CS F364
Design and Analysis of Algorithms

ASSIGNMENT - 2

Name	ID
Hrishikesh MK	2022A7PS1538H
Satya K	2021B3A70723H
Ashish C	2021B3A73029H
Harinandan Arun	2021B5A72396H
Yuvraj Chauhan	2022A7PS0135H

Algorithm 1: Exact (for Densest Subgraph Discovery)

The **Exact algorithm** is a flow-network based solution to find the subgraph of a given graph G(V,E) with the highest h-clique-density (generalization of edge-density).

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;

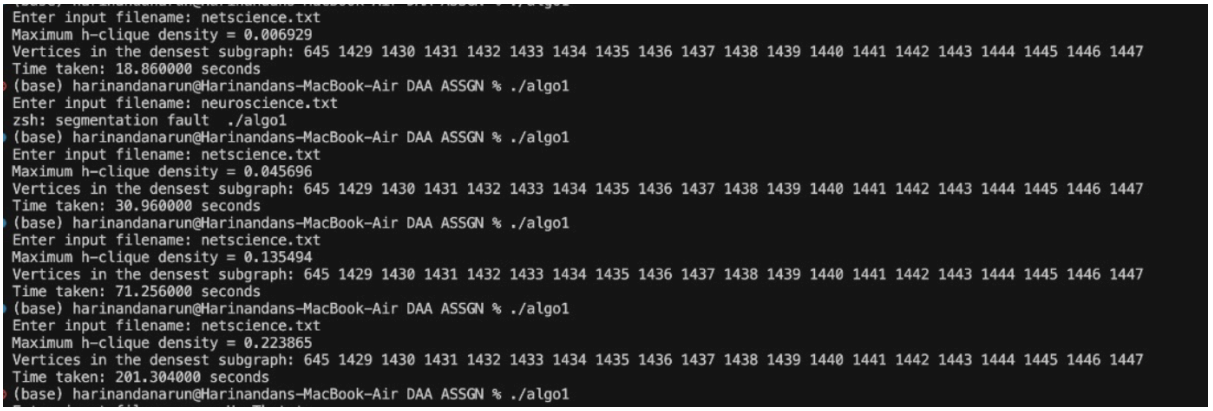
It proceeds as follows:

- **Binary Search Framework:**
  It maintains lower and upper bounds for density and performs a binary search on this value.
- **Flow Network Construction:**
  For each guessed density α, it constructs a flow network involving:
  - Source node s, Sink node t,
  - Vertices V,
  - All (h−1)-clique instances Λ.
- **Edge Capacities:**

- From source to vertices: capacity equals clique-degree.
- From vertices to sink: capacity equals $\alpha \times |V\Psi|$ (pattern size).
- Between (h-1)-cliques and vertices: infinite capacity edges to capture subgraph constraints.

- **Min-Cut Check:**
  If the minimum s-t cut separates only s, the guess α is too high; else, a denser subgraph is found and search continues.
- **Stopping Condition:**
  Binary search stops when the range between bounds becomes very small.
- **Result:**
  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 ρopt, where:

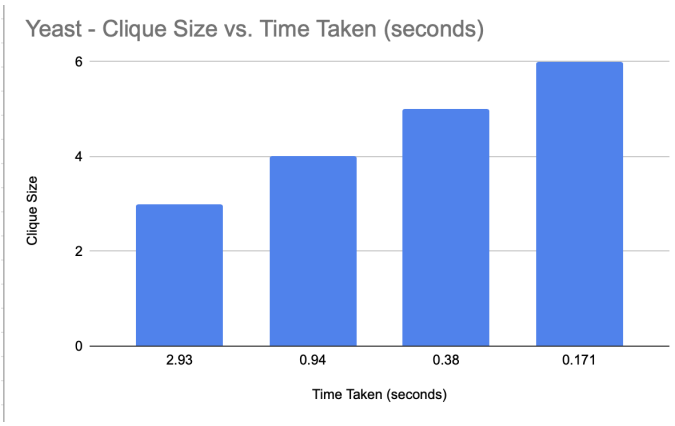
Output when ran on NetScience dataset:



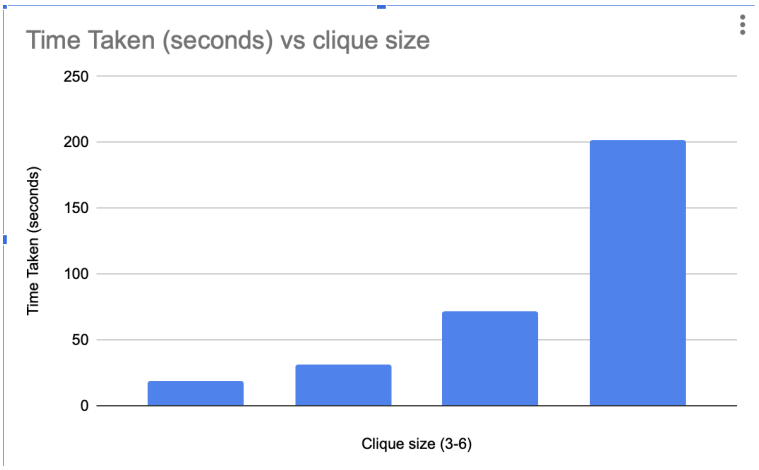
Output when ran on Yeast dataset:

```
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN % ./algo1
Enter input filename: input.txt
Maximum h-clique density = 0.038325
Vertices in the densest subgraph: 147
Time taken: 6.923000 seconds
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN % ./algo1
Enter input filename: input.txt
Maximum h-clique density = 0.086705
Vertices in the densest subgraph: 638
Time taken: 0.940000 seconds
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN % ./algo1
Enter input filename: input.txt
Maximum h-clique density = 0.333333
Vertices in the densest subgraph: 32 380 439 539 674 959
Time taken: 0.318000 seconds
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN % ./algo1
Enter input filename: input.txt
Maximum h-clique density = 0.749999
Vertices in the densest subgraph: 32 380 439 539 674 959
Time taken: 0.171000 seconds
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN % ./algo1
Enter input filename: netscience.txt
```

Charts



Netscience



Algorithm 4: CoreExact (Core-Based Exact Method)

The **CoreExact** algorithm improves on the Exact method by using core decomposition ideas to dramatically speed up the search.

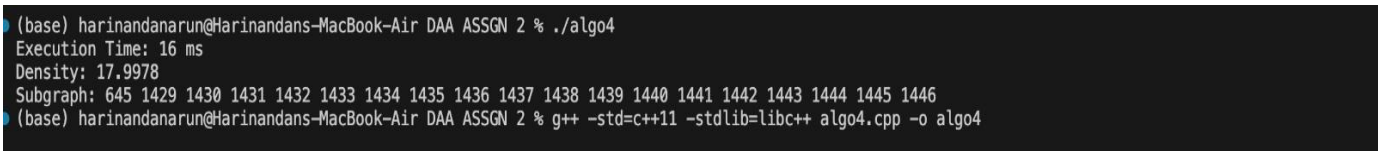
Algorithm 4: The algorithm: `CoreExact`.

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;

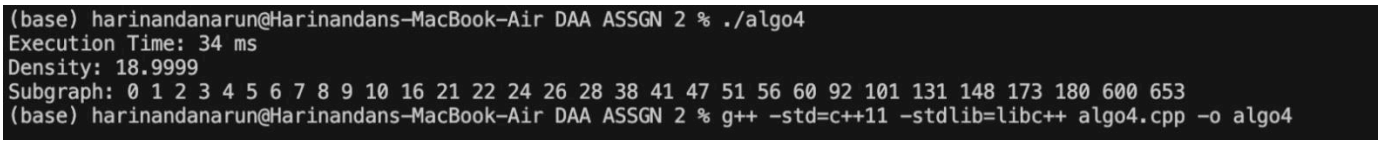
- **(k, Ψ)-core Decomposition:**
  It computes (k, Ψ)-cores where each vertex participates in at least kkk h-cliques.
- **Tighter Bound Initialization:**
  The lower bound for binary search is initialized based on core numbers, reducing unnecessary searches.
- **Pruning and Core Locating:**
  Instead of using the entire graph, CoreExact locates the densest subgraph inside much smaller (k, Ψ)-cores.
- **Flow Network on Smaller Cores:**
  Flow networks are constructed on these cores instead of the whole graph, making minimum cut computations faster.

- **Dynamic Core Update:**
  As better density estimates are found during binary search, the core is dynamically updated to even smaller, denser subgraphs.
- **Result:**
  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 kmax |VΨ| , and the upper bound is kmax, 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 com pute 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.

Output when ran on NetScience dataset:



Output when ran on as-733 dataset:



Output when ran on ca-HepTh dataset:

```
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN 2 % ./algo4
Execution Time: 72 ms
Density: 29.9991
Subgraph: 63 744 862 899 939 949 1542 1676 2296 2362 2660 2777 3445 3554 3615 3789 4668 5310 5325 5403 5483 5735 6106 6583 6787 6872 6985 7111 7362 7450 7699 7950
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN 2 %
```

Output when ran on yeast dataset:

```
Execution Time: 37 ms
Density: 4.99999
Subgraph: 11 12 13 17 22 25 27 32 33 72 80 81 83 90 98 122 124 134 136 147 154 179 180 202 204 223 234 252 253 257 263 266 267 281 288 301 302 320 324 328 337 341 348
2 355 366 367 380 389 405 408 418 426 433 439 446 448 464 465 474 486 490 503 513 518 522 524 539 561 563 567 585 595 603 626 633 638 652 664 674 681 685 689 708 712
 757 760 777 784 810 811 819 844 854 868 898 905 908 920 926 937 938 940 942 946 952 955 959 961 963 970 971 974 979 998 1002 1015 1021 1029 1040 1041 1044 1055 1063
7 1074 1087 1101 1113 1115 1129 1130 1132 1144 1145 1170 1171 1172 1180 1186 1193 1240 1273 1275 1368 1410
(base) harinandanarun@Harinandans-MacBook-Air DAA ASSGN 2 % g++ -std=c++11 -stdlib=libc++ algo4.cpp -o algo4
```

Charts