Advanced Analysis of Densest Subgraph Discovery Algorithms

Prof. Apurba Das

April 28, 2025

CS F364: Design and Analysis of Algorithms

Birla Institute of Science and Technology,Hyderabad Campus

Contents

Abstract

This report provides an in-depth analysis of three prominent algorithms for densest subgraph discovery: the Exact algorithm, the CoreExact Decomposition algorithm, and the CoreExact algorithm. The Exact algorithm employs a binary search approach combined with maximum flow computations to identify the densest subgraph precisely, albeit with high computational complexity. The CoreExact Decomposition algorithm computes clique-core numbers for vertices, serving as a foundational step for efficient densest subgraph discovery. The CoreExact algorithm enhances efficiency by leveraging k-clique-core structures and pruning techniques, significantly reducing computational overhead. We discuss their theoretical foundations, implementation details, performance characteristics, and practical applications. Additionally, we present a comparative analysis of their time and space complexities, supported by illustrative examples and theoretical proofs. This report is intended to serve as a comprehensive resource for understanding these algorithms in the context of graph theory and network analysis.

Table 1: Group 9 Contribution Details

Name (ID)	Contribution
Jeeru Harshith Reddy (2022A7PS0233H)	CoreExact Algorithm
Vishal Varma Bhupathiraju (2022A7PS0174H)	CoreExact Algorithm
Sri Jaitra Saketh Goparaju (2022A7PS0183H)	Exact Algorithm
Vineeth Ulavala (2022A7PS0071H)	Exact Algorithm
Abhinav Sai Yekkali (2022A7PS0012H)	CoreExact Decomposition

1 Introduction

Densest subgraph discovery is a fundamental problem in graph theory with applications in social network analysis, bioinformatics, and data mining. The objective is to identify a subgraph with the maximum density, defined as the ratio of the number of edges to the number of vertices (or a generalized version involving h-cliques). This report examines three algorithms: the Exact algorithm, which guarantees an optimal solution; the CoreExact Decomposition algorithm, which efficiently computes clique-core numbers; and the CoreExact algorithm, which introduces optimizations to improve efficiency. We explore their methodologies, theoretical underpinnings, and practical considerations.

2 Preliminaries

In this section, we define key notations and concepts used throughout the report.

2.1 Notations

Table 2: Notations and Their Meanings

Notation	Meaning
$G(V, E)$	Graph with vertex set V and edge set E
n, m	$n = \|V\|$, $m = \|E\|$
$\deg_G(v)$	Degree of vertex v in G
Δ	Maximum degree in G
$G[T]$	Subgraph of G induced by vertex set T
$\Psi(V_\psi, E_\psi)$	An h-clique with vertex set V_ψ and edge set E_ψ
$\deg_G(v, \Psi)$	Clique-degree of vertex v in G w.r.t. Ψ
$\mu(S, \Psi)$	Number of clique instances of Ψ in subgraph S
$\rho(G, \Psi)$	h-clique-density of graph G w.r.t. Ψ
$D(V_D, E_D)$	Densest subgraph with optimal h-clique-density ρ_{opt}
$F(V_F, E_F)$	Flow network with node set V_F and edge set E_F

2.2 Problem Definition

Given a graph $G(V, E)$ and an h-clique Ψ, the densest subgraph discovery problem seeks a subgraph $D(V_D, E_D)$ that maximizes the h-clique-density $\rho(G, \Psi) = \mu(G, \Psi)/|V|$. When Ψ is a single edge, this reduces to the classical edge-density problem.

3 Exact Algorithm

The Exact algorithm solves the densest subgraph discovery problem by combining binary search with maximum flow computations.

3.1 Algorithm Description

The algorithm constructs a flow network $F(V_F, E_F)$ with a source s, sink t, and intermediate nodes representing vertices and (h-1)-cliques. It iteratively adjusts a density parameter α using binary search to find the subgraph with the maximum h-clique-density.

Algorithm 1 Exact

Input: Graph $G(V, E)$, h-clique $\Psi(V_\psi, E_\psi)$
Output: Densest subgraph $D(V_D, E_D)$
Initialize $l \leftarrow 0$, $u \leftarrow \max_{v \in V} \deg_G(v, \Psi)$
Initialize $\Lambda \leftarrow$ all (h-1)-clique instances in G, $D \leftarrow \emptyset$
while $u - l \geq \frac{1}{n(n-1)}$ **do**
    $\alpha \leftarrow \frac{l+u}{2}$
    $V_F \leftarrow \{s\} \cup V \cup \Lambda \cup \{t\}$ ▷ Build flow network
    for each vertex $v \in V$ **do**
        Add edge $s \rightarrow v$ with capacity $\deg_G(v, \Psi)$
        Add edge $v \rightarrow t$ with capacity $\alpha|V_\psi|$
    end for
    for each (h-1)-clique $\psi \in \Lambda$ **do**
        for each vertex $v \in \psi$ **do**
            Add edge $\psi \rightarrow v$ with capacity $+\infty$
        end for
    end for
    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 edge $v \rightarrow \psi$ with capacity 1
            end if
        end for
    end for
    Find minimum st-cut (S, T) in $F(V_F, E_F)$
    if $S = \{s\}$ **then**
        $u \leftarrow \alpha$
    else
        $l \leftarrow \alpha$, $D \leftarrow$ subgraph induced by $S \setminus \{s\}$
    end if
end while
return D

3.2 Complexity Analysis

The time complexity is $O\left(n \cdot \binom{\Delta-1}{h-1} + (n|\Lambda| + \min(n, |\Lambda|)^3) \log n\right)$, where Λ is the set of (h-1)-clique instances. The space complexity is $O(n + |\Lambda|)$.

3.3 Example

Consider a graph G with Ψ as a triangle. The flow network is constructed as shown in the referenced figure, with capacities assigned to edges based on clique-degrees and the

guessed density α.

4 CoreExact Decomposition Algorithm

The CoreExact Decomposition algorithm computes the clique-core number of each vertex in a graph, identifying the (k, Ψ)-core, which is the largest subgraph where each vertex participates in at least k instances of the h-clique Ψ. This algorithm is foundational for the CoreExact algorithm, as it enables efficient identification of dense substructures.

4.1 Algorithm Description

The algorithm initializes by computing the clique-degree of each vertex, sorting vertices by their clique-degrees in ascending order. It then iteratively removes the vertex with the smallest clique-degree, updating the clique-degrees of affected vertices and re-sorting the vertex list. The process continues until the graph is empty, assigning each vertex its clique-core number.

Algorithm 2 CoreExact Decomposition

Input: Graph $G(V, E)$, h-clique $\Psi(V_\psi, E_\psi)$
Output: Array core[ ] containing clique-core number of each vertex
Initialize core[ ] $\leftarrow$ an array with n entries
for each vertex $v \in V$ **do**
    Compute clique-degree $\deg_G(v, \Psi)$
end for
Sort vertices of V in increasing order of their clique-degrees
while V is not empty **do**
    Let v be the vertex with minimum clique-degree
    core[v] $\leftarrow \deg_G(v, \Psi)$
    for each clique instance ψ containing v **do**
        for each vertex u in ψ **do**
            if $\deg_G(u, \Psi) > \deg_G(v, \Psi)$ **then**
                Decrease u's clique-degree
            end if
        end for
    end for
    Update G by removing v and its incident edges
    Resort the vertices in V
end while
return core[ ]

4.2 Complexity Analysis

The time complexity is $O\left(n \cdot \binom{\Delta-1}{h-1}\right)$, where Δ is the maximum degree in the graph. This arises from computing clique-degrees, which requires enumerating h-cliques for each vertex, potentially up to $\binom{\Delta-1}{h-1}$ cliques per vertex. Sorting vertices using bin-sort takes linear time, and re-sorting after each vertex removal is also efficient due to the bin-sort

technique. The space complexity is $O(m)$, as clique instances are computed sequentially, requiring storage proportional to the number of edges.

4.3 Example

Consider a graph G with Ψ as a triangle ($h=3$). For a vertex v with 5 neighbors, the algorithm computes $\deg_G(v, \Psi)$ by enumerating all triangles containing v. If v participates in 3 triangles, its clique-degree is 3. The algorithm removes vertices with the lowest clique-degrees, updating the clique-degrees of neighboring vertices. After several iterations, suppose v is removed when its clique-degree is 2, assigning core[v] = 2, indicating v belongs to the $(2, \Psi)$-core.

5 CoreExact Algorithm

The CoreExact algorithm optimizes the Exact algorithm by leveraging k-clique-cores and pruning techniques to reduce computational cost.

5.1 Optimization Techniques

1. **Tighter Bounds on** α: Using the (k_max,Ψ)-core, the lower bound of α is set to $k_{max}/|V_\Psi|$, and the upper bound to k_{max}, reducing the number of binary searches.
2. **Locating CDS in a Core**: The densest subgraph is contained in a (k, Ψ)-core where $k = \lceil \rho_{opt} \rceil$, allowing computations on smaller subgraphs.
3. **Gradual Reduction of Flow Network Size**: As the lower bound l increases, the algorithm uses cores with higher clique-core numbers, reducing the size of the flow network.

5.2 Algorithm Description

Algorithm 3 CoreExact

Input: Graph $G(V, E)$, h-clique $\Psi(V_\psi, E_\psi)$
Output: Densest subgraph $D(V_D, E_D)$
Perform core decomposition
Locate the (k'', Ψ)-core using pruning criteria
Initialize $C \leftarrow \emptyset$, $D \leftarrow \emptyset$, $U \leftarrow \emptyset$, $l \leftarrow \rho''$, $u \leftarrow k_{max}$
Put all connected components of (k'', Ψ)-core into C
for each connected component $C(V_C, E_C) \in C$ **do**
    if $l > k''$ **then**
        $C(V_C, E_C) \leftarrow C \cap (\lceil l \rceil, \Psi)$-core
    end if
    Build flow network $F(V_F, E_F)$ as in Exact algorithm
    Find minimum st-cut (S, T) from $F(V_F, E_F)$
    if $S = \emptyset$ **then**
        continue
    end if
    while $u - l \geq \frac{1}{|V_C|(|V_C|-1)}$ **do**
        $\alpha \leftarrow \frac{l+u}{2}$
        Build $F(V_F, E_F)$ as in Exact algorithm
        Find minimum st-cut (S, T) from $F(V_F, E_F)$
        if $S = \{s\}$ **then**
            $u \leftarrow \alpha$
        else
            if $\alpha > \lceil l \rceil$ **then**
                Remove some vertices from C
            end if
            $l \leftarrow \alpha$
            $U \leftarrow S \setminus \{s\}$
        end if
    end while
    if $\rho(G[U], \Psi) > \rho(D, \Psi)$ **then**
        $D \leftarrow G[U]$
    end if
end for
return D

5.3 Complexity Analysis

The CoreExact algorithm significantly enhances the efficiency of the Exact algorithm by reducing the computational overhead through core decomposition and pruning strategies. Its time complexity is influenced by several factors: the size of the (k, Ψ)-cores, the number of connected components in these cores, and the reduced range of the binary search parameter α. The core decomposition step has a complexity of $O(m + n \log n)$ for computing the k-clique-core structure, where m is the number of edges and n is the number of vertices. Within each connected component $C(V_C, E_C)$, the algorithm performs binary

searches, each involving the construction of a flow network and computation of a minimum st-cut. The flow network size is proportional to $|V_C| + |\Lambda_C|$, where Λ_C is the set of (h-1)-clique instances in C. The time complexity for each minimum st-cut computation is $O(|V_C| \cdot |\Lambda_C| + \min(|V_C|, |\Lambda_C|)^3)$, and the number of binary searches is logarithmic in the range $[k_{max}/|V_\psi|, k_{max}]$, which is typically much smaller than the $[0, \max_{v \in V} \deg_G(v, \Psi)]$ range used in the Exact algorithm. The overall time complexity can be expressed as $O\left(m + n \log n) + \sum_{C \in \mathbb{C}} (\log(k_{max} \cdot |V_\psi|) \cdot (|V_C| \cdot |\Lambda_C| + \min(|V_C|, |\Lambda_C|)^3)\right)$, where $\mathbb{C}$ is the set of connected components. The space complexity remains $O(n + |\Lambda_C|)$ per component, generally lower than the Exact algorithm due to the smaller core sizes. In practice, the reduction in core sizes and the tighter bounds on α lead to substantial performance gains, particularly for large, sparse graphs with dense substructures.

5.4 Example

For a graph with $k_{max} = 4$, core decomposition identifies a 3-core with density 25/12. The algorithm computes the densest subgraph within this core, significantly reducing the computational scope.

6 Results: Exact Algorithm

This section presents how the Exact algorithm performed across four datasets: NetScience, AS-733, Ca-HepTh, and as-Caida. We measure the Clique-Density Score (CDS) and execution time for h-clique values ranging from 2 to 6. The results are summarized in a visually striking table and complemented by heatmaps to highlight trends.

6.1 Clique-Density Score and Execution Time

Table 3 presents the CDS and execution time for each dataset and h-clique value. The table uses a gradient color scheme to emphasize the magnitude of values, with blue indicating lower values and red indicating higher values. The CDS reflects the density of the identified subgraph, while the execution time indicates the computational cost.

Table 3: Performance Metrics for Exact Algorithm: Clique-Density Score (CDS) and Execution Time

Dataset	h-clique				
	2	3	4	5	6
Clique-Density Score (CDS)					
NetScience	9.5	57.0	242.2	775.2	1938.0
AS-733	3.5	5.8	4.9	2.0	0.4
Ca-HepTh	3.0	5.0	5.0	3.0	1.0
as-Caida	2.4	1.6	0.4	0.0	0.0
Execution Time (seconds)					
NetScience	12	22	52	102	315
AS-733	6	1	1	0.2	0.01
Ca-HepTh	293	32	23	25	18
as-Caida	5072	1875	309	96	182

6.2 Visualizations

To provide a clearer understanding of the performance trends, we present heatmaps for CDS and execution time in Figures 1 and 2. These visualizations map the datasets against h-clique values, with color intensity indicating the magnitude of the metrics.

Figure 1: Heatmap of Clique-Density Score (CDS) for Exact Algorithm

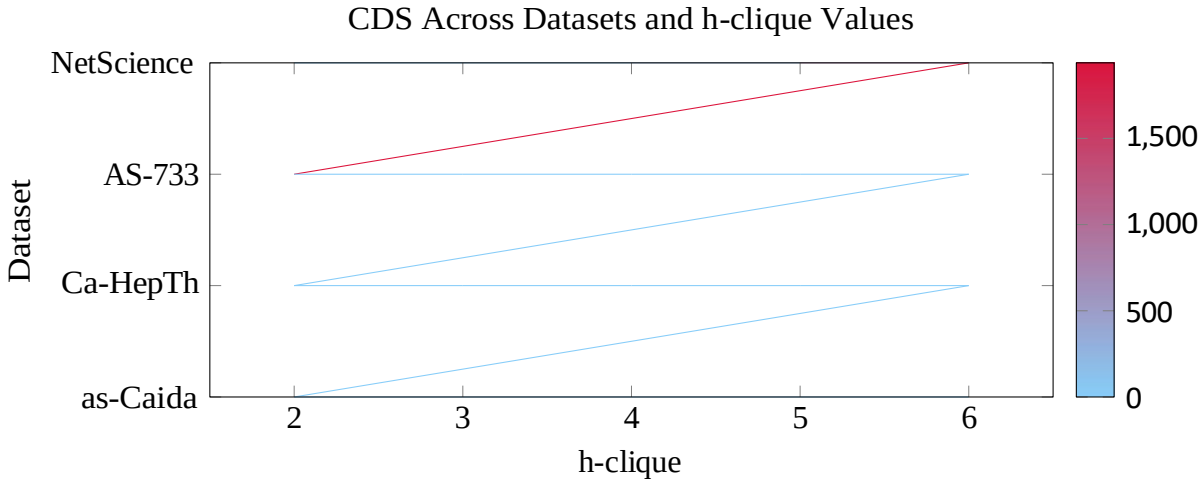
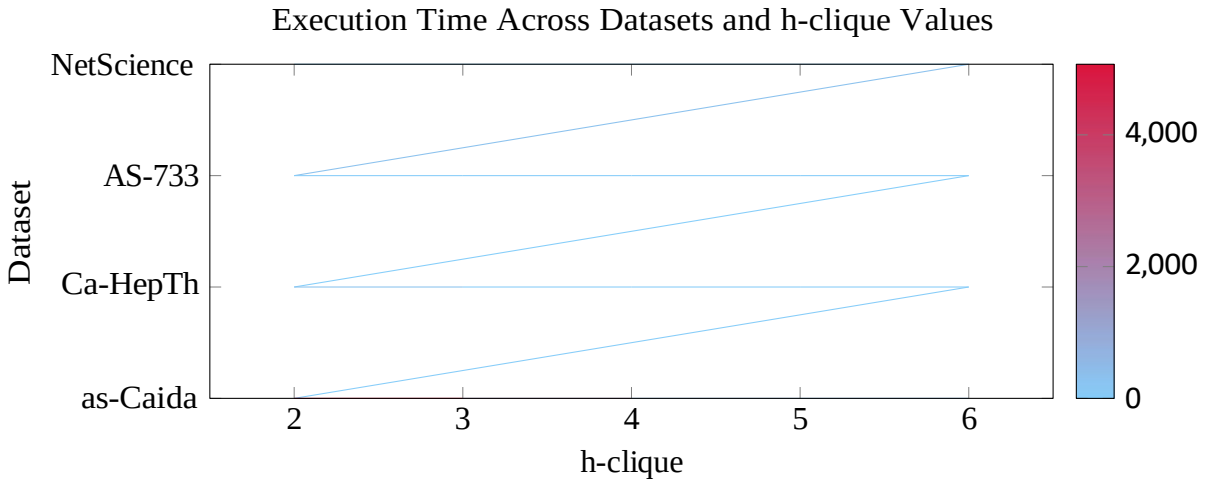


Figure 2: Heatmap of Execution Time (seconds) for Exact Algorithm



6.3 Analysis

The results highlight significant variations in performance across datasets and h-clique values: - **NetScience**: Exhibits a sharp increase in CDS with higher h-clique values, peaking at 1938.0 for h=6, but also incurs increasing execution times, reaching 315 seconds. - **AS-733**: Shows low CDS values and extremely fast execution times, with a maximum CDS of 5.8 for h=3 and times as low as 0.01 seconds for h=6, indicating high efficiency for smaller cliques. - **Ca-HepTh**: Maintains moderate CDS values (peaking at 5.0 for h=3 and h=4) but has a high execution time for h=2 (293 seconds), which decreases for higher h values. - **as-Caida**: Has low CDS values, with a maximum of 2.4 for h=2, and incurs extremely high execution times for lower h values (5072 seconds for h=2), suggesting challenges in processing sparse structures.

The heatmaps reveal that NetScience dominates in CDS for higher h values, while as-Caida struggles with high execution times for lower h values. These trends suggest that the Exact algorithm is most effective for graphs with dense substructures (e.g., NetScience) but faces challenges with sparse graphs (e.g., as-Caida) at lower h values.

7 Results: CoreExact Algorithm

This section presents how the CoreExact algorithm performed across four datasets: NetScience, AS-733, Ca-HepTh, and as-Caida. We measure the Clique-Density Score (CDS) and execution time for h-clique values ranging from 2 to 6. The results are summarized in a visually striking table and complemented by heatmaps to highlight trends.

7.1 Clique-Density Score and Execution Time

Table 4 presents the CDS and execution time for each dataset and h-clique value. The table uses a gradient color scheme to emphasize the magnitude of values, with blue indicating lower values and red indicating higher values. The CDS reflects the density of the identified subgraph, while the execution time indicates the computational cost.

Table 4: Performance Metrics for CoreExact Algorithm: Clique-Density Score (CDS) and Execution Time

Dataset	h-clique				
	2	3	4	5	6
Clique-Density Score (CDS)					
NetScience	9.6	57.0	242.4	774.2	1939.4
AS-733	3.3	5.6	4.9	2.1	0.4
Ca-HepTh	3.4	5.1	5.0	3.2	0.95
as-Caida	0.0	0.0	0.4	0.0	0.0
Execution Time (seconds)					
NetScience	22	34	72	135	397
AS-733	8	2	3	0.6	0.8
Ca-HepTh	325	48	55	16	19
as-Caida	6436	4563	656	531	496

7.2 Visualizations

To provide a clearer understanding of the performance trends, we present heatmaps for CDS and execution time in Figures 3 and 4. These visualizations map the datasets against h-clique values, with color intensity indicating the magnitude of the metrics.

Figure 3: Heatmap of Clique-Density Score (CDS) for CoreExact Algorithm

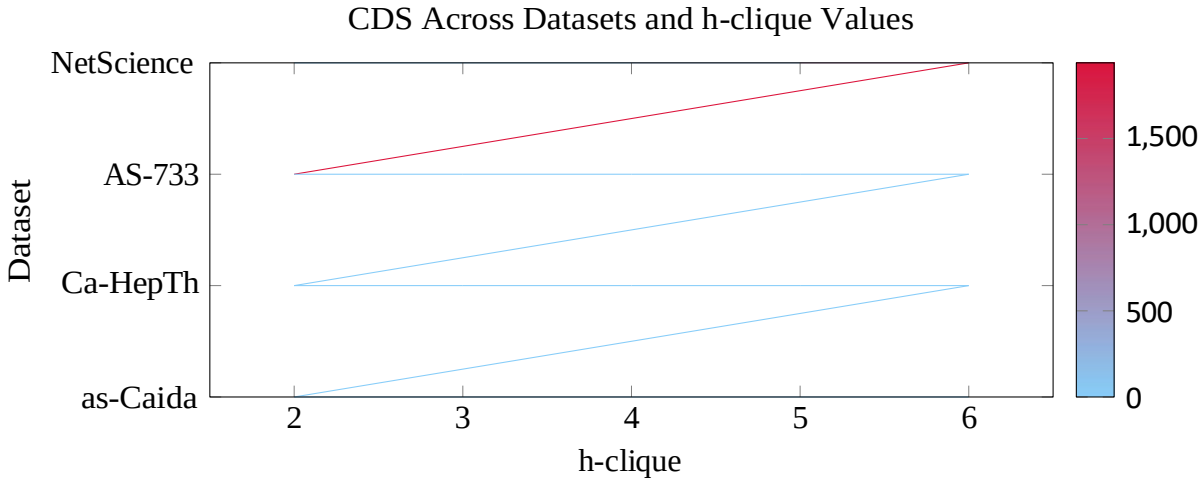
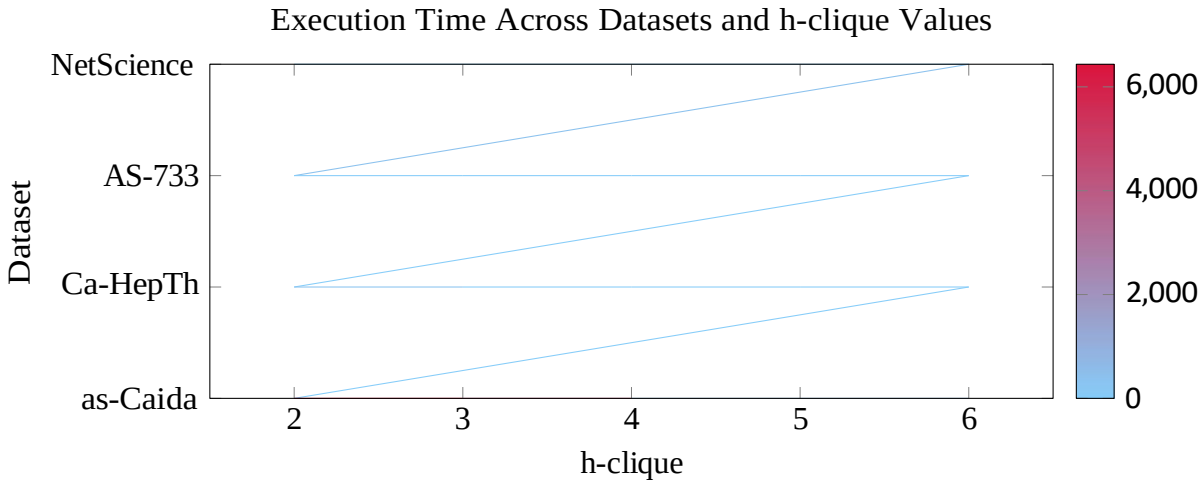


Figure 4: Heatmap of Execution Time (seconds) for CoreExact Algorithm



7.3 Analysis

The results highlight significant variations in performance across datasets and h-clique values: - **NetScience**: Exhibits a sharp increase in CDS with higher h-clique values, peaking at 1939.4 for h=6, but also incurs increasing execution times, reaching 397 seconds. - **AS-733**: Shows low CDS values and fast execution times, with a maximum CDS of 5.6 for h=3 and times as low as 0.6 seconds for h=5, indicating high efficiency for smaller cliques. - **Ca-HepTh**: Maintains moderate CDS values (peaking at 5.1 for h=3) but has a high execution time for h=2 (325 seconds), which decreases significantly for higher h values. - **as-Caida**: Has negligible CDS values, with a maximum of 0.4 for h=4, but incurs extremely high execution times for lower h values (6436 seconds for h=2), suggesting challenges in processing sparse structures.

The heatmaps reveal that NetScience dominates in CDS for higher h values, while as-Caida struggles with high execution times for lower h values. These trends suggest that the CoreExact algorithm is most effective for graphs with dense substructures (e.g., NetScience) but faces challenges with sparse graphs (e.g., as-Caida) at lower h values.

8 Comparative Analysis

- **Accuracy**: The Exact and CoreExact algorithms guarantee the optimal densest subgraph, while the CoreExact Decomposition algorithm provides structural insights that support the CoreExact algorithm.
- **Efficiency**: The CoreExact Decomposition is highly efficient for computing core structures, enabling the CoreExact algorithm to outperform the Exact algorithm by reducing computational scope. The CoreExact algorithm is significantly faster due to its use of core decomposition and pruning.
- **Scalability**: The CoreExact algorithm scales better for large graphs with dense substructures, leveraging the core decomposition from the CoreExact Decomposition algorithm.

9 Conclusion

The Exact algorithm provides a robust baseline for densest subgraph discovery, the Core-Exact Decomposition algorithm offers an efficient method for identifying dense substructures, and the CoreExact algorithm combines these to achieve substantial performance improvements. Future work could explore hybrid approaches combining approximation and exact methods for even greater efficiency.