Uncovering the Largest Community in Social Networks at Scale

Shohei Matsugu[1] , **Yasuhiro Fujiwara**[2] and **Hiroaki Shiokawa**[3]
[1]Graduate School of Science and Technology, University of Tsukuba, Japan
[2]NTT Communication Science Laboratories, Japan
[3]Center for Computational Sciences, University of Tsukuba, Japan
matsugu@kde.cs.tsukuba.ac.jp, yasuhiro.fujiwara.kh@hco.ntt.co.jp, shiokawa@cs.tsukuba.ac.jp

Abstract

The Maximum *k*-Plex Search (MPS) can find the largest *k*-plex, which is a generalization of the largest clique. Although MPS is commonly used in AI to effectively discover real-world communities of social networks, existing MPS algorithms suffer from high computational costs because they iteratively scan numerous nodes to find the largest *k*-plex. Here, we present an efficient MPS algorithm called *Branch-and-Merge (BnM)*, which outputs an exact maximum *k*-plex. BnM merges unnecessary nodes to explore a smaller graph than the original one. Extensive evaluations on real-world social networks demonstrate that BnM significantly outperforms other state-of-the-art MPS algorithms in terms of running time.

1 Introduction

The largest community discovery is an essential tool to understand user behaviors in social networks [Shiokawa *et al.*, 2013; Shiokawa *et al.*, 2019]. To this end, traditional applications employ the maximum clique search [Tomita and Seki, 2003] or the maximum *k*-core search [Seidman, 1983]. However, these searches are difficult to apply to real-world social networks for two reasons. First, social networks have diverse densities due to the scale-free and the power-law properties [Faloutsos *et al.*, 1999]. Second, the identified subgraphs using the above search methods tend to be too dense for actual communities [Matsugu *et al.*, 2021]. Consequently, these methods fail to reproduce the largest community in real-world social networks [Pattillo *et al.*, 2013].

To overcome these limitations, the Maximum *k*-Plex Search (MPS) [Balasundaram *et al.*, 2007] has recently attracted much attention in the AI community [Conte *et al.*, 2017; Conte *et al.*, 2018]. MPS finds the maximum *k*-plex, which is the largest subgraph containing n nodes where each node is adjacent to at least $n - k$ nodes within the subgraph [Seidman, Stephen B and Foster, Brian L, 1978]. Unlike traditional methods, MPS can balance the density and the size of a discovered subgraph to meet the diverse density of communities in social networks [Leskovec and Krevl, 2014]. As described below, MPS has recently been applied in various AI-powered applications:

Criminal network analysis: MPS effectively uncovers suspicious communications among terrorist cells, criminal gangs, and religious cults in social networks. For example, Wiil *et al.* applied MPS to public social network data to detect terrorist groups engaged in the September 11 attacks on the World Trade Center in 2001 [Wiil *et al.*, 2010]. MPS uncovered more terrorists than other dense subgraph models. Similarly, MPS achieved a better performance in detecting narcotics trades and money laundering in criminal networks than the maximum clique model [Balasundaram, 2007].

Personalized social search: MPS can enhance a personalized social search, which reranks Web search results based on friendship communities in social networks [Noll and Meinel, 2007; Carmel *et al.*, 2009]. Danezis et al. proposed an MPS-based algorithm that estimates search preferences shared in the largest friendship community [Danezis *et al.*, 2010]. The algorithm first extracts the maximum *k*-plex from a social network. It infers preferences in the *k*-plex via Bayesian models. Their MPS-based algorithm achieved more accurate search results than other dense subgraph models.

Graph neural network (GNN): MPS contributes to enhanced performances of GNN which has played a key role in social network analysis. In GNN, graph pooling is essential to capture the context of nodes, and many clique-based methods have been used [Ying *et al.*, 2018; Lee *et al.*, 2019; Wang *et al.*, 2020]. Bianchi et al. recently proposed an MPS-based graph pooling method [Bianchi *et al.*, 2020]. They reported that the MPS-based method successfully mitigated the oversmoothing issues in GNN [Li *et al.*, 2018], improving the accuracy compared to other GNN models.

MPS can also be used in other applications such as viral marketing [Doerr *et al.*, 2012; Bodaghi and Oliveira, 2022], disease propagation analysis [Wang *et al.*, 2003; Wang and Dai, 2008], etc. Due to space limitations, we omit the details here.

Although MPS is effective for many AI-powered applications, it has an NP-hard complexity [Balasundaram *et al.*, 2007]. Specifically, the naïve MPS algorithm requires $O(2^{|V_G|}\ |V_G|)$ time to find the maximum *k*-plex [Balasundaram *et al.*, 2007], where V_G is a node set. In the 2010s, MPS was applied to graphs composed of only 100 nodes. By contrast, recent AI-based applications handle massive graphs with at least 10,000 nodes [Jiang *et al.*, 2021]. Hence, MPS requires several days to obtain the maximum *k*-plex.

1.1 Existing Approaches and Challenges

Many studies have improved the efficiency of MPS [Conte *et al.*, 2018; Zhu *et al.*, 2020]. Here, we focus on those most related to this study. For a broader discussion, please refer the Section 5. Classical algorithms include *greedy branching* [Moser *et al.*, 2012], *branch-and-cut* [Balasundaram *et al.*, 2007], and *branch-and-search* [Xiao *et al.*, 2017]. To improve the efficiency, these algorithms attempt to reduce candidates of the maximum *k*-plex by bounding the *k*-plex size. Although they are faster than the naïve MPS algorithm, quickly computing graphs with 10,000 nodes remains a challenge since they must explore numerous nodes due to the large search space. Hence, classical MPS algorithms fail to output the maximum *k*-plex in a reasonable time.

Gao *et al.* recently proposed *branch-and-bound* (BnB) as an alternative [Gao *et al.*, 2018]. The key idea of BnB is to avoid unnecessary searches using *graph reduction algorithms*, which remove nodes that are unlikely to be the largest *k*-plex. Based on these reduction algorithms, BnB achieves a faster search than the classical ones. However, BnB still requires expensive costs to handle massive graphs due to insufficient node removal. To overcome this issue, several algorithms have introduced heuristic upper bounding methods [Zhou *et al.*, 2021; Jiang *et al.*, 2021; Chang *et al.*, 2022]. Unfortunately, they are also unsuitable for large social networks since their graph reduction algorithms must scan the whole graph repeatedly. For example, the state-of-the-art method [Chang *et al.*, 2022], incurs $O(|V_G|^2)$ time to reduce the graph size. Thus, it is still a challenging task to develop an efficient MPS algorithm for massive social networks.

1.2 Our Approaches and Contributions

Our goal is to develop an efficient MPS algorithm for large social networks with million-scale nodes. Here, we present a novel algorithm called *Branch-and-Merge* (BnM). The existing approaches discussed in Section 1.1 suffer from a quadratic cost against the number of nodes since they must repeatedly scan all nodes to reduce the graph size. In contrast, the basic idea underlying BnM is to reduce the graph size by scanning only unpromising nodes. In social networks, many nodes are not included in the maximum *k*-plex due to the well-known scale-free property. Since BnM can efficiently reduce the graph size by removing unpromising nodes, it avoids redundant iterative traversals of all nodes.

We derived BnM using three steps: First, we theoretically derive *safe-to-merge nodes*, which are never included in the maximum *k*-plex together. (Section 3.2). Second, we provide a *node-merging strategy* to efficiently find safe-to-merge nodes (Section 3.3). Finally, we propose BnM using a node-merging strategy to efficiently find the maximum *k*-plex (Section 3.4). Furthermore, we theoretically show that the proposed node-merging strategy more efficiently reduces the graph size than state-of-the-art graph reduction algorithms. As a result, BnM has the following attractive characteristics:

- **Efficiency:** BnM is faster than recently proposed MPS algorithms (Section 4.1). BnM has a better time complexity than the state-of-the-art methods (Theorem 1).
- **Exactness:** BnM always outputs the maximum *k*-plex because it does not sacrifice the search quality of MPS algorithms (Theorem 2).
- **Scalability:** BnM is more scalable than other MPS algorithms (Section 4.3). BnM shows nearly linear scalability against the number of nodes in a graph.

BnM is the first solution that achieves an efficient and exact MPS on massive social networks. BnM outperforms other state-of-the-art MPS algorithms in terms of running time. For example, BnM finds the maximum *k*-plex from a scale-free graph that has 1.1 million nodes within 7 seconds, while the state-of-the-art algorithm [Chang *et al.*, 2022] does not output a result within 100 seconds. Therefore, BnM can improve the quality in a wide variety of AI-based applications.

Symbol	Definition
G	An unweighted, undirected, and connected graph
V_G	A set of nodes in G
E_G	A set of edges in G
$d_G(u)$	Degree of a node u in G
$G[V_S]$	An induced subgraph of G ($V_S \subseteq V_G$)
G_{-v}	Remaining graph after removing v from G
α	The size of searching *k*-plex
U_G	A set of unfruitful nodes in G
$\hat{G}$	A converged graph of G
$F_{\hat{G}}$	A set of fragile nodes of $\hat{G}$
M	A set of safe-to-merge nodes
G^*	A graph after merging sage-to-merge nodes

Table 1: Definitions of the main symbols

2 Preliminaries

First, we define the basic notations used in this paper. Table 1 summarized the main symbols and their definitions. Similar to previous studies [Gao *et al.*, 2018; Zhou *et al.*, 2021; Jiang *et al.*, 2021], let $G = (V_G, E_G)$ be an unweighted, undirected, and connected graph, where V_G and E_G are sets of nodes and edges in G, respectively. The degree of node u in G is denoted as $d_G(u)$. $G[V_S]$ represents an induced subgraph of G, where $V_S \subseteq V_G$. G_{-v} is defined as the remaining graph after removing node v from G (*i.e.,* $G_{-v} = G[V_G \backslash \{v\}]$).

MPS is a task to find the largest *k*-plex in a graph G, where *k*-plex is a generalization of a clique proposed by [Abello *et al.*, 2002]. Formally, *k*-plex is defined as:

Definition 1 (*k*-plex)**.** *Given a graph G, and a positive integer k, a subgraph $G[V_S] \subseteq G$ is a k-plex if $d_{G[V_S]}(v) \geq |V_S| - k$ for all $v \in V_S$.*

If $G[V_S]$ is a *k*-plex, then each node $v \in V_S$ is adjacent to at least $|V_S| - k$ nodes. Thus, the MPS is defined as:

Problem 1 (MPS)**.** *Given a positive integer k, MPS is a task to find a k-plex $G[V_S] \subseteq G$ that maximizes $|V_S|$.*

If $k = 1$, Problem 1 is clearly equivalent to the maximum clique search [Bomze *et al.*, 1999]. Since MPS is NP-hard [Abello *et al.*, 2002; McClosky and Hicks, 2012; Xiao and Kou, 2017], an efficient algorithm must be developed to compute large social networks.

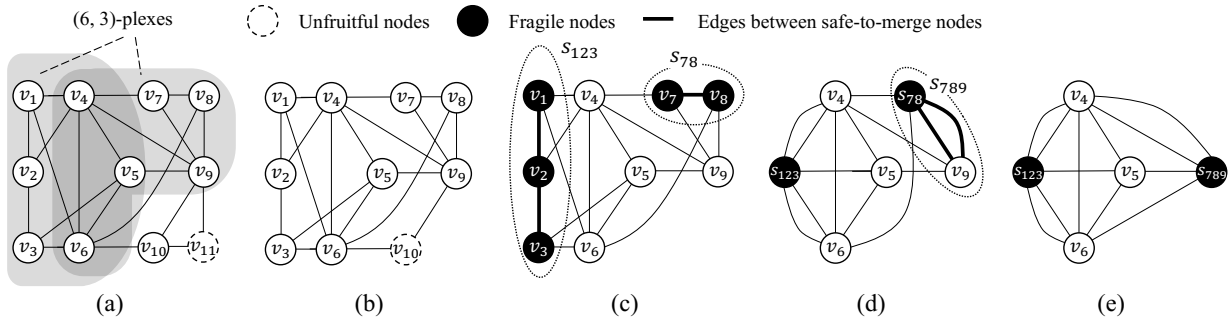


Figure 1: Running example of the node-merging strategy with $\alpha = 6$ and $k = 3$. s_{123}, s_{78}, and s_{789} are supernodes.

3 Branch-and-Merge (BnM)

This paper proposes BnM to efficiently find the maximum k-plex. In this section, we overview the ideas underlying BnM and then provide a full description.

3.1 Overview

The existing approaches discussed in Section 1.1 iteratively run graph reduction algorithms to remove the unpromising nodes. These reduction algorithms require a quadratic cost against the number of nodes since all nodes are scanned repeatedly. In contrast, BnM can efficiently reduce the graph size by scanning the unpromising nodes. Herein, we first theoretically identify *safe-to-merge nodes*, which are not included together in the maximum k-plex (Section 3.2). We then introduce a *node-merging strategy* to efficiently remove the safe-to-merge nodes from a graph (Section 3.3). Finally, we describe BnM to drastically reduce the graph size and efficiently explore the maximum k-plex (Section 3.4).

Our ideas have two advantages. First, BnM is more efficient than existing approaches that repeatedly scan all nodes. Social networks contain many unpromising low-degree nodes due to the scale-free property [Faloutsos *et al.*, 1999]. Because BnM scans only adjacent nodes for each unpromising node to reduce the graph size, it can drastically decrease the graph size on social networks. Second, BnM always outputs exact results, although it removes unpromising nodes via a node-merging strategy. This is because BnM detects nodes not included in the maximum k-plex while performing node merging. Hence, BnM does not sacrifice the k-plex search quality. Consequently, BnM can efficiently find the exact maximum k-plex included in massive social networks.

3.2 Safe-to-merge Nodes

BnM reduces the graph size by removing nodes not included together in the maximum k-plex. We call such nodes *safe-to-merge nodes*. To theoretically derive the safe-to-merge nodes, we introduce Definitions 2, 3, and 4:

Definition 2 ((α, k)-plex)**.** *A subgraph $G[V_S] \subseteq G$ is an (α, k)-plex in a graph G if $G[V_S]$ is a k-plex such that $|V_S| = \alpha$.*

The (α, k)-plex is a k-plex whose size is α. For a given α, a graph can have multiple (α, k)-plexes. For instance, the graph in Figure 1 (a) has two (6, 3)-plexes: $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $\{v_4, v_5, v_6, v_7, v_8, v_9\}$.

Based on Definition 2, *unfruitful nodes*, which cannot be included in any (α, k)-plexes, are defined as:

Definition 3 (Unfruitful Nodes)**.** *For α and k, unfruitful nodes U_G are defined as $U_G = \{v \in V_G \mid d_G(v) < \alpha - k\}$.*

Definition 3 indicates that unfruitful nodes have a degree less than $\alpha - k$. From Definitions 2 and 3, they are never included in any (α, k)-plexes. In Figure 1 (a), the set of unfruitful nodes is $U_G = \{v_{11}\}$ since $d_G(v_{11}) = 2 < \alpha - k = 3$.

Unfruitful nodes can be removed from a graph G since they are not included in any (α, k)-plexes. Because the node degrees decrease after node removal, G can have additional unfruitful nodes. Thus, we define a *converged graph*, which is obtained after removing all unfruitful nodes in G.

Definition 4 (Converged Graph)**.** *For α and k, let $G^{(i)}$ be the i-th remaining graph after removing unfruitful nodes as*

$$G^{(i)} = \begin{cases} G[V_G \backslash U_G] & (i = 1) \\ G[V_{G^{(i-1)}} \backslash U_{G^{(i-1)}}] & (i > 1). \end{cases}$$

A converged graph $\hat{G}$ is defined as $\hat{G} = G[V_{G^{(i)}} \backslash U_{G^{(i)}}]$ such that $U_{G^{(i)}} = \emptyset$.

As shown in Definition 4, a converged graph $\hat{G}$ has no unfruitful nodes for any (α, k)-plexes. Suppose that the graph in Figure 1 (a) is used to generate the converged graph for (6, 3)-plexes. Recall that it has unfruitful nodes $U_G = \{v_{11}\}$. Figure 1 (b) shows $G^{(1)}$, which is the graph after removing U_G from G. From Definition 3, $G^{(1)}$ also has a set of unfruitful nodes $U_{G^{(1)}} = \{v_{10}\}$. Figure 1 (c) shows graph $G^{(2)}$, which has no unfruitful nodes since all nodes have at least $\alpha - k = 3$ neighbor nodes. Hence, $G^{(2)}$ is the converged graph (*i.e.,* $\hat{G} = G^{(2)}$).

We finally derive the safe-to-merge nodes using Definitions 2, 3, and 4 as:

Definition 5 (Safe-to-Merge Nodes)**.** *Given two adjacent nodes $u, v \in \hat{G}$, nodes u and v are safe-to-merge nodes and denoted as $u \leftrightarrow_{\hat{G}} v$, iff $u \in U_{\hat{G}_{-v}}$ and $v \in U_{\hat{G}_{-u}}$.*

Definition 5 indicates that node u becomes an unfruitful node after removing node v from $\hat{G}$ if and only if $u \leftrightarrow_{\hat{G}} v$ holds, and vice versa. In Figure 1 (c), the bold line links the safe-to-merge nodes. For example, two adjacent nodes v_7 and v_8 are the safe-to-merge nodes for (6, 3)-plexes because the degree

Algorithm 1 Node-merging algorithm

Input: A converged graph $\hat{G}$;
Output: A graph G^* after merging safe-to-merge nodes;

1: **procedure** MERGE ($\hat{G}, \alpha, k$):
2: $G^* \leftarrow \hat{G}$;
3: $F_{G^*} \leftarrow \{v \in V_{G^*} \mid d_{G^*}(v) = \alpha - k\}$;
4: $M \leftarrow$ a connected node set in $G[F_{G^*}]$;
5: **while** $M \neq \emptyset$ **do**
6: Add a supernode s to V_{G^*};
7: **for each** $v \in M$ **do**
8: **for each** $u \in N(v) \backslash M$ **do**
9: Add an edge (u, s) to E_{G^*};
10: Remove v from V_{G^*};
11: $F_{G^*} \leftarrow F_{G^*} \cup \{s\} \backslash M$;
12: $M \leftarrow \emptyset$;
13: **for each** $v \in N_{G^*}(s)$ **do**
14: **if** $v \leftrightarrow_{G^*} s$ **then**
15: $M \leftarrow M \cup \{v\}$;
16: **if** $M = \emptyset$ **then**
17: $M \leftarrow$ a connected node set in $G[F_{G^*}]$;
18: **return** G^*;

of node v_7 becomes 2 in G_{-v_8}, which is smaller than $\alpha - k = 3$. Additionally, node v_8 has a degree smaller than 3 in G_{-v_7}. For Definition 5, we derive the following lemma:

Lemma 1. *For $u \leftrightarrow_{\hat{G}} v$, node u is not included in an (α, k)-plex if node v is not included in the (α, k)-plex.*

Proof. We prove by contradiction. Given safe-to-merge nodes $u \leftrightarrow_{\hat{G}} v$, we assume that u is included in an (α, k)-plex if v is not included in the (α, k)-plex. That is, we have $u \notin U_{\hat{G}_{-v}}$. From Definition 5, this is a clear contradiction since $u \leftrightarrow_{\hat{G}} v \Rightarrow u \in U_{\hat{G}_{-v}}$ holds. This completes the proof of Lemma . □

Lemma 3.2 indicates that safe-to-merge nodes never become members of an (α, k)-plex if one of them is not included in the (α, k)-plex. Conversely, they are members of an (α, k)-plex if one is included in the (α, k)-plex. Thus, removing the safe-to-merge nodes can effectively reduce the graph size if one of them is not included in the (α, k)-plex. However, it needs $O(|V_{\hat{G}}|^2)$ time to list the safe-to-merge nodes for exploring all node pairs. To avoid this, we propose a node-merging strategy that dynamically merges unnecessary nodes.

3.3 Node-merging Strategy

We introduce a *node-merging strategy* to efficiently find safe-to-merge nodes from a converged graph. We overview a key property underlying our strategy and then provide its details.

Node-merging Property To find the safe-to-merge nodes efficiently, we present the node-merging property. First, we define *fragile nodes* for a converged graph $\hat{G}$ as:

Definition 6 (Fragile Nodes)**.** *Given a converged graph $\hat{G}$, fragile nodes $F_{\hat{G}}$ are defined as:*

$$F_{\hat{G}} = \{v \in V_{\hat{G}} \mid d_{\hat{G}}(v) = \alpha - k\}.$$

In Figure 1 (c), fragile nodes are those whose degree is $\alpha - k = 3$. Thus, $\hat{G}$ has $F_{\hat{G}} = \{v_1, v_2, v_3, v_7, v_8\}$. Fragile nodes become unfruitful if one of their adjacent nodes is removed since the removed node decreases the degree of the fragile nodes. In Figure 1 (c), v_7 is unfruitful after removing v_8 from $\hat{G}$. Formally, safe-to-merge nodes can be obtained from $G[F_{\hat{G}}]$ using the following property:

Lemma 2. *Let $G[F_{\hat{G}}]$ be the induced subgraph comprised of fragile nodes. If a path $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_N$ exists in $G[F_{\hat{G}}]$, $v_i \leftrightarrow_{\hat{G}} v_{i+1}$ always holds for $i \in \{1, 2, ..., N-1\}$.*

Proof. From Definition 5, $d_{\hat{G}}(v_{i+1}) = \alpha - k$ holds. If v_i is removed from $\hat{G}$, we have $d_{\hat{G}_{-v_i}}(v_{i+1}) = \alpha - k - 1$ since $(v_i, v_{i+1}) \in E_{\hat{G}}$. From Definition 3, we have $v_{i+1} \in U_{\hat{G}_{-v_i}}$. Furthermore, $v_i \in U_{\hat{G}_{-v_{i+1}}}$ holds since we have $d_{\hat{G}}(v_i) = \alpha - k$. Hence, $v_i \leftrightarrow_{\hat{G}} v_{i+1}$ holds from Definition 5. □

If nodes are connected in the induced subgraph composed of fragile nodes, they are safe-to-merge nodes. In Figure 1 (c), nodes v_1, v_2, and v_3 are included in $F_{\hat{G}}$. Because there is a path of $v_1 \rightarrow v_2 \rightarrow v_3$ in $G[F_{\hat{G}}]$, $v_1 \leftrightarrow_{\hat{G}} v_2$ and $v_2 \leftrightarrow_{\hat{G}} v_3$ hold. Clearly, we can find the path in $O(|V_{\hat{G}}|)$ time, at most, by obtaining the fragile nodes. Hence, Lemma 2 can be used to efficiently find safe-to-merge nodes.

Node-merging Algorithm Our algorithm efficiently finds safe-to-merge nodes by initially enumerating all safe-to-merge nodes based on Lemma 2. Then the nodes are removed based on Lemma 3.2 by merging the nodes to a *supernode*. This efficiently reduces the graph size. Here, we formally define a *supernode* as:

Definition 7 (Supernode)**.** *Given a graph $\hat{G}$ and a set of connected safe-to-merge nodes $M \in G[F_{\hat{G}}]$, a supernode s is defined as a node obtained after merging nodes in M. Here, s is adjacent to nodes in $\{v \in V_{\hat{G}} \mid \forall u \in M, (u, v) \in E_{\hat{G}}\}$. We also define its degree $d_{\hat{G}}(s)$ as $\alpha - k$. By merging M, nodes of M and edges linked to them are removed from $\hat{G}$. $|s| = |M|$ denotes the number of merged nodes.*

A supernode is a special class of nodes in which safe-to-merge nodes are represented as a single node. The degree of a supernode is always $\alpha - k$ since all merged nodes are fragile nodes. Additionally, a supernode can be linked by several edges since edges in M are reconnected to the supernode. For example, in Figure 1 (d), two edges (v_1, v_4) and (v_2, v_4) are reconnected as (s_{123}, v_4) after merging $\{v_1, v_2, v_3\}$ to s_{123}.

If $u, v \in F_{\hat{G}}$ are connected in $G[F_{\hat{G}}]$, they are safe-to-merge nodes based on Lemma 2. Using this property, Algorithm 1 is designed to output a graph after merging the nodes by finding the safe-to-merge nodes in $\hat{G}$. Algorithm 1 first constructs fragile nodes F_{G^*}. Then it finds a connected node set M in $G[F_{G^*}]$ using depth-first-search (DFS) [1] based on Lemma 2 (lines 3–4). For each connected node set M, Algorithm 1 replaces M with the corresponding supernode s using Definition 7 (lines 6–10). Then it adds s to F_{G^*} since the degree of s is $\alpha - k$ (line 11). Let $N_{G^*}(u) = \{v \in V_{G^*} \mid (u, v) \in E_{G^*}\}$ be a set of adjacent nodes of node u in G^*.

[1] Any search method is acceptable if it yields equivalent search results as DFS.

Algorithm 2 Proposed method: Branch-and-Merge (BnM)

Input: A given graph G, a positive integer k;
Output: A maximum k-plex G_{max};

1: $\hat{G} \leftarrow G, G_{max} \leftarrow \emptyset, \alpha \leftarrow k+1$;
2: **while true do**
3: $\quad G_{curr} \leftarrow$ BRANCH $(\hat{G}, \alpha, k)$;
4: $\quad$ **if** $G_{curr} = \emptyset$ **then**
5: $\quad\quad$ **return** G_{max};
6: $\quad \hat{G} \leftarrow G[V_{\hat{G}} \backslash F_{\hat{G}}]$;
7: $\quad G_{max} \leftarrow G_{curr}$;
8: $\quad \alpha \leftarrow \alpha + 1$;
9:
10: **procedure** BRANCH $(\hat{G}, \alpha, k)$
11: $\quad \hat{G} \leftarrow$ a converged graph of $\hat{G}$ for α and k;
12: $\quad n \leftarrow 0$;
13: $\quad$ **for each** $v \in V_{\hat{G}}$ **do**
14: $\quad\quad$ **if** v is a supernode **then**
15: $\quad\quad\quad n \leftarrow n + |v|$;
16: $\quad\quad$ **else**
17: $\quad\quad\quad n \leftarrow n + 1$;
18: $\quad$ **if** $n = \alpha$ **then**
19: $\quad\quad$ **return** $\hat{G}$;
20: $\quad$ **if** $n < \alpha$ **then**
21: $\quad\quad$ **return** $\emptyset$;
22: $\quad G^* \leftarrow$MERGE $(\hat{G}, \alpha, k)$;
23: $\quad G_{curr} \leftarrow \emptyset$;
24: $\quad$ **for each** $v \in V_{G^*}$ **do**
25: $\quad\quad G_{curr} \leftarrow$ BRANCH (G^*_{-v}, α, k);
26: $\quad\quad$ **if** $G_{curr} \neq \emptyset$ **then**
27: $\quad\quad\quad$ **return** G_{curr};
28: $\quad$ **return** $\emptyset$;

Then Algorithm 1 updates the connected node set M by exploring the nodes in $N_{G^*}(s)$. If N_{G^*} has the safe-to-merge nodes to s, Algorithm 1 adds the nodes to M (lines 13–15). Otherwise, it explores another connected node set (lines 16–17). Finally, Algorithm 1 is terminated if no safe-to-merge nodes are found (lines 5–17).

Figures 1 (c), (d), and (e) illustrate a running example of Algorithm 1. Figure 1 (c) shows that the converged graph $\hat{G}$ has $F_{\hat{G}} = \{v_1, v_2, v_3, v_7, v_8\}$. By merging $\{v_7, v_8\}$, Algorithm 1 adds supernode s_{78} as shown in Figure 1 (d). Based on Definition 7, the edges (v_7, v_4), (v_7, v_9), (v_8, v_6), and (v_8, v_9) are replaced by (s_{78}, v_4), (s_{78}, v_9), (s_{78}, v_6), and (s_{78}, v_9), respectively. Then Algorithm 1 merges $\{v_1, v_2, v_3\}$ into s_{123}. In Figure 1 (d), since s_{78} and v_9 are linked by two edges, they become the safe-to-merge nodes by Definition 5. Hence, our algorithm merges them into s_{789} in Figure 1 (e), and the graph has no safe-to-merge nodes.

Finally, we theoretically assess the time complexity to merge all safe-to-merge nodes in $\hat{G}$ using Algorithm 1.

Lemma 3. *Algorithm 1 incurs $O(|F_{\hat{G}}| \cdot (\alpha - k))$ time.*

Proof. Algorithm 1 can be split into two processes: finding the safe-to-merge nodes (lines 4, 12–17) and reconnecting the edges to merge the nodes (lines 5–9). We first discuss the time complexity to find all safe-to-merge nodes. As proved in Lemma 2, if a path exists in $G[F_{\hat{G}}]$, the nodes in the path are safe-to-merge nodes. Thus, we employ the depth-first search (DFS) to enumerate the connected node set in $G[F_{\hat{G}}]$. We start DFS from a fragile node in $F_{\hat{G}}$ to explore adjacent fragile nodes in $\hat{G}$ recursively. If the search is terminated, the fragile nodes explored by the DFS are safe-to-merge nodes. By performing DFS iteratively by selecting unvisited fragile nodes as the starting node, we can enumerate all connected node sets in $G[F_{\hat{G}}]$. This computation incurs $O(|F_{\hat{G}}| \cdot (\alpha - k))$ time for two reasons. First, each fragile node has exactly $\alpha - k$ adjacent nodes in $\hat{G}$. Second, DFS is performed only for fragile nodes $F_{\hat{G}}$ and their adjacent nodes. Next, we discuss the time complexity to reconnect edges of the merging nodes. From Definition 5, each safe-to-merge node $v \in M$ becomes an unfruitful node if nodes in $M \backslash \{v\}$ are removed from $\hat{G}$. Thus, v has less than $\alpha - k$ adjacent nodes in $\hat{G} \backslash M$ from Definition 3. It indicates that less than $\alpha - k$ edges are reconnected for each merged node. Since the number of merged nodes is approximately $|F_{\hat{G}}|$ from Lemma 2, reconnecting edges incurs $O(|F_{\hat{G}}| \cdot (\alpha - k))$ time. Hence, Algorithm 1 needs $O(|F_{\hat{G}}| \cdot (\alpha - k))$ time in total. □

As described in Section 3.1, real-world social networks typically contain many low-degree nodes due to the scale-free property; the expected number of nodes with d adjacent nodes is proportional to $|V_G| d^{-\gamma}$, where γ is a positive constant representing the strength of the degree skewness [Albert and Barabási, 2002]. Since fragile nodes are adjacent to exactly $d = \alpha - k$ nodes, we have $O(|F_{\hat{G}}|) \approx O(|V_{\hat{G}}|(\alpha - k)^{-\gamma})$ in practice. Thus, Algorithm 1 practically incurs $O(|F_{\hat{G}}| \cdot (\alpha - k)) \approx O(|V_{\hat{G}}|/(\alpha - k)^{(\gamma - 1)})$ time to reduce the graph size, which is a sublinear time to the number of nodes in real-world social networks. In contrast, existing state-of-the-art algorithms require $O(|V_G|^2)$ time to reduce the graph size. Thus, Lemma 3 indicates that Algorithm 1 reduces the graph size more efficiently than state-of-the-art MPS algorithms.

3.4 BnM Algorithm

Algorithm 2 fully describes BnM. It iteratively explores (α, k)-plexes by increasing α until the maximum k-plex is found (lines 2–8). Initially, BnM sets $\alpha = k + 1$ since any k nodes can be a (k, k)-plex (line 1). In each iteration, BnM invokes the BRANCH function (line 3) to find (α, k)-plexes from $\hat{G}$. The BRANCH function is composed of two steps: (1) graph reduction (line 22) and (2) graph branching (lines 24–27). The first step reduces the graph size using the MERGE function shown in Algorithm 1, which finds and merges the safe-to-merge nodes to efficiently reduce the graph size. In the subsequent step, BnM recursively invokes the BRANCH function by removing each node in V_{G^*} one-by-one (lines 24–25). After recursive computations, if the graph size of $\hat{G}$ reaches α, $\hat{G}$ is returned as an (α, k)-plex (lines 18–19). Otherwise, the BRANCH function outputs no graph (lines 20–21). Finally, BnM outputs ($\alpha - 1$, k)-plexes as the maximum k-plex if the BRANCH function does not return a graph (lines 4–5).

Theoretical analysis: We theoretically assessed the efficiency and the exactness of BnM. By letting τ be the total number of BRANCH functions invoked in Algorithm 2, the

Graphs	$\|V_G\|$	$\|E_G\|$	γ	α_{max}	k	BnM	BnB	Maplex	KpLeX	kPlexS
soc-LiveMocha	104,103	2,193,083	1.75	19	2	16.23 (± 0.31) sec.	467.19 (± 4.42) sec.	48.96 (± 0.91) sec.	**12.01** (± 0.32) sec.	29.01 (± 0.44) sec.
				22	3	**40.80** (± 0.55) sec.	DNF	329.74 (± 1.37) sec.	42.77 (± 0.81) sec.	47.20 (± 0.72) sec.
				25	4	**529.31** (± 2.76) sec.	DNF	5,122.63 (± 7.72) sec.	631.10 (± 4.28) sec.	580.10 (± 3.10) sec.
				28	5	**8,899.67** (± 8.42) sec.	DNF	DNF	DNF	DNF
soc-douban	154,908	327,162	2.13	12	2	**0.16** (± 0.01) sec.	0.81 (± 0.06) sec.	0.21 (± 0.03) sec.	0.28 (± 0.02) sec.	0.25 (± 0.02) sec.
				14	3	**0.16** (± 0.01) sec.	0.71 (± 0.05) sec.	0.26 (± 0.03) sec.	0.31 (± 0.03) sec.	0.23 (± 0.01) sec.
				16	4	**0.14** (± 0.01) sec.	0.59 (± 0.07) sec.	0.16 (± 0.02) sec.	0.21 (± 0.02) sec.	0.16 (± 0.01) sec.
				17	5	0.10 (± 0.01) sec.	0.20 (± 0.03) sec.	**0.09** (± 0.01) sec.	0.16 (± 0.02) sec.	0.10 (± 0.01) sec.
soc-twitter-follows	404,719	713,319	1.18	8	2	1.53 (± 0.07) sec.	11.42 (± 0.23) sec.	0.81 (± 0.09) sec.	0.98 (± 0.05) sec.	**0.58** (± 0.04) sec.
				9	3	1.28 (± 0.05) sec.	23.56 (± 0.36) sec.	0.75 (± 0.08) sec.	0.72 (± 0.04) sec.	**0.47** (± 0.03) sec.
				11	4	0.61 (± 0.02) sec.	24.44 (± 0.32) sec.	0.52 (± 0.08) sec.	0.68 (± 0.07) sec.	**0.35** (± 0.03) sec.
				13	5	**0.29** (± 0.02) sec.	28.71 (± 0.47) sec.	0.32 (± 0.03) sec.	0.44 (± 0.06) sec.	0.31 (± 0.02) sec.
soc-youtube	495,957	1,936,748	1.88	20	2	**2.49** (± 0.18) sec.	17.23 (± 1.72) sec.	3.26 (± 0.21) sec.	4.52 (± 0.31) sec.	4.09 (± 0.14) sec.
				21	3	19.31 (± 0.89) sec.	30.06 (± 1.96) sec.	16.25 (± 0.56) sec.	5.81 (± 0.39) sec.	**5.27** (± 0.19) sec.
				24	4	21.31 (± 0.85) sec.	20.12 (± 2.01) sec.	50.46 (± 0.91) sec.	23.22 (± 0.80) sec.	**18.21** (± 0.31) sec.
				26	5	**11.90** (± 0.52) sec.	25.00 (± 1.75) sec.	3,701.25 (± 20.31) sec.	63.00 (± 2.13) sec.	21.80 (± 0.45) sec.
soc-delicious	536,108	1,365,961	2.14	23	2	0.20 (± 0.01) sec.	1.51 (± 0.15) sec.	**0.19** (± 0.05) sec.	0.89 (± 0.04) sec.	0.25 (± 0.01) sec.
				27	3	**0.21** (± 0.02) sec.	1.79 (± 0.17) sec.	**0.21** (± 0.04) sec.	0.82 (± 0.04) sec.	0.27 (± 0.01) sec.
				29	4	**0.14** (± 0.01) sec.	1.11 (± 0.14) sec.	0.16 (± 0.02) sec.	0.77 (± 0.04) sec.	0.27 (± 0.02) sec.
				30	5	**0.08** (± 0.01) sec.	0.79 (± 0.11) sec.	0.16 (± 0.03) sec.	0.63 (± 0.02) sec.	0.12 (± 0.01) sec.
soc-FourSquare	639,014	3,214,986	1.52	35	2	**621.23** (± 2.31) sec.	5,291.21 (± 24.33) sec.	798.20 (± 8.21) sec.	680.21 (± 4.12) sec.	740.81 (± 1.90) sec.
				39	3	886.14 (± 2.91) sec.	DNF	1,302.39 (± 10.04) sec.	**430.99** (± 3.10) sec.	656.74 (± 2.86) sec.
				42	4	910.26 (± 3.09) sec.	DNF	5,788.21 (± 25.42) sec.	604.49 (± 4.47) sec.	**598.90** (± 2.23) sec.
				44	5	3,490.63 (± 6.89) sec.	DNF	DNF	4,289.75 (± 21.08) sec.	**2,809.31** (± 5.89) sec.
soc-youtube-snap	1,134,890	2,987,624	1.72	20	2	**1.87** (± 0.19) sec.	32.80 (± 0.90) sec.	3.92 (± 0.42) sec.	3.57 (± 0.29) sec.	2.02 (± 0.19) sec.
				21	3	9.89 (± 0.79) sec.	79.56 (± 1.16) sec.	49.95 (± 1.89) sec.	4.99 (± 0.41) sec.	**3.89** (± 0.26) sec.
				24	4	10.02 (± 0.90) sec.	77.37 (± 1.20) sec.	200.34 (± 3.55) sec.	10.11 (± 0.85) sec.	**6.16** (± 0.30) sec.
				26	5	16.20 (± 1.44) sec.	101.02 (± 2.18) sec.	3,922.66 (± 17.16) sec.	79.22 (± 2.15) sec.	**14.22** (± 0.80) sec.
soc-lastfm	1,191,805	4,519,330	1.93	18	2	**4.23** (± 0.35) sec.	261.19 (± 4.74) sec.	9.88 (± 1.14) sec.	7.90 (± 0.21) sec.	19.31 (± 0.82) sec.
				21	3	**6.96** (± 0.39) sec.	823.35 (± 6.45) sec.	62.10 (± 2.43) sec.	10.12 (± 0.30) sec.	101.24 (± 1.53) sec.
				24	4	**32.77** (± 1.02) sec.	DNF	1,166.42 (± 8.70) sec.	42.29 (± 0.38) sec.	217.92 (± 1.81) sec.
				27	5	**201.00** (± 1.42) sec.	DNF	DNF	312.10 (± 2.51) sec.	305.38 (± 2.11) sec.
soc-pokec	1,632,803	22,301,964	2.88	31	2	**16.73** (± 1.31) sec.	112.42 (± 1.39) sec.	39.92 (± 1.36) sec.	21.10 (± 0.71) sec.	44.72 (± 0.70) sec.
				32	3	**17.33** (± 1.48) sec.	151.04 (± 1.46) sec.	44.86 (± 1.82) sec.	23.54 (± 0.59) sec.	39.20 (± 0.59) sec.
				32	4	**17.32** (± 1.29) sec.	96.34 (± 1.21) sec.	36.21 (± 1.27) sec.	22.16 (± 0.62) sec.	27.70 (± 0.64) sec.
				34	5	**18.11** (± 1.50) sec.	77.00 (± 0.00) sec.	32.05 (± 1.14) sec.	26.35 (± 0.91) sec.	23.79 (± 0.39) sec.
soc-flixster	2,523,386	7,918,801	1.88	38	2	34.50 (± 0.82) sec.	941.23 (± 5.12) sec.	110.79 (± 4.28) sec.	28.10 (± 1.22) sec.	**9.11** (± 0.15) sec.
				42	3	290.90 (± 1.74) sec.	DNF	DNF	344.80 (± 4.67) sec.	**179.04** (± 1.16) sec.
				46	4	1,984.52 (± 7.21) sec.	DNF	DNF	2,721.11 (± 6.72) sec.	**415.57** (± 1.72) sec.
				49	5	4,016.59 (± 13.36) sec.	DNF	DNF	DNF	**569.30** (± 1.79) sec.
soc-livejournal	4,033,137	27,933,062	2.47	214	2	**2.12** (± 0.21) sec.	17.34 (± 0.38) sec.	5.82 (± 0.27) sec.	20.21 (± 0.71) sec.	4.09 (± 0.31) sec.
				214	3	**2.24** (± 0.20) sec.	17.23 (± 0.32) sec.	6.21 (± 0.24) sec.	19.20 (± 0.49) sec.	3.96 (± 0.30) sec.
				214	4	**2.25** (± 0.21) sec.	17.80 (± 0.35) sec.	8.11 (± 0.27) sec.	15.70 (± 0.42) sec.	3.64 (± 0.39) sec.
				214	5	**2.39** (± 0.22) sec.	17.60 (± 0.38) sec.	8.23 (± 0.23) sec.	13.10 (± 0.31) sec.	3.29 (± 0.20) sec.

Table 2: Running times for real-world social networks.

time complexity of BnM is analyzed as follows:

Theorem 1. *BnM incurs $O(\tau(|V_{\hat{G}}| + |F_{\hat{G}}|(\alpha - k)))$ time.*

Proof. As shown in line 22 of Algorithm 2, the BRANCH function invokes the MERGE function, which incurs $O(|F_{\hat{G}}|(\alpha - k))$ time, as proved in Lemma 3. Furthermore, as shown in lines 24–27 in Algorithm 2, BnM performs $|V_{\hat{G}}|$ iterations to explore (α, k)-plexes. Hence, BnM requires $O(\tau(|V_{\hat{G}}| + |F_{\hat{G}}|(\alpha - k)))$ time. □

State-of-the-art MPS algorithms require $O(|V_G|^2)$ time for each graph reduction. Thus, they incur $O(\tau|V_G|^2)$ time to find the maximum k-plex. In contrast, BnM drastically reduces the time complexity. Since $O(|F_{\hat{G}}|) \approx O(|V_G|/(\alpha - k)^{(\gamma-1)})$ holds due to the scale-free property, we have $O(\tau(|V_G|+|F_{\hat{G}}|(\alpha-k))) \approx O(\tau|V_G|(1+1/(\alpha-k)^{(\gamma-2)})))$ for the time complexity as proved in Theorem 1. Since $O(1 + 1/(\alpha - k)^{(\gamma-2)}) \approx O(1)$ in practice, BnM incurs $O(\tau|V_G|)$ time. Because τ theoretically depends on $|V_{G^*}|$, which dynamically decreases as the search progresses, the practical size of τ is much smaller than $|V_G^*|$. Thus, BnM has an almost linear scalability against the number of nodes. In Section 4, we further assess the practical efficiency and scalability using real-world social networks.

We then assess the exactness of BnM as follows:

Theorem 2. *BnM outputs the exact maximum k-plex.*

Proof. BnM is composed of two steps: (1) construct $\hat{G}$ (line 11) and (2) invoke the BRANCH function (lines 3, 10–28). As shown in line 11 of Algorithm 2, $\hat{G}$ is constructed while removing only unfruitful nodes not included in the (α, k)-plex from Definition 4. Second, as shown in lines 24–27 of Algorithm 2, BnM recursively searches (α, k)-plex from $\hat{G}$ without v while varying $v \in \hat{G}$. As we proved in Lemma 1, if v is a member of a supernode s, then all nodes merged into s are never included in any (α, k)-plexes in $\hat{G}_{-v}$, even if each node is not computed by Algorithm 2. That is, BnM does not fail to find all (α, k)- plexes in each iteration, although it simultaneously removes all nodes of the supernode by a node-merging strategy. Therefore, BnM outputs the exact maximum k-plex. □

Theorem 2 indicates that BnM efficiently finds the exact maximum k-plex, although BnM dynamically prunes numerous nodes. As a result, BnM achieves an efficient MPS algorithm without sacrificing the quality of AI-powered applications.

4 Experimental Evaluation

We experimentally evaluated the efficiency of BnM.

Experimental Setup: We compared our proposed algorithm BnM with state-of-the-art MPS algorithms: BnB [Gao *et al.*, 2018], Maplex [Zhou *et al.*, 2021], KpLeX [Jiang *et al.*, 2021], and kPlexS [Chang *et al.*, 2022]. All experiments were conducted on a Linux server with Intel Xeon Gold 6246R CPU 3.40 GHz and 128 GiB RAM. All algorithms were implemented in C/C++ using "-O3" option as a single-threaded program with the entire graph held in the main memory. All results are averaged over ten independent runs.

Datasets: We tested 11 real-world social networks used in previous works [Gao *et al.*, 2018; Zhou *et al.*, 2021; Jiang *et al.*, 2021; Chang *et al.*, 2022] with more than 100,000 nodes, which are originally published by the Network Repository [Rossi and Ahmed, 2015][2]. Table 2 shows their statistics, where α_{max} and γ denote the size of the maximum k-plex and the skewness of the degree distribution, respectively. If γ is large, the graph has many low-degree nodes.

4.1 Efficiency

Table 2 shows the running times of MPS on the datasets, where DNF indicates that the runtime exceeded 10,000 seconds. Similar to previous studies [Gao *et al.*, 2018; Zhou *et al.*, 2021], we varied k from 2 to 5. Although BnM guarantees the same results as the Naïve algorithm by Theorem 2, it is up to 305.2, 311.0, 9.53, and 14.54 times faster than BnB, Maplex, KpLeX, and kPlexS, respectively. Specifically, BnM significantly outperforms the other methods on graphs with a large γ value (*e.g.,* soc-lastfm, soc-pokec, soc-livejournal, etc.). Compared to other algorithms, BnM has an improved time complexity as proved in Theorem 1. It is responsible for the superior running time.

In many datasets, BnM can find the maximum k-plex even if k is set to 5, unlike the graph reduction algorithms of BnB, Maplex, KpLeX, and kPlexS. As k increases, the computation cost increases since the maximum k-plex would be large. Regardless of the degree distribution, the reduction algorithms repeatedly compute all nodes to remove unpromising nodes. However, real-world social networks have many low-degree nodes (*i.e.*, large γ values in Table 2) that are unlikely to be the maximum k-plex due to the scale-free property. Thus, reduction algorithms fail to return the maximum k-plex for large k settings on massive graphs. In contrast, BnM efficiently removes unpromising nodes using a node-merging strategy based on the scale-free property if graphs have a large γ value. Therefore, BnM can efficiently compute massive social networks with large k settings compared to other MPS algorithms.

4.2 Effectiveness

We experimentally evaluated the effectiveness of our approach by comparing the runtime of BnM to its variant that excludes the node-merging strategy (BnM (w/o merge)). Table 3 shows the runtimes, where DNF indicates the algorithm did not finish within 10,000 seconds. Table 3 also includes the graph reduction ratio (GRR), which is the fraction of edges visited by BnM compared to BnM (w/o merge), to visualize the reduced graph size by the node-merging strategy.

Graphs	k	BnM	BnM (w/o merge)	GRR
soc-LiveMocha	2	**16.23** sec.	DNF	0.05%
	3	**40.8** sec.	DNF	0.03%
	4	**529.31** sec.	DNF	0.03%
	5	**8,899.67** sec.	DNF	0.03%
soc-douban	2	**0.16** sec.	358.84 sec.	0.04%
	3	**0.16** sec.	291.14 sec.	0.05%
	4	**0.14** sec.	285.92 sec.	0.04%
	5	**0.1** sec.	202.42 sec.	0.05%
soc-twitter-follows	2	**1.53** sec.	2,156.17 sec.	0.06%
	3	**1.28** sec.	7,829.11 sec.	0.01%
	4	**0.61** sec.	7,356.13 sec.	0.01%
	5	**0.29** sec.	7,001.65 sec.	0.01%
soc-youtube	2	**2.49** sec.	3,801.64 sec.	0.06%
	3	**19.31** sec.	7,301.28 sec.	0.25%
	4	**21.31** sec.	7,016.16 sec.	0.26%
	5	**11.9** sec.	5,792.48 sec.	0.20%
soc-delicious	2	**0.2** sec.	371.38 sec.	0.05%
	3	**0.21** sec.	480.72 sec.	0.04%
	4	**0.14** sec.	301.18 sec.	0.05%
	5	**0.08** sec.	266.02 sec.	0.03%
soc-FourSquare	2	**621.23** sec.	DNF	0.05%
	3	**886.14** sec.	DNF	0.04%
	4	**910.26** sec.	DNF	0.04%
	5	**3,490.63** sec.	DNF	0.03%

Table 3: Effectiveness of a node-merging strategy.

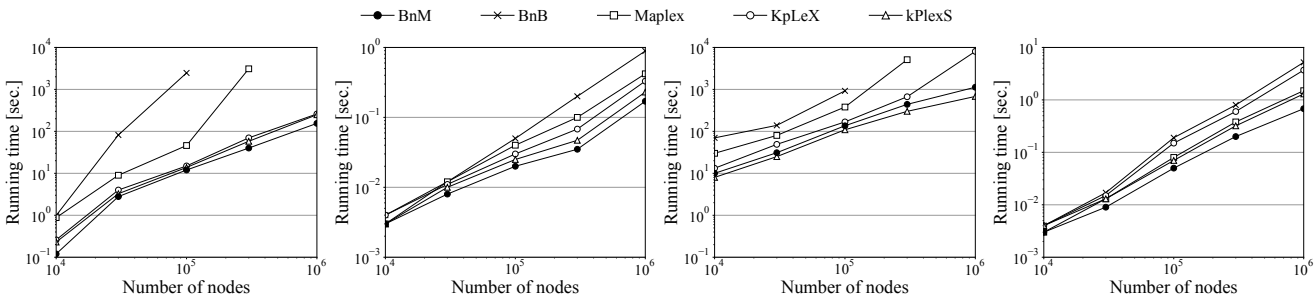
On average, BnM reduces the running time by 99.96% compared to BnM (w/o merge). These results correspond to the values derived from GRR where the node-merging strategy reduces 99.93% of the computed edge sizes compared to BnM (w/o merge). These results indicate that the node-merging strategy enhances the efficiency since BnM dramatically reduces the search space. Our strategy can remove all the safe-to-merge nodes simultaneously if one of them is not included in the (α, k)-plexes (Lemma 3.2). Furthermore, the average number of GRR is 0.07%, which implies that the size of τ in Theorem 1 is quite small since the size of τ depends on the graph size obtained after merging safe-to-merge nodes G^*. Consequently, the node-merging strategy effectively reduces the running time.

4.3 Scalability

We assessed the scalability of BnM. Figure 2 plots the running time on the four largest social networks in Table 2 as a function of the number of nodes. We sampled 1×10^4, 3×10^4, 1×10^5, 3×10^5, and 1×10^6 nodes from the original graphs using the Metropolis-Hastings Random Walk [Cui *et al.*, 2022] and evaluated the runtime on each sampled graph with $k = 5$. If the algorithm did not finish within 10,000 seconds, the results are omitted in Figure 2.

Overall, BnM shows a nearly linear scalability against the number of nodes on most graphs due to the skewness of the degree distribution in social networks. BnM requires $O(\tau|V_G|(1+1/(\alpha-k)^{(\gamma-2)}) \approx O(\tau|V_G|)$ time if γ is sufficiently large. Hence, BnM is scalable if a degree distribution is highly skewed. Therefore, BnM achieves a nearly linear scalability since most graphs have a large γ value (Table 2).

[2] All graphs are publicly available online from https://networkrepository.com.



(a) soc-lastfm (b) soc-pokec (c) soc-flixster (d) soc-livejournal

Figure 2: Scalability. Results are omitted if their running time exceeds 10,000 seconds.

5 Related Works

The problem of finding large dense communities has been studied for several decades [Matsugu *et al.*, 2020; Shiokawa and Takahashi, 2020; Shiokawa, 2021]. Here we review some of more successful MPS methods.

MPS algorithms: MPS is a problem to find the largest k-plex included in a graph. MPS was first proposed by [Balasundaram *et al.*, 2007] in 2007 as a generalization of a maximum clique search. However, the naïve MPS algorithm is unresponsive to large social networks because it requires $O(2^{|V_G|}\ |V_G|)$ time, where $|V_G|$ is the number of nodes included in a graph [Balasundaram *et al.*, 2007].

Several branching algorithms have been proposed to overcome this limitation. Examples include GuidedBranching [Moser *et al.*, 2012] and BS [Xiao *et al.*, 2017]. The key idea of these algorithms is to recursively partition search spaces explored during the search process by branching each search space according to whether a node can be included in the maximum k-plex or not. As reported in [Xiao *et al.*, 2017], branching algorithms reduce the running times for graphs with 700 nodes. However, they still require an expensive cost to handle large social networks since the algorithms need $O(1.98^{|V_G|}\ |V_G|^2\ \log(|V_G|))$ time for MPS at $k = 5$.

Recently, Gao *et al.* proposed branch and bound (BnB) to mitigate the expensive costs of MPS. To enhance the computational efficiency, BnB introduced graph reduction approaches that dynamically exclude unpromising nodes, which are unlikely to be the maximum k-plex. BnB approaches still incur a large computational cost to find unpromising nodes because they must repeatedly scan the whole graph. To mitigate the overhead of graph reduction approaches, several bounding methods have recently been proposed such as Maplex [Zhou *et al.*, 2021], KpLeX [Jiang *et al.*, 2021], and kPlexS [Chang *et al.*, 2022]. These methods employ tighter upper bounds for graph reduction to further reduce the number of visited nodes/edges. However, these cannot handle large social networks with a few million nodes since their state-of-the-art approaches require $O(|V_G|^2)$ time to exclude unpromising nodes.

Our work differs from these existing MPS algorithms in that BnM provides a very efficient graph reduction algorithm based on a node-merging strategy (Section 3.3). As we theoretically discussed in Section 3.4, BnM incurs $O(\tau|V_G|)$ time in practice, which is more efficient than the above algorithms. Furthermore, our experimental analysis in Section 4 shows that BnM has a higher performance in terms of the running time and the node-merging strategy successfully removes 99.93% of visited edges.

Other k-plex search algorithms: Several other types of k-plex search problems have also been studied in recent years. An example is the maximal k-plex search [Wang *et al.*, 2017; Conte *et al.*, 2018; Zhou *et al.*, 2020]. Given a user-specified query nodes V_Q, the maximal k-plex search is a problem to find a k-plex G_S such that G_S includes all V_Q and no other k-plex that contains G_S exists. Unlike MPS, the maximal k-plex search is not guarantee to output the largest k-plex because it requires user-specified query nodes. Conversely, the maximal k-plex search algorithms can find the maximum k-plex if all possible query patterns can be tested, but this clearly incurs $O(2^{|V_G|})$ time. Unfortunately, even state-of-the-art algorithms [Dai *et al.*, 2022] still require exponential complexity with the number of nodes. Thus, it is difficult to use a maximal k-plex search for MPS.

Unlike the above approaches, our proposed method, BnM, guarantees to output the maximum k-plex. Furthermore, we theoretically and experimentally verified that BnM has an almost linear scalability with the number of nodes included in a graph. That is, BnM is a more effective approach than the above ones in terms of the maximum k-plex finding.

6 Conclusion

Here, we propose an efficient MPS algorithm, BnM, for massive social networks. BnM effectively handles the scale-free property of social networks. It reduces the graph size by finding and merging safe-to-merge nodes. Experiments reveal that BnM offers an improved efficiency compared to state-of-the-art MPS algorithms. Consequently, BnM can enhance the effectiveness of AI-powered applications for massive graphs.

Acknowledgements

This paper is partly supported by JST PRESTO JPMJPR2033, JST AIP Acceleration Research JPMJCR23U2, and JSPS KAKENHI 22J10972.

References

[Abello *et al.*, 2002] James Abello, Mauricio G. C. Resende, and Sandra Sudarsky. Massive Quasi-Clique Detection. *Proceedings of the 5th Latin American Symposium on Theoretical Informatics*, page 598–612, 2002.

[Albert and Barabási, 2002] Réka Albert and Albert-László Barabási. Statistical Mechanics of Complex Networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[Balasundaram *et al.*, 2007] B Balasundaram, Sergiy Butenko, I. Hicks, and S Sachdeva. Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem. *Operations Research*, 01 2007.

[Balasundaram, 2007] Balabhaskar Balasundaram. *Graph theoretic generalizations of clique: Optimization and extensions*. Texas A&M University, 2007.

[Bianchi *et al.*, 2020] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[Bodaghi and Oliveira, 2022] Amirhosein Bodaghi and Jonice Oliveira. The Theater of Fake News Spreading, Who Plays Which Role? A Study on Real Graphs of Spreading on Twitter. *Expert Systems with Applications*, 189:116110, 2022.

[Bomze *et al.*, 1999] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. The Maximum Clique Problem. *Handbook of Combinatorial Optimization*, 4:1–74, 05 1999.

[Carmel *et al.*, 2009] David Carmel, Naama Zwerdling, Ido Guy, Shila Ofek-Koifman, Nadav Har'el, Inbal Ronen, Erel Uziel, Sivan Yogev, and Sergey Chernov. Personalized Social Search Based on the User's Social Network. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM'09, page 1227–1236, New York, NY, USA, 2009. Association for Computing Machinery.

[Chang *et al.*, 2022] Lijun Chang, Mouyi Xu, and Darren Strash. Efficient Maximum k-Plex Computation over Large Sparse Graphs. *Proceedings of the VLDB Endowment*, 16(2):127–139, 2022.

[Conte *et al.*, 2017] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Fast Enumeration of Large K-Plexes. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'17, page 115–124, New York, NY, USA, 2017. Association for Computing Machinery.

[Conte *et al.*, 2018] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. KDD'18, page 1272–1281, New York, NY, USA, 2018. Association for Computing Machinery.

[Cui *et al.*, 2022] Yingan Cui, Xue Li, Junhuai Li, Huaijun Wang, and Xiaogang Chen. A Survey of Sampling Method for Social Media Embeddedness Relationship. *ACM Comput. Surv.*, 02 2022. Just Accepted.

[Dai *et al.*, 2022] Qiangqiang Dai, Rong-Hua Li, Hongchao Qin, Meihao Liao, and Guoren Wang. Scaling Up Maximal K-Plex Enumeration. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM'22, page 345–354, New York, NY, USA, 2022. Association for Computing Machinery.

[Danezis *et al.*, 2010] George Danezis, Tuomas Aura, Shuo Chen, and Emre Kıcıman. How to share your favourite search results while preserving privacy and quality. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 273–290. Springer, 2010.

[Doerr *et al.*, 2012] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Why Rumors Spread so Quickly in Social Networks. *Commun. ACM*, 55(6):70–75, jun 2012.

[Faloutsos *et al.*, 1999] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, 08 1999.

[Gao *et al.*, 2018] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. An Exact Algorithm for Maximum k-Plexes in Massive Graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 1449–1455. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[Jiang *et al.*, 2021] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. A New Upper Bound Based on Vertex Partitioning for the Maximum K-plex Problem. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, IJCAI'21, pages 1689–1696. International Joint Conferences on Artificial Intelligence Organization, 8 2021.

[Lee *et al.*, 2019] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 09–15 Jun 2019.

[Leskovec and Krevl, 2014] J. Leskovec and A. Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data, June 2014.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18. AAAI Press, 2018.

[Matsugu *et al.*, 2020] Shohei Matsugu, Hiroaki Shiokawa, and Hiroyuki Kitagawa. Fast and Accurate Community

Search Algorithm for Attributed Graphs. In *Database and Expert Systems Applications: 31st International Conference*, DEXA'20, pages 233–249. Springer, 2020.

[Matsugu *et al.*, 2021] Shohei Matsugu, Hiroaki Shiokawa, and Hiroyuki Kitagawa. Fast Algorithm for Attributed Community Search. *Journal of Information Processing*, 29:188–196, 2021.

[McClosky and Hicks, 2012] Benjamin McClosky and Illya V Hicks. Combinatorial Algorithms for the Maximum k-Plex Problem. *Journal of Combinatorial Optimization*, 23(1):29–49, 2012.

[Moser *et al.*, 2012] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact Combinatorial Algorithms and Experiments for Finding Maximum k-Plexes. *Journal of combinatorial optimization*, 24(3):347–373, 2012.

[Noll and Meinel, 2007] Michael G. Noll and Christoph Meinel. Web Search Personalization Via Social Bookmarking and Tagging. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 367–380, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Pattillo *et al.*, 2013] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On Clique Relaxation Models in Network Analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

[Rossi and Ahmed, 2015] Ryan Rossi and Nesreen Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 2015.

[Seidman, Stephen B and Foster, Brian L, 1978] Seidman, Stephen B and Foster, Brian L. A Graph-theoretic Generalization of the Clique Concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[Seidman, 1983] Stephen B. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5(3):269–287, 1983.

[Shiokawa and Takahashi, 2020] Hiroaki Shiokawa and Tomokatsu Takahashi. DSCAN: Distributed Structural Graph Clustering for Billion-edge Graphs. In *Database and Expert Systems Applications: 31st International Conference*, DEXA'20, pages 38–54. Springer, 2020.

[Shiokawa *et al.*, 2013] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Fast Algorithm for Modularity-Based Graph Clustering. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, AAAI'13, pages 1170–1176, 2013.

[Shiokawa *et al.*, 2019] Hiroaki Shiokawa, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Scaling Fine-grained Modularity Clustering for Massive Graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pages 4597–4604, 2019.

[Shiokawa, 2021] Hiroaki Shiokawa. Scalable Affinity Propagation for Massive Satasets. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, volume 35 of *AAAI'21*, pages 9639–9646, 2021.

[Tomita and Seki, 2003] Etsuji Tomita and Tomokazu Seki. An Efficient Branch-and-Bound Algorithm for Finding a Maximum Clique. pages 278–289, 01 2003.

[Wang and Dai, 2008] Lin Wang and Guan-zhong Dai. Global Stability of Virus Spreading in Complex Heterogeneous Networks. *SIAM Journal on Applied Mathematics*, 68(5):1495–1502, 2008.

[Wang *et al.*, 2003] Yang Wang, D. Chakrabarti, Chenxi Wang, and C. Faloutsos. Epidemic Spreading in Real Networks: an Eigenvalue Viewpoint. In *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.*, pages 25–34, 2003.

[Wang *et al.*, 2017] Yue Wang, Xun Jian, Zhenhua Yang, and Jia Li. Query Optimal k-Plex Based Community in Graphs. *Data Science and Engineering*, 2(4):257–273, 2017.

[Wang *et al.*, 2020] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montufar, Xiaosheng Zhuang, and Yanan Fan. Haar Graph Pooling. In *International Conference on Machine Learning*, pages 9952–9962. PMLR, 2020.

[Wiil *et al.*, 2010] Uffe Wiil, Nasrullah Memon, and Panagiotis Karampelas. Detecting New Trends in Terrorist Networks. pages 435–440, 08 2010.

[Xiao and Kou, 2017] Mingyu Xiao and Shaowei Kou. Exact Algorithms for the Maximum Dissociation Set and Minimum 3-Path Vertex Cover Problems. *Theor. Comput. Sci.*, 657(PA):86–97, January 2017.

[Xiao *et al.*, 2017] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. A Fast Algorithm to Compute Maximum k-Plexes in Social Network Analysis. In *Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 2017.

[Ying *et al.*, 2018] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4805–4815, Red Hook, NY, USA, 2018. Curran Associates Inc.

[Zhou *et al.*, 2020] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. Enumerating Maximal k-plexes with Worst-case Time Guarantee. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34 of *AAAI'20*, pages 2442–2449, 2020.

[Zhou *et al.*, 2021] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. Improving Maximum k-Plex Solver via Second-Order Reduction and Graph Color Bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35 of *AAAI'21*, pages 12453–12460, 2021.

[Zhu *et al.*, 2020] Jinrong Zhu, B. Chen, and Yifeng Zeng. Community Detection Based on Modularity and k-Plexes. *Information Sciences*, 513:127–142, 2020.