Accelerating Maximal Clique Enumeration via Graph Reduction

Wen Deng
Fudan University
Shanghai, China
wdeng21@m.fudan.edu.cn

Weiguo Zheng
Fudan University
Shanghai, China
zhengweiguo@fudan.edu.cn

Hong Cheng
The Chinese University of Hong Kong
Hong Kong, China
hcheng@se.cuhk.edu.hk

ABSTRACT

As a fundamental task in graph data management, maximal clique enumeration (MCE) has attracted extensive attention from both academic and industrial communities due to its wide range of applications. However, MCE is very challenging as the number of maximal cliques may grow exponentially with the number of vertices. The state-of-the-art methods adopt a recursive paradigm to enumerate maximal cliques exhaustively, suffering from a large amount of redundant computation. In this paper, we propose a novel reduction-based framework for MCE, namely RMCE, that aims to reduce the search space and minimize unnecessary computations. The proposed framework RMCE incorporates three kinds of powerful reduction techniques including global reduction, dynamic reduction, and maximality check reduction. Global and dynamic reduction techniques effectively reduce the size of the input graph and dynamically construct subgraphs during the recursive subtasks, respectively. The maximality check reduction minimizes the computation for ensuring maximality by utilizing neighborhood dominance between visited vertices. Extensive experiments on 18 real graphs demonstrate that the existing approaches achieve remarkable speedups powered by the proposed techniques.

PVLDB Reference Format:
Wen Deng, Weiguo Zheng, and Hong Cheng. Accelerating Maximal Clique Enumeration via Graph Reduction. PVLDB, 17(10): 2419 - 2431, 2024.
doi:10.14778/3675034.3675036

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/DengWen0425/RMCE.

1 INTRODUCTION

As one of the most important cohesive structures in graph data, clique is closely related to other fundamental problems like independent set problem [41] and graph coloring problem [14]. In an undirected graph G, a clique refers to a subgraph of G where every pair of vertices are adjacent. A clique is maximal when no other vertices can be included to form a larger clique. Maximal clique enumeration (MCE) is the task of listing all the maximal cliques in a graph G and can be applied in a variety of fields, such as computational biology [1, 30, 40, 48], social network [28, 47], and wireless communication networks [2].

Proceedings of the VLDB Endowment, Vol. 17, No. 10 ISSN 2150-8097.
doi:10.14778/3675034.3675036

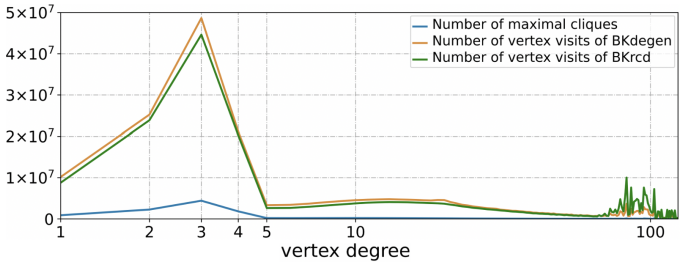


Figure 1: Illustration of the gap between maximal cliques and vertex visits (the horizontal axis is log-scaled).

1.1 Existing Methods and Limitations

BKdegen [15] and BKrcd [27] are two state-of-the-art algorithms for MCE, both of which adopt the Bron-Kerbosch (BK) framework [3] in Algorithm 1 that recursively enumerates all maximal cliques. The framework involves three sets, i.e., R, P, and X, where R stores the partial clique, P records the candidate set, and X contains vertices that have already been visited to ensure the maximality (also called a forbidden set). The recursive function BK initializes R, P, and X as $\emptyset, V$, and $\emptyset$, respectively. To expand a new branch, a vertex v is moved from P to R, Then P and X are updated as $P \cap N(v)$ and $X \cap N(v)$ (lines 3-4). After completing this branch, vertex v is moved from P to X (line 5). Once both P and X are empty, R is reported as a maximal clique (lines 1-2). BKdegen [15] combines the degeneracy order and pivot selection, effectively bounding the subproblem scale of each vertex by the degeneracy λ of graph G. BKrcd [27] leverages the dense nature of subproblems to enumerate maximal cliques in a top-down manner. A question naturally arises "can we make the task of maximal clique enumeration even faster?"

In practice, a substantial amount of overhead is incurred due to the need for repeated visits to specific vertices within the graph during the process of maximal clique enumeration. Figure 1 presents the distribution of the number of maximal cliques in which each vertex appears and the number of visits to each vertex by different algorithms. The results are averaged over 7 real graphs from

Algorithm 1: $BK(R, P, X)$

Input: Partial clique R, Candidate set P, Forbidden set X
Output: All maximal cliques in $G[P]$ restricted by X
1 **if** $P = \emptyset$ *and* $X = \emptyset$ **then**
2 | report R as a maximal clique
3 **for** $v \in P$ **do**
4 | $BK(R \cup \{v\}, P \cap N(v), X \cap N(v))$
5 | $P \leftarrow P \setminus \{v\}, X \leftarrow X \cup \{v\}$

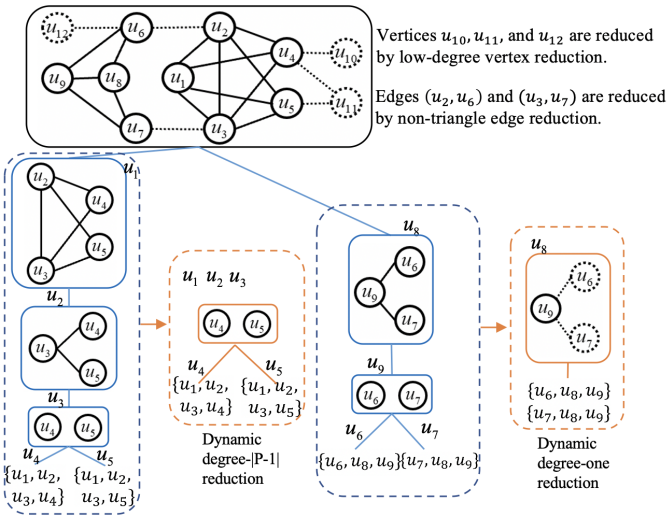


Figure 2: The search process of a toy graph, where the branches in blue dashed boxes denote the original branches following the BK algorithm, while those in orange dashed boxes denote the branches by applying dynamic reduction.

SNAP [24]. We observe a notable gap between the number of maximal cliques and the number of vertex visits, especially for low-degree vertices. For instance, BKdegen and BKrcd visit vertices with a degree of 3 or less around 84.06 million and 77.27 million times on average. However, the average number of maximal cliques that involve vertices with a degree of 3 or less is notably lower, at approximately 7.55 million. This large gap suggests that there is significant scope for further improvement in time efficiency.

Motivated by this observation, in order to enhance the performance, the key idea is to develop effective techniques to *bridge this gap by reducing the graph size and minimizing unnecessary vertex visits during the recursive computation process, while preserving the completeness of maximal cliques.*

1.2 Our Approach and Contributions

Example 1. *Consider the graph at the top of Figure 2, vertex u_{10} forms a single maximal 2-clique with its sole neighbor u_4. However, following Algorithm 1, there are at most 5 (i.e., the number of neighbors of u_4) subproblems involved in the recursive computation, where the candidate set P and the forbidden set X may be intersected with u_4's neighborhood. This will lead to unnecessary visits of u_{10}. By removing u_{10} and its corresponding edge, we can prevent their duplicate visits in the recursion, significantly reducing the computation cost. Meanwhile, we can ensure the completeness of solutions by reporting the maximal cliques that include the removed vertex in advance.*

Inspired by the example above, we propose a novel reduction-based framework for maximal clique enumeration, namely RMCE, that integrates three powerful reduction techniques including global reduction, dynamic reduction, and maximality check reduction.
(1) Global Reduction. Given a graph G, RMCE removes the low-degree vertices (whose degree is less than 3) and identifies the maximal cliques involving these deleted vertices beforehand. Moreover, we remove edges that do not form triangles with other edges throughout the graph, as they directly constitute maximal 2-cliques. For example, in Figure 2, the edges (u_2, u_6) and (u_3, u_7) can be removed as they are not contained in any other cliques, forming maximal cliques themselves. Removing these vertices and edges can reduce redundant computations during the recursive procedure.
(2) Dynamic Reduction. Enumerating maximal cliques operates in a recursive manner, creating an extensive number of subtasks. In each subtask, a subgraph will be dynamically constructed where new low-degree vertices may appear. RMCE recursively reduces the subgraph size by removing these vertices. Moreover, the subgraph is usually very dense and may contain vertices that are adjacent to all other vertices in the candidate set P. Since every maximal clique in this subtask must contain these vertices, we can move them from P into the partial clique R directly, and thus reduce the number of recursive calls (we call it dynamic degree-$|P-1|$ reduction). In the BK algorithm with pivot selection, each recursive call takes $O((|P|+|X|)^2)$ to choose a pivot. If k recursive calls are eliminated in a subproblem, the time cost will be reduced to $O((|P|+|X|)^2)$ from $O((k+1)(|P|+|X|)^2)$. For example, in the subproblem created by expanding u_8 in Figure 2, u_6 and u_7 become new low-degree vertices that can be reduced. In the left branch, u_1, u_2, and u_3 are adjacent to all other vertices in the subgraph, we can move them to R together, decreasing the number of recursive calls from 3 to 1.
(3) Maximality Check Reduction. The concept "maximal clique" involves two criteria: being a clique and achieving maximality. However, the existing methods have primarily focused on reducing the candidate vertices, with little attention given to optimizing the forbidden set that is used for maximality checks. The forbidden set size $|X|$ can be quite large, but not every vertex in X is required, especially when the neighbors of a vertex (e.g., $u_i \in X$) are contained by the neighbors of another vertex (e.g., $u_j \in X$). In such cases, this vertex (e.g., u_i) can be removed safely from X without reporting cliques that are not maximal. As studied in [18], set intersections take 73.6% of the running time in MCE. Since the forbidden set is frequently intersected during the recursion reducing the size of X can reduce the cost of set intersection operations. We develop an efficient algorithm with linear space overhead that minimizes the forbidden set, significantly reducing unnecessary computations.

In summary, we make the following contributions in this paper.

- To the best of our knowledge, we are the first to propose a reduction-based framework, namely RMCE, for enumerating maximal cliques.
- We develop powerful global reduction techniques and dynamic reduction techniques that effectively reduce the size of the input graph globally and the size of subgraphs in recursive subtasks, respectively.
- We introduce a novel concept of maximality check reduction for maximal clique enumeration and propose an efficient algorithm to minimize the forbidden set X.
- The proposed techniques above are orthogonal to the existing BK-based methods for maximal clique enumeration.
- Extensive experiments on 18 real networks demonstrate that our proposed algorithms achieve significant speedups compared to state-of-the-art approaches.

2 PROBLEM DEFINITION AND PRELIMINARY

2.1 Problem Formulation

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $m = |E|$ edges. The neighbor vertices of a vertex v in G are denoted as $N(v)$, and the degree of v is denoted as $d(v) = |N(v)|$. The common neighbors of vertices in S are denoted as $C(S) = \cap_{v \in S} N(v)$.

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

Let $N_S(v)$ denote the set of vertices in S that are adjacent to vertex v in graph G, and let $d_S(v)$ denote the number of such vertices, that is, $N_S(v) = N(v) \cap S$ and $d_S(v) = |N_S(v)|$.

Definition 2. ***(Degeneracy Order)*** *Given a graph $G = (V, E)$, the order of vertices $\overrightarrow{V} = \{v_1, v_2, v_3, \ldots, v_n\}$ is called the degeneracy order of G if vertex v_i has the minimum degree in every induced subgraph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$. The degeneracy of G, denoted by λ, is equal to the largest value of $d_{G[\{v_i, v_{i+1}, \ldots, v_n\}]}(v_i)$.*

The degeneracy order can be efficiently computed in linear time by iteratively removing the vertex with the smallest degree from the graph. In this order, each vertex v_i has at most λ neighbors in the induced graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ among its later neighbors. We use $N^-(v)$ or $N^+(v)$ to denote the neighbors ordered before or later than the vertex v, respectively.

Definition 3. ***(Clique)***. *An induced subgraph $G[S]$ of $G = (V, E)$ is called a clique if there is an edge $(u, v) \in E$ for any two vertices in S. We denote a clique with k vertices as a k-clique.*

Definition 4. ***(Maximal Clique)*** *A clique $G[S]$ is a maximal clique in graph G if and only if $\forall v \in V \setminus S$, the subgraph induced by $S \cup \{v\}$ is not a clique.*

Example 2. *Let us consider the graph shown in Figure 2. The vertices u_1, u_2, and u_3 induce a 3-clique, but it is not maximal since we can add vertex u_4 or u_5 to form a larger 4-clique. The clique $\{u_1, u_2, u_3, u_4\}$ is maximal because there are no other vertices in the graph that can be included to form a larger clique with these vertices.*

Problem Statement. ***(Maximal Clique Enumeration)*** *Given a graph G, the set of maximal cliques in G is denoted by $mc(G)$. Maximal clique enumeration (shorted as MCE) aims to report $mc(G)$.*

2.2 Existing Solutions

BKdegen [15]: Eppstein et al. introduce the degeneracy ordering before calling the recursive function BKpivot in [39]. BKpivot utilizes a pivot selection strategy that selects a vertex u from $X \cup P$ that has the most neighbors in P. This choice ensures that only u and its non-neighbors will be involved in the subsequent search branches. The degeneracy order is calculated at the beginning. Then, for each vertex v, X is initialized as $N^-(v)$, and P is initialized as $N^+(v)$. This ensures that every subproblem starting from each v has a candidate set P whose size is no larger than the degeneracy λ. Consequently, the worst-case time complexity is reduced to $O(n3^{\frac{\lambda}{3}})$.

BKrcd [27]: The subgraph induced by $N^+(v)$ may be very dense due to the degeneracy ordering. For a dense subgraph, it only needs to delete a few vertices to obtain a maximal clique. BKrcd removes a vertex with the fewest neighbors in P until the remaining vertices form a maximal clique. It then repeats this process on the removed vertex and its neighborhood. When P is a clique and passes the maximality check, $P \cup R$ is reported as a maximal clique.

BKfacen [22]: This approach employs a hybrid data structure, whereby the adjacency list of a graph is maintained globally, while an adjacency matrix is utilized in subtasks. The adjacency matrix accelerates queries for adjacency relationships between vertices. However, frequent adjacency matrix creation incurs significant overhead. It deviates from the pivot mechanism in [39], consequently lacking a guaranteed theoretical complexity. Thus, this approach introduces additional time overhead in numerous graphs.

BKrevised [33]: This approach identifies the pivot by recording the degree of each vertex in $P \cup X$. It returns a pivot as soon as a vertex v satisfying that $|P \setminus N(v) \cap P| \leq 2$, which reduces the time required for pivot selection in certain boundary conditions. However, it may not always be the optimal choice for more intricate subproblems.

Algorithm 2: RMCE(G)

```
   Input: Graph G
   Output: All maximal cliques in G
1  G ← apply global reduction on G
2  compute an order of the reduced graph G
3  for i = 1 : n do
4  |  X ← apply maximality check reduction on X
5  |  recursive (R, P, X)
   Procedure recursive(R, P, X)
6  |  R, P, X ← apply dynamic reduction on (R, P, X)
7  |  choose a pivot u
8  |  for w ∈ (P \ N(u) ∩ P) do
9  |  |  recursive (R ∪ {w}, P ∩ N(w), X ∩ N(w))
10 |  |  P ← P \ {w}, X ← X ∪ {w}
```

3 REDUCTION-BASED FRAMEWORK RMCE

Reduction-based Maximal Clique Enumeration (shorted as RMCE). The basic idea is to reduce the graph size and recursive search space by removing vertices and edges prior to performing the exhaustive search.

Algorithm 2 depicts the details of the proposed RMCE framework. Initially, we apply the global reduction on the graph G (line 1) before computing its vertex order (line 2). Subsequently, for each vertex in the ascending order of the reduced graph G, we employ the maximality check reduction before entering the *recursive* function (lines 3-5). The *recursive* function can be any BK-based algorithm, such as BKpivot and BKrcd. The dynamic reduction is conducted at the beginning of each *recursive* function (line 6).

Superiority of RMCE. Our reduction algorithm offers two significant advantages. Firstly, it effectively reduces the search branches during the maximal clique enumeration process, leading to a more efficient exploration of the solution space. Secondly, enormous set intersections are involved in the recursive process. RMCE can reduce the number of neighbors for vertices, enabling faster set intersections to enhance time efficiency.

RMCE consists of three powerful reduction techniques:

(1) **Global Reduction**: Global reduction means deleting some vertices and edges and reporting their maximal cliques in advance

without breaking the completeness of the solution. Formally, if we use $mc(G)$ denote all maximal cliques of a graph G, then we delete some vertices ΔV and edges ΔE such that

$$mc(G) = mc(G') + \alpha(\Delta V, \Delta E),$$

where $G' = (V \setminus \Delta V, E \setminus \Delta E)$, where $\alpha(\Delta V, \Delta E)$ denotes the maximal cliques that contain vertices in ΔV or edges in ΔE. This reduction technique significantly reduces the scale of the input graph, enhancing the efficiency of subsequent computations.

(2) **Dynamic Reduction**: The RMCE framework finds the maximal cliques by using a recursive search in which subproblems will be built dynamically.

Definition 5. ***(Subproblem)***. *In algorithms following the BK framework, a subproblem is represented by a partial clique R, a candidate set P, and a forbidden set X, denoted as (R, P, X). The maximal cliques in this subproblem are denoted by $\tilde{mc}(R, P, X)$.*

During the recursive search, the subproblem undergoes continuous changes, providing opportunities for vertices that cannot be pruned globally to be pruned within the subproblem dynamically. This brings the need for dynamic reduction.
Formally, we delete some vertices ΔP_1 from the candidate set and move some vertices ΔP_2 into R such that

$$\tilde{mc}(R, P, X) = \tilde{mc}(R', P', X') + \tilde{\alpha}(R, \Delta P_1, X),$$

where $R' = R \cup \Delta P_2, P' = P \setminus (\Delta P_1 \cup \Delta P_2), X' = X \cap C(\Delta P_2)$, and $\tilde{\alpha}(R, \Delta P_1, X)$ denotes the maximal cliques that contain R and vertices in ΔP_1 in the subproblem (R, P, X). This technique further reduces the size of the subproblem, efficiently minimizing the search space required for the subsequent search.

(3) **Maximality Check Reduction**: Maximality check reduction refers to reducing the size of forbidden set without producing any non-maximal or losing any maximal cliques, that is

$$\tilde{mc}(R, P, X) = \tilde{mc}(R, P, X \setminus \Delta X),$$

where ΔX denotes the deleted vertices from the forbidden set. This reduction technique prunes unnecessary computations by identifying and eliminating vertices that can be safely ignored during the maximality check process.

Our framework efficiently reduces both the graph size and the recursive search space. Meanwhile, the reduction itself is expected to take as little time as possible. To achieve this, we carefully develop reduction rules to avoid introducing excessive extra computation.

4 GLOBAL REDUCTION

4.1 Intuition

Degeneracy order, proposed by Eppstein et al. [15], is widely employed as the vertex ordering technique in the MCE task. However, we argue that degeneracy order suffers from two major problems:

(1) **Redundant computations for low-degree vertices**: Computing the degeneracy order needs to iteratively remove the vertex with the minimum degree. In fact, identifying maximal cliques containing low-degree vertices is straightforward due to their inherent simplicity. However, these vertices and their associated edges will be redundantly traversed during recursions of the MCE algorithm. This redundancy is expected to be mitigated through an effective reduction technique.

(2) **Redundant computations for edges**: In addition to low-degree vertices, certain edges also undergo repetitive traversals, especially for edges that do not form triangles with other edges. The redundant computations associated with these edges can be minimized through appropriate techniques.

To tackle these challenges, we propose two kinds of reductions to prevent revisiting deleted edges and vertices in subsequent processes, allowing the algorithm to operate on the reduced graph.

4.2 Low-Degree Vertex Reduction

In this subsection, we introduce three efficient reduction rules to eliminate vertices with a degree no larger than 2.

Lemma 1. ***(Degree-Zero Reduction)*** *A vertex u with zero degree can be removed such that $mc(G) = mc(G')$ where $G' = (V \setminus \{u\}, E)$.*

Proof. The proof is straightforward since a clique contains at least two vertices. □

Lemma 2. ***(Degree-One Reduction)*** *Let u be a degree-one vertex with the only neighbor v. The vertex u and its edge can be removed such that $|mc(G)| = |mc(G')| + 1$, where $G' = (V \setminus \{u\}, E \setminus \{(u, v)\})$.*

Proof. The vertex set $\{u, v\}$ forms a 2-clique by definition, and $N(v) \cap N(u) = \emptyset$ ensures its maximality. □

Lemma 3. ***(Degree-Two Reduction)*** *For a degree-two vertex u with neighbors v and w, we have the following scenarios:*

(1) If $(v, w) \notin E$, the edges (u, v) and (u, w) are two maximal 2-cliques. u and its edges can be deleted such that $|mc(G)| = |mc(G')| + 2$, where $G' = (V \setminus \{u\}, E \setminus \{(u, v), (u, w)\})$.
(2) If $(v, w) \in E$ and $N(v) \cap N(w) = \{u\}$, the three edges (u, v), (u, w), and (v, w) form a maximal 3-clique. Vertex u and the three edges can be safely deleted such that $|mc(G)| = |mc(G')| + 1$, where $G' = (V \setminus \{u\}, E \setminus \{(u, v), (u, w), (v, w)\})$.
(3) If $(v, w) \in E$ and $N(v) \cap N(w) \supseteq \{u\}$, the three edges (u, v), (u, w), and (v, w) form a maximal 3-clique. Vertex u and two edges (u, v) and (u, w) can be safely deleted such that $|mc(G)| = |mc(G')| + 1$, where $G' = (V \setminus \{u\}, E \setminus \{(u, v), (u, w)\})$.

Proof. For the first condition, $N(u) \cap N(v) = \emptyset$ means $\{u, v\}$ forms a maximal 2-clique. Since $\nexists u' \in V, N(u') \supseteq \{u, v\}$, (u, v) can be deleted. And (u, w) is similar to (u, v). When $(v, w) \in E$, (u, v), (u, w), and (v, w) form a maximal 3-clique since $N(u) \cap N(v) \cap N(w) = \emptyset$. In the second scenario, $N(v) \cap N(w) = \{u\}$, ensuring that $\{v, w\}$ cannot be enlarged by any vertex except u, which requires the deletion of (u, v) to avoid reporting it as a maximal 2-clique again. Otherwise, it implies that $\exists S \subseteq V, S \cup \{v, w\}$ is also a maximal clique, indicating that (v, w) cannot be deleted. □

Example 3. *Consider the graph presented in Figure 3. Vertices v_1, v_2, v_3, and v_6 can be deleted by our degree-two Reduction. The vertices v_7 and v_8 can be deleted by our degree-one Reduction.*

Algorithm 3 provides an overview of the *VertexReduction* procedure. First, Q is initialized with all vertices whose degree is at most 2 (line 1). Then, for each vertex v in Q, we apply either degree-two reduction, degree-one reduction, or degree-zero reduction based

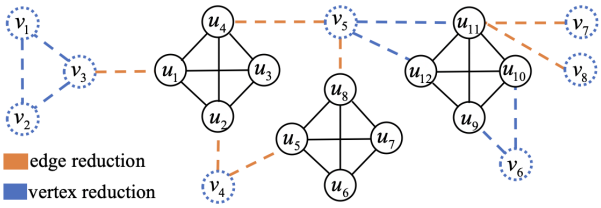


Figure 3: Illustration of global reduction. Vertices and edges in blue dashed lines can be deleted by vertex reduction while the orange dashed edges can be removed by edge reduction.

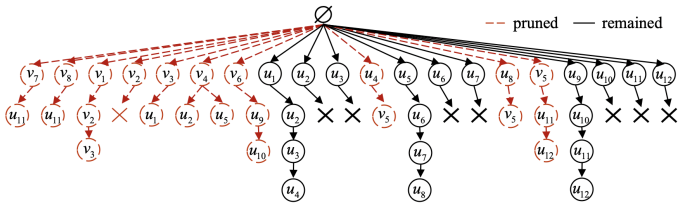


Figure 4: Search tree of BKdegen. The root is initialized to ∅.

on its degree (lines 4-9). We also update Q whenever a new vertex with a degree of no larger than 2 is encountered (line 10).

Complexity Analysis. Algorithm 3 examines a total of n vertices. Checking the existence of a common neighbor in degree-two reduction has a worst-case time complexity of $O(2d_{max})$ by merge-based algorithm [50], where d_{max} is the largest degree in graph G. Hence, the overall time complexity is $O(nd_{max})$. The space cost of $O(n)$ is required to maintain the vertices that need to be removed.

4.3 Non-triangle Edge Reduction

Apart from the vertex-based reduction technique, we also propose non-triangle edge reduction from the perspective of edges.

Definition 6. ***(Non-triangle Edge)***. *Edge (u, v) is defined as a non-triangle edge if $N(v) \cap N(u) = \emptyset$.*

A non-triangle edge (u, v) means no other vertex in the graph is adjacent to both u and v. The objective is to remove all non-triangle edges from the graph, guaranteed by the following lemma.

Lemma 4. ***(Non-triangle Edge Reduction)*** *A non-triangle edge (u, v) directly forms a maximal 2-clique and it can be deleted from G such that $|mc(G)| = |mc(G')| + 1$, where $G' = (V, E \setminus \{(u, v)\})$.*

Proof. It is straightforward that set $\{u, v\}$ cannot be expanded since adding any vertex into it will break the property of clique. □

Algorithm 4 examines each edge in the graph G to determine whether it can be removed. Initially, for an unvisited edge (u, v), if u and v have no common neighbors (line 4), (u, v) is classified as a non-triangle edge, and we can delete it while reporting $\{u, v\}$ as a maximal clique (lines 5-6). Conversely, if a vertex w is their common neighbor, we mark the three edges (u, v), (u, w), and (v, w) as visited, since they form a 3-clique (lines 7-8). Once an edge is visited, this edge will not be checked again in the future (line 3).

Algorithm 3: VertexReduction(G)

```
Input: Graph G
Output: Reduced graph G'
1  G' ← G, Q ← {v | d_G(v) ≤ 2}
2  for v ∈ Q do
3      Q ← Q \ {v}
4      if d_G'(v) = 2 then
5          G' ← apply the degree-two reduction rule on v
6      else if d_G'(v) = 1 then
7          G' ← apply the degree-one reduction rule on v
8      else
9          G' ← apply the degree-zero reduction rule on v
10     Q ← Q ∪ {u|u ∈ N_G'(v) ∧ d_G'(u) = 2}
11 return G'
```

Example 4. *Consider the graph in Figure 3. The edges in orange dashed lines (e.g., (u_1, v_3) and (v_4, u_5)) are non-triangle edges and can be safely deleted. Note that after deleting (u_4, v_5) and (v_5, u_8), v_5 becomes a degree-two vertex and can be removed by vertex reduction. Figure 4 shows a search tree for the graph in Figure 3. By global reduction, we avoid traversing the red paths, resulting in a notable reduction in computational cost. Furthermore, even along the remaining black search paths, the set intersection operations can be accelerated due to the decreased neighborhood size for multiple vertices.*

Complexity Analysis. Algorithm 4 examines a total of m edges. Finding a common neighbor takes a worst-case time complexity of $O(2d_{max})$. Consequently, the overall time complexity is $O(md_{max})$. The space overhead of recording the visited edges is $O(m)$.

5 DYNAMIC REDUCTION

5.1 Intuition

Beyond the global reduction, the recursive procedure of the MCE program will explore numerous subgraphs, presenting opportunities for further reducing the redundant computations.

Example 5. *Figure 5(a) depicts a subgraph involved in a subproblem, where all vertices are adjacent to the current partial clique R. The blue vertices form the forbidden set X, while the orange vertices represent the candidate set P. The corresponding recursion tree for this subproblem is shown in Figure 6(a). By scrutinizing the recursion tree, we make the following observations:*
(1) New low-degree vertices (e.g., u_7, u_8, and u_9) appear in this subproblem and can be handled similar to global scenarios, without creating additional recursion or performing set intersection operations.

Algorithm 4: EdgeReduction(G)

```
Input: Graph G = (V, E)
Output: Reduced graph G'
1  G' ← G
2  for (u, v) ∈ E do
3      if (u, v) is not visited then
4          if u and v has no common neighbors then
5              G' ← delete the edge (u, v) from G'
6              report {u, v} as a maximal clique
           else
7              w ← a common neighbor of u and v
8              mark edges (u, v), (u, w), and (v, w) visited
9  return G'
```

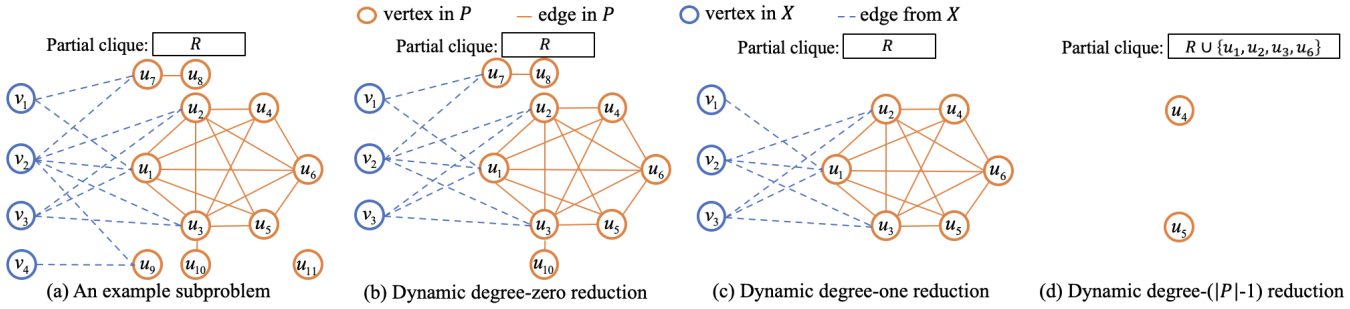


Figure 5: Illustration of dynamic reduction in the subgraph of a subproblem (R, P, X).

(2) As presented in Figure 6(a), there are no other branches from u_1 to u_6 in the search path $R \to u_3 \to u_1 \to u_2 \to u_6$. Because vertices u_1, u_2, and u_6 connect to all other vertices in $P \cap \{u_3\}$ in the subproblem $(R \cup \{u_3\}, P \cap \{u_3\}, X \cap \{u_3\})$, we can move u_1, u_2, and u_6 into partial clique $R \cup \{u_3\}$ together instead of creating three additional recursive calls. Clearly, this will further reduce the computation cost.

5.2 Dynamic Vertex Reduction

Dynamic reduction aims to reduce the subgraph size of subproblem (R, P, X). Different from global reduction, a maximal clique in subgraph $G[P]$ may not be maximal in G, because it may be expanded by a vertex in X. We develop three dynamic reduction techniques to effectively optimize the search process while ensuring that only truly maximal cliques are reported. For clarity, a vertex v is called a dynamic degree-k vertex in the subproblem (R, P, X) if $|N_P(v)| = k$.

Lemma 5. ***(Dynamic Degree-Zero Reduction)*** *For a dynamic degree-zero vertex $u \in P$ in the subproblem (R, P, X), we have*

(1) If $N(u) \cap X = \emptyset$, $R \cup \{u\}$ is a maximal clique. We can remove u from P, and $|\tilde{mc}(R, P, X)| = |\tilde{mc}(R, P', X)|+1$, where $P' = P \setminus \{u\}$.
(2) If $N(u) \cap X \neq \emptyset$, $R \cup \{u\}$ is not maximal. Thus, we can remove u from P and $\tilde{mc}(R, P, X) = \tilde{mc}(R, P', X)$, where $P' = P \setminus \{u\}$.

Proof. For the dynamic degree-zero vertex u, we have $u \in P$, which guarantees the clique property. When $N(u) \cap X = \emptyset$, it indicates that $R \cup \{u\}$ is maximal since $N_P(u) = \emptyset$, which proves the first condition. Otherwise, adding u into R will produce a non-empty X, making $R \cup \{u\}$ not maximal. Thus, u can be removed. □

Lemma 6. ***(Dynamic Degree-One Reduction)*** *Consider a dynamic degree-one vertex $u \in P$ with its only neighbor $v \in P$ in the subproblem (R, P, X). There are two scenarios:*

(1) If $\exists w \in X$ such that $u, v \in N(w)$, vertex u can be immediately removed such that $\tilde{mc}(R, P, X) = \tilde{mc}(R, P', X)$ where $P' = P \setminus \{u\}$. If v is also a dynamic degree-one vertex before u's removal, then v should also be removed, that is $\tilde{mc}(R, P, X) = \tilde{mc}(R, P \setminus \{u, v\}, X)$.
(2) If $\nexists w \in X$ such that $u, v \in N(w)$, $R \cup \{u, v\}$ is a maximal clique and vertex u is deleted such that $|\tilde{mc}(R, P, X)| = |\tilde{mc}(R, P', X)|+1$ where $P' = P \setminus \{u\}$. If v is also a dynamic degree-one vertex before u's removal, then v should be removed as well, that is $|\tilde{mc}(R, P, X)| = |\tilde{mc}(R, P \setminus \{u, v\}, X)| + 1$.

Proof. If $\exists w \in X$ such that $u, v \in N(w)$, moving $\{u, v\}$ into R would result in an empty P but non-empty X, as $w \in X$. This indicates a non-maximal clique. Otherwise, X will be empty, signifying the discovery of a maximal clique. v will become a dynamic degree-zero vertex if it is a dynamic-one vertex before. It needs to be removed because $N(u) \supseteq R \cup \{v\}$ indicates that v cannot form a maximal clique in the subproblem $(R, P \setminus \{u\}, X)$. □

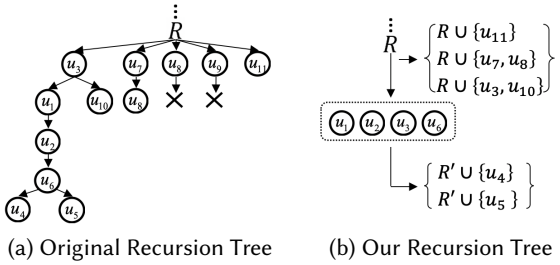


Figure 6: Comparison of two recursion trees.

In Lemma 6, for each dynamic degree-one vertex $u \in P$ in the subproblem, we need to determine whether u and v share a common vertex in the forbidden set X. Thus, the overall time cost is $O((|X| + d_{max})|P|)$. Since the reduction is expected to introduce as little extra computation cost as possible, we develop a relaxed version of dynamic degree-one reduction as follows.

Lemma 7. ***(Relaxed Dynamic Degree-One Reduction)*** *Consider a dynamic degree-one vertex $u \in P$ with its only neighbor $v \in P$ in the subproblem (R, P, X). If $N(u) \cap X = \emptyset$ or $N(v) \cap X = \emptyset$, $R \cup \{v, u\}$ forms a maximal clique and vertex u can be removed from P such that $|\tilde{mc}(R, P, X)| = |\tilde{mc}(R, P', X)| + 1$ where $P' = P \setminus \{u\}$. If v is also a dynamic degree-one vertex before u's removal, v will be removed as well, that is $|\tilde{mc}(R, P, X)| = |\tilde{mc}(R, P \setminus \{u, v\}, X)| + 1$.*

Proof. The condition implies that $\nexists w \in X$ such that $v, u \in N(w)$. We can deduce the conclusion using Lemma 6. □

Note that the cost of conducting this reduction rule is trivial since we can efficiently maintain $N(v) \cap X$ by traversing the neighbors of each vertex in the forbidden set X just once. Subsequently, applying Lemma 7 on dynamic degree-one vertices only requires traversing the candidate set P. The overall time complexity is $O(|X||P|)$.

Example 6. *Let us consider the subproblem presented in Figure 5(a). The vertices in the candidate set P are in orange color and the forbidden set X consists of vertices in blue color. Through the reduction rules,*

Algorithm 5: dynamicVertexReduction(R, P, X)

Input: Partial clique R, Candidate set P, Forbidden set X
Output: Reduced R', P', X'

1 **for** $v \in X$ **do**
2 **for** $u \in N_P(v)$ **do**
3 mark u
4 **for** $v \in P$ **do**
5 **if** $d_P(v) = 0$ **then**
6 $P' \leftarrow$ apply the dynamic degree-zero reduction rule on v
7 **else if** $d_P(v) = 1$ **then**
8 $u \leftarrow v$'s only neighbor
9 **if** *v is not marked or u is not marked* **then**
10 $P' \leftarrow$ apply the relaxed dynamic degree-one reduction rule on v
11 **for** $v \in P'$ **do**
12 **if** $d_{P'}(v) = |P'| - 1$ **then**
13 $P', R' \leftarrow$ apply the dynamic degree-$(|P|-1)$ reduction rule on v
14 $X' \leftarrow X \cap C(R')$; // Update the forbidden set X
15 **return** R', P', X'

we can safely remove the dynamic degree-zero vertices u_9 and u_{11}, resulting in the new subgraph of the subproblem shown in Figure 5(b). Additionally, removing the degree-one vertices u_7, u_8, and u_{10} leads to the subgraph of this subproblem in Figure 5(c).

Lemma 8. ***(Dynamic Degree-$(|P|-1)$ Reduction)*** *If a vertex $u \in P$ satisfies $|N_P(u)| = |P| - 1$ in the subproblem (R, P, X), u is called a dynamic degree-$(|P|-1)$ vertex. In this case, we can directly move u from P into R, ensuring that $\tilde{mc}(R, P, X) = \tilde{mc}(R', P', X')$, where $R' = R \cup \{u\}$, $P' = P \setminus \{u\}$, and $X' = X \cap N(u)$.*

Proof. Given a dynamic degree-$(|P|-1)$ vertex u, we have $|N_P(u)| = |P| - 1$, which means $\forall S \subseteq P$ and $u \notin S$, $u \in C(S)$. Assume there exists a subset $S \subseteq P$ that $u \notin S$ and $S \cup R$ is a clique. Then $S \cup R$ is not maximal due to $u \in P \cap C(S)$. That is any maximal clique in the subproblem (R, P, X) must contain the vertex u. □

Example 7. *Let us revisit the subproblem presented in Figure 5(a). Based on the updated subgraph in Figure 5(c), u_1, u_2, u_3, and u_6 are identified as dynamic degree-$(|P|-1)$ vertices. Thus, adding them to R will make the subgraph of this subproblem much smaller, with only two isolated vertices, as shown in Figure 5(d). Hence, the overall search tree of this subproblem derived from our reduction rules is shown in Figure 6(b). Eliminating low-degree vertices from the original subproblem enhances the efficiency of dynamic degree-$(|P|-1)$ reduction, further reducing the scale of the subproblem.*

Algorithm 5 outlines our dynamic reduction procedure in detail. It starts by marking each neighbor u of each vertex v in forbidden set X if $u \in P$, where a marked vertex u means $N(u) \cap X \neq \emptyset$ (lines 1-3). Subsequently, we iterate through each vertex $v \in P$. For dynamic degree-zero vertices, we remove them according to Lemma 5 (lines 5-6). For dynamic degree-one vertices, we selectively remove some of them following the relaxed dynamic degree-one reduction rule in Lemma 7 (lines 7-10). Afterward, we reiterate through reduced P' again to perform the dynamic degree-$(|P|-1)$ reduction (lines 11-13). Once all reduction operations are completed, we update the forbidden set X so that $\forall v \in X', R' \subseteq N(v)$ (line 14).

Algorithm 6: forbiddenSetReduction($v, P, X, ignoreId$)

Input: Vertex v inducing the subproblem with partial clique $\{v\}$, candidate set P, and forbidden set X; an array $ignoreId$
Output: Reduced X'

1 $X' \leftarrow \{u | u \in X \wedge ignoreId[u] \geq v$'s order$\}$
2 **for** $u \in P$ **do**
3 **if** $P \subseteq N^+(u)$ **then**
4 $ignoreId[v] \leftarrow \min(u$'s order, $ignoreId[v])$
5 **else if** $N^+(u) \subseteq P$ **then**
6 $ignoreId[u] \leftarrow \min(v$'s order, $ignoreId[u])$
7 **return** X'

Complexity Analysis. Marking the vertices in P takes the cost $O(\lambda|X|)$ (lines 2-4), where λ is the degeneracy of the graph G. Traversing the vertices in P and their neighbors costs $O(\lambda|P|)$. Updating the set X also requires $O(\lambda|X|)$ time. Thus, the overall time cost of Algorithm 5 is $O(\lambda(|X| + |P|))$ in the worst case. The space overhead is $O(|P|)$ as we need to mark the vertices in P.

6 MAXIMALITY CHECK REDUCTION

Considerable effort has been focused on reducing the candidate set to enhance the efficiency, while the cost of the maximality check has often been overlooked. Here, we shed light on the observation that the forbidden set involves substantial redundant computation and introduce a technique that efficiently reduces the forbidden set.

6.1 Forbidden Set Reduction

The insight above leads to a method of reducing the forbidden set by checking the neighborhood dominance between vertices in X.

Lemma 9. ***(Forbidden Set Reduction by Neighbor Containment)*** *For two vertices $u, v \in X$, if $N_P(u) \subseteq N_P(v)$, it holds that $\tilde{mc}(R, P, X) = \tilde{mc}(R, P, X')$, where $X' = X \setminus \{u\}$.*

Proof. Assume a non-maximal clique $R \cup \Delta R$ becomes maximal due to the removal of u where $\Delta R \subseteq P$, which means $\Delta R \subseteq N_P(u)$. While $X \cap C(\Delta R) = \emptyset$ implies that $\Delta R \supseteq N_P(v)$, which contradicts the condition $N_P(u) \subseteq N_P(v)$. Thus, the Lemma 9 holds. □

Example 8. *Consider the graph in Figure 5(a). $N_P(v_1) = \{u_1, u_7\}$, $N_P(v_3) = \{u_1, u_2, u_3\}$, and $N_P(v_4) = \{u_9\}$ are subsets of $N_P(v_2) = \{u_1, u_2, u_3, u_7, u_9\}$. If we remove v_1, v_3, and v_4 from X, the solution of the current subproblem will remain unaffected, that is, any non-maximal cliques will not be reported and any maximal cliques will not be lost. For example, $R \cup \{u_9\}$ is not a maximal clique due to the adjacent relationship with v_4 and v_2. After removing v_4 from X, the clique $R \cup \{u_9\}$ will not be maximal due to the existence of vertex v_2.*

Establishing the neighbor containment relationship between vertices in X by traversing them and their neighbors in each subtask incurs significant time overhead. Thus, we design an efficient approach (i.e., Algorithm 6) that continuously builds the neighbor containment relationship between vertices during the MCE process, which can prune X without incurring additional time overhead.

A subproblem induced by vertex v refers to the subproblem with $R = \{v\}$, $P = N^+(v)$, and $X = N^-(v)$. Algorithm 6 aims to establish neighbor containment relationships between v and vertices in P of the subproblem induced by v, and efficiently prunes X. It utilizes

an array of size n, called *ignoreId*, to record the order in which vertices can be excluded from constructing the set X. Initially, all elements are set to n, indicating that no vertex will be ignored. For a subgraph induced by the later neighbors of a vertex v, if a candidate vertex u satisfies $P \subseteq N^+(u)$, i.e., $N^+(v) \subseteq N^+(u)$, then v can be ignored in all subsequent subproblems after completing the j-th iteration, where j is the order of u. This is because in any subproblem after that, if $v \in X$, then u must be contained in X, which allows us to prune v following Lemma 9. Thus, the *ignoreId* of vertex v will be updated as the minimum of j and its current value (line 4). Conversely, if $N^+(u) \subseteq P$, it means u can be ignored instantly after this iteration since $N_P(u)$ will be dominated by $N_P(v)$ in all subsequent subproblems (line 6).

Complexity Analysis. Reducing the size of the forbidden set X runs in $O(|X|)$, which is linear in the size of X. The process of updating *ignoreId* for each vertex in P involves traversing their later neighbors. Thus, the time complexity of Algorithm 6 is $O(\lambda|P|)$. The space complexity is $O(n)$ due to the array *ignoreId* of size n.

6.2 Correctness of RMCE framework

Based on all the reductions, we give the correctness proof of the RMCE framework. For a more detailed proof, please refer to our technical report. Here, we provide an outline of the proof.

Lemma 10. *Algorithm 2 delivers all and only maximal cliques in the input graph G.*

Proof. (1) Firstly, we establish the correctness of the global reduction in the RMCE framework by proving $mc(G) = mc(G') + \alpha(\Delta V, \Delta E)$, where ΔV and ΔE represent the deleted vertices and edges in the global reduction and $\alpha(\Delta V, \Delta E)$ denotes the maximal cliques that contain vertices in ΔV or edges in ΔE whose correctness has been proved in Section 4. To establish this, we only need to show the following cases: case (i) all maximal cliques in G' are also maximal cliques in G and, case (ii) all maximal cliques not in $\alpha(\Delta V, \Delta E)$ must be included by $mc(G)$. The two cases can be proved by contradiction, where it can be shown that all counterexamples would conflict with global reduction conditions.

(2) Secondly, we assume that the recursive function in any BK framework returns all and only the maximal cliques that contain all vertices in R, certain vertices in P, and no vertices in X in each subproblem (R, P, X) (as demonstrated in existing works such as [3, 15, 27, 39]). As the dynamic reduction and maximality check reduction only modify the subproblem, we only need to establish $\tilde{mc}(R, P, X) = \tilde{mc}(R', P', X') + \tilde{\alpha}(R, \Delta P_1, X)$, where $\tilde{\alpha}$ represents the maximal cliques containing the deleted vertices ΔP_1 that are determined through dynamic reduction. For maximality check reduction, we have $\Delta P_1 = \emptyset$ and $\tilde{\alpha}(R, \Delta P_1, X) = 0$. $\tilde{mc}(R, P, X) = \tilde{mc}(R', P', X')$ is directly proven using Lemma 9.

For subproblems modified by dynamic reduction, $\tilde{\alpha}(R, \Delta P_1, X)$ includes all maximal cliques containing any vertices in ΔP_1 under the subproblem (R, P, X), which is proved in Section 5. We only need to demonstrate that (i) all maximal cliques in (R', P', X') are also maximal cliques in (R, P, X), and (ii) all maximal cliques in (R, P, X) that are not in $\tilde{\alpha}(R, \Delta P_1, X)$ must be included in $\tilde{mc}(R', P', X')$. Similarly, the two cases can be proved by showing that all counterexamples will conflict with dynamic reduction conditions. □

Table 1: Graph statistics, where λ represents the degeneracy.

Graph	Abbr.	#Vertices	#Edges	d_{max}	λ
as-skitter	as	1696415	11095298	35455	111
ca-CondMat	ca	23133	93439	279	25
cit-Patents	cp	3774768	16518947	793	64
com-dblp	cd	317080	1049866	343	113
com-orkut	co	3072441	117185083	33313	253
com-youtube	cy	1134890	2987624	28754	51
email-EuAll	ee	265009	364481	7636	37
flickr	fl	105938	2316948	5425	573
inf-road-usa	in	23947346	28854311	9	3
large_twitch	lt	168114	6797557	35279	149
loc-gowalla	lg	196591	950327	14730	51
roadNet-CA	rc	1965206	2766607	12	3
sc-delaunay_n23	sd	8388608	25165784	28	4
soc-pokec	sp	1632803	22301964	14854	47
soc-twitter-higgs	st	456631	12508440	51386	125
web-Google	wg	875713	4322051	6332	44
web-Stanford	ws	281903	1992636	38625	71
wiki-Talk	wt	2394385	4659565	100029	131

7 APPLYING TO TEMPORAL GRAPHS

We will further explore the possibilities of extending our reduction techniques to special graphs, with temporal graphs as an example.

Definition 7. ***(Temporal Graph)*** *A temporal graph G_T is a triple (V, E, T) where V denotes the vertex set, $E \subseteq V \times V \times T$ denotes a set of time-edges, and $T = [t_s, t_e]$ denotes the time interval of the graph. $t_e - t_s$ is called the lifetime of the graph G_T where $t_s, t_e \in \mathbb{N}$.*

Definition 8. ***(Δ-clique)*** *Let $\Delta \in \mathbb{N}$, a Δ-clique in a temporal graph $G_T = (V, E, T)$ is defined as a tuple $C = (S, I = [a, b])$ with $S \subseteq V$, $b - a \geq \Delta$, and $I \subseteq T$ such that for all $\tau \in [a, b - \Delta]$ and for all $u, v \in S$ there exists a $(\{v, w\}, t) \in E$ with $t \in [\tau, \tau + \Delta]$.*

A Δ-clique $C = (S, I)$ is called time-maximal if we cannot increase the time interval I without removing any vertex in S. It is called vertex-maximal if we cannot enlarge the vertex set S without decreasing the time interval I. A Δ-clique is maximal when it is both time-maximal and vertex-maximal.

Our reduction techniques can be applied to the enumeration of maximal Δ-cliques with careful modifications. We have developed Temporal Degree-One Reduction, Temporal Degree-Two Reduction and Temporal Non-triangle Edge Reduction for the global scenario, Temporal Dynamic Degree-Zero Reduction, Temporal Dynamic Degree-One Reduction and Temporal Dynamic Degree-$(|P| - 1)$ Reduction for the dynamic scenario, as well as Temporal Forbidden Set Reduction. Further details can be found in our technique report.

8 EXPERIMENTS

8.1 Experimental Settings

Dataset. As referenced in Table 1, we use 18 real networks from SNAP [24] and Network Repository [35] in the experiments.

Algorithms. We evaluate the performance of enhancing four existing methods BKdegen, BKrcd, BKfacen, and BKrevised.

- BKdegen [15]: The degeneracy-based algorithm, which is the fastest algorithm among existing methods in most cases.
- BKrcd [27]: The top-down algorithm for MCE.
- BKfacen [22]: MCE algorithm that uses hybrid data structure with adjacency list and partial adjacency matrix.

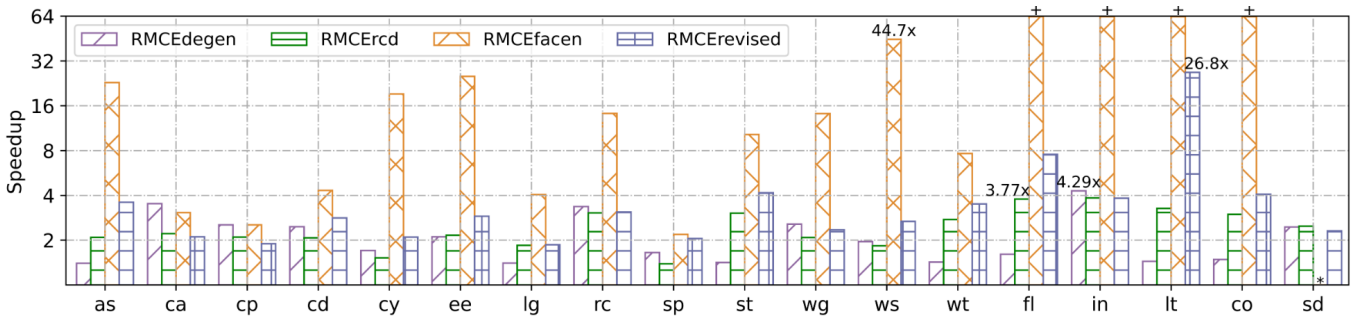


Figure 7: Overall performance: each bar represents the speedups of its corresponding algorithm enhanced by our reduction methods over the original algorithm (e.g., RMCEdegen represents $\frac{\text{running time of BKdegen}}{\text{running time of RMCEdegen}}$). The "+" symbol denotes that our method completes within 12 hours while the original one fails. The "*" symbol denotes that both our method and the original algorithm failed to complete within 12 hours.

- BKrevised [33]: MCE algorithm with a revised pivot strategy.
- RMCEdegen: Our method uses the recursion of BKdegen.
- RMCErcd: Our method uses the recursion of BKrcd.
- RMCEfacen: Our method uses the recursion of BKfacen.
- RMCErevised: Our method uses the recursion of BKrevised.

All experiments are conducted on a Linux Server equipped with Intel(R) Xeon(R) CPU E5-2696 v4 @ 2.20GHz and 128G RAM. All algorithms except the temporal solutions (implemented in Python) are implemented in C++ and compiled with *-O3* option.

8.2 Overall Results

Taking the running time cost of each original algorithm as the baseline, we compute the speedups of our methods over BKdegen, BKrcd, BKfacen, and BKrevised, respectively. Figure 7 presents the results for 18 graphs. we observe that our methods consistently outperform the original methods. Specifically, RMCEdegen, RMCErcd, RMCEfacen, and RMCErevised achieve peak speedups of 4.29×, 3.77×, 44.7×, and 26.8× in graphs in, fl, ws, and lt, respectively. The detailed running time is presented in Table 2. Due to the space limit, we omit the running time of BKfacen (it can be inferred based on the speedup depicted in Figure 7). It is clear that each method exhibits noteworthy improvements by incorporating our reduction techniques. The average speedups are 2.2, 2.5, 13.4, and 4.4 for BKdegen, BKrcd, BKfacen, and BKrevised, respectively.

From the result shown in Figure 7 and Table 2, we can see that BKdegen runs faster among the existing methods in 7 graphs (e.g., as and lg). While in other 9 graphs (e.g., cp and cd), BKrcd outperforms other existing methods. However, its improvement over BKdegen is marginal and it may lead to significant degradation on certain graphs (e.g., lt and co). BKfacen exhibits the slowest performance among all graphs, possibly due to the lack of a robust pivot selection mechanism, which often leads to performance degradation in larger graphs. Additionally, the frequent creation of adjacency matrices in memory can result in out-of-memory issues in certain graphs (e.g., sd). In the dataset sd, BKrevised runs faster than other existing methods, and it closely approaches BKdegen on numerous graphs. This is attributed to its continued utilization of the BKdegen framework, with modifications made to the pivot selection criteria in boundary cases.

Table 2: Detailed running time (in seconds). The "-" symbol denotes that the method failed to complete within 12 hours.

Graph	BKdegen	RMCEdegen	BKrcd	RMCErcd	BKrevised	RMCErevised	RMCEfacen
as	80.55	**57.49**	126.75	60.63	206.68	58.22	200.67
ca	0.20	0.05	0.09	**0.04**	0.12	0.06	0.15
cp	56.11	**22.14**	47.19	22.52	41.94	22.54	29.82
cd	1.64	**0.67**	1.39	**0.67**	1.88	0.69	1.10
cy	6.84	4.01	6.07	**3.99**	8.41	4.22	10.02
ee	0.99	0.47	0.81	**0.38**	1.36	0.48	0.80
lg	2.68	1.91	2.79	**1.51**	3.57	1.91	5.32
rc	3.19	0.95	2.89	0.95	2.93	0.94	**0.82**
sp	73.97	44.77	61.89	**44.54**	91.86	44.82	103.62
st	554.99	391.48	1184.87	**389.70**	1632.51	394.27	1541.28
wg	6.55	2.55	5.30	**2.54**	6.01	2.67	5.65
ws	2.95	1.51	2.33	**1.27**	4.04	1.52	4.98
wt	109.40	**76.68**	213.42	77.51	268.63	77.01	305.84
fl	287.84	178.86	664.91	**176.01**	1348.76	180.19	801.14
in	49.38	11.51	43.98	11.42	44.07	11.43	**10.97**
lt	469.48	**325.24**	1077.11	329.65	8792.72	328.09	1302.74
co	3554.64	**2393.59**	7176.81	2404.62	9742.07	2431.44	8957.91
sd	28.25	**11.52**	29.14	11.72	26.61	12.67	-

After integrating our reduction techniques, all methods have shown improved runtime speeds. RMCEdegen exhibits the fastest performance on 7 graphs (e.g., as and cp), while RMCErcd shows the fastest execution on 9 graphs (e.g., ca and cd). In datasets fl, lt, and co, the gap between RMCEdegen and RMCErcd has been significantly narrowed compared to previous results. Overall, RMCEdegen proves to be relatively superior and stable in larger datasets, while RMCErcd exhibits better performance in smaller datasets.

Discussion. Let us consider the state-of-the-art methods BKdegen, BKrcd, and BKrevised. Although our methods have achieved an average speedup of around 3 times compared to the original methods, it is worth mentioning that optimizing maximal clique enumeration itself, being a fundamental task in the field of graph mining, is yet highly challenging. For instance, BKrcd, built upon BKdegen, only exhibits marginal improvements on a few graphs (up to 2.17× speedup on the ca dataset), and even performs much worse than BKdegen in graphs like fl, lt, and co. On average, its speedup is merely 0.98 (i.e., it performs worse than BKdegen). Therefore, our method's average improvement of 3 times is relatively significant (and the best speedups achieved in BKdegen, BKrcd, and BKrevised are 4.29, 3.77, and 26.8 times, respectively). Furthermore, we have

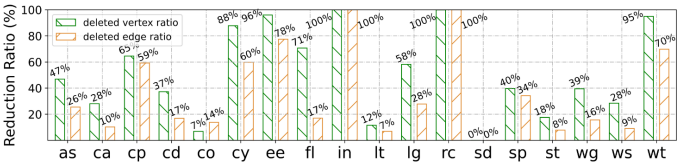


Figure 8: Reduction ratio of global reduction.

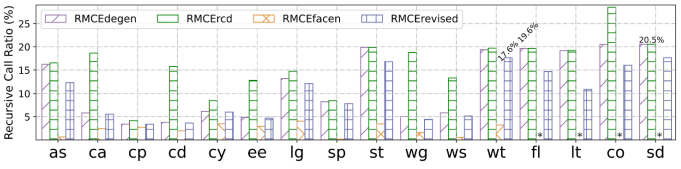


Figure 9: Ratio of recursive calls. The symbol "*" denotes that the original method fails to complete.

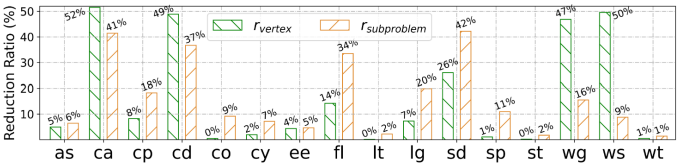


Figure 10: Reduction ratio of maximality check reduction.

surprisingly found that our reduction techniques can achieve significant speedup improvements on particular graphs (e.g., temporal graphs) as well. For more details, please refer to Section 8.6.

8.3 Detailed Evaluation

The Effect of Global Reduction. To evaluate the effect of global reduction, Figure 8 shows the ratio of deleted vertices and the ratio of deleted edges compared to the original graph. There are over 35% vertices in 12 graphs (e.g., cp and cy) and over 20% edges in 9 graphs (e.g., ee and wt) that can be removed by global reduction. In particular, all vertices and edges in graphs in and rc have been deleted due to their sparsity (thus, we exclude them from subsequent experiments). In contrast, no vertices and edges in the sd dataset have been deleted, verifying the effectiveness of dynamic reduction and maximality check reduction on the other hand.

Number of Recursive Calls. We investigate the number of recursive calls during the MCE algorithm to show the pruning power of our reduction method. We define the ratio of recursive calls as

$$\frac{\text{\#recursive calls of a method (like RMCEdegen)}}{\text{\#recursive calls of original algorithm (like BKdegen)}}$$

As depicted in Figure 9, RMCEdegen, RMCErcd, RMCEfacen, and RMCErevised reduce the number of recursive calls to no more than 17.6%, 28.5%, 4.5%, and 20.5% of the original, respectively. In instances like in and rc, no recursive calls are needed since all the vertices and edges have been removed by global reduction. The ratio of the number of recursive calls serves as an indicator of the effectiveness of a method, as each recursive call typically involves set intersection and pivot selection. The superior pruning ability of our algorithms is evident from these results.

Effect of Forbidden Set Reduction. To evaluate the impact of forbidden set reduction, we examine two key metrics: the ratio of pruned vertices when constructing the forbidden set X for each subproblem ($r_{vertex} = \frac{\sum|X'|}{\sum|X|}$) and the ratio of subproblems where forbidden set reduction occurs ($r_{subproblem} = \frac{\#\{subproblem|X' \subset X\}}{\#subproblem}$) in the outer iteration. The results are depicted in Figure 10. Remarkably, we observe that in many datasets like ca and cd, the ratio of pruned vertices (i.e., r_{vertex}) can reach close to 50%. Additionally, in datasets like fl and sd, the ratio of reduced subproblems ($r_{subproblem}$) achieves nearly 40%, indicating the significant pruning effect of our maximality check reduction technique. These results highlight the effectiveness of our method in reducing the computational overhead of the maximality check process.

Reduction of Vertex Visits. We also report the distribution of vertex visits with respect to their degrees. Due to limited space, we report the results of 4 graphs. As shown in Figure 11, RMCEdegen yields a substantial reduction in the number of vertex visits across different degrees. In wg (Figure 11(a)), RMCEdegen reduces 88% vertex visits compared to BKdegen (70% compared to BKrcd) at degree 20. In cp (Figure 11(b)), RMCEdegen reduces 82% vertex visits compared to BKdegen (82% compared to BKrcd) at degree 15. And in cd (Figure 11(d)), RMCEdegen reduces 73% vertex visits compared to BKdegen (61% compared to BKrcd) at degree 3. While for graphs whose vertices may be with a higher degree such as sp in Figure 11(c), our method can also reduces the vistis of high-degree vertices. For example, RMCEdegen reduces 46% vertex visits compared to BKdegen (54% compared to BKrcd) at degree 100 in sp. Similar results can be seen in cp as shown in Figure 11(b).

Time Overhead. The time overhead is measured by the ratio of the time consumed by the reduction techniques to the total running time (i.e., $\frac{\text{time cost of reduction technique}}{\text{total running time}}$). As depicted in Figure 12, the global reduction technique often introduces a substantial time overhead, accounting for 78% of the total runtime in the rc dataset. However, the benefits obtained from this technique often outweigh the time overhead, since all cliques can be efficiently identified through this technique in the rc dataset. In complex datasets like co and lt, global reduction imposes minimal time overhead (3% and 1% for co and lt, respectively). In dataset sd, additional overhead is incurred as no vertices and edges are pruned. Nonetheless, in most scenarios, the sparsity of the graph ensures the effectiveness of global reduction. In most cases, the time ratio of dynamic reduction is less than 10%, such as in datasets cd, lt, and co. At most, it accounts for approximately 11.6% of the total running time in the ca dataset. Note that maximality check reduction costs less than 1.3% of the total time overhead in all algorithms, as observed in the wg dataset. In numerous datasets, this overhead is even below 0.1% (e.g., lt and fl), which shows that maximality check reduction can improve the running speed without introducing significant overhead.

Memory Overhead. The memory overhead is shown in Figure 13. The memory overhead of BKdegen and RMCEdegen is measured by the peak memory usage during the execution. The global reduction and maximality check reduction incur the same memory overhead. They use an integer array to track whether a vertex has been globally deleted and the neighborhood containment relationship between vertices, respectively. Dynamic reduction incurs slightly

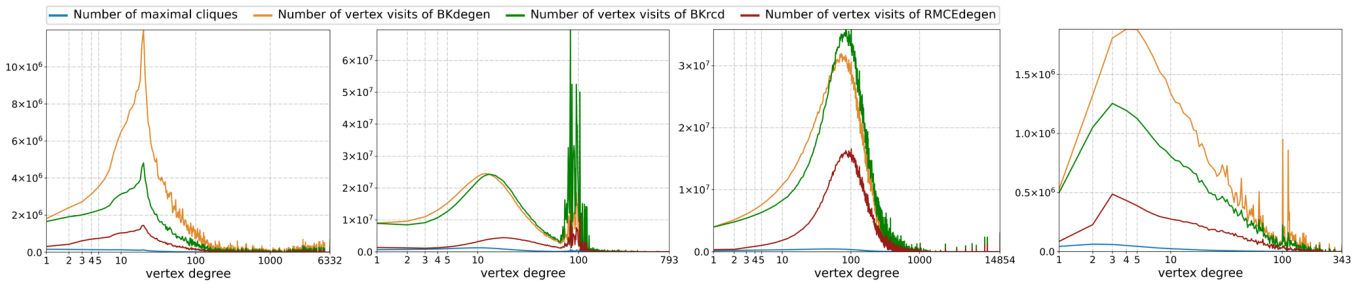


Figure 11: Illustration of the gaps between maximal cliques and vertex visits on four graphs (the horizontal axis is log-scaled).

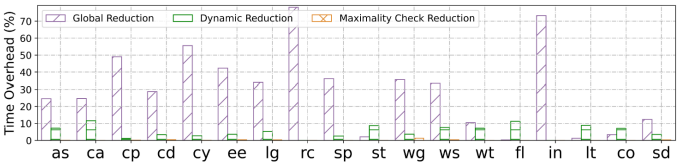


Figure 12: Time overhead.

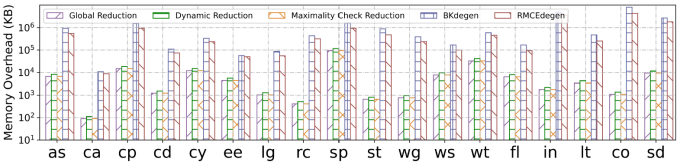


Figure 13: Memory overhead.

higher memory overhead, employing a boolean array to mark vertices in P and an integer array to store the degree of each vertex in the current subgraph. The memory overhead of these three reductions is linear to the number of vertices, which is approximately two orders of magnitude smaller than the peak memory usage. Furthermore, RMCEdegen incurs even less memory overhead compared to BKdegen, as the reductions can reduce the recursive calls.

8.4 Ablation Study

To evaluate the individual effectiveness of the proposed three reduction methods, we implement four variant algorithms, named *Variant1*, *Variant2*, *Variant3*, and *Variant4*. We have incorporated Global Reduction into the BKdegen algorithm, naming it *Variant1*. Building upon it, we added Dynamic Reduction, resulting in *Variant2*. *Variant1* and *Variant2* serve to analyze the step-by-step impact of our reduction techniques on the efficiency of the algorithm. Additionally, for the sake of completeness, we have also removed the Global Reduction and Dynamic Reduction techniques from the RMCEdegen algorithm, creating *Variant3* and *Variant4*, respectively.

From Table 3, we can observe that the complete version RMCEdegen outperforms all other variants in 11 datasets. However, *Variant3* runs faster in 7 datasets. As we can see that *Variant1* slows down the original BKdegen algorithm in datasets like as and sd. Nevertheless, in datasets like cp and rc, Global Reduction provides a significant acceleration. The runtime gap between *Variant3* and RMCEdegen is relatively small, confirming the overall effectiveness

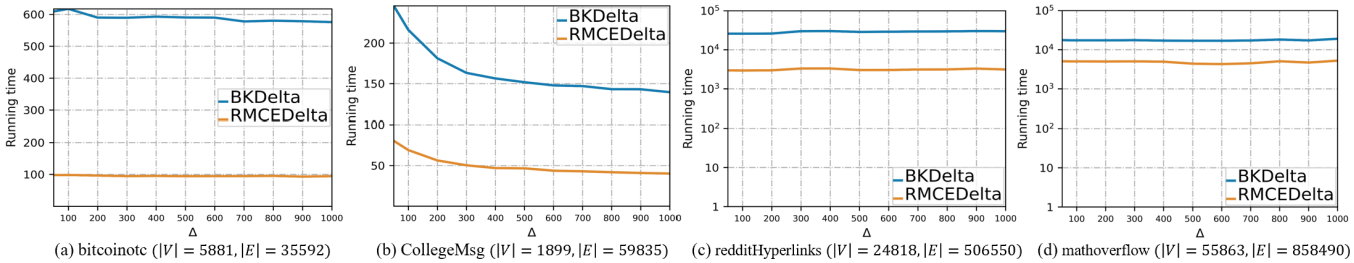
Table 3: Ablation study (in seconds)

Graph	BKdegen	Variant1	Variant2	RMCEdegen	Variant3	Variant4
as	80.55	84.78	60.77	57.49	**51.22**	70.52
ca	0.20	0.14	0.11	**0.05**	0.05	0.06
cp	56.11	31.91	24.86	**22.14**	25.71	25.85
cd	1.64	1.03	0.90	**0.67**	0.75	0.90
cy	6.84	6.23	4.19	4.01	**3.74**	4.47
ee	0.99	0.97	0.44	0.47	**0.39**	0.48
lg	2.68	2.62	2.06	1.91	**1.74**	2.38
rc	3.19	1.19	0.96	**0.95**	1.41	0.97
sp	73.97	71.28	48.93	44.77	**43.69**	49.62
st	554.99	548.24	415.12	**391.48**	405.62	478.73
wg	6.55	4.38	2.69	**2.55**	2.57	3.00
ws	2.95	2.42	1.53	**1.51**	1.52	2.08
wt	109.40	113.08	80.63	76.68	**75.63**	90.74
fl	287.84	282.91	185.40	**178.86**	184.36	249.78
in	49.38	11.54	11.62	**11.51**	19.07	11.82
lt	469.48	447.39	344.67	**325.24**	341.99	408.66
co	3554.64	3479.7	2451.96	**2393.59**	2475.37	2867.58
sd	28.25	32.27	12.04	11.52	**9.28**	13.53

of global reduction. Removing the dynamic reduction significantly degrades the performance in most datasets (e.g., as and fl) since it effectively prunes a substantial number of search branches during recursion. The benefits can also be observed in *Variant2*. The last column of the table demonstrates that maximality check reduction consistently improves the time efficiency across all datasets. As it only takes linear space overhead without incurring additional computations. These results validate the efficiency and effectiveness of our reduction methods in optimizing the MCE algorithm.

8.5 Applying to Parallel Algorithms

To further demonstrate the efficiency of our algorithm, we integrate the reduction techniques into a parallelized MCE algorithm. We have implemented the recent shared-memory parallel algorithm called **parMCE** proposed in [11]. ParMCE parallelizes both the expansion of subproblems and the pivot selection process. Our reduction techniques can be applied to parMCE easily since parMCE also adopts the BK framework. Besides, our reduction techniques can also be parallelized. Table 4 shows the running time of partial datasets to save space, while the complete version can be found in the technique report. The algorithm enhanced by our reduction techniques is called **parRMCE** and the largest threads are set to 32. From the table, we can find our parRMCE runs faster than parMCE in all datasets. Note that it takes even more time than the sequential methods mentioned above in some large graphs like co and lt. This is primarily due to the imbalance in workload among subproblems



(a) bitcoinotc ($|V| = 5881, |E| = 35592$) (b) CollegeMsg ($|V| = 1899, |E| = 59835$) (c) redditHyperlinks ($|V| = 24818, |E| = 506550$) (d) mathoverflow ($|V| = 55863, |E| = 858490$)

Figure 14: Running time vs. Δ on temporal graphs (in seconds).

Table 4: Result of applying to parallel experiment (in seconds)

Graph	as	cp	co	in	lt	rc	sp	wg
parMCE	180.72	20.30	6354.41	29.31	1100.96	2.76	40.71	5.16
parRMCE	**110.71**	**9.82**	**5208.12**	**7.41**	**759.67**	**0.64**	**24.89**	**3.05**

which is a common issue in the field of parallelized MCE algorithms. However, this phenomenon does not undermine the evidence of the effectiveness of our reduction techniques in parallel algorithms.

8.6 Experiments on Temporal Graphs

Since the source code of existing solution BKDelta [20] is implemented in Python, we implement our reduction version RMCEDelta by incorporating our reduction techniques in Python for fair comparison. We use four real temporal graphs in this experiment, which are downloaded from SNAP [24].

The running time of BKDelta and RMCEDelta is shown in Figure 14. The average speedups of RMCEDelta over BKDelta on datasets bitcoinotc, CollegeMsg, mathoverflow, and redditHyperlinks are 6.2, 3.3, 9.2, and 3.6, respectively. Due to the introduction of the temporal dimension in temporal graphs, the solution space of maximal cliques is often more complex. Surprisingly, effective reduction techniques can yield significant time savings, allowing for more efficient computations. Our reduction techniques can achieve an average of 5.6 speedups on these temporal graphs, further demonstrating the effectiveness and scalability of our reduction techniques.

9 RELATED WORK

Maximal Clique Enumeration. The BK algorithm[3] is a classic recursive backtracking algorithm that solves MCE. They also first propose the naive pivot technique which chooses the first vertex as the pivot. Tomita et al. [39] prove that the worst-case time complexity is $O(n3^{\frac{n}{3}})$ by choosing the pivot u which maximizes $|N(u) \cap P|$ from $P \cup X$. Eppstein et al. [15] further introduce the degeneracy order and reduce the worst-case time complexity to $O(n3^{\frac{\lambda}{3}})$. Li et al. [27] propose a top-to-down approach, that repeatedly removes the vertex with the smallest degree until a clique is reached, which aims to efficiently solve the MCE in these dense neighborhoods. Other studies have explored solving MCE using external memory [7] or using GPUs to accelerate the BK algorithm [46]. Additionally, the output-sensitive algorithm [6, 8, 29, 41] is a branching algorithm that guarantees the time interval between two consecutive outputs (also known as delay) at the polynomial level. The enumeration time of this algorithm is related to the number of all output maximal cliques. Many works focus on solving the MCE problem in other variant graphs, such as uncertain graphs[10, 26, 32], dynamic graphs[12, 38], temporal graphs[20, 31, 42], heterogeneous graphs[21], and attributed graphs[34, 49].

Parallel Approach. Numerous parallel algorithms have been designed for MCE [11, 13, 25, 36, 37]. Du et al. [13] implement a method that regards each vertex and its neighborhood $N(v)$ as a basic task, with each processor responsible for multiple basic tasks according to a simple serial number mapping. Schmidt et al. [37] improve load balancing of parallel algorithms through a work stealing strategy, where an idle thread randomly polls one or more search tree nodes at the bottom of the stack of other threads. Lessley et al. [25] introduce an approach consisting of data-parallel operations on shared-memory, multi-core architectures, aiming to achieve efficient and portable performance across different architectures. Das et al. [11] also present a shared-memory parallel method that parallelizes both pivot selection and sub-problem expansion.

Cohesive Subgraph Mining. Similar to MCE, many tasks also focus on the mining of cohesive subgraphs [16, 23, 43–45]. For example, Lijun Chang [4] introduces several efficient reduction rules to tackle the maximum clique problem. k-clique densest subgraph detection is an important task for identifying "near-cliques", and a highly efficient algorithm incorporating reduction techniques has been proposed in [19] to achieve the state-of-the-art performance, significantly contributing to the advancement of cohesive subgraph mining. Reduction rules have also been applied to another classical task maximum independent set [5, 9, 17].

10 CONCLUSION

In this paper, we introduce a novel reduction-based framework RMCE for enumerating maximal cliques. To reduce the computation cost, RMCE incorporates powerful graph reductions including global reduction, dynamic reduction, and maximality check reduction. We conduct comprehensive experiments over 18 real graphs. The empirical results confirm the effectiveness of our proposed reduction techniques.

ACKNOWLEDGMENTS

This work was supported by Key Projects of the National Natural Science Foundation of China (Grant No. U23A20496), Huawei Technologies Co., Ltd. (Grant No. TC20220804023), and the Research Grant Council of the Hong Kong Special Administrative Region, China (Grant No. CUHK 14217622). Weiguo Zheng is the corresponding author.

REFERENCES

[1] Faisal N Abu-Khzam, Nicole E Baldwin, Michael A Langston, and Nagiza F Samatova. 2005. 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*. 1–10.

[2] Kamanashis Biswas, Vallipuram Muthukkumarasamy, and Elankayer Sithirasenan. 2013. Maximal clique based clustering scheme for wireless sensor networks. In *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. IEEE, 237–241.

[3] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding All Cliques of an Undirected Graph. *Commun. ACM* 16, 9 (sep 1973), 575–577.

[4] Lijun Chang. 2019. Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 529–538.

[5] Lijun Chang, Wei Li, and Wenjie Zhang. 2017. Computing a near-maximum independent set in linear time by reducing-peeling. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1181–1196.

[6] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. 2013. Fast maximal cliques enumeration in sparse graphs. *Algorithmica* 66 (2013), 173–186.

[7] James Cheng, Yiping Ke, Ada Wai-Chee Fu, Jeffrey Xu Yu, and Linhong Zhu. 2011. Finding maximal cliques in massive networks. *ACM Transactions on Database Systems (TODS)* 36, 4 (2011), 1–34.

[8] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. 2016. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, Vol. 148. 1–148.

[9] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F Werneck. 2016. Accelerating local search for the maximum independent set problem. In *Experimental Algorithms: 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings 15*. Springer, 118–133.

[10] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, Hongzhi Chen, and Guoren Wang. 2022. Fast maximal clique enumeration on uncertain graphs: A pivot-based approach. In *Proceedings of the 2022 International Conference on Management of Data*. 2034–2047.

[11] Apurba Das, Seyed-Vahid Sanei-Mehri, and Srikanta Tirthapura. 2018. Shared-memory parallel maximal clique enumeration. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*. IEEE, 62–71.

[12] Apurba Das, Michael Svendsen, and Srikanta Tirthapura. 2019. Incremental maintenance of maximal cliques in a dynamic graph. *The VLDB Journal* 28 (2019), 351–375.

[13] Nan Du, Bin Wu, Liutong Xu, Bai Wang, and Xin Pei. 2006. A parallel algorithm for enumerating all maximal cliques in complex network. In *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*. IEEE, 320–324.

[14] Igor Dukanovic and Franz Rendl. 2007. Semidefinite programming relaxations for graph coloring and maximal clique problems. *Mathematical Programming* 109, 2-3 (2007), 345–365.

[15] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing all maximal cliques in sparse graphs in near-optimal time. In *Algorithms and Computation: 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I 21*. Springer, 403–414.

[16] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive Subgraph Search over Big Heterogeneous Information Networks: Applications, Challenges, and Solutions. In *Proceedings of the 2021 International Conference on Management of Data*. 2829–2838.

[17] Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. 2009. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)* 56, 5 (2009), 1–32.

[18] Shuo Han, Lei Zou, and Jeffrey Xu Yu. 2018. Speeding up set intersections in graph algorithms using simd instructions. In *Proceedings of the 2018 International Conference on Management of Data*. 1587–1602.

[19] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2023. Scaling Up k-Clique Densest Subgraph Detection. *Proc. ACM Manag. Data* 1, 1 (2023), 69:1–69:26. https://doi.org/10.1145/3588923

[20] Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. 2016. Enumerating maximal cliques in temporal graphs. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 337–344.

[21] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian YH Lam. 2019. Discovering maximal motif cliques in large heterogeneous information networks. In *ICDE*. 746–757.

[22] Yan Jin, Bowen Xiong, Kun He, Yangming Zhou, and Yi Zhou. 2022. On fast enumeration of maximal cliques in large graphs. *Expert Systems with Applications* 187 (2022), 115915.

[23] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. *A Survey of Algorithms for Dense Subgraph Discovery*. Springer US, Boston, MA, 303–336.

[24] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[25] Brenton Lessley, Talita Perciano, Manish Mathai, Hank Childs, and E Wes Bethel. 2017. Maximal clique enumeration with data-parallel primitives. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 16–25.

[26] Rong-Hua Li, Qiangqiang Dai, Guoren Wang, Zhong Ming, Lu Qin, and Jeffrey Xu Yu. 2019. Improved algorithms for maximal clique search in uncertain networks. In *ICDE*. 1178–1189.

[27] Yinuo Li, Zhiyuan Shao, Dongxiao Yu, Xiaofei Liao, and Hai Jin. 2019. Fast maximal clique enumeration for real-world graphs. In *International Conference on Database Systems for Advanced Applications*. Springer, 641–658.

[28] Zhenqi Lu, Johan Wahlström, and Arye Nehorai. 2018. Community detection in complex networks via clique conductance. *Scientific reports* 8, 1 (2018), 5982.

[29] Kazuhisa Makino and Takeaki Uno. 2004. 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, 260–272.

[30] Tsutomu Matsunaga, Chikara Yonemori, Etsuji Tomita, and Masaaki Muramatsu. 2009. Clique-based data mining for related genes in a biomedical database. *BMC bioinformatics* 10 (2009), 1–9.

[31] Hendrik Molter, Rolf Niedermeier, and Malte Renken. 2021. Isolation concepts applied to temporal clique enumeration. *Network Science* 9, S1 (2021), S83–S105.

[32] Arko Provo Mukherjee, Pan Xu, and Srikanta Tirthapura. 2015. Mining maximal cliques from an uncertain graph. In *ICDE*. 243–254.

[33] Kevin A Naudé. 2016. Refined pivot selection for maximal clique enumeration in graphs. *Theoretical Computer Science* 613 (2016), 28–37.

[34] Minjia Pan, Rong-Hua Li, Qi Zhang, Yongheng Dai, Qun Tian, and Guoren Wang. 2022. Fairness-aware maximal clique enumeration. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 259–271.

[35] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, Blai Bonet and Sven Koenig (Eds.). AAAI Press, 4292–4293.

[36] Pablo San Segundo, Jorge Artieda, and Darren Strash. 2018. Efficiently enumerating all maximal cliques with bit-parallelism. *Computers & Operations Research* 92 (2018), 37–46.

[37] Matthew C Schmidt, Nagiza F Samatova, Kevin Thomas, and Byung-Hoon Park. 2009. A scalable, parallel algorithm for maximal clique enumeration. *Journal of parallel and distributed computing* 69, 4 (2009), 417–428.

[38] Shengli Sun, Yimo Wang, Weilong Liao, and Wei Wang. 2017. Mining Maximal Cliques on Dynamic Graphs Efficiently by Local Strategies. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 115–118.

[39] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42.

[40] Armin Töpfer, Tobias Marschall, Rowena A Bull, Fabio Luciani, Alexander Schönhuth, and Niko Beerenwinkel. 2014. Viral quasispecies assembly via maximal clique enumeration. *PLoS computational biology* 10, 3 (2014), e1003515.

[41] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. 1977. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.* 6, 3 (1977), 505–517.

[42] Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. 2016. Computing maximal cliques in link streams. *Theoretical Computer Science* 609 (2016), 245–252.

[43] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Shunyang Li. 2022. Discovering Hierarchy of Bipartite Graphs with Cohesive Subgraphs. In *ICDE*. 2291–2305.

[44] Kai Wang, Gengda Zhao, Wenjie Zhang, Xuemin Lin, Ying Zhang, Yizhang He, and Chunxiao Li. 2023. Cohesive Subgraph Discovery Over Uncertain Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2023), 11165–11179.

[45] Kai Wang, Gengda Zhao, Wenjie Zhang, Xuemin Lin, Ying Zhang, Yizhang He, and Chunxiao Li. 2023. Cohesive Subgraph Discovery Over Uncertain Bipartite Graphs. *IEEE Trans. Knowl. Data Eng.* 35, 11 (2023), 11165–11179.

[46] Yi-Wen Wei, Wei-Mei Chen, and Hsin-Hung Tsai. 2021. Accelerating the Bron-Kerbosch algorithm for maximal clique enumeration using GPUs. *IEEE Transactions on Parallel and Distributed Systems* 32, 9 (2021), 2352–2366.

[47] Xuyun Wen, Wei-Neng Chen, Ying Lin, Tianlong Gu, Huaxiang Zhang, Yun Li, Yilong Yin, and Jun Zhang. 2016. A maximal clique based multiobjective evolutionary algorithm for overlapping community detection. *IEEE Transactions on Evolutionary Computation* 21, 3 (2016), 363–377.

[48] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics* 22, 7 (2006), 823–829.

[49] Qi Zhang, Rong-Hua Li, Minjia Pan, Yongheng Dai, Qun Tian, and Guoren Wang. 2023. Fairness-Aware Maximal Clique in Large Graphs: Concepts and Algorithms. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2023), 11368–11387.

[50] Weiguo Zheng, Yifan Yang, and Chengzhi Piao. 2021. Accelerating Set Intersections over Graphs by Reducing-Merging. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2349–2359.