More Than Pivot for Maximal Clique Enumeration

Zhaoyi Zhong[†], Rui Zhou[†], Lu Chen[†], Xiaofan Li[‡], Chengfei Liu[†]
[†]Swinburne University of Technology, Australia
[‡]Nanyang Technological University, Singapore
[†]{zzhong,rzhou,luchen,cliu}@swin.edu.au
[‡]xiaofan.li@ntu.edu.sg

***Abstract*—The Maximal Clique Enumeration (MCE) problem is a classic and fundamental task in graph data mining and analysis. It has attracted widespread attention due to its broad applications in areas such as social network analysis and bioinformatics. A widely adopted framework for solving the MCE problem is the Bron–Kerbosch (BK) algorithm. Its efficiency can be significantly improved by incorporating the pivot strategy, which is the most effective pruning technique in BK-based algorithms. However, the pivot strategy is not an optimal solution for reducing search branches, and there remains room for further pruning unnecessary branches. To fill this gap, we propose a new heuristic pruning algorithm, called moreThanPivot, which focuses on further reducing search branches in the BK framework. This algorithm iteratively selects multiple vertices as splitters. For each splitter, the algorithm constructs its Residual Cover Set and Residual Pillar Set, ultimately returning refined search branches with fewer branches than the pivot strategy. Three alternative selection ranges and three greedy objectives are further proposed for selecting new splitters. Extensive experimental results on 16 real-world datasets demonstrate the superiority of our method. And experiments on synthetic datasets reflect that our method is especially effective on graphs with high edge density.**

***Index Terms*—Maximal Clique Enumeration, Pivot, MTP**

I. Introduction

Graph-structured data has become increasingly prominent in modern data-driven applications, including areas such as social network analysis [1], [2], bioinformatics [3]–[6], and knowledge graph construction [7]. A fundamental task in these areas is to enumerate the maximal cliques, which are widely used to reveal dense substructures such as tightly cooperating social groups [8], functional protein modules [9], [10], or sensor clusters [11].

In an undirected graph, a clique is defined as a subset of vertices in which every pair of vertices is connected by an edge. A maximal clique is a clique that cannot be extended by including any additional vertex. The task of enumerating all maximal cliques, known as the Maximal Clique Enumeration (MCE) problem, is a foundational challenge in graph mining and analysis, enabling a broad spectrum of applications in both scientific research and real-world systems [12]–[16].

Over the years, extensive efforts have been dedicated to improving the efficiency of MCE. The most influential and widely adopted frameworks is the Bron–Kerbosch (BK) algorithm [17]. It follows a recursive branch-and-bound strategy and has served as the foundation for many subsequent MCE algorithms. Based on the BK framework, a well-known pruning algorithm called BK_pivot was proposed [17]–[19]. BK_pivot introduces the pivot strategy, which selects the vertex with the most neighbors in the candidate set as the pivot and prunes its neighbors from further exploration. Since then, more algorithms have been proposed to further improve the efficiency of the BK algorithm. Their approaches can be categorized into the following three directions:

Optimizing the pivot selection. Building on BK_pivot, several algorithms have explored more efficient pivot selection strategies. These methods aim to either relax the pivot selection criteria or directly enhance the efficiency of pivot selection, thereby improving the overall performance of MCE. BK_refine [20] relaxes the pivot selection criteria by allowing certain vertices that satisfy special cases to be directly chosen as the pivot, thereby terminating the pivot search process early. BK_degree [21] aims to reduce the time cost of pivot selection by maintaining a descending degree order of vertices within each search branch. During enumeration, it directly selects the first vertex in the order as the pivot. BK_facen [22] maintains the degeneracy order within the candidate set and accelerates pivot selection by always choosing the last vertex as the pivot.

Designing vertex enumeration order. Some other algorithms observe that the vertex enumeration order also affects the scale of the search tree. The BK_degen algorithm [23] introduces the degeneracy order as the vertex enumeration order. The size of the candidate set in each branch is thereby bounded by the graph's degeneracy λ. In this way, the worst-case time complexity of BK_degen is limited to $O(\lambda n 3^{\lambda/3})$. HBBMC [24] adopts a truss-based edge order instead of degeneracy order to guide the enumeration process. By doing so, HBBMC obtains a tighter bound compared to the degeneracy on the candidate size at each branch.

Handling special cases. Some algorithms use reduction or early termination techniques to handle special cases in the subgraph of a branch and report certain maximal cliques in advance without further recursion, thereby improving efficiency. BK_rcd [25] introduces a top-down enumeration framework designed for dense graphs. This approach terminates early and reports a maximal clique as soon as the candidate set is found to form a clique. RMCE [26] analyzes the state of vertices in both the global graph and the current search branch. When certain conditions are met, it directly reports maximal cliques and terminates the enumeration of the associated branches, thereby improving overall performance. HBBMC [24] introduces an early termination technique for special cases where the candidate set forms a 2-plex or 3-plex. In such cases,

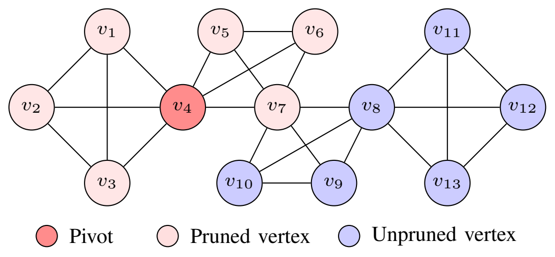

Fig. 1: An example of the pivot strategy.

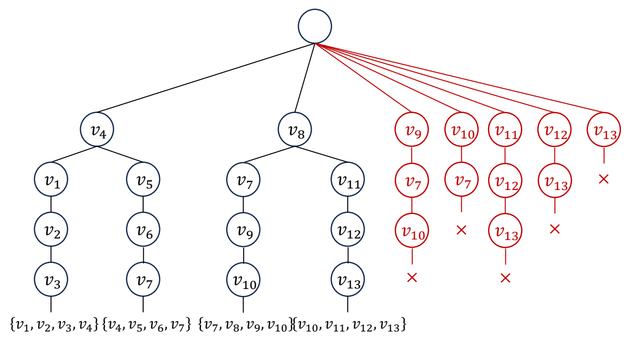

Fig. 2: Further pruned search tree beyond pivot pruning.

the algorithm can immediately report several maximal cliques without further recursion.

A new explorable direction. Although the above methods improve the efficiency of the BK algorithm, they do not focus on reducing search branches. Surprisingly, other than the pivot strategy, for nearly 20 years, branch reduction has remained unexplored, although branch pruning is widely recognized to be crucial in enumeration problems. For MCE, there remains a significant gap between the pivot strategy and the optimal search branch reduction. We illustrate the idea with Fig. 1 and Fig. 2. In the current candidate set $\{v_1, \ldots, v_{13}\}$, the pivot strategy selects vertex v_4 as the pivot, prunes its neighbors $\{v_1, v_2, v_3, v_5, v_6, v_7\}$, and keeps its non-neighbors $\{v_8, \ldots, v_{13}\}$. However, the optimal reduction can enumerate all maximal cliques by retaining only v_4 and v_8. The enumeration tree for this graph is shown in Fig. 2. Existing algorithms using the pivot strategy must explore all branches, including the red branches that are unnecessary. In contrast, the optimal search branch reduction only enumerates the two branches corresponding to v_4 and v_8, avoiding any redundant search. This illustrates that there is room for improvement in search branch reduction.

Achieving optimal search branch reduction is challenging. It is equivalent to τ-cover finding, which is NP-hard [27]. Motivated by the observation, we propose moreThanPivot, a heuristic algorithm for search branch reduction in MCE. We begin by analyzing the effectiveness of the pivoting strategy, which lies in its ability to reduce the number of search branches while bounding the scale of the subproblems. We then investigate the theoretically optimal search branch reduction strategy under the constraint that no maximal cliques are missed. To obtain a better search branch reduction in practice, we introduce a heuristic algorithm called moreThanPivot. It iteratively processes the candidate set to construct refined search branches. Specifically, in each iteration, the algorithm selects a vertex called splitter. For each selected splitter u, its neighbors are added to a Residual Cover Set RCS_u, and the neighbors of RCS_u are further added to a Residual Pillar Set RPS_u. By selecting the vertex with the largest number of neighbors as the initial splitter, moreThanPivot further reduces the number of search branches while maintaining subproblem scale comparable to those of the pivoting strategy. We introduce three alternative selection ranges to guide the splitter selection in each iteration. Additionally, we analyze three greedy objectives for selecting a new splitter. Notably, this algorithm does not need to process all vertices in the branch. By introducing a configurable threshold, we control the number of iterative pruning rounds. Furthermore, we introduce a depth threshold parameter to control when moreThanPivot reverts to the classical pivot method, thus reducing overhead.

Overall, our contributions are summarized as follows:

- To the best of our knowledge, this is the first work that explicitly focuses on optimizing the reduction of search branches within each recursive step of the BK algorithm.
- We propose a heuristic algorithm, called moreThanPivot, which reduces search branches by iteratively removing unnecessary candidates. Our method is able to prune more search branches than the traditional pivot strategy, while still guaranteeing the same worst-case time complexity.
- We design alternative selection ranges and greedy objectives for the iterative splitter selection in moreThanPivot, and demonstrate the effectiveness of different settings through comprehensive experiments.
- The proposed moreThanPivot is modular and can be seamlessly integrated into any existing BK-based maximal clique enumeration algorithms.
- Extensive experimental results demonstrate the effectiveness of moreThanPivot, achieving state-of-the-art performance on 16 real-world datasets. Experiments on synthetic datasets further show that our method is especially effective on graphs with high edge density.

The rest of this paper is organized as follows. Section II introduces the preliminaries and problem definition. In Section III, we describe our proposed method in detail. SectionIV presents experimental results. Section V reviews related work. Finally, Section VI concludes the paper.

II. Preliminaries and Problem Definition

A. Problem Formulation

We consider an undirected graph $G = (V, E)$, where V and E denote the sets of vertices and edges, respectively, with $|V| = n$ and $|E| = m$. Two vertices are said to be adjacent, or neighbors, if there is an edge connecting them. For a vertex $v \in V$, we let $N(v)$ represent the set of all vertices that are

Algorithm 1: BK_pivot Algorithm

```
Function BK_pivot($P, R, X$):
    if $P \cup X = \emptyset$ then
        report $R$ as a maximal clique;
        return;
    Select a pivot $u \in P \cup X$ that maximizes
      $|N(u) \cap P|$;
    foreach vertex $v \in P \setminus N(u)$ do
        BK_pivot($P \cap N(v), R \cup \{v\}, X \cap N(v)$);
        $P \leftarrow P \setminus \{v\}$;
        $X \leftarrow X \cup \{v\}$;
```

adjacent to v in a graph $G = (V, E)$, i.e., $N(v) = \{w \in V \mid (v, w) \in E\}$. For a subset of vertices $S \subseteq V$, we extend this notation by defining $N(S) = \bigcap_{v \in S} N(v)$, i.e., the set of all vertices that are adjacent to all vertices in S.

Definition 1. (Induced Subgraph) Given a graph $G = (V, E)$ and a subset of vertices $S \subseteq V$, the induced subgraph $G[S]$ is defined as $G[S] = (S, \{(u, v) \in E \mid u \in S, v \in S\})$.

Definition 2. (Clique) Given a graph $G = (V, E)$, an induced subgraph $G[S]$ is said to be a clique if every pair of distinct vertices in S is adjacent, i.e., $(u, v) \in E$ for all $u, v \in S$ with $u \neq v$.

Definition 3. (Maximal Clique) Given a graph $G = (V, E)$, a clique $G[S]$ is said to be a *maximal clique* if there does not exist a vertex $v \in V \setminus S$ such that $G[S \cup \{v\}]$ is also a clique.

Problem Statement: (Maximal Clique Enumeration) Given a graph $G = (V, E)$, the objective of the Maximal Clique Enumeration problem is to enumerate and report all maximal cliques in G.

B. BK_pivot Algorithm

In this section, we focus on the well-known BK_pivot algorithm [18], regarded as one of the most straightforward and effective pruning methods for MCE. The BK_pivot algorithm (see Algorithm 1) adopts the recursive backtracking framework of the BK algorithm, which maintains three sets R, P, and X at each level of recursion: R is the current partial clique, P is the candidate set contains vertices that may be added to R, and X stores previously visited vertices to avoid duplicates. In each recursive call, the algorithm selects a vertex u as a pivot from $P \cup X$ and replaces the original search branches P with $P \setminus N(u)$. Any maximal clique involving vertices from the discarded set $P \cap N(u)$ will be found either in the branch starting from pivot u or in branches exploring non-neighbors of u. Therefore, the new branching $P \setminus N(u)$ will not miss any maximal cliques. Choosing a pivot u that minimizes $|P \cap N(u)|$ can make $P \setminus N(u)$ as small as possible. The BK_pivot algorithm has been proven to enumerate all maximal cliques with a worst-case time complexity of $O(n3^{n/3})$.

III. Methodology

In this section, we will further analyze the effectiveness of the BK_pivot algorithm. Based on the preceding analysis, we will introduces a heuristic method that remove the unnecessary branches in the candidate set to improve the search efficiency.

A. Why does BK_pivot algorithm work?

We first revisit the BK_pivot algorithm from a structural perspective. Specifically, the pivot vertex u covers all vertices in $P \cap N(u)$, ensuring that any maximal clique involving these vertices will be found either in the search branch starting from u or in branches exploring non-neighbors of u, thus safely removing vertices in $P \cap N(u)$ does not miss any maximal cliques. To describe this relationship more formally, we introduce the Definition 4:

Definition 4. Cover Set. Given a pivot vertex $u \in P \cup X$, the cover set of u is defined as $CS_u = P \cap N(u)$. Any vertex $v \in CS_u$ is said to be covered by the pivot u.

We say that the pivot vertex u covers the vertices in the corresponding Cover Set CS_u. Based on this definition, we further analyze how the BK_pivot algorithm restricts the scale of subproblems. Since the pivot vertex u is selected to maximize $|CS_u|$, it ensures that the remaining search branches $P \setminus N(u)$ are as small as possible. Consequently, the subgraph induced by each vertex $v \in P \setminus N(u)$ is strictly bounded in size by $n - |CS_u|$, while the branch exploring u corresponds to the largest subgraph of size $n - |CS_u|$.

By recursively expanding branches in this manner, BK_pivot is able to bound the scale of each subproblem, thereby controlling the growth of the recursion tree. This strategy is key to the algorithm's theoretical guarantee that its worst-case time complexity does not exceed $O(n3^{n/3})$.

B. How can we obtain the optimal search branch reduction?

Based on the above analysis, we realize that the BK_pivot algorithm prunes the Cover Set based on the indispensability of the pivot for potential maximal cliques. A natural question then arises: how can we obtain the optimal search branch reduction? For example, in Fig. 1, we could retain only two vertices from the original search branches without missing any maximal cliques. These two vertices together are sufficient to cover all maximal cliques, as each maximal clique contains at least one of them. Based on this intuition, we propose that an optimal search branch reduction $P' \subseteq P$ must include at least one vertex from every maximal clique in $G[P]$ while minimizing the size $|P'|$.

First, we want to clarify the correctness of retaining at least one vertex from each maximal clique in $G[P]$ in the reduced search branches P' without causing the loss of maximal clique enumeration. For a maximal clique C_i, $v_i \in C_i$ is retained in P' while $C_i \setminus \{v_i\}$ is discarded. Since $C_i \setminus \{v_i\}$ is adjacent to v_i and belongs to P, in the current recursive call, the search branch starting from v_i can access $C_i \setminus \{v_i\}$. Therefore, v_i ensures that the maximal clique C_i will not be lost. In the

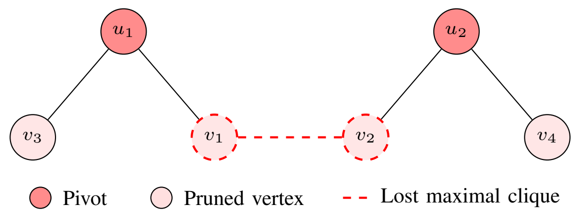

Fig. 3: An example of losing a maximal clique.

following Lemma 1, we introduce the notion of optimal search branch reduction.

Lemma 1. *A subset $P' \subseteq P$ is called an optimal search branch reduction if it intersects every maximal clique in $G[P]$, and no smaller subset $P'' \subset P$ with $|P''| < |P'|$ has the same property.*

Proof. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$ be the collection of vertex sets of all maximal cliques in $G[P]$, where each $C_i \subseteq P$ induces a maximal clique $G[C_i]$. To ensure that no maximal clique is missed during the enumeration, any reduced search branch $P' \subseteq P$ must satisfy: $\forall C_i \in \mathcal{C},\ C_i \cap P' \neq \emptyset$. Among all subsets $P' \subseteq P$ that satisfy this coverage condition, the one with the smallest cardinality is considered the optimal search branch reduction, as it retains the minimal number of vertices necessary to enumerate all maximal cliques in $G[P]$. □

Although we have proven that P' in Lemma 1 is the optimal search branch reduction, obtaining P' is not an easy task. In fact, this problem is equivalent to τ-cover finding [27], which is NP-hard.

C. Why can not we simply select multiple pivots?

Since directly solving for the optimal search branch reduction is an NP-hard problem, we aim to use a heuristic method to effectively futher reduce the search branches. Considering the simplicity and effectiveness of the BK_pivot algorithm, it naturally follows that we might be able to improve the effect by selecting multiple pivots. After all, each pivot can cover a new Cover Set, thereby further reducing the search branches. However, in fact, simply selecting multiple pivots and pruning multiple Cover Sets can result in missing maximal cliques during enumeration. Let's assume we select two pivots, u_1 and u_2. The vertices covered by u_1 and u_2 form the Cover Sets CS_{u_1} and CS_{u_2}, respectively. According to the idea in Section III-A, any vertices in CS_{u_1} and CS_{u_2} that can form a maximal clique will be found by the search branches of u_1 and u_2.

However, as shown in Fig. 3, if there exist vertices $v_1 \in CS_{u_1}$ and $v_2 \in CS_{u_2}$ that are adjacent and participate in the formation of a maximal clique, this maximal clique may not be found. This is because $v_2 \notin N(u_1)$ and $v_1 \notin N(u_2)$, meaning that when the search branch of u_1 calls the recursive function `BK_pivot` $(P \cap N(u_1), R \cup \{u_1\}, X \cap N(u_1))$, $v_2 \notin P \cap N(u_1)$ in Algorithm 1. Therefore, the search branch of u_1 cannot cover v_2. The same applies to u_2.

We observe that the vertices in the Cover Set CS_u of a pivot u are closely related to some additional vertices in the candidate set P. These additional vertices are those adjacent to any vertex in CS_u. These related vertices may be essential for preserving potential maximal cliques during enumeration. Therefore, when discarding CS_u, we must ensure that these related vertices are retained in the new search branches. We call this group of vertices the Pillar Set. To describe this formally, we present the Definition 5:

Definition 5. Pillar Set Given a pivot $u \in P \cup X$ and its cover set $CS_u \subseteq P$, the pillar set with respect to u is defined as $PS_u = \{v \in P \mid \exists w \in CS_u \text{ such that } (v, w) \in E\}$.

The Pillar Set PS_u includes all vertices in P that are adjacent to at least one vertex in CS_u, and the pivot u itself. If we choose a pivot u and discard the Cover Set CS_u, then all vertices in PS_u must be retained in the search branches to avoid missing potential maximal cliques. Although the BK_pivot algorithm does not explicitly mention the inclusion of PS_u, it implicitly retains it because $PS_u \subseteq P \backslash N(u)$. From this, we derive Lemma 2.

Lemma 2. *Given a pivot u and the corresponding CS_u, if CS_u is discarded, then PS_u must be retained in the search branches.*

Proof. Suppose there is a vertex p that has already been discarded from the current candidate set. Now the algorithm selects a new pivot u. We aim to prove Lemma 2 by contradiction. Suppose there is a vertex q in PS_u that does not need to be retained in the new search branches without causing the loss of maximal clique enumeration. Now, let us assume a situation where the discarded vertex p is adjacent to vertex q, and vertex p and vertex q form a maximal clique. In this case, since both vertex p and vertex q are both discarded, the maximal clique cannot be enumerated by any search branch below the current recursive call. This conclusion contradicts our assumption that there is a vertex q in PS_u that does not need to be retained in the new search branches without causing the loss of maximal clique enumeration. Therefore, Lemma 2 is proved. □

D. More Than Pivot

As discussed in the previous section, simply selecting multiple pivots and pruning their neighbors may result in missing maximal cliques. To overcome this challenge, we propose a heuristic algorithm to iteratively reduce the search branches, while ensuring the correctness of enumeration.

According to Lemma 2, the pruned Cover Set CS_u derived from a vertex u bounds a Pillar Set PS_u. In this section, for distinction, we refer to such vertex u as a splitter. Furthermore, we clarify the iterative pruning process as follows: (1) a vertex is first selected as the initiate splitter within the current search state $X \cup P$; (2) a Cover Set is identified based on the neighbors of the splitter and can be safely discarded; (3) a corresponding Pillar Set, consisting of vertices adjacent to the Cover Set, is retained. After this, a new splitter is selected from the remaining vertices to continue the pruning. This process repeats iteratively, progressively dividing P into smaller parts until all vertices in P have been processed through inclusion in a Cover Set, a Pillar Set, or as a splitter.

It should be noted that for any newly selected splitter u', its corresponding Cover and Pillar Sets must be computed only from the unprocessed part of the candidate set P; that is, previously assigned vertices should be excluded. Therefore, we further define the Residual Cover Set in Definition 7 and define the Residual Pillar Set in Definition 8, which are restricted to vertices not yet included in any prior Cover or Pillar Sets. We first define the union of all previously processed sets with respect the newly selected splitter in Definition 6.

Definition 6. (Processed Set) Given a splitter $u \in P$, the processed sets with respect to u, denoted by $Processed_u$, are defined as:

$$Processed_u = U \cup \bigcup_{v \in U} (CS_v \cup PS_v),$$

where U denotes the set of all vertices that have been selected as splitters prior to u.

Definition 7. (Residual Cover Set) Given a splitter $u \in P$, the Residual Cover Set with respect to u, is defined as:

$$RCS_u = (P \cap N(u)) \setminus Processed_u.$$

Definition 8. (Residual Pillar Set) Given the Residual Cover Set RCS_u, the Residual Pillar Set with respect to splitter u, is defined as:

$$RPS_u = (N(RCS_u) \cap P) \setminus Processed_u.$$

According to Definition 7 and Definition 8, we present the following lemma to guide the iterative reduction of search branches in the algorithm.

Lemma 3. *Given a splitter u and the corresponding RCS_u, if RCS_u is discarded, then RPS_u must be retained in the new search branches.*

Since $RCS_u \subseteq CS_u$ and $RPS_u \subseteq PS_u$, and according to Lemma 2, which guarantees that PS_u must be retained when CS_u is discarded, Lemma 3 evidently holds.

Moreover, discarding RCS_u does not lead to the loss of any maximal clique. One might be concerned that a previously discarded vertex could form a maximal clique together with some vertex $q \in RCS_u$, causing it to be missed. However, this is not possible: if a discarded vertex were adjacent to q, then q would have already been included in some prior RPS_v and thus excluded from the current RCS_u.

Based on the above analysis, each selected splitter u and its corresponding Residual Pillar Set RPS_u must be retained in the new search branches to ensure completeness. And all these splitters and Residual Pillar Sets form the new search branches after pruning. For the sake of simplicity, we collectively refer to all splitters and their associated Residual Pillar Sets as the

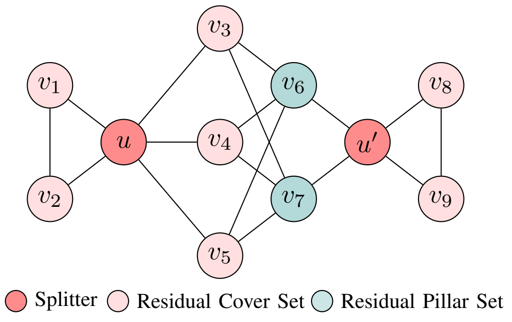

Fig. 4: Graph Representation of moreThanPivot Algorithm

Dominating Set of the original candidate set P. The formal definition of the Dominating Set is given in Definition 9.

Definition 9. (Dominating Set) Given the set of splitters U and their corresponding Residual Pillar Sets $\{RPS_u \mid u \in U\}$, the dominating set, denoted by DS, is defined as:

$$DS = \bigcup_{u \in U} (RPS_u \cup \{u\}). \tag{1}$$

Now we illustrate the process of the moreThanPivot algorithm in Fig. 4, while the complete and detailed procedure is provided in Algorithm 2 of Section III-E. In Fig. 4, moreThanPivot first selects an initial splitter $u \in P \cup X$ that maximizes $|N(u) \cap P|$. To simplify the illustration, only vertices in P are shown, while those in X are omitted. And identifies its Residual Cover Set ($RCS_u = \{v_1, v_2, v_3, v_4, v_5\}$) and Residual Pillar Set ($RPS_u = \{v_6, v_7\}$). Then, a vertex u' is selected as the next splitter from the unprocessed vertices and RPS_u, and the algorithm constructs $RCS_{u'} = \{v_8, v_9\}$. After all vertices in P have been processed, moreThanPivot finally returns $DS = RPS_u \cup \{u, u'\}$ as the refined search branches.

Furthermore, we analyze the worst-case time complexity of the moreThanPivot-enhanced algorithm under the BK framework. We demonstrate that the overall time complexity for enumerating and reporting all maximal cliques remains bounded by $O(n3^{n/3})$, which is consistent with that of BK_pivot.

Let BK_recursionMTP($G[X \cup P]$) denote an arbitrary recursive call within the moreThanPivot-enhanced algorithm, where $|X \cup P| = n$. Define $T_p(n)$ as the worst-case running time of BK_recursionMTP($G[X \cup P]$) when the moreThanPivot procedure returns a dominating set DS of size p. Suppose the initial splitter selected by moreThanPivot is u, and let $|P \setminus N(u)| = k$. Then $p \leq k$. Therefore, we have:

$$T_p(n) \leq \sum_{i=1}^{p} T(|G[X \cup P]_{v_i}|) + P(n),$$

where $v_i \in DS$, and $G[X \cup P]_{v_i} = G[X \cup P] \cap N(v_i)$. $P(n)$ accounts for the overhead of computing set intersections and executing the moreThanPivot process.

We observe the following property: $|G[X \cup P]_{v_i}| \leq n-k \leq n-1$. This holds because the initial splitter u is selected to maximize $|P \cap N(u)|$, which ensures that the subgraph explored in the branch of u is the largest among all branches, with size $n-k$. Consequently, for any branch $v_i \in DS$, the induced subgraph $G[X \cup P]_{v_i}$ is bounded by $n-k$, which is consistent with BK_pivot. As a result, we derive this recurrence relation:

$$T(n) = \max_{p}\{T_p(n)\} \leq C3^{n/3} - Q(n),$$

where C is a constant and $Q(n)$ is a polynomial term. The proof process for this recurrence relation is the same as the time complexity analysis of BK_pivot [18], so we omit it. Therefore, the worst-case running time of BK_recursionMTP is $O(3^{n/3})$, and $O(n3^{n/3})$ when outputting all maximal cliques. Similarly, the worst-case time complexity of calling BK_recursionMTP to enumerate and report maximal cliques under degeneracy order does not exceed $O(\lambda n 3^{\lambda/3})$, where λ is the degeneracy of the graph.

E. Selection Range and Objectives of New Splitters

In Section III-D, we have not explicitly defined the range for selecting splitters. Previous work [18] has shown that when selecting a single pivot, the maximum selection range is $P \cup X$. Now, we will discuss the maximum selection range for each new splitter in moreThanPivot. According to Lemma 3, we know that the Residual Cover Set bounds the Residual Pillar Set. Therefore, if the selection range of each new splitter u' in moreThanPivot includes the Residual Cover Set RCS_u with respect to the last splitter u, then RCS'_u with respect to the new splitter u' will overlap with RPS_u. This contradicts Lemma 3, Definition 7 and Definition 8.

Thus, the selection range of each new splitter u' in moreThanPivot should exclude the vertices in the previous Residual Cover Set. Additionally, the new splitter u' cannot be a vertex that has already been selected as a splitter. Otherwise, the newly selected splitter would be meaningless. Based on this analysis, we define the maximum selection range for each new splitter u' in moreThanPivot as $(X \cup P \setminus RCS_{global}) \setminus U$, where RCS_{global} is the set of all vertices that have been added to a Residual Cover Set, and U is the set of vertices that have already been selected as splitters. Since $(X \cup P \setminus RCS_{global}) \setminus U$ is the maximum selection range, choosing any vertex within this range as the new splitter in moreThanPivot is reasonable. We then present the three selection ranges considered in our method:

- Range (1):
  Select the new splitter u' from $(X \cup P \setminus RCS_{\text{global}}) \setminus U$. This selection range represents the most extensive range, as previously discussed. It is more likely to identify a splitter that achieves the best pruning effect, but the enlarged candidate scope also leads to increased computational overhead.
- Range (2):
  Select the new splitter u' from the Residual Pillar Set RPS_u with respect to the last splitter u. Choosing this selection range enables an immediate pruning of at least one vertex in the branching process.
- Range (3):
  Select the new splitter u' from RPS_u with respect to the last splitter u; if $RPS_u = \emptyset$, select the new splitter from $(X \cup P \setminus RCS_{\text{global}}) \setminus U$. This selection range tries to strike a balance between Range (1) and Range (2).

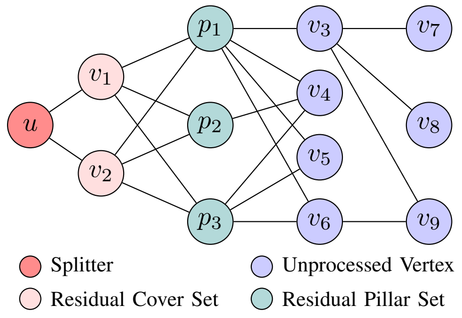

Fig. 5: Graph Representation of Three Greedy Objectives

Given the selection range, we further discuss the strategy for choosing a new splitter u'. In previous works where only a single pivot u needed to be chosen, a common strategy involved selecting the pivot that maximizes $P \cap N(u)$, as mentioned in Section II-B. However, in moreThanPivot, a new splitter should be chosen in each iteration of the loop. Therefore, a greedy objective is necessary to guide the algorithm in selecting the new splitter at each iteration.

There are two key factors that influence the greedy objective for selecting a new splitter u': (a) the number of vertices in the Residual Cover Set, $|RCS_{u'}|$, and (b) the number of vertices in the Residual Pillar Set, $|RPS_{u'}|$. Intuitively, a large $RCS_{u'}$ indicates more vertices can be pruned, while a small $RPS_{u'}$ implies fewer vertices need to be retained. Therefore, we explore three alternative greedy objectives for evaluating candidate splitters u': (1) $\arg\max_{v \in RPS_u} |RCS_v|$, (2) $\arg\min_{v \in RPS_u} |RPS_v|$, and (3) $\arg\max_{v \in RPS_u} (|RCS_v| - |RPS_v|)$. Fig. 5 illustrates an example where a new splitter u' is to be selected from $RPS_u = \{p_1, p_2, p_3\}$ with respect to the last splitter u. In the following, we take Range (2) as a representative selection range for choosing new splitters and analyze the computational cost associated with each of the three greedy objectives through illustrative examples.

- Objective (1):
  $u' = \arg\max_{v \in RPS_u} |RCS_v|$. This objective try to maximize the number of vertices in the Residual Cover Set RCS_v, which can be reduced from the search branches. In the example shown in Fig. 5, this objective selects p_1 as the new splitter due to its largest Residual Cover Set in RPS_u, where $RCS_{p_1} = \{v_3, v_4, v_5, v_6\}$, $RCS_{p_2} = \{v_4\}$, and $RCS_{p_3} = \{v_4, v_5, v_6\}$. To evaluate this objective, the algorithm iterates through each vertex $v \in RPS_u$, identifies its Residual Cover Set RCS_v by checking the neighbors of v in the current subgraph $G[P]$.

Since RCS_v only depends on v's direct neighbors in $G[P]$, the total cost of evaluating this objective is proportional to the number of edges in $G[P]$, denoted $|E_P|$. Hence, the time complexity for this greedy objective is $O(|E_P|)$.

- Objective (2): $u' = \arg\min_{v \in RPS_u} |RPS_v|$. This objective focuses solely on reducing the number of vertices in the Residual Pillar Set. In Fig. 5, this objective leads to the selection of p_2 as the new splitter due to its smallest Residual Pillar Set, where $RPS_{p_2} = \emptyset$, $RPS_{p_1} = \{v_7, v_8, v_9\}$, and $RPS_{p_3} = \{v_9\}$. This objective requires iterating through each vertex v in RPS_u, identifying v's Residual Cover Set RCS_v by checking v's neighbors in the current subgraph $G[P]$. Then, for each vertex in RCS_v, its neighbors in $G[P]$ are iterated to compute RPS_v, which involves inspecting the 2-hop neighbors of v in $G[P]$. Let $\deg_P(v)$ denote the degree of v in $G[P]$. The time complexity is thus $O\left(\sum_{v \in P} \deg_P(v)^2\right)$. In the worst case, this results in a time complexity of $O(|P|^2)$.
- Objective (3): $u' = \arg\max_{v \in RPS_u} (|RCS_v| - |RPS_v|)$. In Fig. 5, this objective leads to selecting p_3 as the new splitter, because $|RCS_{p_3}| - |RPS_{p_3}| = 2$, compared to $|RCS_{p_1}| - |RPS_{p_1}| = 1$ and $|RCS_{p_2}| - |RPS_{p_2}| = 1$. This objective balances two goals: minimizing the number of vertices retained in the new search branches and maximizing the potential for pruning. Similar to Objective (2), this objective similarly iterates through each vertex v in RPS_u to search $|RCS_v|$ and then explores the neighbors within $G[P]$ to compute $|RPS_v|$. Similarly, the time complexity of this greedy objective is $O(|P|^2)$ in the worst case.

We have introduced three selection ranges and three corresponding greedy objectives for choosing the new splitter. It is worth noting that, as introduced in Section III-D, moreThanPivot continues selecting splitters until no further selection is necessary. The purpose of this is to obtain the optimal search branch reduction as much as possible. However, it is not strictly necessary to do so. Since we have used RPS to ensure that selecting multiple splitters and discarding the RCS does not result in missing any maximal cliques, it is acceptable to choose any number of splitters and stop iterating at any time. In practice, due to diminishing marginal benefits, selecting too many splitters may incur excessive overhead that outweighs the benefits of further reducing search branches. Therefore, we introduce a iteration limit parameter S to limit the number of iterations for searching splitters in moreThanPivot. Moreover, we introduce a depth control parameter D, where moreThanPivot falls back to traditional pivot strategy once the recursion depth exceeds D. Based on these observations, we present the Algorithm 2 to obtain the new search branches DS'.

The worst-case time complexity of any moreThanPivot-enhanced algorithm remains bounded by $O(n3^{n/3})$. This is achieved by selecting the initial splitter as the vertex with the largest number of neighbors within P, which effectively limits the scale of subproblems during recursion, as discussed in Section III-D.

Algorithm 2: moreThanPivot Algorithm

```
Input: Graph $G$, Candidate set $P$, Exclusion set $X$,
       Recursive Depth $depth$, $D$, $S$.
Output: New search branches $DS'$.
if $depth > D$ then
    Select a pivot $u \in P \cup X$ that maximizes
      $|N(u) \cap P|$;
    return $DS' = P \setminus N(u)$;
$Processed \leftarrow \emptyset$, $Splitters \leftarrow \emptyset$, $DS' \leftarrow \emptyset$;
Select an initial splitter $u \in P \cup X$ that maximizes
  $|N(u) \cap P|$;
for $i = 1$ to $S$ do
    if $P \setminus Processed = \emptyset$ then
        break;
    $Splitters \leftarrow Splitters \cup \{u\}$;
    $RCS_u \leftarrow (P \cap N(u)) \setminus Processed$;
    $RPS_u \leftarrow (N(RCS_u) \cap P) \setminus Processed$;
    $Processed \leftarrow Processed \cup RCS_u \cup RPS_u \cup \{u\}$;
    $DS' \leftarrow DS' \cup RPS_u \cup \{u\}$;
    if $RPS_u \neq \emptyset$ then
        Select a new splitter $u'$ from the specified
          selection range using a greedy objective;
    else
        break;
foreach $v \in P \setminus Processed$ do
    $DS' \leftarrow DS' \cup \{v\}$;
return $DS'$
```

TABLE I: Statistics of Real-World Datasets

Graph	$\|V\|$	$\|E\|$	$d_{\max}$	λ	ρ
AS	1,696,415	11,095,298	35,455	111	7.71×10^{-6}
BY	6,008	156,945	2,557	64	8.70×10^{-3}
CA	18,771	198,050	504	56	1.12×10^{-3}
CH	27,769	352,285	2,468	37	9.10×10^{-4}
DP	300	21,928	229	98	4.89×10^{-1}
EC	986	16,064	345	34	3.31×10^{-2}
EE	36,692	183,831	1,383	43	2.70×10^{-4}
EF	4,039	88,234	1,045	115	1.08×10^{-2}
LB	58,228	214,078	1,134	52	1.30×10^{-4}
LG	196,591	950,327	14,730	51	4.92×10^{-5}
SE	75,879	405,740	3,044	67	1.40×10^{-4}
SS	82,168	504,230	2,552	55	1.50×10^{-4}
TC	1,791,489	25,444,207	238,342	99	1.59×10^{-5}
WB	685,230	6,649,470	84,230	201	2.83×10^{-5}
WT	2,394,385	4,659,565	100,029	131	1.63×10^{-6}
WV	7,115	100,762	1,065	53	3.98×10^{-3}

IV. Experiment

A. Experimental Settings

Datasets. In our experiments, we utilize both real-world and synthetic datasets. A total of 16 real-world datasets are used, all of which are publicly available from SNAP [28], BioGrid [29], and DIMACS challenge [30]. Details of these datasets are summarized in Table I. Due to space limitations, the full dataset names are provided in Table II. Here, $d_{\max}$ denotes the maximum degree of the graph, λ denotes the degeneracy of the graph, and ρ denotes the edge density of the graph, defined as $\rho = \frac{2|E|}{|V|(|V|-1)}$. Prior to conducting the experiments, we performed preprocessing on each dataset. Specifically, we removed self-loops, ignored edge directions and weights, and discarded any vertex attributes and weights. Synthetic datasets are generated based on the Erdős–Rényi random graph model [31]. There are two common variants of this model: $G(n, p)$, where each possible edge between n vertices is independently generated with probability p; and $G(n, m)$, where exactly m edges are randomly selected from all possible edges. In our setting, we adopt the $G(n, m)$ model. Specifically, we specify the number of vertices $|V|$ and the desired edge density ρ, then compute the number of edges, and randomly connect m edges among the vertices. For each parameter setting, we generate five independent synthetic datasets to ensure robustness. We run experiments on each group of synthetic datasets and report the average results.

Algorithms. We evaluate the performance of four maximal clique enumeration algorithms: the BK_degen algorithm, the BK_rcd algorithm, the RMCE algorithm,and the HBBMC algorithm. We integrate the proposed moreThanPivot into each of these algorithms to assess its effectiveness.

- BK_degen [23] is one of the most influential variants under the BK framework and generally achieves strong performance across most datasets.
- BK_rcd [25] partially follows the BK_degen framework and incorporates a top-down enumeration strategy to better handle dense regions in the graph.
- RMCE_degen [26] enhances the BK_degen algorithm by incorporating multiple graph reduction techniques to improve efficiency.
- HBBMC [24] utilizes an edge-oriented search order alongside a termination strategy to enhance enumeration efficiency.
- BK_degenMTP, BK_rcdMTP, RMCE_degenMTP, and HBBMCMTP are moreThanPivot-enhanced variants of their respective baselines.

All algorithms are implemented in C++. The source codes of BK_degen, BK_rcd, and RMCE_degen are obtained from the authors' GitHub repositories[1][2][3]. Since the source code of HBBMC is not available[4], we reimplement it based on the codebase of BK_degen. Our source code and used datasets are available at https://github.com/Zhaoyi199875/moreThanPivot. All experiments are conducted on a Linux server equipped with an AMD EPYC 7543 32-Core Processor @ 2.80GHz and 128G RAM.

[1]https://github.com/darrenstrash/quick-cliques
[2]https://github.com/CGCL-codes/BKrcd
[3]https://github.com/DengWen0425/RMCE
[4]https://github.com/wangkaixin219/HBBMC

B. Overall Results

We evaluate the overall efficiency of the eight maximal clique enumeration algorithms listed in the section IV-A across 16 real-world datasets. In this experiment, for all moreThanPivot-enhanced algorithms, the depth control parameter is set to $D = 1$, restricting the use of moreThanPivot to recursion depth 1. The iteration limit parameter is set to $S = 2$, limiting the number of splitters selected in moreThanPivot to at most two. And we choose Range (2) and Objective (3) described in Section III-E for selecting new splitters. The running times of the eight algorithms are reported in Table II. Table III summarizes the speedup achieved by our moreThanPivot-enhanced algorithms compared to their respective baseline methods across all real-world datasets. The speedup is defined as:

$$\text{Speedup\%} = \left(\frac{T_{\text{baseline}}}{T_{\text{MTP}}} - 1\right) \times 100\%,$$

where T_{baseline} and T_{MTP} denote the running time of the baseline and moreThanPivot-enhanced algorithms, respectively. Our speedup measure is adapted from [26], where the original formulation is defined as $T_{\text{baseline}}/T_{\text{new}}$, in order to highlight the performance difference.

Our methods consistently achieve state-of-the-art performance across all datasets, effectively enhancing the efficiency of each baseline. On average, the MTP variants improve BK_degen, BK_rcd, RMCE_degen, and HBBMC by 26.1%, 80.2%, 10.1%, and 17.8%, respectively. For example, BK_degenMTP achieves performance gains of 32.4%, 32.4%, 45.5%, and 39.2% over BK_degen on AS, DP, EF, and WB, respectively. BK_rcdMTP demonstrates substantial speedups of 134.5% (AS), 193.9% (DP), 300% (EF), and 169.2% (WT), with the most significant improvement observed on the high-density graph EF. RMCE_degenMTP also achieves better performance over RMCE_degen, improving efficiency by 13.7%, 14.4%, and 15.1% on DP, EF, and WV, respectively. Similarly, HBBMCMTP outperforms HBBMC on AS, BY, SE, WB, and WV by 22.5%, 22.1%, 21.5%, 23.0%, and 22.0%. Among all variants, RMCE_degenMTP achieves the best overall performance, recording the lowest running time on 14 out of 16 datasets and outperforming the baseline RMCE_degen consistently. Although RMCE_degenMTP does not exhibit substantial improvements over RMCE_degen on the TC and WB datasets, the state-of-the-art algorithms on these two datasets are BK_degenMTP and BK_rcdMTP, respectively. These methods achieve 11.5% and 13.2% improvements over their corresponding baselines on TC and WB, respectively. Moreover, BK_degenMTP is 143.7% faster than RMCE_degen on TC, and BK_rcdMTP outperforms RMCE_degen by 259.7% on WB. These results demonstrate the effectiveness of moreThanPivot and confirm that moreThanPivot is highly effective in combination with or even in place of reduction-based strategies on challenging datasets.

We further investigate the impact of edge density on the effectiveness of the moreThanPivot-enhanced algorithms.

TABLE II: Overall Running Time (in seconds)

Abbr	Graph	BK_degen	BK_degenMTP	BK_rcd	BK_rcdMTP	RMCE_degen	RMCE_degenMTP	HBBMC	HBBMCMTP
AS	as-skitter	56.7047	42.8215	99.9735	42.6380	42.4329	**38.5377**	74.3639	60.6948
BY	biogrid-yeast	0.8837	0.6866	1.2015	0.6933	0.6411	**0.5615**	1.0919	0.8945
CA	ca-AstroPh	0.2499	0.2093	0.2359	0.2110	0.1217	**0.1725**	0.2658	0.2305
CH	cit-HepTh	0.7553	0.6260	0.8415	0.6344	0.5021	**0.4691**	0.8922	0.7454
DP	dimacs-p_hat300-2	59.9650	45.2829	134.4256	45.7434	42.3350	**37.2495**	90.0880	76.0121
EC	email-Eucore	0.0475	0.0366	0.0581	0.0372	0.0321	**0.0284**	0.0593	0.0492
EE	email-Enron	0.3444	0.2820	0.3917	0.2906	0.2401	**0.2130**	0.4026	0.3515
EF	ego-Facebook	596.4452	410.0462	1661.7520	415.4614	344.6504	**301.2422**	1551.3860	1392.6540
LB	loc-Brightkite	0.3442	0.2892	0.4200	0.2909	0.1870	**0.1725**	0.4580	0.3905
LG	loc-Gowalla	1.6555	1.4515	1.8408	1.4442	1.2603	**1.1847**	1.9696	1.8145
SE	soc-Epinions1	1.8042	1.4329	2.4312	1.4273	1.2890	**1.1320**	2.2646	1.8647
SS	snap-slashdot0902	0.9474	0.8015	1.1699	0.8051	0.6234	**0.5787**	1.1823	1.0056
TC	wiki-topcats	53.2432	**47.7542**	55.9614	48.5288	116.3630	115.1734	60.2078	54.0913
WB	web-BerkStan	7.7973	5.6031	3.9550	**3.4939**	12.5670	12.5295	6.3509	5.1638
WT	snap-soc-wiki-Talk	80.1285	61.6752	167.4576	62.1941	57.9559	**51.1633**	111.0546	93.0989
WV	snap-wiki-Vote	0.3459	0.3497	0.5588	0.3465	0.3193	**0.2774**	0.5431	0.4452

TABLE III: Relative Speedup of moreThanPivot-Enhanced Algorithms on Real-World Datasets

Graph	BK_degen↑	BK_rcd↑	RMCE_degen↑	HBBMC↑
AS	32.4%	134.5%	10.1%	22.5%
BY	28.7%	73.3%	14.2%	22.1%
CA	19.4%	11.8%	11.1%	15.3%
CH	20.6%	32.6%	7.0%	19.7%
DP	32.4%	193.9%	13.7%	18.5%
EC	29.8%	56.1%	13.0%	20.7%
EE	22.1%	34.8%	12.7%	14.5%
EF	45.5%	300.0%	14.4%	11.4%
LB	19.0%	44.4%	8.5%	17.3%
LG	14.1%	27.5%	6.4%	8.5%
SE	25.9%	70.3%	13.9%	21.4%
SS	18.2%	45.3%	7.7%	17.6%
TC	11.5%	15.3%	1.0%	11.3%
WB	39.2%	13.2%	0.03%	23.0%
WT	29.9%	169.2%	13.3%	19.3%
WV	28.8%	61.3%	15.1%	22.0%
Average	26.1%	80.2%	10.1%	17.8%

Among all datasets, DP, EC, and EF exhibit the high edge densities, 0.4489, 0.0331, and 0.0108 respectively, significantly higher than those of the other datasets. These datasets serve as representative dense graphs, where the search branches for clique enumeration are typically more numerous and contain a substantially greater number of maximal cliques. This results in increased computational intensity. On these dense graphs, our moreThanPivot-enhanced algorithms achieve performance improvements that are significantly higher than the average gains observed across all datasets. For instance, BK_degenMTP outperforms BK_degen by 32.4%, 29.8%, and 45.5% on DP, EC, and EF, respectively. BK_rcdMTP improves over BK_rcd by 193.9%, 156.1%, and 300% on the same datasets. Similarly, RMCE_degenMTP achieves gains of 13.7%, 13.0%, and 14.4%. And HBBMCMTP outperforms HBBMC by 18.5% and 20.7% on DP and EC, respectively.

These results are consistent with the pruning rationale behind the moreThanPivot algorithm. In dense graphs, each vertex is connected to a large number of others, leading to a huge increase in the size of potential candidate set. Consequently, selecting a new splitter can eliminate a substantially larger portion of the search branches, making the pruning more effective and the performance improvement more pronounced.

C. Performance Evaluation of Different Selection Ranges

In this subsection, we aim to compare the effectiveness of the three different selection ranges for selecting new splitters, as introduced in Section III-E. We apply these three selection ranges to the BK_degenMTP algorithm, denoted as BK_degenMTP-R1, BK_degenMTP-R2, and BK_degenMTP-R3, which correspond to Range (1), Range (2), and Range (3), respectively. For brevity, these variants are referred to as R1, R2, and R3 in Table IV. In this experiment, we set $D = 1$, $S = 2$ in moreThanPivot, and choose Objective (3) for selecting new splitters. We conducted experiments on the real-world datasets, recording the running times of the three methods.

Overall, the performance differences among the three methods are not significant. R2 and R3 together achieve the best running time on 11 out of 16 datasets, suggesting the effectiveness of selecting new splitters from the Local Pillar Set, which can immediately reduce one search branch. On the other hand, Range (2) has the smallest selection scope among the three, indicating its advantage in avoiding high computational overhead. R1 outperforms the others on 5 datasets, showing that exploring a broader selection range can also lead to good splitter choices.

D. Performance Evaluation of Different Greedy Objectives

In this subsection, we aim to compare the effectiveness of the three different greedy objectives for selecting new splitters, as introduced in Section III-E. We apply these three greedy objectives to the BK_degenMTP algorithm, denoted as BK_degenMTP-O1, BK_degenMTP-O2, and BK_degenMTP-O3, which correspond to Objective (1), Objective (2), and Objective (3), respectively. For brevity, these variants are referred to as O1, O2, and O3 in Table V. In this experiment, we set $D = 1$, $S = 2$ in moreThanPivot, and choose Range (2) as the selection range for selecting new splitters. We conducted experiments on 16 real-world datasets , reporting the running times.

Overall, BK_degenMTP-O3 achieves faster running times compared to BK_degenMTP-O1 and BK_degenMTP-O2 on eight datasets. This demonstrates the effectiveness of the

TABLE IV: Running Time of BK_degenMTP with Different Selection Ranges (in seconds)

Range	AS	BY	CA	CH	DP	EC	EE	EF	LB	LG	SE	SS	TC	WB	WT	WV
R1	43.6783	0.7053	0.2346	0.6612	45.8536	0.0369	0.3200	415.1162	**0.2789**	**1.5418**	1.6139	**1.0151**	67.1650	6.2778	**66.3249**	**0.3753**
R2	43.7721	0.7206	**0.2102**	0.6665	**45.7564**	**0.0367**	0.2945	**412.8616**	0.2900	1.5423	**1.5938**	1.0891	**66.8531**	6.2571	67.9666	0.3764
R3	**43.4693**	**0.6984**	0.2195	**0.6485**	47.3141	0.0370	**0.2885**	416.4384	0.2850	1.5780	1.6852	1.0470	66.9697	**6.2307**	68.0079	0.3881

TABLE V: Running Time of BK_degenMTP with Different Greedy Objectives (in seconds)

Objective	AS	BY	CA	CH	DP	EC	EE	EF	LB	LG	SE	SS	TC	WB	WT	WV
O1	43.5253	0.7022	0.2155	0.6332	46.1867	0.0380	0.2893	430.1092	**0.2811**	1.4566	1.4505	**0.7832**	48.0027	5.6718	62.8594	0.3497
O2	42.8644	0.6910	0.2101	0.6335	**44.9092**	**0.0364**	0.2876	**409.4788**	0.2889	1.4921	**1.4057**	0.7907	48.3753	5.6156	**61.3730**	**0.3428**
O3	**42.8215**	**0.6866**	**0.2093**	**0.6260**	45.2829	0.0366	**0.2820**	410.0462	0.2891	**1.4515**	1.4329	0.8015	**47.7542**	**5.6031**	61.6752	0.3458

TABLE VI: Running Time of BK_degenMTP with Different Depth Control Parameter D (in seconds)

D	AS	BY	CA	CH	DP	EC	EE	EF	LB	LG	SE	SS	TC	WB	WT	WV
1	**42.3554**	**0.6680**	**0.2018**	**0.6008**	46.5235	**0.0365**	**0.2815**	**409.3287**	**0.2630**	1.4282	**1.4074**	**0.7369**	**47.4266**	**5.5967**	**62.7636**	**0.3534**
2	42.5131	0.6873	0.2034	0.6200	46.3798	0.0371	0.2874	415.0957	0.2668	**1.4203**	1.4438	0.7922	48.5215	5.6659	63.2898	0.3574
3	42.5640	0.6989	0.2053	0.6194	**46.2455**	0.0377	0.2916	414.6340	0.2689	1.4682	1.4674	0.7848	48.1588	5.7550	63.8606	0.3657
∞	47.0189	0.7649	0.2087	0.6536	54.3862	0.0419	0.3117	506.2397	0.2915	1.5858	1.6047	0.8032	48.6499	6.0650	72.2662	0.3925

TABLE VII: Running Time of BK_degenMTP with Different Iteration Limits S (in seconds)

S	AS	BY	CA	CH	DP	EC	EE	EF	LB	LG	SE	SS	TC	WB	WT	WV
2	41.9727	**0.6508**	0.2019	0.6169	44.2862	**0.0349**	**0.2753**	390.4243	0.2706	1.3705	**1.3044**	**0.6607**	42.5639	5.3748	**59.6232**	0.3318
3	42.2390	0.6642	0.2049	0.6093	44.3779	0.0351	0.2794	391.8393	**0.2634**	**1.3238**	1.3167	0.6657	42.4999	5.3880	60.0433	**0.3317**
4	**41.7298**	0.6825	**0.2006**	0.5917	45.0579	0.0350	0.2827	**389.7513**	0.2720	1.3631	1.3280	0.6757	42.4987	5.4415	59.8256	0.3330
5	42.0645	0.6891	0.2077	0.6135	**44.1161**	0.0350	0.2812	398.3500	0.2754	1.4439	1.3352	0.6825	42.5036	5.6013	60.0202	0.3329
∞	42.1367	0.6658	0.2081	**0.5789**	44.2399	0.0352	0.2822	398.9097	0.2654	1.3497	1.3434	0.6627	**42.2748**	**5.3598**	59.6364	0.3328

greedy objective $u' = \arg\max_{v \in RPS_u} (|RCS_v| - |RPS_v|)$ that balances between pruning and preserving branches. It is worth noting that on the high-density dataset EF, both BK_degenMTP-O2 and BK_degenMTP-O3 significantly outperform BK_degenMTP-O1. On the other hand, BK_degenMTP-O1 and BK_degenMTP-O2 also show competitive performance on several datasets, indicating that all three greedy objectives can be effective in different scenarios. Considering overall performance, we recommend using BK_degenMTP-O3 as the default choice in practice.

E. Performance Evaluation across Recursive Depths

In this subsection, we evaluate the effectiveness of the moreThanPivot algorithm under different settings of its depth control parameter D. Specifically, we integrate moreThanPivot into the BK_degen algorithm and set D to 1, 2, 3, and infinity (denoted as ∞). When $D = \infty$, moreThanPivot never reverts to the traditional pivoting strategy. For other values of D, moreThanPivot reverts to traditional pivoting only when the current recursion depth exceeds D. In this experiment, we use Range (2) and Objective (3) in Section III-E for selecting new splitters in moreThanPivot, and set $S = 2$. The running times of these four configurations are presented in Table VI. Overall, among the 16 tested datasets, setting $D = 1$ achieved the best performance on 14 of them. For example, at $D = 1$, the running time on the EF dataset is 409.33 seconds, which is clearly better than other settings. When the recursion depth increases to 2 and 3, although the running time decreases on some datasets such as DP and LG, the overall improvement is limited. On the other hand, applying moreThanPivot across the full recursion tree leads to a significant increase in runtime on most datasets. For instance, on AS and EF, the running times increase from 42.36 seconds and 409.33 seconds to 47.02 seconds and 506.24 seconds, respectively. This suggests that the extra computational overhead caused by increasing the recursion depth cannot be offset by efficiency gains. These results indicate that while moreThanPivot can be applied at greater recursion depths, setting $D = 1$ is sufficient in most cases. This is because, at depth 1, the candidate set for branching is usually larger, making it more likely that selecting multiple splitters is worthwhile.

F. Performance Evaluation under Varying Iteration Limits

In this subsection, we evaluate the impact of the iteration limit S in moreThanPivot, which restricts the number of new splitters that can be selected. We apply moreThanPivot to the BK_degen algorithm and restrict its usage to recursion depth 1 only. We use Range (2) and Objective (3) described in Section III-E for selecting new splitters. Specifically, we vary the maximum number of allowed iterations S from 2 to 5, and $S = \infty$, where no iteration limit is imposed and the algorithm continues until no further splitter can be found. Table VII reports the average running time across 16 datasets under each setting.

Overall, the experimental results indicate that no single setting yields an overwhelming advantage across all 16 datasets. Among the different settings, $S = 2$ achieves the best performance on 6 datasets, suggesting it as a generally effective choice. In contrast, the unlimited iteration setting ($S = \infty$) does not consistently provide better results, only achieves

TABLE VIII: Runtime of Each Algorithm on Synthetic Datasets ($|V| = 10,000$) with Different Edge Densities ρ (in seconds)

ρ	BK_degen	BK_degenMTP	BK_rcd	BK_rcdMTP	RMCE_degen	RMCE_degenMTP	HBBMC	HBBMCMTP
0.02	2.5051	1.9127	2.3328	1.9177	2.0183	1.5579	2.3352	2.0494
0.03	6.3829	5.0262	6.7413	4.9586	5.6880	4.1010	7.1653	5.3539
0.04	15.3927	10.7091	16.1657	11.1182	13.5127	9.9710	17.3527	12.4002
0.05	35.1820	24.1575	40.1391	23.9772	28.7652	20.2178	36.7147	24.9540
0.06	71.3129	45.7866	87.0001	44.7051	55.6910	38.9986	72.2721	44.8292
0.07	140.993	84.4920	188.082	82.0688	114.620	71.9194	142.516	86.3842
0.08	257.828	142.465	386.878	145.660	224.655	129.687	277.921	149.770
0.09	484.591	264.889	764.724	247.123	381.059	208.006	478.543	249.103
0.10	811.229	444.967	1440.110	433.426	703.753	366.952	876.899	438.980

TABLE IX: Relative Speedup of moreThanPivot-Enhanced Algorithms on Synthetic Datasets with Varying Densities ρ

ρ	BK_degen↑	BK_rcd↑	RMCE↑	HBBMC↑
0.02	31.0%	21.6%	29.5%	13.9%
0.03	27.0%	36.0%	38.7%	33.8%
0.04	43.7%	45.4%	35.5%	39.9%
0.05	45.6%	67.4%	42.3%	47.1%
0.06	55.8%	94.6%	42.8%	61.2%
0.07	66.9%	129.2%	59.4%	65.0%
0.08	81.0%	165.6%	73.2%	85.6%
0.09	82.9%	209.5%	83.2%	92.1%
0.10	82.3%	232.3%	91.8%	99.8%

best performance on CH, TC and WB. In fact, on several datasets such as EF, $S = \infty$ yields noticeably higher running times. This suggests that excessive iterations may lead to diminishing returns or even introduce unnecessary overhead. Moreover, in some cases, even when the iteration limit is set to $S = \infty$, the algorithm may not find more splitters compared to other settings. These findings indicate that it is not necessary to exhaustively search for the best splitter in every call to moreThanPivot.

G. Performance on Synthetic Datasets with Varying Edge Densities

To evaluate the performance of our proposed moreThanPivot algorithm on datasets with varying edge densities, we conducted experiments on a series of synthetic datasets with fixed vertex count ($|V| = 10,000$) and edge density gradually increasing from 0.02 to 0.1. Table VIII reports the average runtime of four baseline algorithms (BK_degen, BK_rcd, HBBMC, and RMCE) alongside their corresponding variants enhanced with moreThanPivot. All moreThanPivot-enhanced variants adopt the Range (2) and Objective (3) described in Section III-E for selecting new splitters, and set $D = 1$ and $S = 2$. Table IX presents the relative speedups achieved by our moreThanPivot-enhanced algorithms compared to their respective baselines.

Overall, we observe consistent performance improvements across all datasets when the baselines are enhanced by moreThanPivot. Moreover, the performance gains become increasingly pronounced as the edge density grows. For instance, the runtime of RMCE drops from 703.7530 seconds to 366.9520 seconds at $\rho = 0.1$, yielding a 91.8% speedup. At $\rho = 0.02$, the improvement is 29.5%. Similar trends are observed for BK_degenMTP and HBBMC_MTP, with peak speedups of 82.9% and 99.8%, respectively. Among the four algorithms, BK_rcd benefits the most from the integration of moreThanPivot, with its speedup increasing steadily from 21.6% at $\rho = 0.02$ to 232.3% at $\rho = 0.1$. At lower densities (e.g., $\rho = 0.03$), the improvements are also noticeable, with BK_degen, BK_rcd, RMCE, and HBBMC achieving 27.0%, 36.0%, 38.7%, and 33.8% speedups, respectively. These results confirm the effectiveness of moreThanPivot across datasets of different densities and highlight its particular advantage in handling dense graphs, where the potential for pruning is significantly higher.

H. Performance on Synthetic Datasets with Varying Graph Size

To evaluate the performance of our proposed moreThanPivot algorithm on relatively dense datasets with varying numbers of vertices, we conducted experiments on a series of synthetic datasets with a fixed edge density of 0.05, where the number of vertices gradually increases from 2k to 40k. Table X reports the average runtime of eight algorithms, while Table XI presents the relative speedups achieved by our moreThanPivot-enhanced algorithms compared to their respective baselines. All moreThanPivot-enhanced variants adopt the Range (2) and Objective (3) described in Section III-E for selecting new splitters, and set $D = 1$ and $S = 2$.

Overall, as the graph size increases, the moreThanPivot-enhanced algorithms consistently achieve better runtime performance than their baselines across all tested scales. Specifically, for the smallest graphs with 2k vertices, the enhanced versions reduce runtime by approximately 20%–30%. As the graph size increases, the relative speedup becomes more pronounced, indicating that moreThanPivot is more effective at reducing the search branches in large-scale graphs. For instance, on the largest dataset with 40k vertices, BK_degen takes about 9594 seconds, whereas BK_degenMTP reduces this to 5481 seconds, achieving a 75.0% runtime reduction. BK_rcd exhibits the most significant improvement, with a speedup of 153.0% at the 40k scale. RMCE and HBBMC also show substantial improvements of 88.5% and 86.5%, respectively. In summary, moreThanPivot demonstrates increasingly significant performance improvements on dense graphs as the number of vertices grows.

V. Related Work

Maximal Clique Enumeration (MCE) is a fundamental and classical problem in the field of graph data analysis

TABLE X: Runtime of Each Algorithm on Synthetic Datasets ($\rho = 0.05$) with Varying Graph Sizes $|V|$ (in seconds)

$\lvert V\rvert$	BK_degen	BK_degenMTP	BK_rcd	BK_rcdMTP	RMCE_degen	RMCE_degenMTP	HBBMC	HBBMCMTP
2k	0.1363	0.1086	0.1397	0.1062	0.1248	0.1000	0.1557	0.1209
4k	1.2390	0.9214	1.3117	0.8856	1.0846	0.7983	1.3481	0.9368
6k	4.9313	3.5707	5.3846	3.4859	4.2070	3.0753	5.2832	3.6220
8k	13.9022	9.7125	15.5027	9.5100	11.6470	8.3756	14.8024	9.9349
10k	31.9520	21.6339	36.3359	21.1961	26.4726	18.7556	33.8505	22.2957
20k	552.256	314.345	690.728	305.051	453.500	275.517	575.401	310.530
30k	2912.83	1645.26	3987.43	1588.51	2465.53	1414.48	3055.30	1558.13
40k	9594.04	5481.91	14708.8	5810.74	9058.81	4805.46	10656.2	5721.58

TABLE XI: Relative Speedup of moreThanPivot-Enhanced Algorithms on Synthetic Datasets with Varying Graph Sizes $|V|$

$\lvert V\rvert$	BK_degen↑	BK_rcd↑	RMCE↑	HBBMC↑
2k	25.4%	31.5%	24.7%	28.8%
4k	34.5%	48.1%	35.9%	43.9%
6k	38.1%	54.5%	36.8%	45.9%
8k	43.1%	63.0%	39.1%	48.9%
10k	47.7%	71.4%	41.2%	51.8%
20k	75.7%	126.4%	64.6%	85.3%
30k	77.0%	151.0%	74.3%	96.1%
40k	75.0%	153.2%	88.5%	86.5%

and mining. Numerous algorithms have been developed to solve this problem in various settings [32]–[54]. A widely recognized algorithmic framework for solving MCE is the Bron–Kerbosch (BK) algorithm [17], first proposed by Bron and Kerbosch in 1973. This algorithm adopts a recursive backtracking framework to enumerate all maximal cliques in the graph without duplication. Following the original BK algorithm, Tomita et al. [18] proved the BK_pivot algorithm with the worst-case time complexity of $O(n3^{n/3})$. Due to its superior performance, the pivot-based pruning strategy has become a standard technique adopted by almost all subsequent MCE algorithms. Subsequently, Eppstein et al. [23] proposed another well-known algorithm called BK_degen with the worst-case time complexity of $O(n3^{\lambda/3})$. They introduced the degeneracy order of the graph as the top-level enumeration sequence within the BK framework. By using the degeneracy order, the size of the candidate set in each recursive branch can be bounded within the graph's degeneracy λ. In addition, to accelerate the pivot selection and reduce space overhead, they designed a space-efficient data structure, further enhancing the overall performance of the algorithm.

Subsequently, some optimizations for the pivot selection strategy were proposed. Naudé et al. [20] designed a set of specific rules to handle various special cases. These rules allow the algorithm to immediately terminate the pivot selection process when such conditions are detected. On the other hand, Segundo et al. [21] proposed maintaining the vertices in the current subgraph in descending order of degree at each recursive branch. This enables the algorithm to directly select the first vertex in the sorted list as the pivot. To improve the efficiency of pivot selection, Jin et al. [22] introduced the FACEN algorithm, which exploits the degeneracy order to enhance pivot selection efficiency. At each recursion level, it preserves the order within the candidate set and selects the last vertex as the pivot, motivated by the observation that later vertices in the order often exhibit higher degrees.

Recently, three state-of-the-art algorithms have attracted considerable attention. Li et al. [25] proposed the BK_rcd algorithm, which adopts a top-down enumeration strategy specifically designed for dense subgraphs. They integrated this approach with the BK_degen algorithm. To further improve efficiency, they developed a formula to adaptively select the most suitable enumeration strategy based on the structural properties of the current subgraph. Deng et al. [26] proposed a reduction-based MCE framework called RMCE_degen. They observed that a large number of branches in the enumeration tree are unnecessarily revisited. Thereby, they proposed the including global reduction, dynamic reduction, and maximality check reduction techniques, to eliminate unnecessary enumeration branches. Wang et al. [24] proposed the HBBMC algorithm, which replaces the degeneracy order with the truss order, thereby achieving better time complexity in the worst-case. Additionally, the authors introduced an early termination mechanism that allows maximal cliques satisfying certain conditions to be reported early.

Although the existing methods use different strategies to speed up MCE, only BK_pivot explores how to prune the candidate set. In this paper, we design a heuristic algorithm called moreThanPivot to obtain an improved search branch reduction.

VI. Conclusion

In this paper, we propose a novel pruning algorithm, moreThanPivot, to further reduce search branches in the Bron-Kerbosch algorithm for maximal clique enumeration, beyond the capability of the traditional pivot strategy. Our method iteratively selects multiple splitters and constructs their Residual Cover Sets and Residual Pillar Sets to generate refined search branches with fewer recursive calls. To guide splitter selection, we introduce three alternative selection ranges and greedy objectives. Extensive experiments on 16 real-world graphs confirm the superiority of moreThanPivot over existing methods. Moreover, evaluations on synthetic datasets show that our method is particularly effective on graphs with high edge density.

One future direction is to explore more efficient and effective search branch reduction, i.e. other branch reduction strategies, better than the pivot strategy. Considering that obtaining the optimal reduction is NP-hard, there could be more interesting heuristic solutions.

References

[1] S. Tabassum, F. S. Pereira, S. Fernandes, and J. Gama, "Social network analysis: An overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 5, p. e1256, 2018.

[2] X. Wen, W.-N. Chen, Y. Lin, T. Gu, H. Zhang, Y. Li, Y. Yin, and J. Zhang, "A maximal clique based multiobjective evolutionary algorithm for overlapping community detection," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 3, pp. 363–377, 2016.

[3] H.-C. Yi, Z.-H. You, D.-S. Huang, and C. K. Kwoh, "Graph representation learning in bioinformatics: trends, methods and applications," *Briefings in Bioinformatics*, vol. 23, no. 1, p. bbab340, 2022.

[4] F. N. Abu-Khzam, N. E. Baldwin, M. A. Langston, and N. F. Samatova, "On the relative efficiency of maximal clique enumeration algorithms, with applications to high-throughput computational biology," in *International Conference on Research Trends in Science and Technology*, 2005, pp. 1–10.

[5] T. Matsunaga, C. Yonemori, E. Tomita, and M. Muramatsu, "Clique-based data mining for related genes in a biomedical database," *BMC bioinformatics*, vol. 10, pp. 1–9, 2009.

[6] H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein, "Predicting interactions in protein networks by completing defective cliques," *Bioinformatics*, vol. 22, no. 7, pp. 823–829, 2006.

[7] S. Bhatt, S. Padhee, A. Sheth, K. Chen, V. Shalin, D. Doran, and B. Minnery, "Knowledge graph enhanced community detection and characterization," in *Proceedings of the twelfth ACM international conference on web search and data mining*, 2019, pp. 51–59.

[8] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Transactions on Database Systems (TODS)*, vol. 36, no. 4, pp. 1–34, 2011.

[9] D. M. Strickland, E. Barnes, and J. S. Sokol, "Optimal protein structure alignment using maximum cliques," *Operations research*, vol. 53, no. 3, pp. 389–402, 2005.

[10] W. Zhang and X. Zou, "A new method for detecting protein complexes based on the three node cliques," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, no. 4, pp. 879–886, 2014.

[11] K. Biswas, V. Muthukkumarasamy, and E. Sithirasenan, "Maximal clique based clustering scheme for wireless sensor networks," in *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. IEEE, 2013, pp. 237–241.

[12] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, "On generating all maximal independent sets," *Information Processing Letters*, vol. 27, no. 3, pp. 119–123, 1988.

[13] J. L. Walteros and A. Buchanan, "Why is maximum clique often easy in practice?" *Operations Research*, vol. 68, no. 6, pp. 1866–1895, 2020.

[14] C. Comin and R. Rizzi, "An improved upper bound on maximal clique listing via rectangular fast matrix multiplication," *Algorithmica*, vol. 80, pp. 3525–3562, 2018.

[15] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in *Algorithm Theory-SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory, Humlebæk, Denmark, July 8-10, 2004. Proceedings 9*. Springer, 2004, pp. 260–272.

[16] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 505–517, 1977.

[17] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[18] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical computer science*, vol. 363, no. 1, pp. 28–42, 2006.

[19] I. Koch, "Enumerating all connected maximal common subgraphs in two graphs," *Theoretical Computer Science*, vol. 250, no. 1-2, pp. 1–30, 2001.

[20] K. A. Naudé, "Refined pivot selection for maximal clique enumeration in graphs," *Theoretical Computer Science*, vol. 613, pp. 28–37, 2016.

[21] P. San Segundo, J. Artieda, and D. Strash, "Efficiently enumerating all maximal cliques with bit-parallelism," *Computers & Operations Research*, vol. 92, pp. 37–46, 2018.

[22] Y. Jin, B. Xiong, K. He, Y. Zhou, and Y. Zhou, "On fast enumeration of maximal cliques in large graphs," *Expert Systems with Applications*, vol. 187, p. 115915, 2022.

[23] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *Journal of Experimental Algorithmics (JEA)*, vol. 18, pp. 3–1, 2013.

[24] K. Wang, K. Yu, and C. Long, " Maximal Clique Enumeration with Hybrid Branching and Early Termination ," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 599–612.

[25] Y. Li, Z. Shao, D. Yu, X. Liao, and H. Jin, "Fast maximal clique enumeration for real-world graphs," in *International Conference on Database Systems for Advanced Applications*. Springer, 2019, pp. 641–658.

[26] W. Deng, W. Zheng, and H. Cheng, "Accelerating maximal clique enumeration via graph reduction," *Proc. VLDB Endow.*, vol. 17, no. 10, pp. 2419–2431, 2024.

[27] X. Li, R. Zhou, L. Chen, C. Liu, Q. He, and Y. Yang, "One set to cover all maximal cliques approximately," in *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Z. G. Ives, A. Bonifati, and A. E. Abbadi, Eds. ACM, 2022, pp. 2006–2019.

[28] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[29] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers, "Biogrid: a general repository for interaction datasets," *Nucleic acids research*, vol. 34, no. suppl_1, pp. D535–D539, 2006.

[30] D. S. Johnson and M. A. Trick, *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. American Mathematical Soc., 1996, vol. 26.

[31] P. ERDdS and A. R&wi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

[32] Q. Chen, C. Fang, Z. Wang, B. Suo, Z. Li, and Z. G. Ives, "Parallelizing maximal clique enumeration over graph data," in *Database Systems for Advanced Applications: 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part II 21*. Springer, 2016, pp. 249–264.

[33] A. Abidi, R. Zhou, L. Chen, and C. Liu, "Pivot-based maximal biclique enumeration," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 3558–3564.

[34] B. Wu, S. Yang, H. Zhao, and B. Wang, "A distributed algorithm to enumerate all maximal cliques in mapreduce," in *2009 Fourth International Conference on Frontier of Computer Science and Technology*. IEEE, 2009, pp. 45–51.

[35] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Mining maximal cliques from an uncertain graph," in *2015 IEEE 31st international conference on data engineering*. IEEE, 2015, pp. 243–254.

[36] M. C. Schmidt, N. F. Samatova, K. Thomas, and B.-H. Park, "A scalable, parallel algorithm for maximal clique enumeration," *Journal of parallel and distributed computing*, vol. 69, no. 4, pp. 417–428, 2009.

[37] Y. Xu, J. Cheng, and A. W.-C. Fu, "Distributed maximal clique computation and management," *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp. 110–122, 2015.

[38] B. Lessley, T. Perciano, M. Mathai, H. Childs, and E. W. Bethel, "Maximal clique enumeration with data-parallel primitives," in *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 2017, pp. 16–25.

[39] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1240–1248.

[40] A. Das, S.-V. Sanei-Mehri, and S. Tirthapura, "Shared-memory parallel maximal clique enumeration from static and dynamic graphs," *ACM Transactions on Parallel Computing (TOPC)*, vol. 7, no. 1, pp. 1–28, 2020.

[41] B. Hou, Z. Wang, Q. Chen, B. Suo, C. Fang, Z. Li, and Z. G. Ives, "Efficient maximal clique enumeration over graph data," *Data Science and Engineering*, vol. 1, pp. 219–230, 2016.

[42] S. Han, L. Zou, and J. X. Yu, "Speeding up set intersections in graph algorithms using simd instructions," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1587–1602.

[43] M. Svendsen, A. P. Mukherjee, and S. Tirthapura, "Mining maximal cliques from a large graph using mapreduce: Tackling highly uneven subproblem sizes," *Journal of Parallel and distributed computing*, vol. 79, pp. 104–114, 2015.

[44] Q. Dai, R.-H. Li, M. Liao, H. Chen, and G. Wang, "Fast maximal clique enumeration on uncertain graphs: A pivot-based approach," in *Proceedings of the 2022 international conference on management of data*, 2022, pp. 2034–2047.

[45] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge, "Enumerating maximal cliques in temporal graphs," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 337–344.

[46] A. Brighen, H. Slimani, A. Rezgui, and H. Kheddouci, "Listing all maximal cliques in large graphs on vertex-centric model," *The Journal of Supercomputing*, vol. 75, pp. 4918–4946, 2019.

[47] N. Du, B. Wu, L. Xu, B. Wang, and X. Pei, "A parallel algorithm for enumerating all maximal cliques in complex network," in *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*. IEEE, 2006, pp. 320–324.

[48] Q. Zhang, R.-H. Li, M. Pan, Y. Dai, Q. Tian, and G. Wang, "Fairness-aware maximal clique in large graphs: concepts and algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 11, pp. 11 368–11 387, 2023.

[49] K. Yu and C. Long, "Fast maximal quasi-clique enumeration: A pruning and branching co-design approach," *Proceedings of the ACM on Management of Data*, vol. 1, no. 3, pp. 1–26, 2023.

[50] A. Das, M. Svendsen, and S. Tirthapura, "Incremental maintenance of maximal cliques in a dynamic graph," *The VLDB Journal*, vol. 28, pp. 351–375, 2019.

[51] L. Lu, Y. Gu, and R. Grossman, "dmaximalcliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution," in *2010 IEEE International Conference on Data Mining Workshops*. IEEE, 2010, pp. 1320–1327.

[52] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient maximal biclique enumeration for large sparse bipartite graphs," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, pp. 1559–1571, 2022.

[53] H. Inoue, M. Ohara, and K. Taura, "Faster set intersection with simd instructions by reducing branch mispredictions," *Proceedings of the VLDB Endowment*, vol. 8, no. 3, pp. 293–304, 2014.

[54] J. Blanuša, R. Stoica, P. Ienne, and K. Atasu, "Manycore clique enumeration with fast set intersections," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2676–2690, 2020.