Approximate Anchored Densest Subgraph Search on Large Static and Dynamic Graphs

Qi Zhang[‡]
University of Science and Technology Beijing
Beijing, China
qizhangcs@163.com

Yalong Zhang
Beijing Institute of Technology
Beijing, China
yalong-zhang@qq.com

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@gmail.com

ABSTRACT

Densest subgraph search, aiming to identify a subgraph with maximum edge density, faces limitations as the edge density inadequately reflects biases towards a given vertex set R. To address this, the R-subgraph density was introduced, refining the doubled edge density by penalizing vertices in a subgraph but not in R, using the degree as a penalty factor. This advancement leads to the Anchored Densest Subgraph (ADS) search problem, which finds the subgraph $\hat{S}$ with the highest R-subgraph density for a given set R. Nonetheless, current algorithms for ADS search face significant inefficiencies in handling large-scale graphs or the sizable R set. Furthermore, these algorithms require re-computing the ADS whenever the graph is updated, complicating the efficient maintenance within dynamic graphs. To tackle these challenges, we propose the concept of integer R-subgraph density and study the problem of finding a subgraph $S^* \subseteq V$ with the highest integer R-subgraph density. We reveal that the R-subgraph density of S^* provides an additive approximation to that of ADS with a difference of less than 1, and hence S^* is termed the Approximate Anchored Densest Subgraph (AADS). For searching the AADS, we present an efficient global algorithm incorporating the re-orientation network flow technique and binary search, operating in a time polynomial to the graph's size. Additionally, we propose a novel local algorithm using shortest-path-based methods for the max-flow computation from s to t around R, markedly boosting performance in scenarios with larger R sets. For dynamic graphs, both basic and improved algorithms are developed to efficiently maintain the AADS when an edge is updated. Extensive experiments and a case study demonstrate the efficiency, scalability, and effectiveness of our solutions.

PVLDB Reference Format:
Qi Zhang[‡], Yalong Zhang, Rong-Hua Li, and Guoren Wang. Approximate Anchored Densest Subgraph Search on Large Static and Dynamic Graphs. PVLDB, 18(3): 623-636, 2024. doi:10.14778/3712221.3712230

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/qizhang1996/aads.

1 INTRODUCTION

A graph, denoted by $G = (V, E)$, where vertices represent entities and edges represent relationships, serves as a fundamental model for various complex real-world networks [1, 2, 35, 41, 51]. Densest subgraph search, a central problem in graph analysis, aims to identify a subgraph S with the maximum edge density, which is defined as the ratio of the number of edges to the number of vertices, i.e., $\frac{|E(S)|}{|V(S)|}$ [8, 9, 15, 29, 38, 54, 61, 62]. This problem has unveiled their extensive applicability across various domains, such as identifying dense structures in scientific collaborations [48], biological networks [28, 57], and social networks [33, 45, 53]. Furthermore, this investigation has significantly advanced graph database, notably in the areas of query processing [19, 37] and visualization [68, 71].

Densest subgraph search based on the edge density identifies a globally densest subgraph, which often does not meet the requirements of practical applications. In many cases, users require a locally cohesive subgraph that not only contains an anchor vertex set A but is also biased toward a specific reference vertex set R. For example, in a co-purchasing network, each vertex represents a product, and an edge between two vertices denotes the co-purchasing of those items. Given the set of items R that the user has browsed and the set of items A that have been purchased, the system anticipates a tightly connected subgraph structure. Items outside of R in this structure not only be closely related to the purchased items but also include as many of the browsed items as possible. Prioritizing these items can enhance recommendation accuracy and boost revenue. In academic research, a professor aims to form a team to tackle a complex problem and he identifies a set of potential collaborators R and a subset of essential researchers $A \subseteq R$. The professor queries an academic collaboration network by providing the vertex sets R and A, expecting to obtain a densely connected community that includes all members of A and incorporates as many members from R as possible, thereby enhancing research efficiency.

To address such real-world applications, Dai *et al.* [20] introduces the concept of R-subgraph density to measure a subgraph S's predisposition towards a reference vertex set R. The R-subgraph density, denoted as $\rho_R(S) = \frac{2|E(S)| - \sum_{v \in S \setminus R} d_G(v)}{|V(S)|}$, refines the doubled traditional density by imposing a penalty $d_G(v)$ (i.e., the degree of a vertex in G) in the numerator for each vertex v in $S \setminus R$. Using the

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 18, No. 3 ISSN 2150-8097.
doi:10.14778/3712221.3712230

[‡]Work partially done at Beijing Institute of Technology.

R-subgraph density, Dai *et al.* [20] formulate the anchored densest subgraph search problem, which seeks to identify the subgraph $\hat{S} \supseteq A$ with the highest $\rho_R(S)$ for a reference set R and an anchor set A. They propose a global algorithm, ADSGA, whose time complexity grows polynomially with the size of G, and a local algorithm, ADSLA, with complexity dependent solely on the size of R. However, since the definition of ADS is based on R-subgraph density, ADSGA and ADSLA must probe every possible real value of $\rho_R(S)$, leading to inefficiencies when processing large-scale graphs or large R sets. Moreover, the ADSLA algorithm's reliance on iterative subgraph expansion results in only marginal efficiency gains compared to ADSGA. Additionally, graph updates require a full recalculation due to the high number of affected edges, posing significant challenges for the efficient maintenance of ADS in dynamic graphs.

To overcome the limitations imposed by R-subgraph density, we propose a novel metric called integer R-subgraph density for a subgraph S, denoted by $\overline{\rho}_R(S) = \lceil \frac{2|E(S)| - \sum_{v \in S \setminus R} d_G(v)}{|V(S)|} \rceil$, and investigate the problem of finding a subgraph $A \subseteq S^* \subseteq V$ of G with the highest $\overline{\rho}_R(S)$ on large static and dynamic graphs. We reveal that $\overline{\rho}_R(S^*) = \lceil \rho_R(\hat{S}) \rceil$, i.e., $\rho_R(\hat{S}) - \rho_R(S^*) < 1$, where $\hat{S}$ represents the ADS, and hence S^* is called the approximate anchored densest subgraph. To search the AADS, we first introduce an efficient global algorithm that incorporates a re-orientation network flow technique and a binary search, addressing concerns of scalability. Subsequently, a novel local algorithm is proposed that leverages shortest-path based methods for localized max-flow computation from s to t around R. This algorithm significantly improves performance in scenarios involving larger values of $|R|$. Additionally, for dynamic graphs, we prove a novel anchored densest subgraph update theorem, based on which both basic and improved algorithms are presented to efficiently maintain the AADS for edge insertions and deletions. In summary, our contributions are as follows.

A global algorithm for AADS search. To compute the AADS, we present an algorithm, named AADSGA, incorporating the re-orientation network flow technique alongside a binary search method. Specifically, the re-orientation network flow technique evaluates whether $\lceil \rho_R(S^*) \rceil \geq \alpha$ using the max-flow method, while the binary search iteratively adjusts the guessed value α to ascertain the AADS. Remarkably, the AADSGA algorithm requires only an integer guess for α, resulting in a time complexity of $O(m^{1.5} \log d_{\max})$ with the classic Dinic's max-flow method [65], where d_{max} is the maximum degree of vertices in a graph. This represents a significant efficiency improvement over the ADSGA algorithm, tailored for ADS search, which has a time complexity of $O(nm \log n)$.

A novel local algorithm for AADS search. To improve the efficiency, we propose a novel local algorithm, denoted as AADSLA. A central innovation of AADSLA is the introduction of the probing method, LReTest, which leverages shortest-path based methods for local max-flow computation from s to t around R. Significantly, for a specified α, whereas the probing method of ADSLA determines the max-flow through a sequence of progressively expanding subgraphs, LReTest eliminates the necessity for this intricate iterative mechanism. Instead, it achieves max-flow computation in tandem with the construction of the locally augmented subgraph. The AADSLA algorithm exhibits a time complexity of $O(\log d_{\max} \mathrm{Vol}^3(R))$ where $\mathrm{Vol}(R)$ is the sum of degrees of vertices in R. In contrast, the ADSLA algorithm, in its last iteration, features $O(\mathrm{Vol}^2(R))$ vertices and $O(\mathrm{Vol}^2(R))$ edges. Using the state-of-the-art max-flow method, the final iteration complexity reaches $O(\mathrm{Vol}^4(R))$, thus establishing the minimum overall complexity of ADSLA at $O(\log n \mathrm{Vol}^4(R))$, markedly surpassing that of AADSLA.

The efficient algorithms for AADS maintenance. To handle dynamic updates of the graph, we prove a novel anchored densest subgraph update theorem, which elucidates that the insertion or deletion of an edge can cause $\lceil \rho_R(\hat{S}) \rceil$ to either remain unchanged, increase by 1, or decrease by 1. Based on this theorem, two fundamental algorithms: Ins and Del are proposed for handling edge insertions and deletions, respectively. These algorithms enable the efficient maintenance of the AADS through merely two probes of the guessed value α, achieving a computational complexity of $O(m^{1.5})$. Furthermore, we present two improved algorithms, Ins+ and Del+, which focus on maintaining an *unreversible orientation* $\vec{G}$ and two important subgraphs to preserve the AADS (details see Section 5.2). The worst-case time complexity of our improved algorithms is $O(\mathrm{Vol}^3(R))$ because they necessitates applying the LReTest whenever $\lceil \rho_R(\hat{S}) \rceil$ changes. In empirical evaluations, both Ins+ and Del+ demonstrate high efficiency as they require merely limited BFS computations in the majority of cases.

Extensive experiments. We conduct comprehensive experiments on 7 large real-life graphs from different domains to evaluate the efficiency of our algorithms for the search and maintenance of AADS. The results show that: (1) AADSGA for AADS search achieves speeds 3 to 13 times faster and reduces memory consumption by 3.5 to 6.5 times compared to ADSGA tailored for ADS search; (2) The AADSLA algorithm demonstrably outperforms ADSLA, with speedups ranging from 5 to 1500 times, while only marginally increasing memory usage; (3) For AADS maintenance, our basic and improved algorithms are significantly superior to the method that recomputes the ADS (AADS) using ADSLA (AADSLA). Ins+ (Del+) consistently outperforms Ins (Del) by at least an order of magnitude across all datasets; (4) The divergence in R-subgraph densities between the ADS $\hat{S}$ and the AADS S^*, namely, $\rho_R(\hat{S}) - \rho_R(S^*)$, invariably remains below 0.2, a figure substantially lower than the theoretical maximum difference of 1. Additionally, we also conduct a case study on dblp to illustrate the effectiveness of the AADS. The results show that AADS can better align a given reference set R compared to the exact ADS, while maintaining a close R-subgraph density to that of the ADS. For reproducibility, the source code of this paper is released (see Artifact availability).

Due to the space limits, all the missing proofs can be found in the full version of this paper [67].

2 PRELIMINARIES

Consider an unweighted and undirected graph $G = (V, E)$, where V is the set of vertices and E denotes the set of edges. Let n and m symbolize the number of vertices and edges in G, respectively. An edge between vertices u and v in G is denoted as (u, v) and equivalently, (v, u). The neighbors of u within G, $N_G(u)$, is defined as $\{v \in V | (u, v) \in E\}$, and the degree of u within G, $d_G(u)$, is given by the cardinality of $N_G(u)$. For a vertex subset S of V, $N_G(S)$ represents the set of neighbors of vertices in S, formally expressed as $N_G(S) = \{v \in V \setminus S | \exists u \in S, (u, v) \in E\}$, and $E(S)$ specifies the edges connecting vertices within S, defined as $E(S) = \{(u, v) \in E | u, v \in S\}$. For two disjoint vertex subsets S and T, the cross edges between S and T are denoted as $E_\times(S, T) = \{(u, v) \in E | u \in S, v \in T\}$, and the additional edges from S with respect to T as $E_{\triangle}(S,T) = E(S) \cup E_{\times}(S,T)$. Given an unweighted and directed graph $\vec{G} = (V, \vec{E})$ with V as the set of vertices and $\vec{E}$ as the set of directed edges. The directed edge from u to v is represented by $\langle u, v\rangle$. A path in this directed graph is a sequence of vertices $v_s = v_0, v_1, ..., v_{l-1}, v_l = v_t$, where $\langle v_{i-1}, v_i\rangle \in \vec{E}$ for $i = 1, ..., l$; for conciseness, this path can also be represented as $v_s \rightsquigarrow v_t$. The set of in-neighbors of u in $\vec{G}$ is denoted as $N_{\vec{G}}(u) = \{v \in V | \langle v, u\rangle \in \vec{E}\}$, and the indegree of u in $\vec{G}$ is $\vec{d}_{\vec{G}}(u) = |N_{\vec{G}}(u)|$. For simplicity, we omit the subscripts G and $\vec{G}$ in the above notations when the context is clear.

Below, we first elucidate the concepts of R-subgraph density and anchored densest subgraph.

Definition 2.1. (R-subgraph Density [20]) Given a graph $G = (V, E)$ and a reference vertex set $R \subseteq V$, the R-subgraph density of a subgraph $S \subseteq V$ is defined as:

$$\rho_R(S) = \frac{2|E(S)| - \sum_{v \in S \backslash R} d_G(v)}{|V(S)|} \tag{1}$$

Definition 2.2. (Anchored Densest Subgraph [20]) Given an undirected graph $G = (V, E)$, an anchor vertex set $A \subseteq V$, a reference vertex set $R \subseteq V$ satisfying $A \subseteq R$ and $E(R) \neq \emptyset$, the ADS, denoted as $\hat{S}$, is a subgraph of G that includes all vertices in A and exhibits the maximum R-subgraph density, i.e., $\hat{S} = \arg\max_{A \subseteq S \subseteq V} \rho_R(S)$.

Example 2.3. Consider a graph $G = (V, E)$ as shown in Figure 1. Let the vertex sets R and A be given as $R = \{v_1, v_3, v_4, v_5, v_6, v_8\}$ and $A = \{v_6\}$, respectively. For the vertex set $S = \{v_1, v_3, v_4, v_5, v_6\}$, the R-subgraph density of S, $\rho_R(S)$, is calculated to be $\frac{2\times 8 - 0}{5} = 3.2$. It is determined that S possesses the highest R-subgraph density with $A \subseteq S$, identifying it as the ADS (i.e., the area within the red dotted line in Figure 1).

Motivated by Definition 2.1 and Definition 2.2, we introduce the concepts of integer R-subgraph density and approximate anchored densest subgraph as follows.

Definition 2.4. (Integer R-Subgraph Density) Given a graph $G = (V, E)$ and a reference vertex set $R \subseteq V$, the integer R-subgraph density of a subgraph $S \subseteq V$ is defined as:

$$\overline{\rho}_R(S) = \lceil \frac{2|E(S)| - \sum_{v \in S \backslash R} d_G(v)}{|V(S)|} \rceil \tag{2}$$

Definition 2.5. (Approximate Anchored Densest Subgraph) Given an undirected graph $G = (V, E)$, an anchor vertex set $A \subseteq V$, a reference vertex set $R \subseteq V$ satisfying $A \subseteq R$ and $E(R) \neq \emptyset$, the AADS, denoted as S^*, is a subgraph of G that includes all vertices in A and exhibits the maximum integer R-subgraph density, i.e., $S^* = \arg\max_{A \subseteq S \subseteq V} \overline{\rho}_R(S)$.

According to Definition 2.2 and Definition 2.5, the following fact is established.

FACT 2.6. *Given the ADS $\hat{S}$ and AADS S^*, $\overline{\rho}_R(S^*) = \lceil \rho_R(\hat{S}) \rceil$ holds, i.e., $\rho_R(\hat{S}) - \rho_R(S^*) < 1$.*

The intuition behind the AADS. AADS relies on the integer R-subgraph density, and the number of possible values of $\overline{\rho}_R(S^*)$ is much less compared to that of $\rho_R(S^*)$, making AADS computationally more efficient. Moreover, compared to the R-subgraph density, the integer R-subgraph density of AADS changes less frequently when edges are inserted or deleted, thus can be efficiently maintained in dynamic graph cases. Finally, Fact 2.6 reveals that $\rho_R(S^*)$ provides an additive approximation to $\rho_R(\hat{S})$, with a minimal loss of precision, suggesting that AADS is a good approximation of ADS.

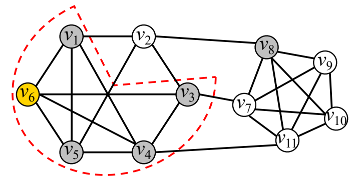


Figure 1: An example graph G

Problem Statement. Given an undirected graph $G = (V, E)$, an anchor vertex set $A \subseteq V$, a reference vertex set $R \subseteq V$ satisfying $A \subseteq R$ and $E(R) \neq 0$, our goal is to search and maintain the AADS S^* on large static and dynamic graphs.

Remark. Fact 2.6 establishes theoretical guarantees for the additive approximation less than 1. However, our empirical analyses indicate that in real-world graphs, the difference between $\rho_R(\hat{S})$ and $\rho_R(S^*)$ is considerably smaller than 1.

3 AN EFFICIENT GLOBAL ALGORITHM

This section proposes an efficient algorithm, namely, AADSGA, for searching the AADS S^*. The essence of AADSGA is to employ a binary search strategy to examine whether $\lceil \rho_R(S^*) \rceil \geq \alpha$. Specifically, if $\lceil \rho_R(S^*) \rceil \geq \alpha$, it is inferred that an increment in α is justified. Conversely, $\lceil \rho_R(S^*) \rceil < \alpha$ indicates a need to reduce α. Below, we begin by checking whether $\lceil \rho_R(S^*) \rceil \geq \alpha$ with the re-orientation network flow technique, followed by a detailed introduction of the AADSGA algorithm integrating a binary search approach.

3.1 Checking whether $\lceil \rho_R(S^*) \rceil \geq \alpha$

Here we propose a probing algorithm, called ReTest, equipped with the re-orientation network flow technique to check whether $\lceil \rho_R(S^*) \rceil \geq \alpha$. Prior to delving into ReTest, we introduce three important concepts: *orientation*, *bounty*, and *re-orientation network*.

Definition 3.1. (Orientation) Given a graph $G = (V, E)$, the orientation of G, represented as $\vec{G} = (V, \vec{E})$, is a directed graph where the vertex set remains identical to that of G; while, for each edge (u, v) in E, $\vec{G}$ features two directed edges between u and v.

Definition 3.2. (Bounty) Given a graph G and its orientation $\vec{G} = (V, \vec{E})$, the bounty of a vertex u, denoted as δ_u, is defined as: (1) $\delta_u = +\infty$ if $u \in A$; (2) $\delta_u = \vec{d}(u)$ if $u \in R \backslash A$; (3) $\delta_u = \vec{d}(u) - d(u)$ if $u \in V \backslash R$.

Utilizing the concept of bounty, for a given value of α, the re-orientation network flow technique enables the construction of an edge-weighted and directed graph $G_\alpha = (V_\alpha, \vec{E}_\alpha, c)$ by augmenting the orientation $\vec{G} = (V, \vec{E})$ as follows. (1) Add a source vertex s and a sink vertex t to V_α, and add all vertices in V to V_α, i.e., $V_\alpha = V \cup \{s, t\}$; (2) For each directed edge $\langle u, v\rangle \in \vec{E}$, add an arc $\langle u, v\rangle$ to E_α with capacity 1; (3) For each vertex $u \in V$, add an arc $\langle s, u\rangle$ to E_α, if $\delta_u < \alpha - 1$ with capacity $(\alpha - 1) - \delta_u$; (4) For each vertex $u \in V$, add an arc $\langle u, t\rangle$ to E_α, if $\delta_u > \alpha - 1$ with capacity $\delta_u - (\alpha - 1)$.

Example 3.3. Consider the graph G depicted in Figure 1. We assume that the orientation $\vec{G}$ of G corresponds to the subgraph of Figure 2, induced by all vertices of G, wherein all directed edges orient towards vertices possessing higher IDs. Given the same sets R and A as in Example 2.3 and $\alpha = 3$, the re-orientation network

Algorithm 1: AADSGA(G, A, R)

Input: An undirected graph $G = (V, E)$, an anchor vertex set $A \subseteq V$, a reference vertex set $R \subseteq V$ satisfying $A \subseteq R$ and $E(R) \neq 0$
Output: The AADS S^*
1 $\alpha_l \leftarrow 1$; $\alpha_u \leftarrow \max_{u \in R} d(u)$;
2 **while** $\alpha_l < \alpha_u$ **do**
3 $\quad \alpha_m \leftarrow \lceil(\alpha_l + \alpha_u + 1)/2\rceil$;
4 $\quad S^* \leftarrow$ ReTest(G, A, R, α_m);
5 $\quad$ **if** $\lceil \rho_R(S^*) \rceil \geq \alpha$ **then** $\alpha_l \leftarrow \alpha_m$;
6 $\quad$ **else** $\alpha_u \leftarrow \alpha_m - 1$;
7 **return** ReTest(G, A, R, α_l);
8 **Procedure** ReTest(G, A, R, α)
9 $\vec{E} \leftarrow \emptyset$;
10 **foreach** $(u, v) \in E$ **do**
11 $\quad \vec{E} \leftarrow \vec{E} \cup \langle u, v\rangle$; $\vec{E} \leftarrow \vec{E} \cup \langle u, v\rangle$;
12 **foreach** $u \in V \backslash R$ **do** $\delta_u \leftarrow \vec{d}(u) - d(u)$;
13 **foreach** $u \in R \backslash A$ **do** $\delta_u \leftarrow \vec{d}(u)$;
14 **foreach** $u \in A$ **do** $\delta_u \leftarrow +\infty$;
15 Construct the re-orientation network $G_\alpha = (V_\alpha, \vec{E}_\alpha, c)$;
16 Compute a maximum flow $f_{\max}$ in $G_\alpha = (V_\alpha, \vec{E}_\alpha, c)$ by Dinic's algorithm;
17 **foreach** $\langle u, v\rangle \in \vec{E}$ **do**
18 $\quad$ **if** *$\langle u, v\rangle$ is satruated in G_α* **then** Reverse the edge $\langle u, v\rangle \in \vec{E}$;
19 **return** $S \leftarrow \{u \in V | \delta_u \geq \alpha$ *or u can reach a vertex v with* $\delta_v \geq \alpha\}$;

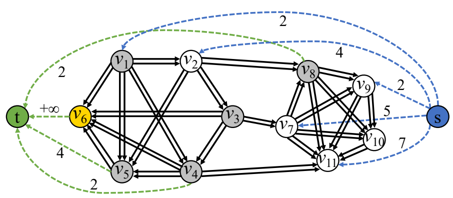


Figure 2: The initial orientation $\vec{G}$ of G

G_α is illustrated as Figure 2. Within G_α, all edges in $\vec{G}$ are assigned a weight of 1, which we omit for clarity.

Based on the re-orientation network, the ReTest algorithm establishes a relationship between the max-flow computation on G_α and the R-subgraph density of a subgraph to determine whether $\lceil \rho_R(S^*) \rceil \geq \alpha$. The pseudo-code of ReTest is outlined in lines 8-19 of Algorithm 1. Initially, ReTest constructs the orientation $\vec{G}$ by inserting the directed edge $\langle u, v\rangle$ twice for each edge (u, v) in G (lines 10-11). It then calculates the bounty of each vertex in G (lines 12-14) and constructs the re-orientation network G_α with a guess value α (line 15). The algorithm proceeds to compute the maximum flow on G_α and reverses the direction of each edge in $\vec{G}$ if it is saturated in G_α (lines 16-18). Ultimately, the ReTest algorithm yields a subgraph S comprising all vertices with a bounty not less than α, or those capable of reaching a vertex with a bounty that meets or exceeds α (line 19). Regarding the subgraph S, the following theorem establishes the correctness of the ReTest algorithm.

Theorem 3.4. *The subgraph S output by the* ReTest *algorithm exhibits the following properties: (1) $A \subseteq S$ and $\lceil \rho_R(S) \rceil \geq \alpha$ if $\lceil \rho_R(\hat{S}) \rceil \geq \alpha$; (2) $A \subseteq S$ and $\lceil \rho_R(S) \rceil < \alpha$ if $\lceil \rho_R(\hat{S}) \rceil < \alpha$.*

3.2 The AADSGA algorithm

With Section 3.1, we propose the AADSGA algorithm which utilizes a binary search on α to identify the AADS. The binary search requires establishing the guess value α's range, where the lower bound is intuitively set to 1, and the upper bound, as inferred from [20], is $\max_{u \in R} d(u)$. The pseudo-code of AADSGA is depicted in Algorithm 1. It first initializes the lower bound α_l and the upper bound α_u and then iteratively computes the middle value α_m and invokes ReTest(G, A, R, α_m) to yield a subgraph S^*. If $\lceil \rho_R(S^*) \rceil \geq \alpha$, then α_l is updated to α_m (line 5); otherwise, α_u is adjusted to $\alpha_m - 1$ (line 6). The iterative process terminates once α_l equals α_u, and the algorithm performs ReTest(G, A, R, α_l) again to capture the AADS S^* (line 7). The following theorem shows the correctness and complexity of Algorithm 1.

Theorem 3.5. *Algorithm 1 outputs the AADS S^* correctly with the time complexity of $O(m^{1.5} \log d_{max})$ and the space complexity of $O(m + n)$.*

Discussions. The time complexity of AADSGA for AADS search, $O(m^{1.5} \log d_{\max})$, is significantly lower than the $O(nm \log n)$ held by ADSGA, tailored for ADS search. This is because AADSGA performs a binary search only on integer values ($\log d_{\max}$), whereas ADSGA requires a binary search on fractional values ($\log n$) and $n > \sqrt{m}$ holds. Our later experiments confirm this theoretical result.

4 A NOVEL LOCAL ALGORITHM

Although the AADSGA algorithm operates within polynomial time complexity, its scalability is still limited when applied to large-scale graphs. This section presents a novel local algorithm, i.e., AADSLA, to solve the AADS search problem. A significant innovation within AADSLA is the introduction of a novel probing method LReTest, which enables the localized computation of the max-flow from s to t around R without necessitating access to (or construction of) the entire augmented graph G_α. Below, we first introduce the AADSLA algorithm and the LReTest algorithm and then proceed to the theoretical analysis.

4.1 The framework of AADSLA algorithm

The AADSGA algorithm initializes the orientation $\vec{G}$ by inserting $\langle u, v\rangle$ into $\vec{G}$ twice for each edge $(u, v) \in G$. After this initialization method, it is impossible to determine the bounty of each vertex, thus the algorithm needs to traverse **all** vertices to construct the re-orientation network, and therefore it is not local.

We introduce a novel initialization approach for $\vec{G}$, namely "bi-directional orientation", which entails inserting both $\langle u, v\rangle$ and $\langle v, u\rangle$ for every edge (u, v). This method ensures that the bounty of the vertices in the orientation after initialization is definite and regular since each vertex u receives a precise tally of $d(u)$ incoming edges. Combined with the definition, it follows that vertices within the set $R \backslash A$ have a bounty equal to $d(u)$, whereas all other vertices in $V \backslash R$ have a bounty of exactly 0. As a result, only the vertices in R are connected to the sink t. Intuitively, the augmenting paths found by the maximum flow algorithm often pass only through vertices near R, while vertices farther from the set R are not considered. This forms the rationale for designing our local algorithm.

With the concept of a bi-directional orientation, we propose the AADSLA algorithm, detailed in Algorithm 2. This algorithm adopts a binary search framework to compute the AADS, with the following distinctions from AADSGA. Firstly, AADSLA determines if $\lceil \rho_R(S^*) \rceil \geq \alpha$ for a specified guess value α by using LReTest (Algorithm 3, detailed in Section 4.2), which locally computes the max-flow from s to t around R in G_α with shortest-path based method (line 13, line 16). Secondly, the orientation $\vec{G}$ is progressively constructed in executing a local search. A set V^A is employed

Algorithm 2: AADSLA(G, A, R)

Input: An undirected graph $G = (V, E)$, an anchor vertex set $A \subseteq V$ and a reference vertex set $R \subseteq V$ satisfying $A \subseteq R$ and $E(R) \neq 0$
Output: The AADS S^*
1 $\mathrm{Vol}(R) \leftarrow 0$;
2 $\vec{G} \leftarrow \emptyset$; $V^A \leftarrow \emptyset$;
3 **foreach** $u \in R$ **do**
4   $\mathrm{Vol}(R) \leftarrow \mathrm{Vol}(R) + d(u)$;
5   $V^A \leftarrow V^A \cup \{u\}$;
6   $\mathsf{InitOri}(u)$;
7   **if** $u \in A$ **then** $\delta_u \leftarrow +\infty$;
8   **else** $\delta_u \leftarrow d(u)$;
9 **foreach** $u \in V \backslash R$ **do** $\delta_u \leftarrow 0$;
10 $\alpha_l \leftarrow 1$; $\alpha_u \leftarrow \max_{u \in R} d(u)$;
11 **while** $\alpha_l < \alpha_u$ **do**
12   $\alpha_m \leftarrow \lceil (\alpha_l + \alpha_u + 1)/2 \rceil$;
13   $S^* \leftarrow \mathsf{LReTest}(\vec{G}, A, R, \alpha_m, V^A)$;
14   **if** $\lceil \rho_R(S^*) \rceil \geq \alpha$ **then** $\alpha_l \leftarrow \alpha_m$;
15   **else** $\alpha_u \leftarrow \alpha_m - 1$;
16 **return** $\mathsf{LReTest}(\vec{G}, A, R, \alpha_l, V^A)$;
17 **Procedure** $\mathsf{InitOri}(u)$
18 **if** $u \notin V^A$ **then**
19   $V^A \leftarrow V^A \cup \{u\}$;
20   **foreach** $v \in N_G(u)$ *and* $v \notin V^A$ **do**
21     $\vec{G} \leftarrow \vec{G} \cup \langle u, v \rangle$; $\vec{G} \leftarrow \vec{G} \cup \langle v, u \rangle$;

Algorithm 3: LReTest$(\vec{G}, A, R, \alpha, V^A)$

Input: An undirected graph $G = (V, E)$, an anchor vertex set $A \subseteq V$, a reference vertex set $R \subseteq V$, a non-negative integer α, and the set V^A
Output: A subgraph S satisfying $A \subseteq S$
1 **if** $\alpha = 1$ **then** $\{S \leftarrow R$; **return** $S\}$;
2 $V^N \leftarrow \{u \in V^A | \delta_u \geq \alpha - 1\}$;
3 **foreach** $u \in V^N$ **do** $\mathsf{InitOri}(u)$;
4 Construct the partial reverse re-orientation network $G_{\tilde{\alpha}}^{-1}$ according to $\vec{E}$;
5 **while** *true* **do**
6   Perform BFS from the source node s in $G_{\tilde{\alpha}}^{-1}$ to try to find a shortest path $\langle s, v_s, \ldots, v_t \rangle$ in $G_{\tilde{\alpha}}^{-1}$ where v_t satisfies $\delta_{v_t} < \alpha - 1$ or $d(v_t) \geq \mathrm{Vol}(R)$;
7   For each vertex u visited by the BFS, invoke $\mathsf{InitOri}(u)$ to update $\vec{E}$;
8   Maintain $G_{\tilde{\alpha}}^{-1}$ according to the updated $\vec{E}$ in the process of BFS;
9   **if** *there exists such a shortest path* $\langle s, v_s, \ldots, v_t \rangle$ *in* $G_{\tilde{\alpha}}^{-1}$ **then**
10   Reverse the path $\langle v_s, \ldots, v_t \rangle$ in $\vec{E}$;
11   **if** $v_t \notin V^A$ *and* $d(v_t) < \mathrm{Vol}(R)$ **then** $V^A \leftarrow V^A \cup v_t$;
12   **if** $\delta_{v_t} = \alpha - 1$ *and* $d(v_t) < \mathrm{Vol}(R)$ **then** $V^N \leftarrow V^N \cup v_t$;
13   **else break;**
14 $S \leftarrow \{u \in V | \delta_u \geq \alpha$ or u can reach a vertex v with $\delta_v \geq \alpha\}$;
15 **return** S;

to determine whether u's adjacent edges have been integrated into $\vec{G}$, thereby ensuring that each vertex u is processed only once (line 2). For each vertex $u \notin V^A$, the InitOri procedure is invoked following the bi-directional orientation construction method. It inserts $\langle u, v \rangle$ and $\langle v, u \rangle$ for each edge (u, v) associated with u into $\vec{G}$ (lines 17-21). Before performing LReTest, AADSLA initializes by invoking InitOri for each vertex in R (line 6) and calculating $\mathrm{Vol}(R)$, defined as the sum of the degrees of all vertices in R (line 4). $\mathrm{Vol}(R)$ is crucial for computing the local max-flow of LReTest (see Lemma 4.1). Additionally, vertices in A are assigned an infinite bounty, vertices in $R \backslash A$ receive a bounty equal to their degree, and others are allocated a bounty of 0 (lines 7-9).

4.2 Locally checking whether $\lceil \rho_R(S^*) \rceil \geq \alpha$

Before introducing LReTest for identifying whether $\lceil \rho_R(S^*) \rceil \geq \alpha$, it is essential to delineate the concept of "reverse re-orientation network", which is instrumental in offering a more intuitive elucidation of the algorithm's locality.

Given an integer α, the reverse re-orientation network, $G_\alpha^{-1} = (V_\alpha, E_\alpha^{-1}, c)$, is constructed by augmenting the bi-directional orientation $\vec{G} = (V, \vec{E})$ as follows. (1) Add a source vertex s and a sink vertex t to V_α, and add all vertices in V to V_α, i.e., $V_\alpha = V \cup \{s, t\}$; (2) For each directed edge $\langle u, v \rangle \in \vec{E}$, add an arc $\langle v, u \rangle$ to E_α^{-1} with capacity 1; (3) For each vertex $u \in V$, add an arc $\langle s, u \rangle$ to E_α^{-1}, if $\delta_u > \alpha - 1$ with capacity $\delta_u - (\alpha - 1)$; (4) For each vertex $u \in V$, add an arc $\langle u, t \rangle$ to E_α^{-1}, if $\delta_u < \alpha - 1$ with capacity $(\alpha - 1) - \delta_u$. After performing the max-flow computation on G_α^{-1}, there is no path from s to t, and we can also yield a set S including the vertices either with a bounty of at least α or that can reach another vertex with a bounty of at least α. According to Theorem 3.4, the correctness of the probing method using G_α^{-1} is also established.

Based on the reverse re-orientation network, our goal is to implement a local max-flow calculation starting from R. The probing method described in ADSLA initially constructs a subgraph, followed by a max-flow computation and iterative subgraph expansion until a solution is obtained. The LReTest algorithm integrates max-flow computation with subgraph construction to simplify this iterative process. It employs a shortest path-based approach which identifies the shortest augmenting path to compute the max-flow. Upon locating this path, all edges present are reversed in $\vec{G}$. In the computation of max-flow, a critical aspect involves determining the termination of the shortest augmenting path during the BFS procedure. Certainly, the absence of a path from source s to sink t in G_α^{-1} indicates that the search halts at a vertex with bounty less than $\alpha - 1$. To enhance the efficiency of the BFS procedure, we propose Lemma 4.1 that provides a novel approach to determine the path's endpoint.

Lemma 4.1. *Given $\alpha \geq 2$ and a subgraph S with $\rho_R(S) > \alpha - 1$, the degree $d(u)$ for each vertex u in S satisfies $d(u) < \mathrm{Vol}(R)$.*

Equipped with Lemma 4.1, we propose the LReTest algorithm detailed in Algorithm 3. This algorithm leverages the bi-directional orientation from the previous iteration to expedite the construction of the reverse re-orientation network. Specifically, it first checks if $\alpha = 1$. If this condition holds, i.e., $\rho_R(\hat{S}) \leq 1$, the algorithm directly outputs the set R as the AADS (line 1). For the case of $\alpha \geq 2$, corresponding to $\rho_R(\hat{S}) > 1$, LReTest first adds the vertices in V^A with a bounty no less than $\alpha - 1$ into V^N (line 2). After initializing V^N, it incorporates the adjacent edges of each vertex in V^N into the orientation for α using the InitOri procedure (line 3).

With V^N and $\vec{G}$, the partial reverse re-orientation network, $G_{\tilde{\alpha}}^{-1}$, is initialized, obviating the need for construction from scratch (line 4). Next, the LReTest algorithm progresses to the construction of the local G_α^{-1} based on $G_{\tilde{\alpha}}^{-1}$ and the max-flow computation phases, utilizing a computation-while-expanding strategy (lines 5-13). It continuously executes the BFS procedure starting from the source node s, aiming to identify the shortest augmenting path $\langle s, v_s, \ldots, v_t \rangle$, where v_t meets $\delta_{v_t} < \alpha - 1$ or $d(v_t) \geq \mathrm{Vol}(R)$, in accordance with Lemma 4.1. Note that during the BFS process, as each vertex is visited, the InitOri procedure is invoked to extend the orientation, and then $G_{\ddot{\alpha}}^{-1}$ is updated (lines 6-8). Upon discovering a shortest augmenting path $\langle s, v_s, \ldots, v_t \rangle$, all edges on this path, except for the one connected to s, are saturated. Consequently, these saturated edges are reversed in the bi-directional orientation, and updates are made to V^A and V^N (lines 10-12). The absence of augmenting paths in $G_{\ddot{\alpha}}^{-1}$ signifies the completion of the max-flow computation. At this juncture, Algorithm 3 outputs the vertex set S, which comprises vertices either possessing a bounty of at least α or capable of reaching another vertex with a bounty of at least α.

4.3 Analysis of correctness and locality

Using Theorem 3.4 and Lemma 4.1, we ascertain the correctness of LReTest, ensuring that the AADSLA algorithm correctly outputs the AADS. Next, we establish the locality of the AADSLA algorithm by demonstrating that LReTest is local. LReTest constructs a local reverse re-orientation network $G_{\ddot{\alpha}}^{-1}$, which we extend to be complete for ease of analysis. Specifically, we introduce $\langle u, v \rangle$ and $\langle v, u \rangle$ into $G_{\ddot{\alpha}}^{-1}$ if u and v are not connected in $G_{\ddot{\alpha}}^{-1}$ but $(u, v) \in E$. Within the complete G_{α}^{-1}, let l be the distance between s and t and $\mathrm{dist}(s, u)$ be the distance between s and u. Denote V_i by the vertex set defined as $V_i = \{u \in V \mid \mathrm{dist}(s, u) \leq i\}$. The following lemma is established.

Lemma 4.2. *In G_{α}^{-1}, $|V_{l-1}| \leq \frac{Vol(R)}{\alpha} + |A| \leq 2Vol(R)$ and $|V_l| \leq 2Vol^2(R)$.*

With the established Lemma 4.2, we analyze the time complexity of LReTest using the Dinic algorithm as a paradigmatic example of a shortest path-based max-flow computation technique. Notably, the implementation of the Dinic algorithm [65] herein includes an optimization: during the BFS process, when exploring the adjacent edges of a vertex u in V_{l-1}, the search will terminate upon reaching the edge (u, t). This optimization halts the BFS prematurely, preventing further traversal of other vertices.

Theorem 4.3. *The* LReTest *algorithm, which employs the Dinic algorithm for maximum flow computation, has a time complexity of $O(Vol^3(R))$ and a space complexity of $O(m + n)$.*

Theorem 4.3 shows that the time complexity of LReTest is bounded by a polynomial in terms of $O(\mathrm{Vol}(R))$, which is independent on the size of graph. With this foundation, AADSLA is strongly local, with a time complexity of $O(\log d_{\max} \times \mathrm{Vol}^3(R))$. The space complexity of AADSLA is $O(m + n)$ as it needs to invoke LReTest.

Discussions. We analyze the time complexity of ADSLA for searching ADS [20]. First, ADSLA needs to perform $(\log n)$ local max-flow computations. Second, the local max-flow computation method in ADSLA iteratively extends the subgraph and performs computations. In the last iteration, the number of vertices and edges of the subgraph are both $O(\mathrm{Vol}^2(R))$, causing the time complexity to reach $O(\mathrm{Vol}^4(R))$ using the state-of-the-art max-flow method. Thus, the total time complexity of ADSLA is $\geq O(\log n \times \mathrm{Vol}^4(R))$. Clearly, the time complexity of AADSLA for searching AADS, i.e., $O(\log d_{\max} \times \mathrm{Vol}^3(R))$, is significantly lower than that of ADSLA. Our subsequent experiments confirm this theoretical analysis.

5 THE MAINTAINANCE OF AADS

This section proposes efficient algorithms for maintaining the AADS amidst updates to the graph. Initially, an ADS update theorem is presented, based on which we propose the Ins and Del algorithms

Algorithm 4: $\mathrm{Ins}(G, A, R, S^*, (u, v))$

Input: An undirected graph $G = (V, E)$, an anchor vertex set A, a reference vertex set R, the AADS S^* and an edge (u, v) to be inserted

Output: The updated AADS $\overline{S}^*$

1 $E \leftarrow E \cup \{(u, v)\}$;
2 $\alpha \leftarrow \lceil \rho_R(S^*) \rceil$;
3 $S^* \leftarrow \mathrm{ReTest}(G, A, R, \alpha)$;
4 **if** $\lceil \rho_R(S^*) \rceil < \alpha$ **then** // Check if $\lceil \rho_R(\hat{S}) \rceil$ decreases by 1
5 $\quad S^* \leftarrow \mathrm{ReTest}(G, A, R, \alpha - 1)$;
6 **else**
7 $\quad \ddot{S} \leftarrow \mathrm{ReTest}(G, A, R, \alpha + 1)$;
8 $\quad$ **if** $\lceil \rho_R(\ddot{S}) \rceil \geq \alpha + 1$ **then** // Check if $\lceil \rho_R(\hat{S}) \rceil$ increases by 1
9 $\quad\quad S^* \leftarrow \mathrm{ReTest}(G, A, R, \alpha + 1)$;
10 **return** S^*;

for edge insertions and deletions, respectively. Moreover, the improved algorithms, Ins+ and Del+, are developed, with a focus on maintaining an unreversible orientation $\vec{G}$ to preserve the AADS.

5.1 ADS update theorem and basic algorithms

We first present the ADS update theorem, which forms the foundation of the proposed Ins and Del algorithms for maintaining AADS.

Theorem 5.1. *After a single edge insertion (resp., deletion) in G, $\lceil \rho_R(\hat{S}) \rceil$ either remains unchanged, increases by 1, or decreases by 1.*

According to Theorem 5.1, $\lceil \rho_R(\hat{S}) \rceil$ remains constant or alters by only one unit following the insertion or deletion of an edge, the same applies to $\lceil \rho_R(S^*) \rceil$. Thus, merely two checks using ReTest are required to identify the updated AADS. Based on this rationale, we develop the Ins and Del algorithms for edge insertion and deletion. The pseudo-code of Ins is outlined in Algorithm 4. Initially, the algorithm inserts the edge (u, v) into G, and calculates the current round-up of the maximum R-subgraph density $\alpha \leftarrow \lceil \rho_R(S^*) \rceil$ (line 2). Subsequently, Ins invokes ReTest with the parameter α, updating the subgraph S^* (line 3). The algorithm then checks whether $\lceil \rho_R(S^*) \rceil < \alpha$ to determine if $\lceil \rho_R(\hat{S}) \rceil$ decreases by 1 (line 4). If so, Ins performs ReTest with guess value $\alpha - 1$ to update S^* as the AADS after inserting edge (u, v); otherwise, the algorithm needs to compute the subgraph $\ddot{S}$ to check if $\lceil \rho_R(\hat{S}) \rceil$ increases by 1 (lines 7-8), and then update S^* if it does increase. For the Del algorithm, which maintains the AADS for edge deletion, its pseudo-code necessitates merely a modification of line 1 in Algorithm 4 to $E \leftarrow E - \{u, v\}$.

The correctness of Ins and Del is affirmed by Theorem 3.4 and Theorem 5.1. Theorem 5.2 shows their time complexity.

Theorem 5.2. *The time complexity of* Ins *and* Del *are $O(m^{1.5})$ because they only need to perform* ReTest *twice.*

5.2 The improved algorithms

The primary limitation of the basic algorithms lies in their dependency on reconstructing the orientation network for an updated graph and performing a max-flow computation. In this subsection, we propose two improved algorithms, Ins+ and Del+, designed to maintain the AADS by preserving an "unreversible orientation" contingent on a parameter α. For a given integer α, we define "reversible path" and "unreversible orientation" as follows.

Definition 5.3. (Reversible Path) Given a graph $G = (V, E)$, its orientation $\vec{G} = (V, \vec{E})$ and an integer α, a path $v_s \rightsquigarrow v_t$ is reversible if: (1) $\delta_{v_s} < \alpha - 1$ and $\delta_{v_t} > \alpha - 1$; or (2) $\delta_{v_s} < \alpha$ and $\delta_{v_t} > \alpha$.

Definition 5.4. (Unreversible Orientation) Given a graph $G = (V, E)$, its orientation $\vec{G} = (V, \vec{E})$ and an integer α, $\vec{G}$ is an unreversible orientation if there is no reversible path in $\vec{G}$.

Algorithm 5: $\mathsf{Ins+}(\vec{G}, A, R, S_\alpha, S_{\alpha+1}, (u, v))$

Input: A graph $G = (V, E)$, an orientation $\vec{G} = (V, \vec{E})$, an anchor vertex set A, a reference vertex set R, the subgraph S_α, $S_{\alpha+1}$, and the edge (u, v) to be inserted
Output: The updated S_α, $S_{\alpha+1}$, and orientation $\vec{G}$

1 $E \leftarrow E \cup \{(u, v)\}$;
2 **if** $u \notin R$ **then** $\mathsf{DecBounty}(u)$;
3 **if** $v \notin R$ **then** $\mathsf{DecBounty}(v)$;
4 **if** *($u \in S_{\alpha+1}$ and $v \notin S_{\alpha+1}$) or ($u \in S_\alpha$ and $v \notin S_\alpha$)* **then** $p \leftarrow u$; $q \leftarrow v$;
5 **else** $p \leftarrow v$; $q \leftarrow u$;
6 $\vec{G} \leftarrow \vec{G} \cup \{\langle p, q \rangle\}$;
7 $\mathsf{IncBounty}(q)$;
8 Repeat lines 4-7;
9 **if** $\lceil \rho_R(S_{\alpha+1}) \rceil \geq \alpha + 1$ **then** $S_\alpha \leftarrow S_{\alpha+1}$; $S_{\alpha+1} \leftarrow \mathsf{LReTest}(\alpha + 2)$; α++;
10 **else if** $\lceil \rho_R(S_\alpha) \rceil < \alpha$ **then** $S_{\alpha+1} \leftarrow S_\alpha$; $S_\alpha \leftarrow \mathsf{LReTest}(\alpha - 1)$; α− −;
11 **return** $(S_\alpha, S_{\alpha+1}, \vec{G})$

12 **Procedure** $\mathsf{IncBounty}(x)$
13 δ_x++;
14 **if** $x \in V \setminus S_\alpha$ *and* $\delta_x = \alpha$ **then**
15   **if** *a path $y \rightsquigarrow x$ can be found, $\delta_y \leq \alpha - 2$* **then** Reverse the path; δ_x− −; δ_y++;
16   **else** $S_\alpha \leftarrow S_\alpha \cup \{x\} \cup \{y | y$ can reach $x\}$;
17 **else if** $x \in S_\alpha \setminus S_{\alpha+1}$ *and* $\delta_x = \alpha + 1$ **then**
18   **if** *a path $y \rightsquigarrow x$ can be found, $\delta_y \leq \alpha - 1$* **then** Reverse the path; δ_x− −; δ_y++;
19   **else** $S_{\alpha+1} \leftarrow S_{\alpha+1} \cup \{x\} \cup \{y | y$ can reach $x\}$;

20 **Procedure** $\mathsf{DecBounty}(x)$
21 δ_x− −;
22 **if** $x \in S_{\alpha+1}$ **then**
23   **if** $\delta_x = \alpha - 1$ **then** Reverse a path $x \rightsquigarrow y$ where $\delta_y \geq \alpha + 1$; δ_y− −; δ_x++;
24     $S_{\alpha+1} \leftarrow \{y | \delta_y \geq \alpha + 1$ or y can reach any vertex with $\delta \geq \alpha + 1\}$;
25 **else if** $x \in S_\alpha \setminus S_{\alpha+1}$ **then**
26   **if** $\delta_x = \alpha - 2$ **then** Reverse a path $x \rightsquigarrow y$, where $\delta_y \geq \alpha$, δ_y− −, δ_x++;
27     $S_\alpha \leftarrow \{y | \delta_y \geq \alpha$ or y can reach any vertex with $\delta \geq \alpha\}$;

The rationale behind defining the unreversible path and the unreversible orientation originates from Theorem 3.4. Note that $\lceil \rho_R(S_\alpha) \rceil \geq \alpha$ and $\lceil \rho_R(S_{\alpha+1}) \rceil < \alpha + 1$ hold if and only if $\alpha = \lceil \rho_R(\hat{S}) \rceil$, where S_* is the subgraph returned by invoking the ReTest algorithm with $*$ as the guess value. It can be proven that S_* comprises the vertices that either have a bounty of at least $*$ or can reach another vertex with a bounty of no less than $*$ in an unreversible orientation $\vec{G}$. Notably, S_α consists of vertices within the AADS. By maintaining an unreversible orientation $\vec{G}$ and two subgraphs S_α and $S_{\alpha+1}$, we can determine whether $\lceil \rho_R(\hat{S}) \rceil$ changes due to the insertion or deletion of edges and thus maintain AADS. This constitutes the central idea of our $\mathsf{Ins+}$ and $\mathsf{Del+}$ algorithms.

According to the above main idea, the unreversible orientation $\vec{G}$ and two subgraphs S_α and $S_{\alpha+1}$ must be taken as inputs of the $\mathsf{Ins+}$ and $\mathsf{Del+}$ algorithms. We provide the following method to compute the inputs $\vec{G}$, S_α and $S_{\alpha+1}$. First, we perform AADSGA (or AADSLA) with $\alpha = \lceil \rho_R(S^*) \rceil$, ensuring that there is no path $v_s \rightsquigarrow v_t$ with $\delta_{v_s} < \alpha - 1$ and $\delta_{v_t} > \alpha - 1$ in $\vec{G}$. Subsequently, we construct the re-orientation network based on $\vec{G}$ and $\alpha + 1$, and invoke ReTest again to ensure no path $v_s \rightsquigarrow v_t$ exists with $\delta_{v_s} < \alpha$ and $\delta_{v_t} > \alpha$ in $\vec{G}$. With the unreversible orientation $\vec{G}$, S_α and $S_{\alpha+1}$ can be derived easily. Below, we introduce our improved algorithms in detail.

The $\mathsf{Ins+}$ algorithm for edge insertion. When an edge (u, v) is inserted, it is imperative to insert two directed edges into $\vec{G}$ and maintain $\vec{G}$ as an unreversible orientation. We present a three-step procedure for edge insertion, detailed as follows: (1) insert (u, v) into the graph G; (2) insert a directed edge into the orientation $\vec{G}$; (3) repeat step 2. This procedure ensures that the bounty of any vertex changes by only one unit at each step, simplifying the identification of reversible paths and thereby aiding in maintaining $\vec{G}$.

The $\mathsf{Ins+}$ algorithm is outlined in Algorithm 5, which accepts three specific parameters: the unreversible orientation $\vec{G}$, S_α and $S_{\alpha+1}$. The main idea of $\mathsf{Ins+}$ is to maintain the AADS by ensuring that $\vec{G}$ remains an unreversible orientation throughout each stage of the three-step process. In Step 1 (lines 1-3), $\mathsf{Ins+}$ inserts (u, v) into G, which increments the degrees of u and v by one each, potentially altering their bounties δ_u and δ_v. Taking u as an example, if $u \in R$, then by Definition 3.2, δ_u remains unchanged, thus preserving the irreversibility of $\vec{G}$. Alternatively, if $u \notin R$, the $\mathsf{DecBounty}$ procedure is invoked to reduce δ_u by 1 and make $\vec{G}$ be unreversible again (line 2). In $\mathsf{DecBounty}$, when x is in $S_{\alpha+1}$, it checks if δ_x decreases to $\alpha - 1$. If it does, since x can reach a vertex y in $\vec{G}$ whose bounty is at least $\alpha + 1$, we reverse the reversible path $x \rightsquigarrow y$ and update δ_y and δ_x (line 23). Importantly, $\mathsf{DecBounty}$ updates the set $S_{\alpha+1}$, regardless of whether δ_x equals $\alpha - 1$ (line 24). For the case of $x \in S_\alpha \backslash S_{\alpha-1}$, if $\delta_x = \alpha - 2$, a reversible path $x \rightsquigarrow y$ is identified as x can reach a vertex y with bounty no less than α. This path is then reversed, and updates are applied to both δ_y and δ_x (line 26). Note that $\mathsf{DecBounty}$ maintains the set S_α once $x \in S_\alpha \backslash S_{\alpha-1}$ (line 27). For all other cases, it is clear that $\vec{G}$ remains unreversible, as there are no reversible paths present in the input orientation $\vec{G}$.

In step 2 of $\mathsf{Ins+}$ (lines 4-7), a directed edge $\langle p, q \rangle$ is inserted into $\vec{G}$, with the orientation determined by the membership of u and v in sets $S_{\alpha+1}$ or S_α. This configuration ensures that after the edge's insertion, $\vec{G}$ hosts at most one reversible path. The insertion allows vertex p to retain its bounty δ_p, simultaneously increasing the in-degree and the bounty of vertex q, regardless of its presence in set R. Following this, the $\mathsf{Ins+}$ algorithm engages the $\mathsf{IncBounty}(q)$ procedure to preserve the irreversibility of $\vec{G}$ by identifying a potential reversal path (lines 12-19). In $\mathsf{IncBounty}$, after increasing x's bounty by 1, if x belongs to $V \backslash S_\alpha$ and δ_x reaches α, the procedure checks for a reversible path $y \rightsquigarrow x$ where $\delta_y \leq \alpha - 2$. If such a path is identified, it is reversed, and both δ_x and δ_y are updated to restore the irreversibility of $\vec{G}$. Conversely, S_α is maintained as x now has a bounty equal to α. For the case where x is in $S_\alpha \backslash S_{\alpha+1}$ and δ_x equals $\alpha + 1$, if the procedure identifies a reversible path $y \rightsquigarrow x$ with $\delta_y \leq \alpha - 1$, then the path is reversed and δ_x and δ_y are adjusted accordingly to make $\vec{G}$ unreversible again. Otherwise, $S_{\alpha+1}$ is maintained since the bounty of x now equals $\alpha + 1$. For other cases where these specific conditions do not apply, $\vec{G}$ remains irreversible. Step 3 repeats the operations of step 2 to continuously uphold the irreversibility of $\vec{G}$ (line 8).

After the three-step procedure, $\mathsf{Ins+}$ ensures that $\vec{G}$ retains an unreversible orientation upon edge insertion, and then it maintains the sets S_α and $S_{\alpha+1}$ as follows. If $\lceil \rho_R(S_{\alpha+1}) \rceil \geq \alpha + 1$, indicating that $\lceil \rho_R(\hat{S}) \rceil = \lceil \rho_R(S_{\alpha+1}) \rceil = \alpha + 1$, and then S_α is updated to $S_{\alpha+1}$, and $\mathsf{Ins+}$ performs $\mathsf{LReTest}(\alpha + 2)$ to re-calculate $S_{\alpha+1}$ and increase α by 1 (line 9). While if $\lceil \rho_R(S_\alpha) \rceil < \alpha$, meaning $\lceil \rho_R(\hat{S}) \rceil =$

Table 1: Graph datasets statistics

Name	Type	n	m	d_{max}
dblp	Citation network	1,653,767	8,159,739	2,119
skitter	Computer network	1,696,416	11,095,299	35,455
indian	Hyperlink network	1,382,868	13,591,473	21,869
pokec	Social network	1,632,804	22,301,964	14,854
livejournal	Social network	3,997,962	34,681,189	14,815
orkut	Social network	3,072,441	117,185,083	33,313
weibo	Social network	58,655,849	261,321,033	278,489

$\lceil \rho_R(S_{\alpha+1}) \rceil = \alpha - 1$, $S_{\alpha+1}$ is adjusted to S_α, and LReTest is employed to manage $S_{\alpha-1}$ and decrease α by 1 (line 10). Ultimately, the Ins+ algorithm outputs S_α, $S_{\alpha+1}$ and $\vec{G}$ where S_α represents the AADS.

Owing to space limitations, the complete proof of correctness for the Ins+ algorithm is provided in the full version [67]. Concerning time complexity, Ins+ requires $O(\mathrm{Vol}^3(R))$ time in the worst-case scenario due to its reliance on LReTest. However, Ins+ exhibits superior efficiency in practical settings for two reasons. Firstly, the BFS process terminates upon encountering vertices with a bounty less than or equal to a specific threshold, limiting the search domain to vertices with bounties exceeding this threshold. Typically, these vertices are located within the graph's densest regions, which are relatively small in practical scenarios, thus offering a restricted search field for BFS. Secondly, LReTest is activated only when there is a change in $\lceil \rho_R(\hat{S}) \rceil$. Given that such changes are rare in real-world applications, the execution of LReTest becomes infrequent.

The Del+ algorithm for edge deletion. The deletion of an edge (u, v) necessitates the removal of two directed edges from $\vec{G}$. Analogous to Ins+, Del+ involves a three-step procedure for edge deletion, summarized as follows: (1) delete (u, v) from the graph G; (2) delete a directed edge between u and v from the orientation $\vec{G}$; (3) repeat step 2. Following such a three-step procedure, the Del+ algorithm maintains the AADS by ensuring that $\vec{G}$ remains in an unreversible orientation. The pseudo-code of Del+ differs from Ins+ in the following slight ways. First, the Del+ algorithm removes the edge (u, v) at line 1 and performs IncBounty at lines 2-3 to complete the first step. In step 2 (step 3), Del+ directly obtains a directed edge $\langle x, y \rangle$ between vertices u and v and invokes DecBounty$(y, \langle x, y \rangle)$ to delete it without executing lines 4-7. Note that DecBounty in Del+ has one additional input parameter, i.e., the directed edge $\langle x, y \rangle$, compared to DecBounty in Ins+, because Del+ needs to delete the directed edge $\langle x, y \rangle$ from the orientation after lines 23, 26, and 27. The correctness and time complexity of Del+ are similar to that of Ins+ (detailed analysis can be found in our full version [67]).

6 EXPERIMENTS

6.1 Experiment settings

Different algorithms. We implement the proposed algorithms, specifically AADSGA (Algorithm 1) and AADSLA (Algorithm 2), for the problem of AADS search. For comparative analysis, algorithms for searching the ADS presented in [20], namely, ADSGA and ADSLA, are also implemented. We also incorporate the flow-based local graph clustering algorithm with seed set inclusion [63], FlowSeed, into our analysis. For dynamic graphs, we implement the maintenance algorithms for AADS, which includes Ins (Algorithm 4) and Ins+ (Algorithm 5) for edge insertion, alongside Del and Del+ for edge deletion. In our experiments, we also compare these maintenance algorithms with the method that uses ADSLA to compute from scratch.

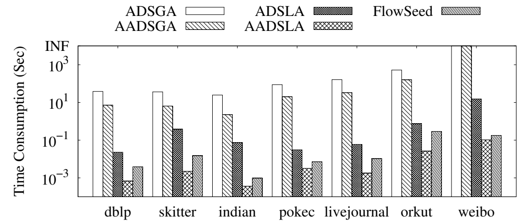


Figure 3: The comparison of computation time

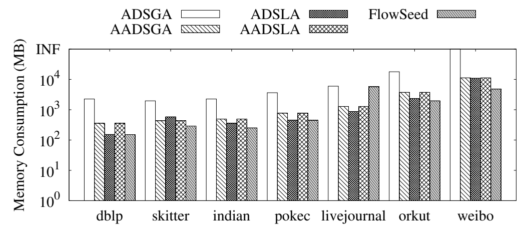


Figure 4: The comparison of memory cost

Datasets. We curate a selection of 7 distinct datasets from two sources: the Network Repository (https://networkrepository.com/) and the Koblenz Network Collection (http://konect.cc/), detailed comprehensively in Table 1. For the purposes of our study, all graphs are treated as undirected and unweighted.

Experiment environment. All algorithms are coded using C++ and compiled with the GCC compiler using O3 optimization. Experiments are conducted on a PC running a Linux operating system, equipped with a 2.2 GHz AMD 3995WX 64-Core CPU and 256 GB of memory. The cutoff time was 1,000 seconds for each query.

Seed vertex set generation. We adopt a query seed set generation approach used in [20]. The process begins with randomly selecting a vertex v from the overall set of vertices. Subsequently, A is constituted by randomly selecting a predefined number of vertices (default is 8) from v's 1-hop and 2-hop neighbors, explicitly including v itself in A. R is then generated for each vertex u in A through several (default: 3) random walks of a specified length (default: 2 steps), aimed at identifying additional members for R. Detailed insights into the query set generation method can be found in [20].

6.2 Performance studies

Exp-1: The runtime and memory of different algorithms. We generate 100 queries for each dataset and perform five algorithms: ADSGA, ADSLA, AADSGA, AADSLA and FlowSeed. The average running time and memory consumption of these algorithms across datasets are illustrated in Figure 3 and Figure 4, respectively. The results show that ADSGA incurs the highest runtime and memory usage across the algorithms evaluated. In contrast, the proposed AADSGA significantly boosts performance, achieving speeds 3 to 13 times faster and consuming 3.5 to 6.5 times less memory than ADSGA. This improvement in efficiency is attributed to the definition of the approximate anchored densest subgraph, necessitating merely integer guesses for α, which aligns with our theoretical analysis in Theorem 3.5. In terms of the local algorithms, AADSLA demonstrably surpasses ADSLA and FlowSeed, exhibiting speedups ranging from 9.5 to 206.5 times and 1.7 to 11 times, respectively, while only slightly increasing memory consumption. This substantial advancement is credited to two primary factors. Firstly, AADSLA reduces the number of LReTest invocations, facilitated

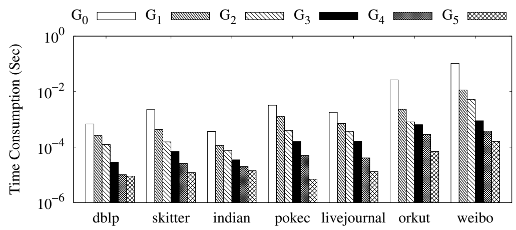


Figure 5: The running time of AADSLA on graph density

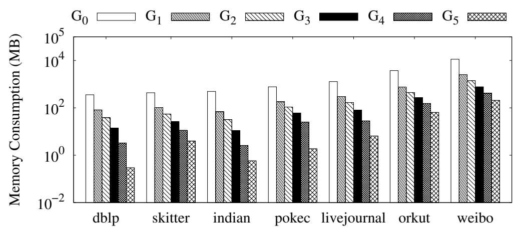


Figure 6: The memory cost of AADSLA on graph density

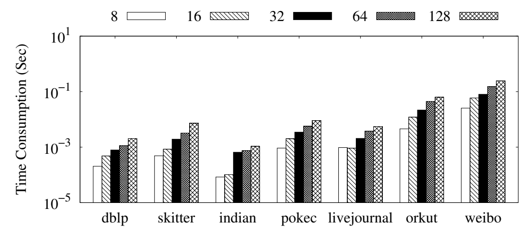


Figure 7: Sensitivity of AADSLA on reference set size

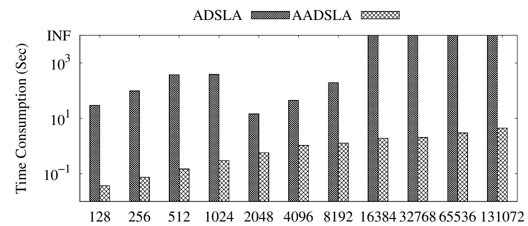


Figure 8: Computation time on larger reference vertex sets

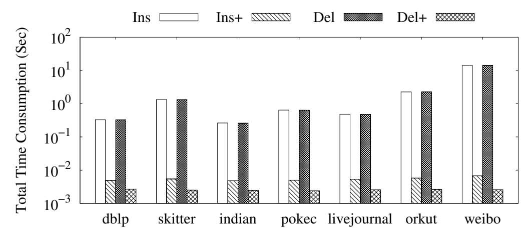


Figure 9: The running time of maintenance algorithms

by adopting integer guesses for α like AADSGA. Secondly, it introduces a "build-as-you-compute" search strategy within LReTest, representing a significant enhancement over the iterative approach utilized by ADSLA and FlowSeed. The results demonstrate the superiority of our AADSGA and AADSLA algorithms.

Additionally, the running time of AADSGA is at least 3 orders of magnitude slower than that of AADSLA in finding the AADS. This lag stems from AADSGA's necessity to construct a complete re-orientation network for each iteration of the probing with α, a requirement not shared by AADSLA. Regarding memory usage, both AADSGA and AADSLA necessitate identical memory allocations, owing to their requirement for linear-sized data structures dedicated to bounties, shortest path distances, and so on. Moreover, the memory consumption of both algorithms scales linearly with the size of the dataset. These results confirm the superior efficiency of AADSLA in comparison to AADSGA.

Exp-2: Sensitivity of AADSLA on graph density. We commence with an original graph, denoted by G_0, and generate five subgraphs, $G_1, \ldots, G_5$, to represent different levels of density. We sample each edge in G_0 with a probability of 0.5^i, and then extract the largest connected component from the edge-induced subgraph as G_i. A suite of 1,000 queries, configured with default parameters, is generated for each G_i. Should a subgraph G_i not surpass the threshold of 128 vertices, it is excluded from our analysis. The average runtime and memory occupancy of AADSLA on the eligible subgraphs are shown in Figure 5 and Figure 6, respectively. It is evident that both time costs and memory consumption of AADSLA increase with the density of the graph. An average escalation in time cost by a factor of 2.85 and in memory requirements by a factor of 3.66 is recorded when the graph density is doubled. These findings highlight the AADSLA algorithm's notable scalability, showcasing its efficiency across various graph densities.

Exp-3: Sensitivity of AADSLA on reference set size. Let k be the cardinality of the reference vertex set R, i.e., $k = |R|$. Here we evaluate the running time of AADSLA by varying k within the set $\{8, 16, 32, 64, 128\}$. We generate the set R with the specified size k using the default method, except that the size of A is resized to $\frac{k}{4}$ when generating A. For each specified k, a total of 100 queries are generated, upon which AADSLA is executed for each dataset. The running time of AADSLA with different k is depicted in Figure 7. It is observed that, across the majority of datasets, the runtime of AADSLA incrementally rises in conjunction with the expansion of R. Remarkably, AADSLA can efficiently search the AADS in under one second within all parameter settings. When the size of $|R|$ is doubled, the average computational overhead incurred by AADSLA increases by a factor of only 0.585. In addition, the runtime of AADSLA and ADSLA with larger k from the set $\{128, 256, ..., 65536, 131072\}$ are depicted in Figure 8. Our AADSLA algorithm efficiently identifies the AADS in under 10 seconds for all values of k. While ADSLA fails to deliver the ADS within the time limitation for larger k. Again, AADSLA achieves a runtime that is an order of magnitude faster than ADSLA. These results demonstrate a significant insensitivity of AADSLA to variations in the size of the reference vertex set, thereby evidencing its robust scalability.

Exp-4: The running time of maintenance algorithms. We generate 10,000 edges at random in each dataset to evaluate our algorithms: Ins and Ins+ for edge insertion, along with Del and Del+ for edge deletion. Given a specific dataset, for a query q_i within Exp-1, where $1 \leq i \leq 100$, the processing time for 10,000 updated edges is denoted as $T(q_i)$. The average time of maintenance algorithms for handling 100 queries, each subjected to 10,000 updates, denoted as $\overline{T} = \frac{1}{100} \sum_{i=1}^{100} T(q_i)$, is presented in Figure 9. Building upon the insights from Exp-1, it becomes apparent that the recomputation time for 10,000 updated edges, as necessitated by algorithms aimed at (approximate) anchored densest subgraph search, significantly exceeds the processing time required by our maintenance algorithms. For instance, on the dblp dataset, the recomputation times mandated by the optimal algorithms, i.e., ADSLA and AADSLA, for insertion or deletion of 10,000 edges, totals to $0.02297 \times 10000 = 229.7$ seconds

Table 2: Comparison between ADS and AADS with different metrics

Dataset	Subgraph	R-subgraph density: $\rho_R(S) = \frac{2\lvert E(S)\rvert - \sum_{v \in S \setminus R} d_G(v)}{\lvert V(S)\rvert}$	Density: $\rho(S) = \frac{\lvert E(S)\rvert}{\lvert V(S)\rvert}$	Local Conductance: $\pi_R(S) = \frac{\lvert E(S,\overline{S})\rvert}{\mathrm{Vol}(R \cap S) - (S \cap \overline{R})}$	Conductance: $\pi(S) = \frac{\lvert E(S,\overline{S})\rvert}{\min(\mathrm{Vol}(S), \mathrm{Vol}(V \setminus S))}$	F_1-score: $F_1(S,R) = \frac{2\lvert S \cap R\rvert}{(\lvert S\rvert + \lvert R\rvert)}$
dblp	ADS	10.543997	5.309548	0.767453	0.765460	0.289283
	AADS	10.527255 **(-0.016742)**	5.299411 **(-0.010137)**	0.770350 **(+0.002897)**	0.768543 **(+0.003083)**	0.303563 **(+0.01428)**
skitter	ADS	10.770263	5.411840	0.920304	0.844299	0.388504
	AADS	10.769975 **(-0.000288)**	5.413363 **(+0.001523)**	0.926973 **(+0.006669)**	0.854317 **(+0.010018)**	0.407545 **(+0.019041)**
indian	ADS	14.143748	7.646244	0.537802	0.513856	0.484087
	AADS	14.110848 **(-0.0329)**	7.653666 **(+0.007422)**	0.543738 **(+0.005936)**	0.520130 **(+0.006274)**	0.523696 **(+0.039609)**
pokec	ADS	6.5524730	3.276936	0.924950	0.924915	0.212633
	AADS	6.5162130 **(-0.03626)**	3.258538 **(-0.018398)**	0.925808 **(+0.000858)**	0.925789 **(+0.000874)**	0.241173 **(+0.02854)**
livejournal	ADS	11.759231	5.899040	0.789256	0.788460	0.286649
	AADS	11.743309 **(-0.015922)**	5.897954 **(-0.001086)**	0.791998 **(+0.002742)**	0.790474 **(+0.002014)**	0.299826 **(+0.013177)**
orkut	ADS	9.6347500	4.817375	0.937277	0.937277	0.209141
	AADS	9.6118910 **(-0.022859)**	4.805945 **(-0.01143)**	0.937234 **(-0.000043)**	0.937234 **(-0.000043)**	0.227251 **(+0.01811)**
weibo	ADS	2.6075910	1.456520	0.991176	0.970494	0.196846
	AADS	2.4221570 **(-0.185434)**	1.296905 **(-0.159615)**	0.994090 **(+0.002914)**	0.993316 **(+0.022822)**	0.447595 **(+0.250749)**

Note: Density and conductance are two baseline community metrics. Density quantifies the average degree of vertices within a subgraph S. Higher density signifies better internal connectivity of S. Conductance evaluates the ratio of the number of edges between a subgraph S and the rest of the graph to the smaller of the sum of vertices' degree in S and its complement. Lower conductance suggests that S is relatively less connected to the rest of the graph. R-subgraph density [20] and local conductance [63] are localized extensions of the density and conductance with respect to a reference set R. The F_1-score uses R as the ground truth to measure the locality; a higher F_1-score indicates that S closely resembles R.

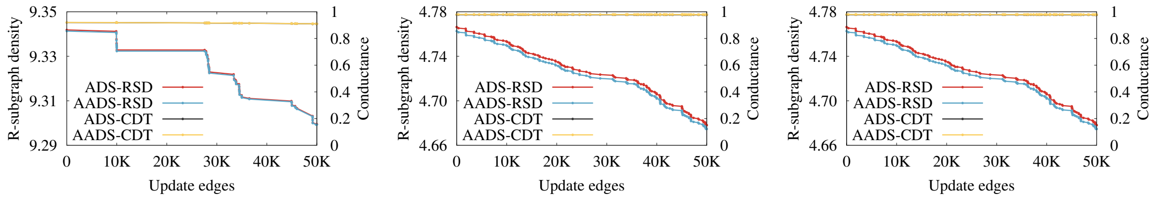
and $0.000683 \times 10,000 = 6.83$ seconds, respectively. Conversely, our Ins and Ins+ algorithms require 0.327774 and 0.004887 seconds, respectively, to maintain the AADS for edge insertion. For edge deletion, Del and Del+ necessitate 0.325508 and 0.002675 seconds, respectively. Moreover, Ins+ (Del+) consistently outperforms Ins (Del) by at least an order of magnitude across all datasets. The comparable processing times for Ins and Del result from both algorithms requiring a mere two max-flow computations. Additionally, the running time of Ins+ and Del+ are comparable in order of magnitude, with the former being slightly longer. These results confirm the efficiency of our algorithms for AADS maintenance.

Exp-5: Comparison of AADS and ADS. We evaluate the ADS and AADS using five distinct metrics to demonstrate the effectiveness of AADS as an approximation of ADS. This assessment generates 100 queries by the procedures detailed in Section 6.1, albeit with modifications to the number of random walks and steps, set to 15 and 4, respectively. Using these queries, the AADSLA and ADSLA algorithms are invoked to search AADS and ADS. For each metric, the average values of AADS and ADS derived from 100 queries are presented in Table 2. Investigations into the R-subgraph density across diverse datasets demonstrate that the divergence between $\rho_R(\hat{S})$ and $\rho_R(S^*)$ invariably remains below 0.2, with $\rho_R(\hat{S})$ consistently larger than $\rho_R(S^*)$. This magnitude of discrepancy is notably smaller than the theoretical maximum difference of 1 suggested by Fact 2.6, affirming the effectiveness of the AADS in approximating the ADS. Concerning metrics of density, local conductance, and conductance, the mean values of AADS over 100 queries are sometimes slightly greater than those of ADS and sometimes slightly less. However, overall, the mean values of AADS and ADS are comparable. Regarding the F_1 score, the average value of AADS slightly exceeds that of ADS across all datasets. The reason for this observation is as follows. When $\lceil \rho_R(\hat{S}) \rceil \geq 2$, AADS contains ADS exactly, and their R-subgraph densities are very close to each other. This indicates that vertices belonging to AADS but not ADS have a higher probability of being included in R, leading to a slightly higher F_1-score for AADS than for ADS. For the case of $\lceil \rho_R(\hat{S}) \rceil \leq 1$, our AADSLA algorithm outputs R directly as AADS, which also results in a slightly higher F_1-score for AADS than for ADS. These results show that AADS is a good approximation to ADS in terms of subgraph density and locality.

Remark. On weibo, the difference $\rho_R(\hat{S}) - \rho_R(S^*)$ is approximately 0.18, which is relatively larger than the differences observed on other datasets. Notably, we find that 60 out of 100 queries satisfy $\lceil \rho_R(\hat{S}) \rceil > 1$, with the average R-subgraph densities of ADS and AADS being 3.135726 and 3.097369, respectively, showing only a small difference of 0.038357. The remaining 40 queries satisfy $\lceil \rho_R(\hat{S}) \rceil = 1$. In these cases, our AADSLA returns R as AADS, while ADSLA searches ADS, leading to a relatively large difference across all 100 queries. However, $\lceil \rho_R(\hat{S}) \rceil = 1$ typically indicates that the ADS is insufficiently dense, rendering it less useful in practice [20]. Our experiments show that when the ADS is denser (i.e., $\rho_R(\hat{S}) > 1$), $\rho_R(\hat{S}) - \rho_R(S^*)$ is very small, indicating that AADS can effectively approximate ADS in real-world applications.

Exp-6: Results on quires corresponding to ADSs with large R-subgraph density. We modify the generation method of the anchor vertex set A in Section 6.1 to produce queries with larger $\rho_R(\hat{S})$. Instead of randomly selecting vertex v, we choose a vertex with higher clustering coefficients from the entire set of vertices. Intuitively, the subgraph containing the set A generated from this vertex is more likely to exhibit a larger $\rho_R(\hat{S})$. For generating R, we use the default parameters of 3 random walks of 2 steps each (i.e., PS1). Considering that a larger $|R|$ may result in an ADS with a higher $\rho_R(\hat{S})$, we also apply another parameter setting: 15 random walks of 4 steps each (i.e., PS2). Eventually, we generate 100 queries with $\rho_R(\hat{S}) \geq 5$ for PS1 and 100 queries with $\rho_R(\hat{S}) \geq 15$ for PS2. Table 3 shows the average R-subgraph density for each dataset under PS1 and PS2. It can be seen that for ADS $\hat{S}$ and AADS S^*, the difference between $\rho_R(\hat{S})$ and $\rho_R(S^*)$ is as low as $0.007 \ll 1$, which again confirms the effectiveness of the AADS in approximating the ADS. Additionally, the average running time of different algorithms across all datasets for PS1 and PS2 are illustrated in Figure 11 and Figure 12, respectively. Our AADSGA algorithm is faster than the ADSGA algorithm, and AADSLA significantly outperforms both ADSLA and FlowSeed for all parameter settings. Again, AADSLA improves upon the AADSGA by at least 3 orders of magnitude. These results are consistent with the previous findings, demonstrating the efficiency of AADSGA and AADSLA.

We also conduct an adversarial experiment on livejournal to further demonstrate the irrelevance of algorithm efficiency to R-subgraph density. Specifically, we set four intervals of R-subgraph density under PS1 and PS2, i.e., $[1, 10]$, $[11, 20]$, $[21, 30]$, $[31, 40]$ and $[11, 20]$, $[21, 30]$, $[31, 40]$, $[41, 50]$, respectively, and generate 100 queries for each interval. Figure 13 depicts the average runtime



(a) WikiElec (b) Epinions (c) Hepph

Figure 10: The R-subgraph densities (RDS) and conductances (CDT) of ADS and AADS on three temporal graphs

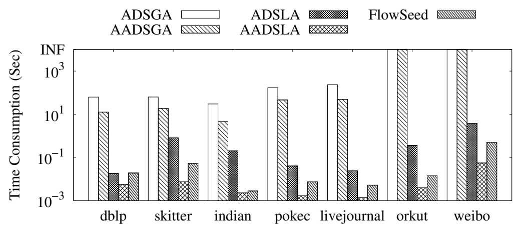


Figure 11: The running time of different algorithms (PS1)

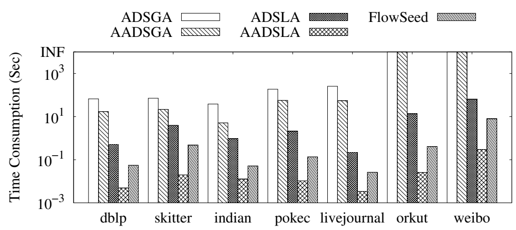


Figure 12: The running time of different algorithms (PS2)

of 100 queries for different algorithms. Consistent with previous findings, our AADSGA and AADSLA algorithms show their superiority in efficiency for all parameter settings. Additionally, the running time of each algorithm in different intervals are of the same order of magnitude, with only minor differences, indicating the insensitivity of these algorithms to the R-subgraph density.

Exp-7: Results on dynamic graphs. Here, we evaluate our maintenance algorithms on temporal graphs, which represent naturally evolving edge updates in real-world scenarios and are inherently dynamic. This experimental setup is widely used in existing studies [17, 22, 23, 25, 34, 40, 42, 43, 52, 55, 59, 64, 69, 70]. Three temporal graphs, WikiElec, Epinions, and Hepph, are used in this experiment, all of which can be downloaded from the Network Repository. To simulate the evolution process of edge deletion, we sort all edges according to their timestamps in ascending order and use the complete graph as the initial graph; then, 50,000 edges are deleted in descending order of their timestamps. For edge insertion, we use the opposite process: the graph consisting of $m - 50,000$ edges is the initial graph, and then 50,000 edges are inserted in ascending order of their timestamps.

As a comparison, since there is no algorithm to maintain ADS directly (except computation from scratch), we use the new graph formed after each edge update as input to ADSLA to compute ADS, which is very costly. Therefore, we roughly estimate the total time required to be $T^* \times 50{,}000$, where T^* is the average runtime of 100 queries per dataset when applying ADSLA. The total runtime for all algorithms to update 50,000 edges in WikiElec, Epinions, and Hepph

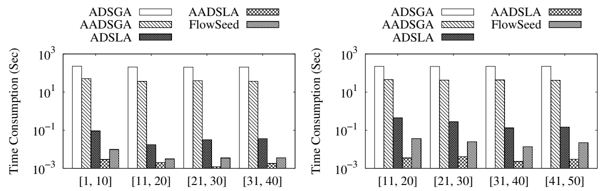
Table 3: R-subgraph densities of subgraphs with PS1 and PS2

Dataset	dblp	skitter	indian	pokec	livejournal	orkut	weibo
ADS(PS1)	8.102760	8.204698	14.558106	7.338885	17.983923	7.003694	6.884859
AADS(PS1)	8.095092	8.201777	14.554896	7.337010	17.983014	7.003446	6.884206
Difference	**0.007668**	**0.002921**	**0.003210**	**0.001875**	**0.000909**	**0.000248**	**0.000653**
ADS(PS2)	28.304661	28.976499	55.890673	19.547565	43.357405	22.910484	16.348741
AADS(PS2)	28.302989	28.975587	55.889356	19.546004	43.356272	22.909059	16.348619
Difference	**0.001672**	**0.000912**	**0.001317**	**0.001561**	**0.001133**	**0.001425**	**0.000122**

is shown in Table 4. Clearly, our proposed maintenance algorithms significantly outperform the re-computation method equipped with ADSLA. The improved algorithms (i.e., Ins+ and Del+) are at least one order of magnitude faster than the basic algorithms (i.e., Ins and Del), again indicating their high efficiency. In addition, we sample the R-subgraph density values and conductance values of AADS S^* and ADS $\hat{S}$ during the edge updating process, as shown in Figure 10. It can be seen that the R-subgraph densities and conductances of AADS and ADS are consistently very close to each other over time for all three datasets, with $\rho_R(S^*)$ being slightly lower than $\rho_R(\hat{S})$. These results again show that AADS is a good approximation of ADS, consistent with the observations obtained from Exp-5.

6.3 Case study

We conduct a case study on a subgraph, dblpCCF, of the dblp dataset, encompassing authors who have published in conferences and journals recommended by the China Computer Federation, along with their collaborative relationships. The dblpCCF subgraph contains 1,207,754 vertices and 5,878,173 edges. To construct the anchored set A and the reference set R, we apply a random walk technique with the parameters set to 15 walks of 4 steps each, following the procedure outlined in Section 6.1. The average runtime of 100 queries for ADSGA and ADSLA are 35.370 seconds and 0.337 seconds, respectively. While AADSGA and AADSLA consume 8.746 seconds and 0.004 seconds, respectively, demonstrating 4x and 72x improvements. Figure 14 shows the ADS and AADS by performing ADSLA and AADSLA, with Lixin Zhou identified as the seed vertex (represented by the red vertex). The yellow vertices, along with the seed vertex, constitute the anchored set A, and all depicted vertices are part of the reference set R. Note that $|A| = 8$, both ADS and AADS implicitly include an isolated vertex within A, which is not shown in Figure 14, in alignment with the definitions. From Figure 14, it is evident that ADS, consisting of 43 vertices, is included in AADS, which includes 61 vertices. All vertices within AADS are elements of the reference set R, and AADS covers a wider subset of vertices in R than ADS. The R-subgraph densities of AADS and ADS are 8.787 and 8.884, respectively, with a marginal difference not surpassing 0.1, which shows that AADS is a good approximation of ADS. Furthermore, Figure 14 distinctly highlights the significant correlation of the additional green vertex on the right side of AADS



(a) livejournal, PS1 (b) livejournal, PS2

Figure 13: The runtime of different algorithms on livejournal

Table 4: Update time on temporal graphs (Second)

Dataset	n	m	ADSLA	Ins	Ins+	Del	Del+
WikiElec	7,116	100,693	1, 200	3.080	**0.015**	3.085	**0.041**
Epinions	131,580	711,210	14, 150	8.044	**0.023**	8.118	**0.093**
Hepph	28,094,	3,148,447	132, 950	25.449	**0.029**	25.640	**0.113**

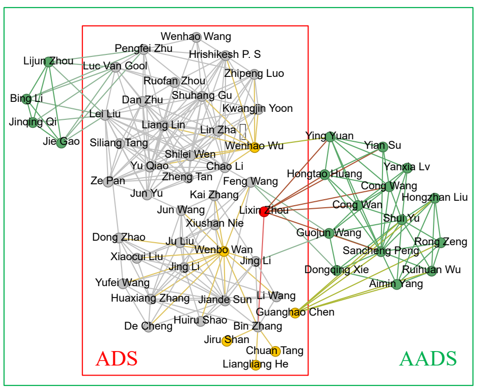


Figure 14: Case study on dblpCCF

with the seed vertex. Similarly, the green vertex positioned in the upper left corner shows a close association with ADS. The results show that AADS is not only a good approximation of ADS, but also better aligns with the reference set R compared to ADS.

7 RELATED WORK

Densest subgraph search. Our work is intricately linked to the Densest Subgraph Search (DSS) problem, which seeks to identify a subgraph exhibiting the highest edge density defined as the ratio of the number of edges to the number of vertices [16, 24, 32]. This problem has been approached through parametric maximum flow methods, achieving a complexity of $O(mn \cdot \log n)$ [32]. However, due to the prohibitive complexity of exact solutions for large-scale graphs, there has been significant advancement in approximation algorithms [8, 16, 24, 60]. Furthermore, the DSS problem has been extensively generalized to various graph types, including weighted [21], directed [16, 38, 44–46], bipartite [4, 33, 49], uncertain [50, 72], and multilayer graphs [30, 31, 36].

A major class of variants of the DSS problem focuses on incorporating additional constraints into edge-density-based DSS problem, including size constraints [5, 11, 14, 27, 38, 56, 66], seed set [20, 26, 58], connectivity constraints [13, 47] and so on. Among them, the seed set-based variation holds particular relevance to our study. Dai *et al.* established the concept of R-subgraph density for a vertex set S, applying penalties to vertices outside a given reference set R, leading to the formulation of the anchored densest subgraph search problem [20]. They proposed a local algorithm to efficiently identify the vertex set S with the maximum R-subgraph density, whose time complexity is independent of the input graph's size. Sozio and Gionis addressed the problem of seeking a set S that includes all query vertices $Q \subseteq V$, with S possessing the highest minimum degree while meeting criteria like the maximum permissible distance between S and Q [58]. They demonstrated the effectiveness of adopting the Charikar greedy peeling algorithm for optimal solutions. Fazzone *et al.* investigated the dense subgraph with attractors and repulsers problem, aiming to identify a dense subgraph S that is proximal to a set of attractors, A, while maintaining distance from a set of repulsers, R [26]. This paper studies the problem of approximate anchored densest subgraph search and maintenance. Due to different problem definitions, the algorithms described in [58] and [26] are not applicable to our problem. The algorithms in [20], while solving our problem, are significantly less efficient when dealing with large-scale graphs or sizable R sets and cannot maintain the ADS for dynamic graphs. In this paper, we present, for the first time, efficient algorithms for searching the AADS on both static and dynamic graphs.

The re-orientation network flow techniques. Our work is also related to the re-orientation network flow techniques, initially developed to address the minimization of the maximum in-degree problem [3, 6, 18]. The re-orientation network flow can also be used to compute pseudoarboricity since the maximum in-degree of the optimal orientation is equal to the pseudoarboricity [10]. Bezáková proposed a renowned method to compute the exact optimal orientation by using the re-orientation network flow technique and binary search, achieving a time complexity of $O(|E|^{3/2} \log p)$ [10]. Blumenstock later enhanced this method, reducing the complexity to $O(|E|^{3/2}\sqrt{\log \log p})$ [12]. In terms of approximation algorithms, Bezáková developed a 2-approximation algorithm with a runtime of $O(m+n)$ [10], while Kowalik proposed a $(1+\epsilon)$-approximation algorithm based on the early-stopped Dinic algorithm, which operates with time complexity of $O(m \log n \max\{1, \log p\}/\epsilon)$ [39]. Additionally, Asahiro investigated the orientation of edges in a weighted graph to minimize the maximum weighted out-degree [7]. To the best of our knowledge, our research constitutes the first application of the re-orientation network flow technique to the search and maintenance of the approximate anchored densest subgraph.

8 CONCLUSION

In this paper, we study the problem of approximate anchored densest subgraph search in static and dynamic graphs, where the R-subgraph densities of the approximate solution and exact anchored densest subgraph are equal after rounding upwards. We propose the AADSGA algorithm equipped with the re-orientation network flow technique and a binary search method. To improve the efficiency, an innovative local algorithm is proposed that utilizes shortest-path based methods to compute the max-flow from s to t around R locally. Furthermore, for dynamic graphs, we develop both basic and improved algorithms aimed at efficiently maintaining the AADS for edge insertions and deletions. Comprehensive experiments and a case study demonstrate the efficiency, scalability, and effectiveness of our solutions.

ACKNOWLEDGMENTS

This work was partially supported by (i) NSFC-Grant U2241211; (ii) the China Postdoctoral Science Foundations 2023TQ0026 and 2023M730251. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Fabeah AduOppong, Casey K Gardiner, Apu Kapadia, and Patrick P Tsang. 2008. Social circles: Tackling privacy in social networks. In *SOUPS*.

[2] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. 2002. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*. 5–16.

[3] Oswin Aichholzer, Franz Aurenhammer, and Gunter Rote. 1995. *Optimal graph orientation with storage applications.* SFB-Report: F003-51, SFB Optimierung und Kontrolle.

[4] Reid Andersen. 2008. A local algorithm for finding dense subgraphs. In *SODA*. 1003–1009.

[5] Reid Andersen and Kumar Chellapilla. 2009. Finding Dense Subgraphs with Size Bounds. In *WAW (Lecture Notes in Computer Science, Vol. 5427)*. 25–37.

[6] Srinivasa Rao Arikati, Anil Maheshwari, and Christos D. Zaroliagis. 1997. Efficient Computation of Implicit Representations of Sparse Graphs. *Discret. Appl. Math.* 78, 1-3 (1997), 1–16.

[7] Yuichi Asahiro, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. 2007. Graph Orientation Algorithms to minimize the Maximum Outdegree. *Int. J. Found. Comput. Sci.* 18, 2 (2007), 197–215.

[8] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest Subgraph in Streaming and MapReduce. *Proc. VLDB Endow.* 5, 5 (2012), 454–465.

[9] Oana Denisa Balalau, Francesco Bonchi, T.H. Hubert Chan, Francesco Gullo, and Mauro Sozio. 2015. Finding Subgraphs with Maximum Total Density and Limited Overlap. In *WSDM*. 379–388.

[10] Ivona Bezakova. 2000. Compact representations of graphs and adjacency testing. (2000).

[11] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. 2010. Detecting high log-densities: an O (n 1/4) approximation for densest k-subgraph. In *STOC*. 201–210.

[12] Markus Blumenstock. 2016. Fast Algorithms for Pseudoarboricity. In *ALENEX*. SIAM, 113–126.

[13] Francesco Bonchi, David GarciaSoriano, Atsushi Miyauchi, and Charalampos E. Tsourakakis. 2021. Finding densest k-connected subgraphs. *Discret. Appl. Math.* 305 (2021), 34–47.

[14] Nicolas Bourgeois, Aristotelis Giannakos, Giorgio Lucarelli, Ioannis Milis, and Vangelis Th. Paschos. 2013. Exact and Approximation Algorithms for Densest *k*-Subgraph. In *WALCOM (Lecture Notes in Computer Science, Vol. 7748)*. 114–125.

[15] Lijun Chang and Miao Qiao. 2020. Deconstruct Densest Subgraphs. In *WWW*. 2747–2753.

[16] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *APPROX 2000 (Lecture Notes in Computer Science, Vol. 1913)*. Springer, 84–95.

[17] Qing Chen, Oded Lachish, Sven Helmer, and Michael H. Bohlen. 2022. Dynamic Spanning Trees for Connectivity Queries on Fully-dynamic Undirected Graphs. *Proc. VLDB Endow.* 15, 11 (2022), 3263–3276.

[18] Marek Chrobak and David Eppstein. 1991. Planar Orientations with Low Outdegree and Compaction of Adjacency Matrices. *Theor. Comput. Sci.* 86, 2 (1991), 243–266.

[19] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[20] Yizhou Dai, Miao Qiao, and Lijun Chang. 2022. Anchored Densest Subgraph. In *SIGMOD*. 1200–1213.

[21] Maximilien Danisch, T.H. Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. In *WWW*. 233–242.

[22] Wenfei Fan, Muyang Liu, Chao Tian, Ruiqi Xu, and Jingren Zhou. 2020. Incrementalization of Graph Partitioning Algorithms. *Proc. VLDB Endow.* 13, 8 (2020), 1261–1274.

[23] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD*. 459–471.

[24] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *Proc. VLDB Endow.* 12, 11 (2019), 1719–1732.

[25] Muhammad Farhan, Qing Wang, and Henning Koehler. 2022. BatchHL: Answering Distance Queries on Batch-Dynamic Networks at Scale. In *SIGMOD*. 2020–2033.

[26] Adriano Fazzone, Tommaso Lanciano, Riccardo Denni, Charalampos E. Tsourakakis, and Francesco Bonchi. 2022. Discovering Polarization Niches via Dense Subgraphs with Attractors and Repulsers. *Proc. VLDB Endow.* 15, 13 (2022), 3883–3896.

[27] Uriel Feige, Guy Kortsarz, and David Peleg. 2001. The Dense *k*-Subgraph Problem. *Algorithmica* 29, 3 (2001), 410–421.

[28] Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. 2006. MotifCut: regulatory motifs finding with maximum density subgraphs. In *ISMB*. 156–157.

[29] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2016. Top-k overlapping densest subgraphs. *Data Min. Knowl. Discov.* 30, 5 (2016), 1134–1165.

[30] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core Decomposition and Densest Subgraph in Multilayer Networks. In *CIKM*. 1807–1816.

[31] Edoardo Galimberti, Francesco Bonchi, Francesco Gullo, and Tommaso Lanciano. 2020. Core Decomposition in Multilayer Networks: Theory, Algorithms, and Applications. *ACM Trans. Knowl. Discov. Data* 14, 1 (2020), 11:1–11:40.

[32] Andrew V Goldberg. 1984. Finding a maximum density subgraph. (1984).

[33] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *KDD*. 895–904.

[34] Guanhao Hou, Qintian Guo, Fangyuan Zhang, Sibo Wang, and Zhewei Wei. 2023. Personalized PageRank on Evolving Graphs with an Incremental Index-Update Scheme. *Proc. ACM Manag. Data* 1, 1 (2023), 25:1–25:26.

[35] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322.

[36] Vinay Jethava and Niko Beerenwinkel. 2015. Finding Dense Subgraphs in Relational Graphs. In *ECML PKDD (Lecture Notes in Computer Science, Vol. 9285)*. Springer, 641–654.

[37] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 2009. 3-HOP: a high-compression indexing scheme for reachability query. In *SIGMOD*. 813–826.

[38] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *ICALP (Lecture Notes in Computer Science, Vol. 5555)*. 597–608.

[39] Lukasz Kowalik. 2006. Approximation Scheme for Lowest Outdegree Orientation and Graph Density Measures. In *ISAAC (Lecture Notes in Computer Science, Vol. 4288)*. Springer, 557–566.

[40] Pei Lee, Laks V. S. Lakshmanan, and Evangelos E. Milios. 2014. Incremental cluster evolution tracking from highly dynamic network data. In *ICDE*. 3–14.

[41] Chengwei Lei and Jianhua Ruan. 2013. A novel link prediction algorithm for reconstructing protein-protein interaction networks by topological similarity. *Bioinform.* 29, 3 (2013), 355–364.

[42] Ziming Li, Youhuan Li, Xinhuan Chen, Lei Zou, Yang Li, Xiaofeng Yang, and Hongbo Jiang. 2024. NewSP: A New Search Process for Continuous Subgraph Matching over Dynamic Graphs. In *ICDE*. 3324–3337.

[43] Xuanming Liu, Tingjian Ge, and Yinghui Wu. 2019. Finding Densest Lasting Subgraphs in Dynamic Graphs: A Stochastic Approach. In *ICDE*. 782–793.

[44] Wensheng Luo, Zhuo Tang, Yixiang Fang, Chenhao Ma, and Xu Zhou. 2023. Scalable Algorithms for Densest Subgraph Discovery. In *ICDE*. 287–300.

[45] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient Algorithms for Densest Subgraph Discovery on Large Directed Graphs. In *SIGMOD*. 1051–1066.

[46] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. On Directed Densest Subgraph Discovery. *ACM Trans. Database Syst.* 46, 4 (2021), 13:1–13:45.

[47] Wolfgang Mader. 1972. Existenz n-fach zusammenhaengender Teilgraphen in Graphen genugend grosser Kantendichte. In *Abhandlungen aus dem mathematischen Seminar der Universitaet Hamburg*, Vol. 37. 86–97.

[48] Alberto O. Mendelzon. 2000. Review - Authoritative Sources in a Hyperlinked Environment. *ACM SIGMOD Digit. Rev.* 2 (2000).

[49] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. 2015. Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling. In *KDD*. 815–824.

[50] Atsushi Miyauchi and Akiko Takeda. 2018. Robust Densest Subgraph Discovery. In *ICDM*. 1188–1193.

[51] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.

[52] Yue Pang, Lei Zou, and Yu Liu. 2023. IFCA: Index-Free Community-Aware Reachability Processing Over Large Dynamic Graphs. In *ICDE*. 2220–2234.

[53] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. 2010. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. In *PAKDD (Lecture Notes in Computer Science, Vol. 6119)*. Springer, 435–448.

[54] Lu Qin, RongHua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally Densest Subgraph Discovery. In *KDD*. 965–974.

[55] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time Constrained Cycle Detection in Large Dynamic Graphs. *Proc. VLDB Endow.* 11, 12 (2018), 1876–1888.

[56] Frederic Roupin and Alain Billionnet. 2008. A deterministic approximation algorithm for the densest k-subgraph problem. *International Journal of Operational Research* 3, 3 (2008), 301–314.

[57] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and XiaoNing Zhang. 2010. Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs. In *RECOMB (Lecture Notes in Computer Science, Vol. 6044)*. 456–472.

[58] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *KDD*. ACM, 939–948.

[59] Kumar Sricharan and Kamalika Das. 2014. Localizing anomalous changes in time-evolving graphs. In *SIGMOD*. 1347–1358.

[60] HsinHao Su and Hoa T. Vu. 2020. Distributed Dense Subgraph Detection and Low Outdegree Orientation. In *DISC (LIPIcs, Vol. 179)*. 15:1–15:18.

[61] Nikolaj Tatti and Aristides Gionis. 2013. Discovering Nested Communities. In *ECML PKDD (Lecture Notes in Computer Science, Vol. 8189)*. 32–47.

[62] Elena Valari, Maria Kontaki, and Apostolos N. Papadopoulos. 2012. Discovery of Top-k Dense Subgraphs in Dynamic Graph Collections. In *SSDBM (Lecture Notes in Computer Science, Vol. 7338)*. 213–230.

[63] Nate Veldt, Christine Klymko, and David F. Gleich. 2019. Flow-Based Local Graph Clustering with Better Seed Set Inclusion. In *ICDM*. 378–386.

[64] Victor Junqiu Wei, Raymond ChiWing Wong, and Cheng Long. 2020. Architecture-Intact Oracle for Fastest Path and Time Queries on Dynamic Spatial Networks. In *SIGMOD*. 1841–1856.

[65] A YEFIM. 1970. DINITZ: Algorithm for solution of a problem of maximum flow in a network with power estimation. In *Soviet Math. Doklady*, Vol. 11. 1277–1280.

[66] Peng Zhang and Zhendong Liu. 2021. Approximating Max k-Uncut via LP-rounding plus greed, with applications to Densest k-Subgraph. *Theor. Comput. Sci.* 849 (2021), 173–183.

[67] Qi Zhang, Yalong Zhang, RongHua Li, and Guoren Wang. 2024. Approximate Anchored Densest Subgraph Search on Large Static and Dynamic Graphs. *Full version: https://github.com/qizhang1996/aads* (2024).

[68] Yang Zhang and Srinivasan Parthasarathy. 2012. Extracting Analyzing and Visualizing Triangle K-Core Motifs within Networks. In *ICDE*. 1049–1060.

[69] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and Efficiency of Truss Maintenance in Evolving Graphs. In *SIGMOD*. 1024–1041.

[70] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A Fast Order-Based Approach for Core Maintenance. In *ICDE*. 337–348.

[71] Feng Zhao and Anthony K. H. Tung. 2012. Large Scale Cohesive Subgraphs Discovery for Social Network Visual Analysis. *Proc. VLDB Endow.* 6, 2 (2012), 85–96.

[72] Zhaonian Zou. 2013. Polynomial-time algorithm for finding densest subgraphs in uncertain graphs. In *Proceedings of MLG Workshop*.