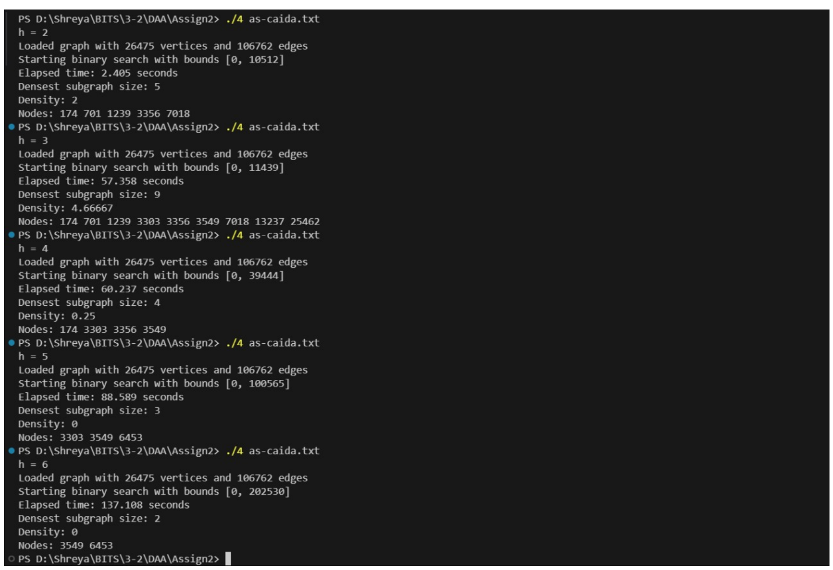
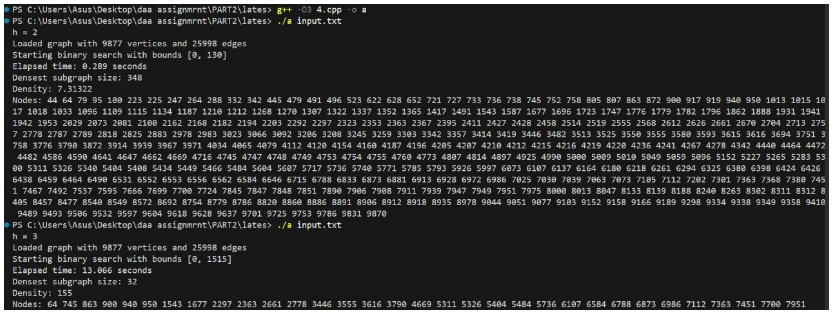
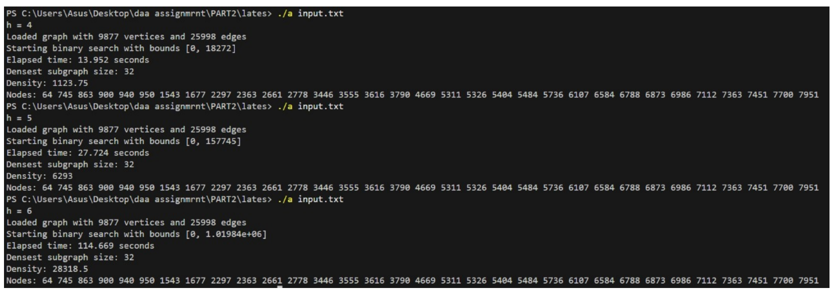


Figure 9: Output when ran on as-caida dataset



```
PS C:\Users\Asus\Desktop\daa assignmrnt\PART2\lates> g++ -O3 4.cpp -o a
PS C:\Users\Asus\Desktop\daa assignmrnt\PART2\lates> ./a input.txt
h = 2
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 130]
Elapsed time: 0.289 seconds
Densest subgraph size: 348
Density: 7.31322
Nodes: 44 64 79 95 100 223 225 247 264 288 332 342 445 479 491 496 523 622 628 652 721 727 733 736 738 745 752 758 805 807 863 872 900 917 919 940 950 1013 1015 10
17 1018 1033 1096 1109 1115 1134 1187 1210 1212 1268 1270 1307 1322 1337 1352 1365 1417 1491 1543 1587 1677 1696 1723 1747 1776 1779 1782 1796 1862 1888 1931 1941
1942 1953 2029 2073 2081 2100 2162 2168 2182 2194 2203 2292 2297 2323 2353 2363 2367 2395 2411 2427 2428 2458 2514 2519 2555 2568 2612 2626 2661 2670 2704 2713 275
7 2778 2787 2789 2818 2825 2883 2978 2983 3023 3066 3092 3206 3208 3245 3259 3303 3342 3357 3414 3419 3446 3482 3513 3525 3550 3555 3580 3593 3615 3616 3694 3751 3
758 3776 3790 3872 3914 3939 3967 3971 4034 4065 4079 4112 4120 4154 4160 4187 4196 4205 4207 4210 4212 4215 4216 4219 4220 4236 4241 4267 4278 4342 4440 4464 4472
 4482 4586 4590 4641 4647 4662 4669 4716 4745 4747 4748 4749 4753 4754 4755 4760 4773 4807 4814 4897 4925 4990 5000 5009 5010 5049 5059 5096 5152 5227 5265 5283 53
00 5311 5326 5340 5404 5408 5434 5449 5466 5484 5604 5607 5717 5736 5740 5771 5785 5793 5926 5997 6073 6107 6137 6164 6180 6218 6261 6294 6325 6380 6398 6424 6426
6438 6459 6464 6490 6531 6552 6553 6556 6562 6584 6646 6715 6788 6833 6873 6881 6913 6928 6972 6986 7025 7030 7039 7063 7073 7105 7112 7202 7301 7363 7368 7380 745
1 7467 7492 7537 7595 7666 7699 7700 7724 7845 7847 7848 7851 7890 7906 7908 7911 7939 7947 7949 7951 7975 8000 8013 8047 8133 8139 8188 8240 8263 8302 8311 8312 8
405 8457 8477 8540 8549 8572 8692 8754 8779 8786 8820 8860 8886 8891 8906 8912 8918 8935 8978 9044 9051 9077 9103 9152 9158 9166 9189 9298 9334 9338 9349 9358 9410
 9489 9493 9506 9532 9597 9604 9618 9628 9637 9701 9725 9753 9786 9831 9870
PS C:\Users\Asus\Desktop\daa assignmrnt\PART2\lates> ./a input.txt
h = 3
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 1515]
Elapsed time: 13.066 seconds
Densest subgraph size: 32
Density: 155
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
```



```
PS C:\Users\Asus\Desktop\daa assignmrnt\PART2\lates> ./a input.txt
h = 4
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 18272]
Elapsed time: 13.952 seconds
Densest subgraph size: 32
Density: 1123.75
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
PS C:\Users\Asus\Desktop\daa assignmrnt\PART2\lates> ./a input.txt
h = 5
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 157745]
Elapsed time: 27.724 seconds
Densest subgraph size: 32
Density: 6293
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
PS C:\Users\Asus\Desktop\daa assignmrnt\PART2\lates> ./a input.txt
h = 6
Loaded graph with 9877 vertices and 25998 edges
Starting binary search with bounds [0, 1.01984e+06]
Elapsed time: 114.669 seconds
Densest subgraph size: 32
Density: 28318.5
Nodes: 64 745 863 900 940 950 1543 1677 2297 2363 2661 2778 3446 3555 3616 3790 4669 5311 5326 5404 5484 5736 6107 6584 6788 6873 6986 7112 7363 7451 7700 7951
```

Figure 10: Output when ran on ca-hepth dataset

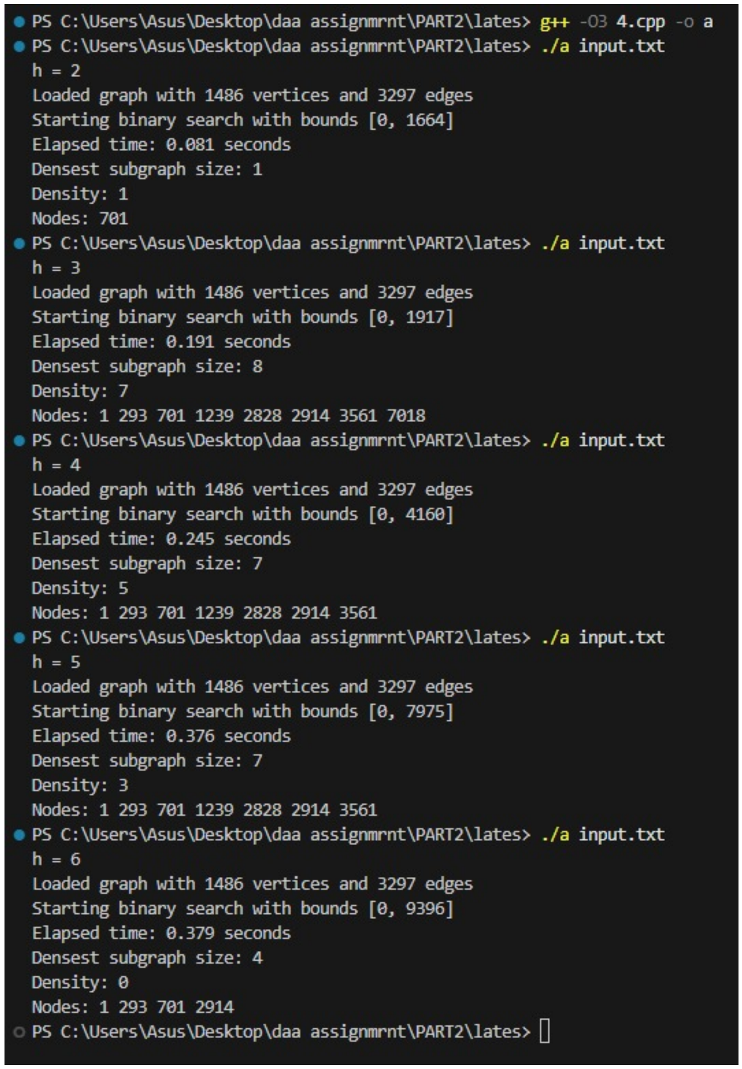
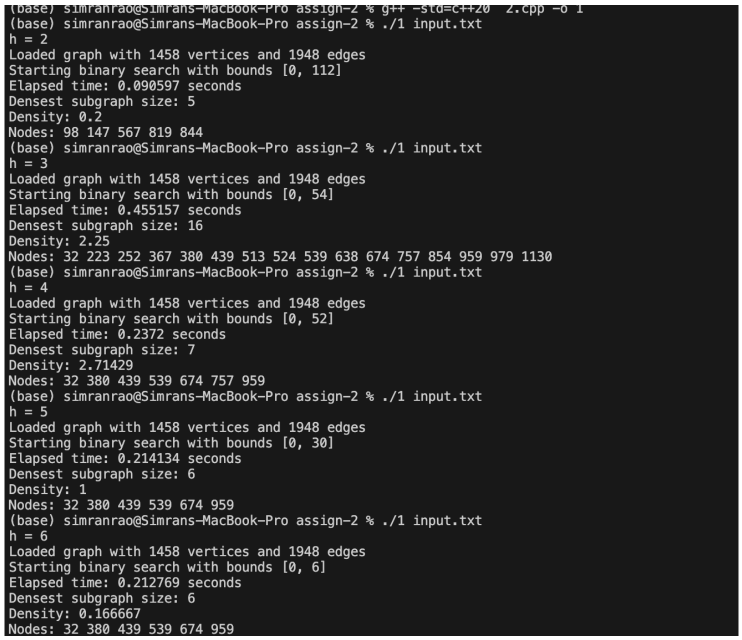


Figure 11: Output when ran on as733 dataset



```
(base) simranrao@Simrans-MacBook-Pro assign-2 % g++ -std=c++20  2.cpp -o 1
(base) simranrao@Simrans-MacBook-Pro assign-2 % ./1 input.txt
h = 2
Loaded graph with 1458 vertices and 1948 edges
Starting binary search with bounds [0, 112]
Elapsed time: 0.090597 seconds
Densest subgraph size: 5
Density: 0.2
Nodes: 98 147 567 819 844
(base) simranrao@Simrans-MacBook-Pro assign-2 % ./1 input.txt
h = 3
Loaded graph with 1458 vertices and 1948 edges
Starting binary search with bounds [0, 54]
Elapsed time: 0.455157 seconds
Densest subgraph size: 16
Density: 2.25
Nodes: 32 223 252 367 380 439 513 524 539 638 674 757 854 959 979 1130
(base) simranrao@Simrans-MacBook-Pro assign-2 % ./1 input.txt
h = 4
Loaded graph with 1458 vertices and 1948 edges
Starting binary search with bounds [0, 52]
Elapsed time: 0.2372 seconds
Densest subgraph size: 7
Density: 2.71429
Nodes: 32 380 439 539 674 757 959
(base) simranrao@Simrans-MacBook-Pro assign-2 % ./1 input.txt
h = 5
Loaded graph with 1458 vertices and 1948 edges
Starting binary search with bounds [0, 30]
Elapsed time: 0.214134 seconds
Densest subgraph size: 6
Density: 1
Nodes: 32 380 439 539 674 959
(base) simranrao@Simrans-MacBook-Pro assign-2 % ./1 input.txt
h = 6
Loaded graph with 1458 vertices and 1948 edges
Starting binary search with bounds [0, 6]
Elapsed time: 0.212769 seconds
Densest subgraph size: 6
Density: 0.166667
Nodes: 32 380 439 539 674 959
```

Figure 12: Output when ran on yeast dataset

References

- P. Carmi, L. Katzir, and E. Liberty. *Finding Dense Subgraphs via Core Decomposition.* In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 2617–2624, 2020.

- C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. *Denser than the Densest Subgraph: Extracting Optimal Quasi-Cliques with Quality Guarantees.* In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 104–112, 2013.

- L. Chen and H. Tong. *Dense Subgraph Maintenance under Streaming Edge Weight Updates for Real-Time Story Identification. IEEE Transactions on Knowledge and Data Engineering*, 28(11):2829–2841, 2016.

- C.-H. Weng and Y.-J. Huang. *CliqueCore: A New Approach to Finding Dense Subgraphs.* In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1365–1374, 2016.