Efficient Maximum s-Bundle Search via Local Vertex Connectivity

YANG LIU, Harbin Institute of Technology, Shenzhen, China
HEJIAO HUANG, Harbin Institute of Technology, Shenzhen, China
KAIQIANG YU*, Nanyang Technological University, Singapore
SHENGXIN LIU*, Harbin Institute of Technology, Shenzhen, China
CHENG LONG, Nanyang Technological University, Singapore

The s-bundle, as a cohesive subgraph model which relaxes the clique, remains connected whenever fewer than $n - s$ vertices are removed, where n is the number of vertices inside. Finding the largest s-bundle is a fundamental problem and has diverse applications in various fields such as social network analysis, graph visualization, and bioinformatics. Existing studies for solving the problem follow the same branch-and-bound framework and improve the efficiency by developing pruning techniques. As a result, all share the same worst-case time complexity of $O^*(2^n)$, where O^* suppresses the polynomial factors. In this paper, we propose a new branch-and-bound algorithm, called SymBD, which achieves improved theoretical guarantees and practical performance. It adopts the existing Symmetric-BK branching strategy whose performance highly depends on the ordering of vertices. We explore various vertex orderings for improving the performance. In particular, we propose two novel vertex orderings based on the local vertex connectivity. With the proposed vertex orderings, SymBD improves the worst-case time complexity to $O^*(\lambda_s^n)$ where λ_s is strictly less than 2. To further boost the practical efficiency, we introduce a heuristic algorithm for computing a large initial solution and a divide-and-conquer strategy. Extensive experiments on 664 graphs demonstrate that our algorithm is up to five orders of magnitude faster than existing solutions.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**.

Additional Key Words and Phrases: Cohesive subgraph search, s-Bundle, Branch-and-bound

ACM Reference Format:
Yang Liu, Hejiao Huang, Kaiqiang Yu, Shengxin Liu, and Cheng Long. 2025. Efficient Maximum s-Bundle Search via Local Vertex Connectivity. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 37 (February 2025), 75 pages. https://doi.org/10.1145/3709687

1 Introduction

Graphs are widely used for capturing relationships among entities across diverse domains, such as social networks, biological networks, financial networks, and collaboration networks [11]. To understand these complex relationships, graph analytics has been increasingly adopted to extract

*Corresponding authors.

Authors' Contact Information: Yang Liu, 23b951003@stu.hit.edu.cn, Harbin Institute of Technology, Shenzhen, Shenzhen, China; Hejiao Huang, Harbin Institute of Technology, Shenzhen, Shenzhen, China, huanghejiao@hit.edu.cn; Kaiqiang Yu, Nanyang Technological University, Singapore, kaiqiang002@e.ntu.edu.sg; Shengxin Liu, Harbin Institute of Technology, Shenzhen, Shenzhen, China, sxliu@hit.edu.cn; Cheng Long, Nanyang Technological University, Singapore, c.long@ntu.edu.sg.

ACM 2836-6573/2025/2-ART37
https://doi.org/10.1145/3709687

interesting information from graphs. In particular, finding cohesive subgraphs is an important problem in graph analytics, which has been extensively studied and applied in a wide range of applications [10, 23, 24, 32, 36]. For example, extracting the densely connected subgraphs would help to identify irregularities within financial networks [3], find groups of research collaborators in publication networks [29], detect unfolding narratives in real-time on social media platforms [4], and recognize protein complexes in biological networks [57].

Clique is known as a classic graph structure for defining cohesive subgraphs, which requires subgraphs to be fully connected, i.e., an edge between every pair of distinct vertices in the subgraph. In the literature, extensive studies are conducted on identifying cliques in big graphs and quite a few clique related problems are explored, including listing k-cliques [51], finding the maximum clique [7], and enumerating all maximal cliques [54]. However, the fully-connected requirement on a large subgraph is shown to be too restrictive for many real applications where data quality is a real concern. As a result, quite a few relaxations of cliques have been explored in the recent years, including s-plex [46], s-defective clique [57], quasi-clique [2], s-block [40, 43], s-bundle [44] and so on. In this paper, we focus on s-bundle and are motivated to study the problem of finding the maximum s-bundle. Specifically, s-bundle requires a subgraph to remain connected whenever fewer than $n - s$ vertices are removed, where n is the number of vertices in the subgraph. We remark that an s-bundle with $s = 1$ is a clique.

Compared to other clique relaxations, i.e., s-plex and s-defective clique, which define the cohesive subgraphs by simply requiring at most certain number of vertices and/or edges being missed in subgraphs, s-bundle defines the cohesive subgraphs via *connectivity*. Given the differences, we have conducted a case study, which shows that s-bundle works better than other clique relaxations, including s-plex and s-block, for the task of community search (details will be presented in Section 6.3). Besides, the problem of finding maximum s-bundle has been widely studied and has found many real applications [28, 30, 44, 66].

Existing methods. The maximum s-bundle search problem is NP-hard [37, 44]. There are quite a few studies on the problem of finding maximum s-bundle [28, 30, 48, 66]. The recent state-of-the-art approaches [30, 66] all utilize the same *branch-and-bound (BB)* framework. In particular, they recursively partition the problem instance into several sub-problem instances via *branching*; each sub-problem instance corresponds to a branch. We note that these approaches adopt the same branching strategy, which would produce $O(2^n)$ branches in the worst case where n is the number of vertices in the input graph. As a result, they all have the worst-case time complexity of $O^*(2^n)$, where O^* suppresses the polynomial factors. We remark that the state-of-the-art worst-case time complexity for finding the maximum s-bundle is $O^*(2^n)$. Besides, many pruning techniques are proposed in these BB algorithms towards improving the practical performance. Nevertheless, they still suffer from the efficiency and scalability issues. For example, the state-of-the-arts [30, 66] cannot find the maximum 5-bundle for those graphs with millions of vertices within a time limit of 3 hours, as verified in our experiments. In summary, this motivates us to advance the state-of-the-arts both theoretically and practically.

Our Solutions. In this paper, we develop a new branch-and-bound algorithm, called `SymBD`, which achieves improved theoretical guarantees and practical performance. `SymBD` adopts the recently proposed Symmetric-BK (Sym-BK) branching strategy [55, 59], which tends to produce fewer branches compared with the binary branching strategy used in [30, 66]. We observe that the performance of the Sym-BK branching depends on the vertex ordering. In particular, with an arbitrary vertex ordering, the Sym-BK branching would produce $O(2^n)$ branches in the worst case. To improve, we are motivated to explore various vertex orderings towards producing fewer branches in both practice and theory. Specifically, we propose two novel vertex orderings that

leverage *local vertex connectivity*, namely the local-vertex-connectivity-based (LVC-based) vertex ordering and the relaxed-local-vertex-connectivity-based (RLVC-based) vertex ordering. With these vertex orderings, SymBD improves the worst-case time complexity to $O^*(\lambda_s^n)$, where λ_s is the largest real root of the equation $x^{s+2} - 2x^{s+1} + 1 = 0$. Note that λ_s is strictly less than 2 and varies depending on s. For example, when $s = 1$, 2, and 3, $\lambda_s = 1.6181$, 1.8393, and 1.9276, respectively. Thus, we advance the state of the art regarding the theoretical time complexity. Further, SymBD incorporates upper-bounding reductions to prune unpromising branches within the branch-and-bound framework. Specifically, we introduce two new upper bounds, both derived from the idea of local vertex connectivity.

To boost the practical efficiency and scalability of SymBD, we provide several reduction rules and a divide-and-conquer strategy. First, the reduction rules are used to refine the input graph by removing those vertices/edges that cannot appear in any s-bundle larger than a lower bound of the largest s-bundle. Clearly, the larger the lower bound is, the more vertices/edges we can prune. To unleash the pruning power, we propose a heuristic algorithm DegenPro for obtaining a large s-bundle which can serve as the lower bound. Second, the divide-and-conquer strategy has been widely used for finding cohesive subgraphs on a big graph [11, 12, 58]. The idea is to divide the graph into several smaller ones and run SymBD on each of them, thus improving the scalability.

Contributions. The main contributions of our paper are as follows.

- We propose a new branch-and-bound algorithm SymBD, which is based on the existing Sym-BK branching and our newly proposed local-vertex-connectivity-based vertex orderings. We remark that SymBD improves the worst-case time complexity to $O^*(\lambda_s^n)$, where λ_s is strictly less than 2. Besides, SymBD also incorporates new upper bounds to effectively prune unpromising branches. (Section 4)
- We further integrate a heuristic method, reduction rules, and the divide-and-conquer strategy into our SymBD for boosting the practical performance. In particular, we design a new heuristic method DegenPro for obtaining a large initial s-bundle in an efficient manner. (Section 5)
- We conduct extensive empirical studies on three benchmark graph collections with 664 graph instances. The results show that (1) SymBD outperforms the state-of-the-art algorithms in solving a greater number of instances. Specifically, in the 10th DIMACS and real-world graph collections, SymBD solves more instances within 1 second than the state-of-the-art algorithms do within 3 hours; (2) on the 40 representative graphs, SymBD is up to five orders of magnitude faster than the state-of-the-art algorithms; (3) our newly devised heuristic method improves the practical efficiency of SymBD. (Section 6)

In addition, Section 2 formulates the problem we studied. Section 3 reviews the state-of-the-art methods. We discuss the related work in Section 7 and conclude the paper in Section 8.

2 Preliminaries

Let $G = (V, E)$ be a graph with the vertex set V and the edge set E, where $|V| = n$ and $|E| = m$. Given a vertex v in G, we denote the set of neighbors of v in G by $N(v, V) = \{u \in V \mid (u, v) \in E\}$ and its degree in G by $d(v, V) = |N(v, V)|$. Given a vertex set $S \subseteq V$, we say $G[S]$ is the subgraph induced by S in G, i.e., $G[S] = (S, \{(u, v) \in E \mid u, v \in S\})$. Let g be an induced subgraph of G. We denote the vertex set and the edge set of g as $V(g)$ and $E(g)$, respectively.

A *vertex cut* of a connected graph G is a set of vertices $\Gamma \subset V$ such that removing Γ from G results in the remaining graph $G[V \setminus \Gamma]$ becomes either disconnected or trivial (i.e., a 1-vertex graph). The *(vertex) connectivity* of G, denoted by $\kappa(G)$, is a non-negative integer given by the size of the *minimum* vertex cut of G, i.e., the minimum number of vertices whose deletion yields a disconnected or a trivial graph. For a disconnected graph G, all of its vertex cuts are defined by the

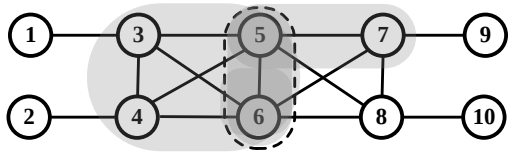


Fig. 1. Illustration of the concepts ($\{v_5, v_6\}$ is a vertex cut and $\{v_3, v_4, v_5, v_6, v_7\}$ induces a 3-bundle)

empty set and its connectivity is defined as 0, i.e., $\kappa(G) = 0$. To illustrate, consider an example in Figure 1 where $\{v_5, v_6\}$ is a vertex cut of the input graph (since after the removal of $\{v_5, v_6\}$, the input graph has two components $\{v_1, v_2, v_3, v_4\}$ and $\{v_7, v_8, v_9, v_{10}\}$).

In this paper, we focus on the s-bundle defined as follows.

Definition 2.1 (s-bundle [44]). Given a graph g and a positive integer s, g is said to be an *s-bundle* if and only if $\kappa(g) \geq |V(g)| - s$.

To illustrate, consider an example in Figure 1 where $\{v_3, v_4, v_5, v_6, v_7\}$ induces a 3-bundle since its connectivity is 2 (note that $\{v_5, v_6\}$ is the minimum vertex cut). Intuitively, an s-bundle g remains connected with fewer than $|V(g)| - s$ vertices being removed (note that a graph containing at most s vertices is an s-bundle). We also note that any disconnected or trivial subgraph of an s-bundle (i.e., the remaining graph after removing a vertex cut) involves at most s vertices (since otherwise there exists a vertex cut with the size smaller than $|V(g)| - s$). Thus, the smaller the value of s, the denser the s-bundle. In particular, 1-bundles are equivalent to cliques. Besides, the s-bundle possesses the hereditary property [44, 66], i.e., any induced subgraph of an s-bundle is still an s-bundle. This is because the difference between the size of the remaining graph and the connectivity of the remaining graph is non-increasing after removing a set of vertices from an s-bundle.

We are ready to define the problem studied in this paper.

Problem Definition. *Given a graph G and an integer s, the* maximum s-bundle search problem (MSBP) *aims to find the largest s-bundle in G.*

We remark that MSBP is NP-hard [44] since the s-bundle enjoys the hereditary property and the NP-hardness of MSBP then follows from the general proof of [37].

3 The state-of-the-art framework

Quite a few algorithms have been developed for solving the problem in the literature [30, 66]. They all follow the same *branch-and-bound* framework called BinSymBD, which recursively solves the problem instance via a process of branching. Specifically, the branching process partitions the current problem instance of finding the largest s-bundle into several sub-problem instances. Each problem instance corresponds to a branch, represented by a pair of disjoint vertex sets (S, C). The partial set S is a set of vertices that induces an s-bundle and must be included in any s-bundle within the branch; the candidate set C is a set of vertices that can be included into S to form a larger s-bundle. Solving a problem instance (or branch) (S, C) involves finding the largest s-bundle in the branch; an s-bundle g is said to be in a branch (S, C) if it (1) contains all vertices in S and (2) is a subgraph of $G[S \cup C]$, i.e., $S \subseteq V(g) \subseteq S \cup C$. Thus, solving the branch $(\emptyset, V)$ finds the largest s-bundle in G. We remark that (1) *a branch (S, C) with its partial set $G[S]$ not being an s-bundle holds no s-bundle* since any subgraph to be found in (S, C) must contain S and cannot be an s-bundle either due to the hereditary property and thus (2) the candidate set C can be refined by removing from C those vertices for which v cannot form an s-bundle with S, i.e., $G[S \cup \{v\}]$ is not an s-bundle, since any superset of $S \cup \{v\}$ to be found in (S, C) cannot induce an s-bundle.

One key part of BinSymBD falls in the branching strategy, i.e., the method of partitioning the current branch into sub-branches. In particular, it adopts a hybrid branching strategy, which combines two kinds of strategies, namely *symmetric-BK (Sym-BK) branching* and *binary branching*. Consider the branching process at a current branch $B = (S, C)$, both strategies rely on a vertex v^*, called the *branching vertex*, from either C or S, which is selected based on the *degree*. Below, we elaborate on the details.

Degree-based Sym-BK branching. The Sym-BK branching forms $|C| + 1$ sub-branches from B, namely $B_1, B_2, ..., B_{|C|+1}$, based on a given ordering of vertices in C, i.e., $C = \{v_1, v_2, ..., v_{|C|}\}$. Each branch $B_i = (S_i, C_i)(1 \leq i \leq |C| + 1)$ holds those s-bundles that (1) include $S \cup \{v_1, ..., v_{i-1}\}$ and (2) exclude v_i. Formally,

$$S_i = S \cup \{v_1, v_2, ..., v_{i-1}\}, \quad C_i = C - \{v_1, v_2, ..., v_i\}, \tag{1}$$

where v_0 in set S_1 and $v_{|C|+1}$ in set $C_{|C|+1}$ are fictitious. Clearly, solving all formed sub-branches finds the largest s-bundle in branch B. Consider one sub-branch B_i and any other sub-branches B_j following B_i, i.e., $1 \leq i < j \leq |C| + 1$. One important property is that S_i *is always a subset of* S_j. As a result, if B_i has its partial set S_i that induces a non-s-bundle and can be pruned, all other branches B_j following B_i would have their partial set violating s-bundle and can be pruned as well. The earlier such a branch B_i is formed, the more branches can be pruned. Therefore, the performance depends on the ordering of vertices in C. BinSymBD uses a degree-based vertex ordering, which is determined by a branching vertex v^* selected from $S \cup C$. Specifically, it has the smallest degree among those vertices in $S \cup C$, i.e., $v^* := \arg\min_{v \in S \cup C} d(v, S \cup C)$, and has the degree smaller than $|S \cup C| - s$, i.e., $d(v, S \cup C) < |S \cup C| - s$. Then, the degree-based vertex ordering is obtained by putting v^* (if it is in C) together with its non-neighbours before the other vertices in C. We call the Sym-BK branching with degree-based vertex ordering *degree-based Sym-BK branching*.

We note that the degree-based Sym-BK branching would produce fewer branches in both practice and theory [30, 59, 66]. In particular, applying it solely (if possible) will produce $O(\lambda_s^n)$ branches in the worst case, where λ_s is strictly less than 2 [59]. However, it cannot be applied in some cases when such a branching vertex does not exist (i.e., all vertices in $S \cup C$ have the degree at least $|S \cup C| - s$). To illustrate, consider an example with $s = 3$ in Figure 2. Given an input graph in Figure 2(a), the degree-based Sym-BK branching on a branch (S, C) with $S = \{v_1, v_2\}$ and $C = \{v_3, v_4, ..., v_{10}\}$ is shown in Figure 2(b). Specifically, v_9 has the smallest degree of 1 and thus serves as the branching vertex. By putting v_9 together with its non-neighbours before the other vertices, we get the degree-based vertex ordering, i.e., $\langle v_9, v_{10}, v_3, v_4, v_5, v_6, v_7, v_8 \rangle$. Besides, we note that the partial set $S \cup \{v_9, v_{10}\}$ in B_3 is not a 3-bundle since it is disconnected and thus has the connectivity of 0, which is smaller than $|S \cup \{v_9, v_{10}\}| - s = 1$. As a result, the following branches $B_4, ..., B_9$ can also be pruned.

Degree-based binary branching. This strategy is adopted when the degree-based Sym-BK branching cannot be applied. Specifically, it selects from C the vertex with the smallest degree as the branching vertex, i.e., $v^* := \arg\min_{v \in C} d(v, S \cup C)$. Then, it creates two sub-branches by either including v^* into the partial set, i.e., $B_1 = (S \cup \{v^*\}, C \backslash \{v^*\})$, or discarding v^* from the candidate set, i.e., $B_2 = (S, C \backslash \{v^*\})$. Clearly, solving B_1 and B_2 finds the largest s-bundle in branch B. We note that applying the degree-based binary branching solely produces $O(2^n)$ branches in the worst case.

Summary and Time complexity. We summarize BinSymBD in Algorithm 1. Specifically, it terminates a branch when $G[S \cup C]$ becomes an s-bundle since $G[S \cup C]$ is the largest s-bundle in the branch (Lines 3-4). It then applies some existing pruning techniques to narrow down the search space, for which we omit the details (Line 5). Finally, it applies the degree-based Sym-BK branching if possible since it would produce fewer branches (Lines 6-10); otherwise, it adopts the degree-based

Algorithm 1: The existing framework: BinSymBD

Input: A graph $G = (V, E)$ and an integer s
Output: The maximum s-bundle g^*

1 $g^* \leftarrow \emptyset$; BinSymBD_Rec($\emptyset, V$);
2 **return** g^*;

Procedure: BinSymBD_Rec(S, C)
3 **if** $G[S \cup C]$ *is an s-bundle* **then**
4 $\quad g^* \leftarrow G[S \cup C]$ if $|V(g^*)| < |S \cup C|$; **return**;
5 Apply pruning techniques for refining the candidate set C;
6 $v^* \leftarrow \arg\min_{v \in S \cup C} d(v, S \cup C)$;
7 **if** $d(v^*, S \cup C) < |S \cup C| - s$ **then**
$\quad$ /* Degree-based Sym-BK branching */
8 $\quad$ Create branches $\{B_1, B_2, ...\}$ via degree-based Sym-BK branching;
9 $\quad$ **foreach** *branch* $B_i = (S_i, C_i)$ *in* $\{B_1, B_2, ...\}$ **do**
10 $\quad\quad$ **if** $G[S_i]$ *is an s-bundle* **then** BinSymBD_Rec(S_i, C_i);
11 **else**
$\quad$ /* Degree-based binary branching */
12 $\quad$ $v^* \leftarrow \arg\min_{v \in C} d(v, S \cup C)$;
13 $\quad$ BinSymBD_Rec($S \cup \{v^*\}, C \backslash \{v^*\}$);
14 $\quad$ BinSymBD_Rec($S, C \backslash \{v^*\}$);

binary branching instead (Lines 11-14). In summary, it would explore $O(2^n)$ branches in the worst case (e.g., the cases where only degree-based binary branching can be applied). Hence, it has the worst-case time complexity of $O^*(2^n)$.

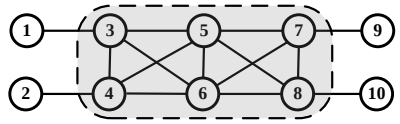
4 Our Branch-and-Bound Algorithms

We observe that the Sym-BK branching suits better for finding the largest s-bundle based on the previous discussion. However, its performance highly relies on the ordering of vertices in the candidate set. We note that the Sym-BK branching with an arbitrary vertex ordering would produce $O(2^n)$ branches in the worst case and thus cannot improve the worst-case time complexity for solving MSBP. Though the degree-based Sym-BK branching would produce fewer branches, it is not always applicable (since the degree-based vertex ordering may not exist) and thus cannot improve the worst-case time complexity either.

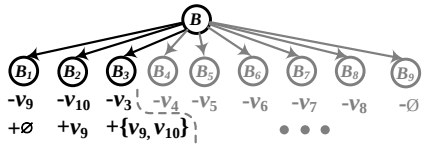
Motivated by all above, we will explore other vertex orderings for Sym-BK branching *towards producing fewer branches both theoretically and practically* in this section. Specifically, we propose two new vertex orderings based on the concept of *local vertex connectivity*: LVC-based vertex ordering (Section 4.1) and RLVC-based vertex ordering (Section 4.2). We then present our branch-and-bound algorithms SymBD-L and SymBD-H, which incorporate the Sym-BK branching with carefully designed vertex orderings, in Section 4.3. We remark that both SymBD-L and SymBD-H improve the worst-case time complexity to $O^*(\lambda_s^n)$, where λ_s is strictly smaller than 2.

4.1 Local-Vertex-Connectivity-based Vertex Ordering

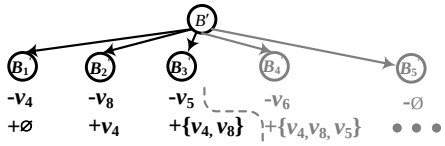
Overview. To design a vertex ordering for the Sym-BK branching, the idea is to *identify a small subset C_{sub} of vertices in the candidate set C such that including them in the partial set S will violate*



(a) Input graph used throughout the paper



(b) Degree-based Sym-BK branching (B=(S, C), S={v_1, v_2}, and C={v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}})



(c) LVC-based Sym-BK branching (B'=(S', C'), S'={v_3, v_7} and C'={v_4, v_5, v_6, v_8})

Fig. 2. A running example graph with $s = 3$, which includes illustrations of (b) degree-based Sym-BK branching and (c) LVC-based Sym-BK branching. The notation "+" means to include a vertex by adding it into the partial set and "-" means to exclude a vertex by removing it from the graph.

the definition of s-bundle. By putting them before other vertices in the ordering, we can prune the sub-branch with the partial set of $S \cup C_{sub}$ and all of its following sub-branches. To this end, we first propose an equivalent condition for a graph to be an s-bundle, which is derived via the concept of local vertex connectivity. Based on this equivalent condition, we then figure out a small subset C_{sub} such that $S \cup C_{sub}$ violates the equivalent condition and thus is not an s-bundle. Below, we elaborate on the details.

Equivalent condition derived via local vertex connectivity. We first review the concept of local vertex connectivity [30, 38, 39, 53, 66] which is defined on a pair of vertices. Specifically, consider two vertices u and v in G, the *local vertex connectivity*, denoted by $\kappa(G, u, v)$, of a pair of vertices $\langle u, v\rangle$ in G falls in the two cases.

- **Case 1:** u is not adjacent to v in G, i.e., $(u, v) \notin E$. The local vertex connectivity $\kappa(G, u, v)$ is the size of the minimum u-v vertex cut of G; Here, the u-v vertex cut of G is a vertex cut Γ of G such that (1) Γ excludes u and v (i.e., $u \notin \Gamma$ and $v \notin \Gamma$) and (2) u disconnects[1] to v after the removal of Γ from G, i.e., $G[V \backslash \Gamma]$.
- **Case 2:** u is adjacent to v in G, i.e., $(u, v) \in E$. The local vertex connectivity $\kappa(G, u, v)$ is defined by $|V| - 1$, and the u-v vertex cut is $V \backslash \{u\}$ or $V \backslash \{v\}$ whose removal yields a trivial graph.

To illustrate, consider an example in Figure 2(a). Given two adjacent vertices v_1 and v_3, the local vertex connectivity is $\kappa(G, v_1, v_3) = |V| - 1 = 9$. Given two non-adjacent vertices v_1 and v_2, one minimum v_1-v_2 vertex cut is $\{v_3\}$ (since v_1 disconnects to v_2 after the removal of v_3) and thus its local vertex connectivity is 1.

Given the definition of local vertex connectivity, we observe that the graph connectivity $\kappa(G)$ is equal to the the minimum local vertex connectivity $\kappa(G, u, v)$ among all possible pairs of vertices u

[1]We say that vertex u disconnects to vertex v in G if and only if there does not exist any path from u to v in G.

and v in G [30], formally,

$$\kappa(G) = \min_{u,v \in V} \kappa(G, u, v). \tag{2}$$

This can be easily verified based on the definitions as follows. First, we have $\kappa(G) \leq \min_{u,v \in V} \kappa(G, u, v)$. This is because $\kappa(G)$ is the size of the minimum vertex cut of G, denoted by Γ^*, and $\kappa(G, u, v)$ is the size of one specific vertex cut of G, i.e., the minimum u-v vertex cut. Second, we can deduce $\kappa(G) \geq \min_{u,v \in V} \kappa(G, u, v)$. Note that if $G[V \backslash \Gamma^*]$ is a trivial graph, G is a clique and we have $\kappa(G) = \min_{u,v \in V} \kappa(G, u, v) = |V| - 1$. Otherwise, $G[V \backslash \Gamma^*]$ is disconnected and thus there exists two vertices u' and v' in $G[V \backslash \Gamma^*]$ such that u' disconnects to v'. Hence, Γ^* is a u'-v' vertex cut of G and thus $\kappa(G) = |\Gamma^*| \geq \kappa(G, u', v') \geq \min_{u,v \in V} \kappa(G, u, v)$.

Given all above, we can derive the following equivalent condition, namely LVC-based condition, for a graph to be an s-bundle.

Lemma 4.1 (LVC-based condition). *A graph g is an s-bundle iff $\kappa(g, u, v) \geq |V(g)| - s$ for every pair of vertices u and v in g.*

LVC-based vertex ordering. We then leverage the local vertex connectivity to design the vertex ordering. Consider a branch $B = (S, C)$. We note that $G[S \cup C]$ is not an s-bundle since otherwise this branch can be terminated. Thus, there exists one pair of vertices $\langle u, v \rangle$ in $S \cup C$ such that its local vertex connectivity is smaller than $|S \cup C| - s$, i.e., $\kappa(G[S \cup C], u, v) < |S \cup C| - s$, based on Lemma 4.1.

We partition the vertex set $S \cup C$ into two disjoint subsets: the first one, denoted by P, corresponds to the minimum u-v vertex cut in $G[S \cup C]$ and the second one, denoted by Q, corresponds to the set of other vertices, i.e., $S \cup C \setminus P$. We remark that the minimum u-v vertex cut may not be unique, and P can be any arbitrary one. Clearly, we have

$$|P| < |S \cup C| - s \text{ and } |Q| > s. \tag{3}$$

We propose to put those vertices in $Q \cap C$ before other vertices (in $C \setminus Q$) in the ordering towards producing fewer branches. In general, there are three scenarios as below.

Scenario 1: $u \in S$ and $v \in S$. In this case, both u and v are in the partial set S. Consider a subset Q_{sub} of $Q \cap C$. We observe that *$P \cap S$ is a u-v vertex cut of subgraph $G[S \cup Q_{sub}]$*, which can be verified by contradiction as follows. Suppose that our observation does not hold, i.e., u still connects to v after removing $P \cap S$ from $G[S \cup Q_{sub}]$. Hence, there exists a path T from u to v in $G[S \cup Q_{sub} \setminus P]$ and the path T consists of vertices from Q only. This implies that u connects to v via the path T in $G[Q]$, which contradicts to the fact that P is a u-v vertex cut of $G[S \cup C]$ and $G[P \cup Q]$. In summary, we have the following lemma.

Lemma 4.2. *Let $P \cup Q$ be the partition on a branch (S, C), and Q_{sub} be a subset of $Q \cap C$. $P \cap S$ is a u-v vertex cut of subgraph $G[S \cup Q_{sub}]$.*

Based on the above lemma, the local vertex connectivity $\kappa(G[S \cup Q_{sub}], u, v)$ of vertex pair $\langle u, v \rangle$ in $G[S \cup Q_{sub}]$ is at most the size of the u-v vertex cut $P \cap S$. Formally, we have

$$\kappa(G[S \cup Q_{sub}], u, v) \leq |P \cap S| = |S| - |Q \cap S|. \tag{4}$$

The above equation implies that $G[S \cup Q_{sub}]$ would violate the LVC-based condition and thus cannot be an s-bundle when $|Q_{sub}| > s - |Q \cap S|$. This is because $\kappa(G[S \cup Q_{sub}], u, v) \leq |S| - |Q \cap S| < |S| + |Q_{sub}| - s$. We thus define

$$x = s - |Q \cap S| \text{ and } y = |Q \cap C|. \tag{5}$$

Specifically, x corresponds to the largest possible number of vertices that we can remove from $Q \cap C$ to S for forming larger s-bundles. Besides, we have

$$0 \leq x < y, \tag{6}$$

since (1) $x \geq 0$, i.e., $|Q \cap S| \leq s$ since otherwise $G[S]$ is not an s-bundle (note that $P \cap S$ is a u-v vertex cut of $G[S]$ and thus $\kappa(G[S], u, v) \leq |P \cap S| = |S| - |Q \cap S| < |S| - s$, which violates the LVC-based condition) and (2) $y - x = |Q \cap C| + |Q \cap S| - s = |Q| - s > 0$ according to Equation (3).

Given all above, we define the ordering as below.

$$\langle v_1, v_2, ..., v_y, v_{y+1}, ..., v_{|C|} \rangle, \tag{7}$$

where $v_1, ..., v_y$ are vertices from $Q \cap C$ in any order and $v_{y+1}, ..., v_{|C|}$ are vertices from the $C \backslash Q$ in any order. Based on the above discussion, branch B_{x+2} has the partial set of $S \cup \{v_1, ..., v_{x+1}\}$, which induces a non-$s$-bundle since the local vertex connectivity of vertex pair $\langle u, v \rangle$ is at most $|S| - |Q \cap S| = |S| + x - s$ according to Equation (4) and thus is smaller than $|S \cup \{v_1, v_2, ..., v_{x+1}\}| - s = |S| + (x + 1) - s$. Hence, branches $B_{x+2}, ..., B_{|C|+1}$ can be pruned and we only need to create the first $x + 1$ branches.

To illustrate, consider an example in Figure 2(c). The current branch is $B' = (S', C')$ with $S' = \{v_3, v_7\}$ and $C' = \{v_4, v_5, v_6, v_8\}$. Note that the degree-based Sym-BK branching cannot be applied since any vertex in $S' \cup C'$ has degree at least 3. Based on vertex pair $\langle v_3, v_7 \rangle$ with the local vertex connectivity smaller than 3, the LVC-based vertex ordering is $\langle v_4, v_8, v_5, v_6 \rangle$ according to Equation (7). Besides, we note that B'_3 has the partial set $S' \cup \{v_4, v_8\}$ that does not form an s-bundle since it is disconnected and has the connectivity of 0 smaller than 1. Thus, branches B'_3, B'_4, B'_5 can be pruned.

Scenario 2: $u \in S$ and $v \in C$. Note that $u \in C$ and $v \in S$ corresponds to a symmetric case, the ordering can be obtained symmetrically. In this case, one vertex v is not in S. Consider a subset Q'_{sub} of $Q \cap C \setminus \{v\}$. From Lemma 4.2, $P \cap S$ is a u-v vertex cut of $G[S \cup \{v\} \cup Q'_{sub}]$ and then the local vertex connectivity is bounded as below.

$$\kappa(G[S \cup \{v\} \cup Q'_{sub}], u, v) \leq |P \cap S| = |S| - |Q \cap S|. \tag{8}$$

Therefore, we can include v together with at most $s - |Q \cap S| - 1$ vertices from $Q \cap C \setminus \{v\}$ to S without violating the LVC-based condition. This is because when $|Q'_{sub}| > s - |Q \cap S| - 1$, we have $\kappa(G[S \cup \{v\} \cup Q'_{sub}], u, v) \leq |S| - |Q \cap S| < |S| + |Q'_{sub}| + 1 - s$.

Refer to the definitions in Equation (5) where $x < y$ still holds in this case. Besides, we can deduce $x \geq 1$. This is because otherwise v can be refined from C since $G[S \cup \{v\}]$ is not an s-bundle (note that $P \cap S$ is a u-v vertex cut of $G[S \cup \{v\}]$ and thus $\kappa(G[S \cup \{v\}], u, v) \leq |P \cap S| = |S| - |Q \cap S| < |S| + 1 - s$). We define the ordering as below.

$$\langle v, v_2, v_3, ..., v_y, v_{y+1}, ..., v_{|C|} \rangle, \tag{9}$$

where $v_2, ..., v_y$ are vertices from the vertex set $Q \cap C \setminus \{v\}$ in any order and $v_{y+1}, ..., v_{|C|}$ are vertices from $C \setminus Q$ in any order. Similar to Scenario 1, branches $B_{x+2}, ..., B_{|C|+1}$ can be pruned and only the first $x + 1$ branches need to be kept.

Scenario 3: $u \in C$ and $v \in C$. In this case, both u and v are not in S. Consider a subset Q''_{sub} of $Q \cap C \setminus \{u, v\}$. Based on Lemma 4.2, $P \cap S$ is a u-v vertex cut of subgraph $G[S \cup \{u, v\} \cup Q''_{sub}]$ and then the local vertex connectivity is bounded as below.

$$\kappa(G[S \cup \{u, v\} \cup Q''_{sub}], u, v) \leq |P \cap S| = |S| - |Q \cap S|. \tag{10}$$

Therefore, we can include $\{u, v\}$ together with at most $s - |Q \cap S| - 2$ vertices from $Q \cap C \setminus \{u, v\}$ without violating the LVC-based condition. This can be verified similar to Scenario 2.

Refer to the definitions in Equation (5) where $x < y$ still holds in this case. We define the ordering as below.

$$\langle v, u, v_3, v_4, ..., v_y, v_{y+1}, ..., v_{|C|} \rangle, \tag{11}$$

where $v_3, ..., v_y$ are vertices from the vertex set $Q \cap C \setminus \{v, u\}$ in any order and $v_{y+1}, ..., v_{|C|}$ are vertices from $C \setminus Q$ in any order. First, if $x < 2$, we can deduce that $G[S \cup \{v, u\}]$ is not an s-bundle since (1) $P \cap S$ is a u-v vertex cut of $G[S \cup \{v, u\}]$ and (2) thus $\kappa(G[S \cup \{v, u\}], u, v) \leq |P \cap S| = |S| - |Q \cap S| < |S| + 2 - s$. Hence, branches $B_3, ..., B_{|C|+1}$ can be pruned. Second, if $x \geq 2$, similar to Scenario 1, branches $B_{x+2}, ..., B_{|C|+1}$ can be pruned. In summary, we only need to keep the first $\max\{2, x + 1\}$ branches.

Summary. We call the Sym-BK branching with the LVC-based vertex ordering *LVC-based Sym-BK branching*. Clearly, it would create at most $x+1$ (which is $s+1$ in the worst case based on Equation (5)) sub-branches. Therefore, with LVC-based Sym-BK branching, our algorithm `SymBD-L` improves the worst-case time complexity to $O^*(\lambda_s^n)$ (details will be discussed in Section 4.3).

4.2 Relaxed-Local-Vertex-Connectivity-based Vertex ordering

Since the local vertex connectivity of any two vertices in $g = G[S \cup C]$ can be computed in $O(|V(g)|^{0.5}|E(g)|)$ using the maximum flow algorithm in [22], the time of obtaining the LVC-based vertex ordering for a branch (S, C) is $O(|V(g)|^{2.5}|E(g)|)$ [53]. This complexity is dominated by (1) that of finding a vertex pair $\langle u, v \rangle$ such that its local vertex connectivity is smaller than $|S \cup C| - s$ and (2) that of computing the minimum u-v vertex cut in g. Clearly, though the LVC-based vertex ordering would help to produce fewer branches, computing the ordering itself incurs significant time costs. Motivated by this, we seek to find a vertex ordering that *can be computed efficiently* and *achieves a similar performance as the LVC-based vertex ordering for branching*. Below, we elaborate on the details.

Consider a current branch $B = (S, C)$. Recall that the LVC-based vertex ordering relies on the partition $P \cup Q$ satisfying the Equation (3). We observe that it is stronger than necessary to require P to be the *minimum* u-v vertex cut in $G[S \cup C]$. In fact, to guarantee the worst-case time complexity, it would be sufficient as long as P is a u-v vertex cut in $G[S \cup C]$ satisfying Equation (3), as Scenarios 1-3 can be verified in the same way. Moreover, we note that such a u-v vertex cut is usually easier to obtain than the minimum one. Given this, our idea is to use a u-v vertex cut in $G[S \cup C]$ with the size less than $|S \cup C| - s$ for computing the vertex ordering.

Let $v_{\min}$ be the vertex with the smallest degree in $G[S \cup C]$, formally,

$$v_{\min} = \arg \min_{v \in S \cup C} d(v, S \cup C). \tag{12}$$

Based on the vertex $v_{\min}$, we can partition $S \cup C$ into two disjoint vertex sets: the first one denoted by P^* corresponds to the set of $v_{\min}$'s neighbours in $G[S \cup C]$ and the second one denoted by Q^* corresponds to the remaining vertices, i.e., set of $v_{\min}$'s non-neighbours in $G[S \cup C]$. Formally, we have

$$P^* = N(v_{\min}, S \cup C) \text{ and } Q^* = S \cup C - N(v_{\min}, S \cup C). \tag{13}$$

Clearly, for a vertex u in Q^*, P^* is a u-$v_{\min}$ vertex cut in $G[S \cup C]$ since after the removal of P^*, the remaining graph $G[Q^*]$ has two vertices u and $v_{\min}$ disconnected (note that $v_{\min}$ is in Q^* as it is not adjacent to itself). In summary, we get the following lemma.

Lemma 4.3. *Let $v_{\min}$ be the vertex with the smallest degree and $P^* \cup Q^*$ be a partition on a branch (S, C). For a vertex u in $Q^* \setminus \{v_{\min}\}$, P^* is a u-$v_{\min}$ vertex cut in $G[S \cup C]$.*

By the above lemma, if the conditions in Equation 3, i.e., $|P^*| < |S|-s$ and $|Q^*| > s$ (or equivalently, $d(v_{\min}, S \cup C) < |S| - s$), are satisfied, the partition $P^* \cup Q^*$ can be used to obtain the LVC-based vertex ordering similarly. Specifically, it has the following scenarios.

- **Scenario A:** $v_{\min} \in S$ and $Q^* \cap S \neq \emptyset$. Let u be an arbitrary vertex in $Q^* \cap S \setminus \{v_{\min}\}$. By Lemma 4.3, P^* is a u-$v_{\min}$ vertex cut in $G[S \cup C]$. Thus, we can derive the vertex ordering, i.e., Equation (7), by following the similar procedures as Scenario 1.
- **Scenario B:** $v_{\min} \in S$ and $Q^* \cap S = \emptyset$. Let u be an arbitrary vertex in $Q^* \setminus \{v_{\min}\}$. Similarly, we can derive the vertex ordering, i.e., Equation (9), by following the similar procedures as Scenario 2.
- **Scenario C:** $v_{\min} \in C$. Let u be an arbitrary vertex in $Q^* \cap C \setminus \{v_{\min}\}$. Note that $Q^* \cap C \setminus \{v_{\min}\}$ is non-empty since otherwise $G[S \cup \{v_{\min}\}]$ cannot be an s-bundle and $v_{\min}$ should be refined. Similarly, we can derive the vertex ordering, i.e., Equation (11), by following the similar procedures as Scenario 3.

Summary. We refer to the resulting ordering as the *relaxed local-vertex-connectivity based (or RLVC-based) vertex ordering*, as it is based on a vertex cut P^* instead of the minimum one (that corresponds to the local vertex connectivity). Clearly, it can be obtained efficiently since there is no need to compute the local vertex connectivity and the minimum vertex cut. Specifically, the time cost is $O(|S \cup C|)$, which is dominated by (1) that of finding the vertex $v_{\min}$ and (2) that of partitioning the branch. However, one potential issue is that the RLVC-based vertex ordering fails to exist when all vertices in $S \cup C$ have the degree at least $|S| - s$, i.e., $d(v_{\min}, S \cup C) \geq |S| - s$ since Equation (3) would not hold for P^* and Q^* as discussed before. Hence, we will show a new BB algorithm, namely SymBD-H, which uses the RLVC-based vertex ordering (if possible) and the LVC-based vertex ordering (otherwise) for the Sym-BK branching.

Remark. We remark that our RLVC-based vertex ordering is similar to the existing degree-based vertex ordering [30, 66] in the following aspects. First, they are both based on a vertex with the smallest degree in $G[S \cup C]$, called branching vertex. Second, they cannot be solely applied for Sym-BK branching for finding the largest s-bundle as such a branching vertex may fail to exist. Third, they both put the set of non-neighbours of the branching vertex before other vertices in the ordering and thus would produce at most $s + 1$ branches in the worst case.

4.3 SymBD-L and SymBD-H: Summary and Analysis

Based on the newly proposed LVC-based and RLVC-based vertex orderings, we develop two new BB algorithms, namely SymBD-L and SymBD-H. We present the pseudocode of SymBD-H in Algorithm 2. The pseudocode of SymBD-L is omitted for the simplicity since it differs with SymBD-H only in the vertex ordering as discussed before.

We note that SymBD-L and SymBD-H differ with the state-of-the-art BinSymBD in three aspects. First, BinSymBD adopts two different branching strategies, i.e., the Sym-BK branching and binary branching, while the latter would produce $O(2^n)$ branches in the worst case when solving the problem. Both SymBD-L and SymBD-H solely adopt the Sym-BK branching which tends to produce fewer branches (Lines 7-16). Second, they use different vertex orderings for the Sym-BK branching. Specifically, BinSymBD adopts the degree-based vertex ordering, which may fail to exist; SymBD-L adopts the newly proposed LVC-based vertex ordering; SymBD-H uses the hybrid vertex ordering which combines the LVC-based vertex ordering (Lines 11-16) and RLVC-based vertex ordering (Lines 7-10), where the latter can be obtained faster than the former. Third, both SymBD-L and SymBD-H employ the following upper bounds for pruning unpromising branches (Line 5). Consider a current branch $B = (S, C)$. The idea is that if the upper bound (of the size of the s-bundle to be

Algorithm 2: A new branch-and-bound method: SymBD-H

Input: A graph $G = (V, E)$ and an integer s
Output: The maximum s-bundle g^*

$g^* \leftarrow \emptyset$; SymBD-H_Rec$(\emptyset, V)$;
return g^*;

Procedure: SymBD-H_Rec(S, C)
if $G[S \cup C]$ *is an* s*-bundle* **then**
  $g^* \leftarrow G[S \cup C]$ if $|V(g^*)| < |S \cup C|$; **return**;
Apply pruning techniques for refining the candidate set;
$v^* \leftarrow \arg\min_{v \in S \cup C} d(v, S \cup C)$;
if $d(v^*, S \cup C) < |S \cup C| - s$ **then**
  /* RLVC-based Sym-BK branching */
  Create branches $\{B_1, B_2, ...\}$ via RLVC-based Sym-BK branching (Section 4.2);
  foreach *branch* $B_i = (S_i, C_i)$ *in* $\{B_1, B_2, ...\}$ **do**
    if $G[S_i]$ *is an* s*-bundle* **then** SymBD-H_Rec(S_i, C_i);
else
  /* LVC-based Sym-BK branching */
  $\langle u, v \rangle \leftarrow$ a vertex pair s.t. $\kappa(G[S \cup C], u, v) < |S \cup C| - s$;
  Partition $S \cup C$ into P and Q;
  Create branches $\{B_1, B_2, ...\}$ via LVC-based Sym-BK branching based on Equations (7), (9) and (11);
  foreach *branch* $B_i = (S_i, C_i)$ *in* $\{B_1, B_2, ...\}$ **do**
    if $G[S_i]$ *is an* s*-bundle* **then** SymBD-H_Rec(S_i, C_i);

found in B) is no greater than the largest s-bundle seen so far, we can prune the branch safely. Below, we elaborate on the details (where proofs can be found in the technical report [1]).

UB1 (Degree Bound [66]). The degree bound in the branch (S, C) is calculated as $\min_{v \in S} d(v, G[S \cup C]) + s$.
UB2 (Connectivity Bound). The connectivity bound in the branch (S, C) is calculated as $\min_{u,v \in S} \kappa(G[S \cup C], u, v) + s$.
UB3 (Connectivity-Partition Bound, CP-bound). Consider a current branch (S, C). We consider all the vertex pairs in S, resulting in a total of $p = \frac{|S|(|S|-1)}{2}$ pairs. These vertex pairs are ordered arbitrarily, which form a sequence $\{vp_1, ..., vp_p\}$. For each vertex pair vp_i, we partition $S \cup C$ into two disjoint subsets P_i and Q_i, where P_i corresponds to a minimum vertex cut for vp_i in $G[S \cup C]$, and $Q_i = S \cup C \setminus P_i$. The CP-bound then partitions C into $p + 1$ subsets, i.e., $\Pi = \{\Pi_0, \Pi_1, ..., \Pi_p\}$ and $\bigcup_{i=0}^{p} \Pi_i = C$. The partition Π_0 includes the set of vertices in C that also belong to P_i for all $1 \le i \le p$, i.e., $\Pi_0 = \{u \in C \mid u \in P_i, \forall 1 \le i \le p\}$. Moreover, a vertex $u \in C$ can be included to partition Π_i if $u \in Q_i$. Note that a vertex can be included in different partitions as it may belong to different Q_i, thus multiple feasible partitioning methods exist. Finally, the CP-bound is calculated as $|S| + |\Pi_0| + \sum_{i=1}^{p} \min\{s - \delta_i, |\Pi_i|\}$, where $\delta_i = |S \cap Q_i|$ denotes the number of vertices in S that also belong to Q_i.

Worst-case time complexity. The worst-case time complexity of both SymBD-L and SymBD-H achieves better asymptotic performance than that of the state-of-the-arts following BinSymBD, which we show in the following theorem.

Theorem 4.4. *Given a graph G and a positive integer s,* SymBD-L *and* SymBD-H *find the largest s-bundle in $O(n^{2.5}m\lambda_s^n)$, where $\lambda_s < 2$ is the largest real root of the equation $x^{s+2} - 2x^{s+1} + 1 = 0$. For example, when $s = 1, 2,$ and 3, $\lambda_s = 1.6181, 1.8393,$ and 1.9276, respectively.*

Proof Sketch. We show the worst-case time complexity of SymBD-L and SymBD-H by 1) each recursion of them runs in the same polynomial time, namely $O(n^{2.5}m)$; 2) the number of sub-branches generated in each branching process is at most $s + 1$ for both of them, such that we can derive the same recurrence equation $x^{s+2} - 2x^{s+1} + 1 = 0$. In particular, the time complexity of SymBD-L is dominated by Scenario 3 of the LVC-based vertex ordering in Section 4.1. For SymBD-H, we know that it utilizes two different vertex orderings, namely the LVC-based and RLVC-based vertex orderings. We remark that the derivation of the recurrence equation for LVC-based part of SymBD-H is the same as that of SymBD-L. Further, as shown in Section 4.2, the three scenarios of the RLVC-based vertex ordering can be respectively transformed into the three scenarios of the LVC-based vertex ordering. This allows us to apply the same analytical approach to derive the recurrence equation for the RLVC-based part of SymBD-H, which is also the same as that of SymBD-L. We defer the detailed proof to Appendix of [1]. □

5 Other Efficiency Boosting Techniques

Though SymBD-H achieves the improved worst-case time complexity, it still suffers from efficiency and scalability issues in practice. To improve, we introduce a practically efficient algorithm called SymBD which is built upon SymBD-H. Specifically, we present the whole algorithm in Section 5.1 and show the details of a heuristic procedure for finding a large s-bundle in Section 5.2.

5.1 SymBD: A Practically Improved Algorithm

We summarize our algorithm SymBD in Algorithm 3, which involves two stages. Below, we elaborate on the details.

Stage-I: refining the input graph. This step relies on a heuristic DegenPro for finding a large initial s-bundle g^* (Line 1). The found s-bundle g^* severs as a lower bound of the largest one, i.e., $lb = |V(g^*)|$. It is used to refine the input graph (Line 2) by removing those unpromising vertices and edges that cannot appear in any s-bundle larger than lb via a series of reduction rules as below.

RR1 (Core-based Reduction Rule) [66]. A vertex can be removed from G if its degree is less than $lb + 1 - s$.

RR2 (Truss-based Reduction Rule) [30]. An edge can be removed from G if it participates in fewer than $lb + 1 - 2s$ triangles.

Clearly, the larger the size of g^* is, the more vertices and edges we can prune. The details of DegenPro for finding an s-bundle as larger as possible will be discussed in the following Section 5.2.

Stage-II: divide-and-conquer strategy. The divide-and-conquer strategy has been widely used for finding cohesive subgraphs on big graphs [11, 12, 58, 59]. The idea is to divide the problem of finding the largest s-bundle in the input graph into several sub-problems, each of which seeks to find the largest s-bundle in a much smaller subgraph (Lines 3-7). Specifically, given an ordering $\langle v_1, v_2, ..., v_n \rangle$, it divides the problem into n sub-problems. The i^{th} one seeks to find the largest s-bundle that (1) must include v_i and (2) must exclude $\{v_1, v_2, ..., v_{i-1}\}$, which can be solved by invoking SymBD-H on a smaller subgraph $G_i = G[\{v_i, v_{i+1}, ..., v_n\}]$ via starting from the branch (S, C) with $S = \{v_i\}$ and $C = V(G_i) \backslash \{v_i\}$. It is no hard to verify that the largest s-bundle in the input graph is the largest one among all those found by solving the sub-problems.

Algorithm 3: SymBD

Input: A graph $G = (V, E)$ and an integer s
Output: The maximum s-bundle g^*

```
/* Stage-I: refine the input graph                                    */
```
1 $g^* \leftarrow$ DegenPro(G, s) ; // g^* is maintained globally
2 Refine G based on g^* via **RR1** and **RR2** exhaustively;
```
/* Stage-II: divide-and-conquer strategy                              */
```
3 Let $(v_1, \ldots, v_n)$ be the degeneracy ordering of G;
4 **for** $i \leftarrow 1$ **to** n **do**
5 $\quad G_i \leftarrow G[\{v_i, v_{i+1}, ..., v_n\}]$;
6 $\quad$ Refine G_i via **RR1** and **RR3** sequentially based on v_i;
7 $\quad$ Invoke SymBD-H_Rec$(\{v_i\}, V(G_i) \setminus \{v_i\})$ on G_i;
8 **if** $|V(g^*)| < 2s - 2$ **then** SymBD-H(G, s);
9 **return** g^*;

We note that subgraph G_i can be refined via **RR1** and the following reduction rule **RR3**. Specifically, it is based on the fact that an s-bundle with at least $2s - 1$ vertices has the diameter at most 2 [66]. Hence, for any s-bundle with the size larger than $2s - 2$ to be found in G_i, all vertices inside are v_i's 2-hop neighbours. Thus, we can refine G_i by removing those vertices that are not 2-hop neighbours of v_i. In summary, we give the following reduction.

RR3 (Diameter-based Reduction Rule) [66]. A vertex v in G_i $(1 \leq i \leq n)$ can be pruned if v is not a 2-hop neighbour of v_i in G_i.

We remark that (1) following the existing studies [12, 59], we use the degeneracy ordering (which can be obtained in $O(n + m)$) for dividing the problem (Line 3). With the degeneracy ordering, the number of vertices in G_i can be bounded by the degeneracy of G, denoted by $O(\delta(G)d)$ where d is the maximum degree of a vertex in G; and (2) the divide-and-conquer strategy may fail to solve the problem when the largest s-bundle is smaller than $2s - 2$. In this case, we directly invoke SymBD-H on the refined graph G (Lines 8). In addition, we clarify that **RR1** - **RR3** are existing reductions that are used to reduce the size of input graph in the literature. However, in SymBD, **RR1** and **RR3** are also used to boost the performance of the newly designed divide-and-conquer strategy (Lines 3-7, Algorithm 3) by reducing the size of produced subgraphs G_i (based on the property that all s-bundles found in G_i must include vertex v_i).

Time complexity analysis. The time cost of SymBD is dominated by that of invoking SymBD-H $O(n)$ times at Lines 4-7 (if the largest s-bundle is at least $2s - 2$) or that of invoking SymDB-H directly on G (otherwise). The latter is bounded by $O(n^{2.5} m \lambda_s^n)$ based on Theorem 4.4. In the former case, we note that (1) the time complexity of newly introduced DegenPro is bounded by $O(nm)$, which will be analyzed in the following section and (2) the number of vertices in a graph G_i is bounded by $O(\delta(G)d)$. Hence, the time complexity is bounded by $O((\delta(G)d)^{2.5} m \lambda_s^{\delta(G)d})$.

5.2 A Large Heuristic Solution

We introduce our heuristic method DegenPro for obtaining a large s-bundle in an efficient manner. DegenPro is detailed in Algorithm 4. In Line 1, we first initialize h which keeps track of the largest s-bundle found so far. Then, DegenPro conducts three heuristic steps, namely *Step-1: degeneracy-ordering heuristic*, *Step-2: one-hop subgraph heuristic*, and *Step-3: two-hop subgraph heuristic*. Step 1 aims to *quickly* obtain an s-bundle based on the degeneracy-ordering to facilitate graph reductions

Algorithm 4: DegenPro

Input: A graph $G = (V, E)$ and an integer s
Output: A large initial s-bundle h

1 $h \leftarrow \emptyset$; // the current largest s-bundle globally
/* Step-1: Degeneracy-ordering heuristic. */
2 $h \leftarrow \texttt{Degen}(G, s)$; Perform graph reductions based on h;
/* Step-2: One-hop subgraph heuristic. */
3 $h \leftarrow \texttt{Degen2}(G, s, 1)$; Perform graph reductions based on h;
/* Step-3: Two-hop subgraph heuristic. */
4 **return** $\texttt{Degen2}(G, s, 2)$;

Procedure: $\texttt{Degen}(G, s)$
5 Let $(v_1, ..., v_n)$ be the degeneracy ordering of G;
6 **for** $i \leftarrow 1$ **to** n **do**
7 **if** $d(v_i, V(G)) \geq n - i - \frac{s}{2}$ **and** $n - i + 1 > |V(h)|$ **then**
8 $h \leftarrow G[V(G) \setminus \{v_1, \ldots, v_{i-1}\}]$;
9 Remove v_i from G and update G;
10 **return** h;

Procedure: Procedure $\texttt{Degen2}(G, s, k)$
11 Let $(v_1, \ldots, v_n)$ be the degeneracy ordering of G;
12 **for** $i \leftarrow 1$ **to** n **do**
13 $g \leftarrow$ the subgraph of G induced by $N^+_{k\text{-hop}}(v_i)$;
14 $h \leftarrow \texttt{Degen}(g, s)$;
15 **return** h;

(which we directly utilize CTCP [11] in DegenPro). Steps 2 and 3 both aim to find a larger s-bundle by considering several subgraphs (1-hop or 2-hop) specified by particular vertices. However, Step 2 serves as an intermediate step, aiming to find a larger solution and thereby facilitating further reductions of graph before finding the final heuristic solution. The details of DegenPro are shown as follows.

- **Step-1: degeneracy-ordering heuristic**. Step 1 obtains an s-bundle via Degen (Lines 5-10) and performs graph reductions (Line 2). Specifically, Degen computes the degeneracy ordering (Line 5) and iteratively checks whether the current graph is a larger s-bundle (Lines 6-8) after removing v_i in each iteration. Note that, to support a fast verification of s-bundle, we only check whether $d(v_i, V(G)) \geq n - i - \frac{s}{2}$ for the i-th iteration (Line 7) and update h when the current graph is a larger s-bundle (Line 8). The correctness of the fast verification is because for any two non-adjacent vertices u and v, the local vertex connectivity $\kappa(G, u, v)$ is at least $d(u, V(G)) + d(v, V(G)) + 2 - |V(G)|$ [49].
- **Step-2: one-hop heuristic**. DegenPro invokes $\texttt{Degen2}(G, s, 1)$ with a focus on the 1-hop neighbour set and conducts graph reductions in Line 3. In particular, $\texttt{Degen2}(G, s, 1)$ computes the degeneracy ordering of the graph (Line 11) and iteratively calls Degen using the subgraph induced by $N^+_{1\text{-hop}}(v_i)$ (Lines 13-14). For a vertex v_i in the degeneracy ordering of G, we use $N^+_{k\text{-hop}}(v_i)$ to denote v_i and v_i's higher-ranked neighbors within k-hop in G per the degeneracy ordering, i.e., $N^+_{k\text{-hop}}(v_i) = \{v_i, \ldots, v_n\} \cap \{u \in V | dist(v_i, u) \leq k\}$, where $dist(u, v)$ represents the shortest distance between vertices u and v in G and we have $dist(u, u) = 0$.

- **Step-3: two-hop heuristic**. Compared with Step 2, Step 3 instead considers the two-hop neighbour set $N^{+}_{2\text{-hop}}(v_i)$ for invoking Degen in Line 14.

Time complexity analysis. First, we use CTCP ([11]) in Lines 2 and 3 for graph reductions, which costs $O(\delta(G)\cdot|E|)$. Second, it is clear that the complexity of Degen (Lines 5-10) is $O(|V|+|E|)$, which is the same as that of computing the degeneracy ordering. Note that Degen is invoked $O(|V|)$ times in Steps 1-3. In summary, the time complexity of DegenPro is $O(\delta(G)\cdot|E|+|V|\cdot(|V|+|E|)) = O(|V||E|)$.

Remark. Our SymBD can be easily adapted to discover multiple communities by finding the top-K maximal s-bundles (where K is a user-specific integer). In particular, there are two approaches commonly used in the literature. First, SymBD can be adapted to find top-K non-overlapping maximal s-bundles, i.e., any two identified s-bundles do not share any vertex. Specifically, we iteratively find the largest s-bundle on the current graph (by invoking SymBD) and then remove it from the current graph, until K s-bundles are returned or the graph becomes empty. Clearly, the worst-case time complexity is K times that of SymBD. Second, SymBD can be adapted to find top-K overlapping maximal s-bundles. In general, we need to modify SymBD to enumerate all large maximal s-bundles and store the set of K currently found largest maximal s-bundles. Besides, we need to (1) change the lower bound lb (used in SymBD) to be the size of the smallest one among the currently found top-K maximal s-bundles, and (2) modify our heuristic algorithm, DegenPro, to find the top-K largest heuristic solutions. However, one potential limitation of this adaptation is that the parameters s and K need to be specified by users in real-world applications, which requires domain knowledge.

6 Experiments

In this section, we conduct extensive experiments to evaluate the practical performance of our SymBD[2]. To this end, we compare with two baselines as below.

- MSB[3]: the state-of-the-art for finding the largest s-bundle [66].
- CMBK[4]: the baseline adapted from the solver of enumerating all maximal s-bundles [30]. Note that the largest s-bundle is the largest one among all maximal s-bundles. Hence, we adapt it for solving the problem by invoking the solver on the input graph and then returning the largest maximal s-bundle.

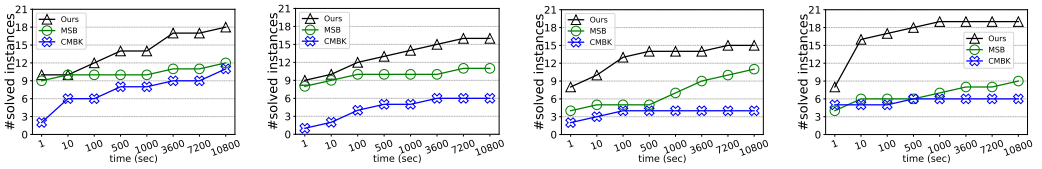
Following the existing studies on finding s-bundle [30, 66], we do not compare our algorithms with any clique discovery solution in the literature since clique (or 1-bundle) enjoys one unique property, i.e., any vertex is adjacent to all others, which is used to design efficient solutions for finding cliques.

Setup. All algorithms are implemented in C++, compiled with -O3, and run on a machine with an Intel(R) Xeon(R) Platinum 8358P CPU @ 2.60GHz and 256GB main memory running Ubuntu 20.04.6. We set the time limit as 3 hours (i.e., 10800s) and measure the total processing time of an algorithm. Note that we use **OOT** (Out of Time) to denote an algorithm cannot solve within a three-hour limit. Moreover, we consider different values of s for MSBP with $s = [2, 15]$ (and defer some results to Appendix of the technical report [1]).

Datasets. We evaluate the algorithms on three collections of graphs that are widely used in previous studies. We remark that all the graphs used in the previous work [66] are covered.

- The **real-world graphs** collection.[5] This dataset contains 500 (real-world) graphs from Network Repository with up to 5.87×10^7 vertices, including biological networks (36), dynamic networks

[2]The source code of SymBD is released at https://github.com/liubufan1998/Maximum-sbundle-computation
[3]https://github.com/joey001/max-s-bundle
[4]The source code is provided by the authors in [30].
[5]https://networkrepository.com/index.php



(a) $s = 3$ (b) $s = 5$ (c) $s = 10$ (d) $s = 15$

Fig. 3. Number of solved instances on 2nd DIMACS graphs

(85), interaction networks (29), labeled networks (104), road networks (15), scientific computing (11), social networks (75), facebook (114), and web networks (31).

- The **2nd DIMACS graphs** collection.[6] This collection contains 80 dense graphs with up to 4,000 vertices from the 2nd DIMACS implementation challenge.
- The **10th DIMACS graphs** collection.[7] This collection contains 84 graphs with up to 5.09×10^7 vertices from the 10th DIMACS implementation challenge.

For better comparisons, we also select 40 representative graphs from the three collections and report the statistics in Table 1, where the graph density is $\frac{2m}{n(n-1)}$ and $|g_5^*|$ denotes the size of the maximum 5-bundle. We observe that none of these identified maximum 5-bundles in 40 representative graphs are cliques. The criteria of selecting the representative graphs are follows. First, we consider large graphs with more than 10^6 vertices, except that the 2nd DIMACS collection do not have graphs of this size. Second, we choose 10 graphs (G1-G10) from the 2nd DIMACS graphs, 10 graphs (G11-G20) from the 10th DIMACS graphs, and 20 graphs (G21-G40) from the real-world graphs, excluding those graphs that can be solved within 5 seconds for all $s \in [2, 15]$ by all solvers or cannot be solved within 3 hours for each $s \in [2, 15]$ by any solver.

6.1 Comparison with Baselines

We first evaluate the efficiency of SymBD against the existing state-of-the-art algorithms MSB and CMBK by comparing the number of solved instances within a specified time limit. The results for $s = 3, 5, 10, 15$ on the 2nd DIMACS graphs, 10th DIMACS graphs, and real-world graphs are reported in Figures 3, 4, and 5, respectively. We can observe that our proposed SymBD outperforms both MSB and CMBK across all values of s in three datasets. The advantage of SymBD is more obvious on the 10th DIMACS graphs and real-world graphs. We observe that in these two datasets, the number of instances that SymBD can solve within 1 second exceeds the number solved by the other two algorithms within three hours. In particular, on the 10th DIMACS graphs, SymBD solves more than *twice* as many instances within 1 second as MSB and CMBK do in three hours. For example, when s=5, SymBD solves 47 instances within 1 second, while MSB and CMBK only solve 15 and 17 instances, respectively, within three hours. This significant practical efficiency improvements achieved by SymBD may be due to the following. First, SymBD incorporates a more efficient branching method, i.e., SymBD−H, which can quickly identify potential s-bundles once all vertices satisfy specific degree requirements. In contrast, BinSymBD, adopted by both MSB and CMBK, can only use a brute-force binary branching approach at this stage, potentially introducing several unnecessary branches. Second, SymBD benefits from an effective heuristic method, DegenPro, which is absent in MSB and CMBK. The heuristic method often provides SymBD with a large initial solution, enabling effective graph reductions that reduce the size of the graphs to be processed (where more details can be found in the technical report [1]). Third, with our new upper bounding techniques **UB2** and **UB3**,

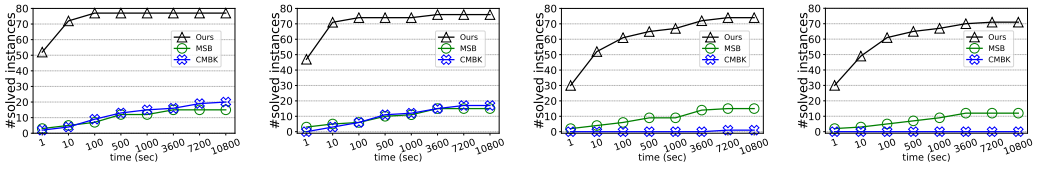
[6]http://archive.dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/
[7]https://sites.cc.gatech.edu/dimacs10/downloads.shtml

Table 1. Statistics of 40 representative graphs

ID	Graph	n	m	density	d	$\delta(G)$	$\lvert g_5^* \rvert$
G1	brock200-2	200	9K	$4.96 \cdot 10^{-1}$	114	84	OOT
G2	c-fat200-1	200	1K	$7.70 \cdot 10^{-2}$	17	14	12
G3	c-fat200-2	200	3K	$1.62 \cdot 10^{-1}$	34	32	24
G4	c-fat500-1	500	4K	$3.57 \cdot 10^{-2}$	20	17	14
G5	c-fat500-2	500	9K	$7.32 \cdot 10^{-2}$	38	35	26
G6	c-fat500-5	500	23K	$1.85 \cdot 10^{-1}$	95	92	64
G7	hamming6-2	64	1K	$9.04 \cdot 10^{-1}$	57	57	48
G8	hamming6-4	64	704	$3.49 \cdot 10^{-1}$	22	22	12
G9	johnson8-4-4	70	1K	$7.68 \cdot 10^{-1}$	53	53	28
G10	p-hat300-1	300	10K	$2.44 \cdot 10^{-1}$	132	49	OOT
G11	channel-500x100x100-b050	4M	42M	$2.24 \cdot 10^{-2}$	18	9	10
G12	delaunay_n21	2M	6M	$2.93 \cdot 10^{-3}$	23	4	9
G13	delaunay_n24	16M	50M	$2.35 \cdot 10^{-2}$	26	4	9
G14	hugetrace-00000	4M	6M	$5.47 \cdot 10^{-2}$	3	2	7
G15	hugetrace-00010	12M	18M	$2.37 \cdot 10^{-2}$	3	2	7
G16	hugetric-00020	7M	10M	$1.06 \cdot 10^{-2}$	3	2	7
G17	inf-great-britain_osm	7M	8M	$5.81 \cdot 10^{-2}$	8	3	7
G18	inf-netherlands_osm	2M	2M	$1.87 \cdot 10^{-2}$	7	3	7
G19	inf-road_central	14M	16M	$1.06 \cdot 10^{-2}$	8	3	7
G20	rgg_n_2_20_s0	1M	6M	$3.21 \cdot 10^{-3}$	36	17	20
G21	ia-wiki-user-edits-page	2M	5M	$5.68 \cdot 10^{-3}$	699K	66	26
G22	rec-amazon-ratings	2M	5M	$8.48 \cdot 10^{-3}$	12K	29	11
G23	road-roadNet-CA	1M	2M	$1.76 \cdot 10^{-3}$	12	3	8
G24	road-roadNet-PA	1M	1M	$1.84 \cdot 10^{-3}$	9	3	8
G25	soc-bitcoin	24M	86M	$2.85 \cdot 10^{-7}$	1M	325	OOT
G26	soc-flixster	2M	7M	$6.79 \cdot 10^{-3}$	1K	68	49
G27	soc-lastfm	1M	4M	$2.96 \cdot 10^{-3}$	5K	70	OOT
G28	soc-linkedin	6M	19M	$1.19 \cdot 10^{-1}$	869	11	15
G29	soc-ljournal-2008	5M	49M	$2.73 \cdot 10^{-1}$	19K	425	412
G30	soc-orkut	2M	106M	$9.56 \cdot 10^{-2}$	27K	230	OOT
G31	soc-orkut-dir	3M	117M	$6.09 \cdot 10^{-2}$	33K	253	76
G32	soc-pokec	1M	22M	$1.41 \cdot 10^{-2}$	14K	47	34
G33	soc-YouTube-ASU	1M	2M	$1.76 \cdot 10^{-3}$	28K	51	26
G34	soc-youtube-snap	1M	2M	$1.58 \cdot 10^{-3}$	28K	51	26
G35	socfb-konect	59M	92M	$8.20 \cdot 10^{-2}$	4K	16	13
G36	socfb-uci-uni	58M	92M	$7.28 \cdot 10^{-2}$	4K	16	13
G37	sx-stackoverflow	2M	28M	$1.60 \cdot 10^{-2}$	44K	198	OOT
G38	tech-as-skitter	1M	11M	$8.27 \cdot 10^{-3}$	35K	111	75
G39	web-wikipedia_link_fr	4M	104M	$1.73 \cdot 10^{-1}$	1M	817	339
G40	web-wikipedia_link_it	2M	86M	$5.88 \cdot 10^{-2}$	825K	894	881

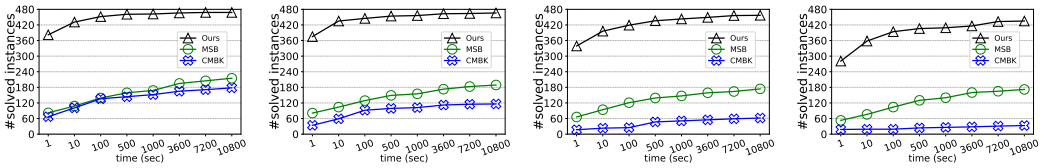
SymBD avoids exploring a vast amount of ineffective solution space, a common issue with MSB and CMBK (where more details can be found in the technical report [1]). Lastly, the framework of SymBD was designed using a divide-and-conquer approach, which effectively reduces a large graph into several smaller ones, accelerating the algorithmic ability to solve large graphs efficiently.

To get a more detailed comparison, we report their running time for $s = 5$ on the 40 representative graph instances in Table 2. In summary, our SymBD runs faster than MSB and CMBK on **38** out of the 40 graphs. Moreover, the running time of SymBD is on average three orders of magnitude faster than



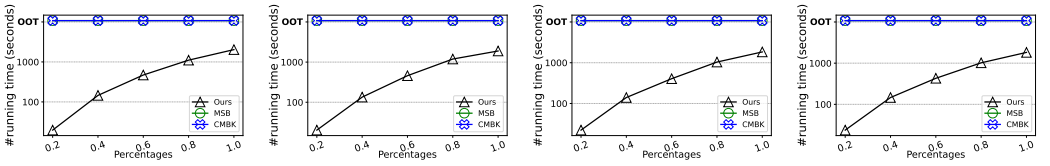
(a) $s = 3$ (b) $s = 5$ (c) $s = 10$ (d) $s = 15$

Fig. 4. Number of solved instances on 10th DIMACS graphs



(a) $s = 3$ (b) $s = 5$ (c) $s = 10$ (d) $s = 15$

Fig. 5. Number of solved instances on real-world graphs



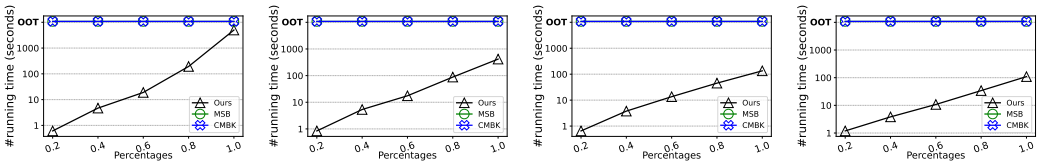
(a) $s = 3$ (b) $s = 5$ (c) $s = 10$ (d) $s = 15$

Fig. 6. Scalability test on the graph com-Friendster

that of MSB and CMBK, and at most five orders of magnitude faster (e.g., G12 and G18 in Table 2). An interesting observation is that MSB and CMBK are only capable of solving some instances in G1-G10, which are from the 2nd DIMACS dataset with a small number of vertices (e.g., no more than 500 vertices). However, in the other two datasets where graphs with more than 10^6 vertices, MSB and CMBK failed to solve any instance. The primary reason for this phenomenon is that both algorithms do not adopt a divide-and-conquer strategy to reduce the search space. Additionally, other potential reasons include the lack of efficient heuristic solutions and upper bounding techniques, such as those proposed in our SymBD. The above experimental results show that our proposed SymBD not only possesses the improved theoretical time complexity in solving the MSBP but also demonstrates the advanced practical performance.

Furthermore, to validate the scalability of the algorithms, we use two large-scale graphs, namely the *com-Friendster*[8] (66M vertices and 2B edges) and *soc-orkut-dir*[9] (3M vertices and 117M edges). Specifically, we randomly sample 20% - 100% of vertices from com-Friendster and soc-orkut-dir to generate 10 subgraphs. We compare our SymBD with the baselines in Figures 6 and 7, and observe the followings. First, our SymBD runs faster than all baselines and can handle the largest graph com-Friendster within the time limit while the baselines cannot. Second, the running time of SymBD increases smoothly as the scale of graph grows in most settings. Hence, the results demonstrate the scalability of our proposed SymBD.

[8]https://snap.stanford.edu/data/com-Friendster.html
[9]https://networkrepository.com/soc-orkut-dir.php



(a) $s = 3$ (b) $s = 5$ (c) $s = 10$ (d) $s = 15$

Fig. 7. Scalability test on the graph soc-orkut-dir

Table 2. Running time in seconds on 40 graphs with $s = 5$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**10.87**	OOT	OOT
G2	**0.02**	0.23	11.05	G22	**1.86**	OOT	OOT
G3	**0.00**	0.06	1897.56	G23	**0.23**	OOT	OOT
G4	**0.03**	0.62	77.36	G24	**0.17**	OOT	OOT
G5	**0.02**	0.56	OOT	G25	OOT	OOT	OOT
G6	**0.40**	0.49	OOT	G26	**31.16**	OOT	OOT
G7	**61.17**	6479.17	OOT	G27	**767.88**	OOT	OOT
G8	**0.12**	51.67	479.57	G28	**3.63**	OOT	OOT
G9	**891.54**	OOT	OOT	G29	**215.02**	OOT	OOT
G10	**4787.32**	OOT	OOT	G30	**5324.32**	OOT	OOT
G11	**39.64**	OOT	OOT	G31	**440.31**	OOT	OOT
G12	**0.56**	OOT	OOT	G32	**3.90**	OOT	OOT
G13	**4.41**	OOT	OOT	G33	**20.93**	OOT	OOT
G14	**2.29**	OOT	OOT	G34	**21.11**	OOT	OOT
G15	**8.85**	OOT	OOT	G35	**7.80**	OOT	OOT
G16	**6.82**	OOT	OOT	G36	**6.72**	OOT	OOT
G17	**1.17**	OOT	OOT	G37	**8112.97**	OOT	OOT
G18	**0.33**	OOT	OOT	G38	**4.46**	OOT	OOT
G19	**1.45**	OOT	OOT	G39	**20.68**	OOT	OOT
G20	**0.30**	OOT	OOT	G40	**959.84**	OOT	OOT

6.2 Ablation Studies

We conduct ablation studies to evaluate the effectiveness of our branching method proposed in Section 4 and efficiency boosting techniques proposed in Section 5.

Effectiveness of Branching Method. We study the effectiveness of our proposed branching method by comparing SymBD with two variants SymBD-L (Algorithm 3 using the branching method SymBD-L) and BinSymBD (Algorithm 3 using the branching method BinSymBD). The results are presented in Table 3, where '-L' and '-Bin' represent SymBD-L and BinSymBD, respectively. We can observe that when $s = 5$, our SymBD with SymBD-H achieves the minimum running time on **28** out of the 40 graphs. Specifically, on G10, G27, G30, and G37, our SymBD can find the result in the given time limit, while both SymBD-L and SymBD-H fail to finish within 3 hours. The primary reason for the poor performance of SymBD-L may be that it solely relies on the local vertex connectivity for branching, and calculating the local vertex connectivity incurs a high time cost. On the other hand, SymBD-H is more effective than BinSymBD in reducing the number of branches, as SymBD-H makes use of the concept of local vertex connectivity, which is an intrinsic property for s-bundle. The above result demonstrates the effectiveness of our proposed branching method SymBD-H.

Table 3. Running time in seconds of different branching methods on 40 graphs with $s = 5$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**10.87**	OOT	125.52
G2	0.02	0.02	**0.01**	G22	1.86	**1.58**	1.69
G3	**0.00**	**0.00**	0.01	G23	**0.23**	0.29	0.28
G4	0.03	**0.02**	0.03	G24	0.17	3.07	**0.16**
G5	0.02	0.02	**0.01**	G25	OOT	OOT	OOT
G6	**0.40**	0.45	0.64	G26	**31.16**	OOT	227.59
G7	**61.17**	3695.70	298.25	G27	**767.88**	OOT	OOT
G8	**0.12**	2.81	0.16	G28	**3.63**	3.65	3.71
G9	**891.54**	OOT	9341.65	G29	**215.02**	OOT	689.50
G10	**4787.32**	OOT	OOT	G30	**5324.32**	OOT	OOT
G11	**39.64**	40.25	40.79	G31	**440.31**	OOT	4303.70
G12	0.56	**0.44**	0.56	G32	**3.90**	9.95	5.68
G13	4.41	4.18	**4.13**	G33	**20.93**	OOT	69.85
G14	**2.29**	OOT	2.68	G34	**21.11**	OOT	69.54
G15	8.85	OOT	**8.45**	G35	7.80	7.69	**6.89**
G16	**6.82**	OOT	7.12	G36	**6.72**	6.95	6.97
G17	**1.17**	1.35	1.30	G37	**8112.97**	OOT	OOT
G18	**0.33**	0.45	**0.33**	G38	**4.46**	8494.52	14.74
G19	**1.45**	1.53	2.11	G39	**20.68**	22.33	23.79
G20	0.30	**0.29**	0.33	G40	**959.84**	1186.09	2010.78

Effectiveness of Heuristic Solutions. We compare SymBD with '-Heu' which represents SymBD without using the heuristic solution DegenPro in Section 5.2. The detailed results are provided in Table 4. Overall, our DegenPro enables SymBD to achieve the shortest running time in 23 out of 40 instances when s is 5. The advantage is more obvious in the 2nd DIMACS graphs and real-world graphs. Note that with the help of DegenPro, SymBD can solve G30, G31, and G39 in less than two hours, whereas '-Heu' fails to solve these instances within three hours. This clearly demonstrates the effectiveness of our proposed DegenPro. The major reason for the efficiency gains lies in DegenPro can potentially find a larger initial solution, which then allows for effective graph reductions of the input graph. The detailed experimental results and discussions related to the graph preprocessing are included in Appendix of [1].

Effectiveness of Upper Bounds. We also compare with a variant -UB, which is SymBD without our proposed **UB2** and **UB3** in Section 4.3. Due to space constraints, the detailed results and discussion are deferred to Appendix of [1]. Here we report our main findings as follows. First, SymBD achieves the shortest running time in 22 out of the 40 graphs when s is set to 5, and the advantage of SymBD expands when s increases. Second, our proposed upper bounding methods (i.e., **UB2** and **UB3**) are more effective on the 10th DIMACS and real-world datasets than on the 2nd DIMACS dataset. This is because the graphs in the 2nd DIMACS dataset are relatively dense, which constrains the ability of our proposed upper bounding methods to identify branches that can be pruned effectively.

6.3 Case Study

To assess the effectiveness of s-bundle, we compare with s-plex [11, 12, 55] and s-block [38, 39, 53] on a community search task. We use the Amazon graph[10] with 151K ground truth communities, where $n = 335$K, $m = 926$K, $d = 549$, and $\delta(G) = 6$. Specifically, the graph is based on the "*Customers Who Bought This Item Also Bought*" feature of the *Amazon* website. Each product corresponds

[10]https://snap.stanford.edu/data/com-Amazon.html

Table 4. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 5$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**10.87**	2918.86
G2	0.02	**0.01**	G22	**1.86**	261.44
G3	**0.00**	0.01	G23	0.23	**0.18**
G4	0.03	**0.02**	G24	0.17	**0.13**
G5	0.02	**0.01**	G25	OOT	OOT
G6	0.40	**0.39**	G26	**31.16**	32.49
G7	**61.17**	61.62	G27	**767.88**	957.93
G8	**0.12**	0.13	G28	3.63	**3.23**
G9	**891.54**	1027.31	G29	**215.02**	248.90
G10	**4787.32**	4870.54	G30	**5324.32**	OOT
G11	39.64	**36.57**	G31	**440.31**	OOT
G12	0.56	**0.44**	G32	**3.90**	6.01
G13	4.41	**3.32**	G33	**20.93**	49.18
G14	2.29	**1.78**	G34	**21.11**	49.68
G15	8.85	**6.16**	G35	**7.80**	10.89
G16	6.82	**6.05**	G36	**6.72**	11.66
G17	1.17	**0.84**	G37	**8112.97**	10629.18
G18	0.33	**0.21**	G38	**4.46**	182.96
G19	**1.45**	2.05	G39	**20.68**	OOT
G20	**0.30**	0.45	G40	**959.84**	1311.08

Table 5. Quality comparison: different cohesive models

model	**precision**	**recall**	**F1-score**	**density**
3-bundle	0.89	0.80	0.84	0.86
4-bundle	0.90	0.90	0.90	0.82
5-bundle	0.91	**1.00**	**0.95**	0.78
3-plex	**1.00**	0.14	0.24	0.83
4-plex	0.70	0.54	0.61	0.73
5-plex	**1.00**	0.79	0.88	0.71
4-block	-	-	-	-
5-block	0.33	0.21	0.26	0.00
6-block	0.97	0.70	0.81	**0.90**

to a vertex in the graph. If a product i is frequently co-purchased with a product j, the graph has an undirected edge between i and j. Each product category provided by *Amazon* defines a ground-truth community. To address the community search task, we find the maximum cohesive subgraph (e.g., maximum s-bundle) and return all vertices in the found subgraph as a community. For the three cohesive models, we explore different values of s. Then, we evaluate the quality of the found community by measuring the **precision**, **recall**, **F1-score**, and **density** in Table 5. We note that since the Amazon graph contains more than one ground-truth community, we adopt the approach in [38, 39] to compute the exact F1-score. More precisely, for each community discovered by the three subgraph models, we compute the F1-score with every ground-truth community in the Amazon graph and choose the largest one as the final F1-score of the discovered community. Additionally, the precision and recall reported in Table 5 are calculated based on the ground-truth community corresponding to the maximum F1-score.

From Table 5, we have the following observations. **First**, the s-bundle achieves the highest recall and F1-score, and it also has relatively high precision and density compared to the other two models.

This suggests that the communities identified by the maximum s-bundle are closer to the ground truth compared to those found by maximum s-plex and s-block. **Second**, the s-plex has the highest precision but relatively low recall, which results in a relatively low F1-score. Moreover, the s-plex has a lower density than the s-bundle which may be due to the fact that an s-bundle is also an s-plex. **Third**, the value of s greatly affects the quality of the s-block. The 6-block has the highest density, while the 5-block has the lowest density. As for the 4-block, we are unable to obtain the solution within 72 hours using the exact algorithm in [53].

Remark. The case study presented here is focused on a specific domain, i.e., Amazon, thus the observations might not simply generalize to other domains. Besides, there are some other clique relaxation models, including k-core[11], k-truss[12], and γ-quasi-clique, in the literature. Compared to s-bundle, the clique relaxation models of k-core and k-truss are *less cohesive* according to the definition but can be found efficiently in *polynomial time* (recall that MSBP is NP-hard). Besides, we note that a γ-quasi-clique (or, γ-QC) g requires that every vertex inside is adjacent to at least a proportional number of vertices, i.e., $\lceil(|V(g)|-1)\times\gamma\rceil$, and the larger a γ-QC, the more missing edges inside are allowed. Hence, γ-QC could be more cohesive and/or flexible than s-bundle, but it does not satisfy the hereditary property (i.e., a subgraph of a γ-QC can be a non-γ-QC) and thus finding maximum γ-quasi-cliques is time-consuming.

7 Related Work

s-bundle. The exploration of MSBP is still in its nascent stages. Pattillo et al. [44] were pioneers in introducing the concept of s-bundle, highlighting it as a novel clique relaxation model grounded in vertex connectivity. Subsequently, Gschwind et al. [28] introduced the problem of the maximum s-bundle and devised an algorithm within the general Russian Doll Search (RDS) framework. Zhou et al. [66] then introduced a branch-and-bound solution, namely MSB, that uses degree-based branching methods. Further, Hu et al. [30] proposed a Bron-Kerbosch-based method, CMBK, for enumerating all maximal s-bundles (referred to as k-RVCCs in their work). However, we remark that state-of-the-art solutions applicable to MSBP, namely MSB and CMBK, are unable to break the trivial time complexity of $O^*(2^n)$. In contrast, our proposed SymBD utilizes a new branching method that depends on local vertex connectivity, which theoretically improves the trivial time complexity and demonstrates the best practical performance compared to all existing algorithms.

s-block. The s-block is another clique relaxation model dependent on vertex connectivity. A graph G is said to be an *s-block*, or s-vertex-connected component, if the vertex connectivity of the graph is at least s, i.e., $\kappa(G)\geq s$, and the graph is maximal [43]. Veremyev et al. [48] studied the maximum s-block problem and proposed a general flow-based integer linear program approach to tackle problems related to vertex connectivity. There is a line of research focusing on the s-block enumeration problem. Sinkovits et al. [47] designed an algorithm tailored for small values of s, such as 2 or 3, while Wen et al. [53] demonstrated the existence of a polynomial-time algorithm for the s-block enumeration problem. Li et al. [39] proposed an approximation algorithm in a bottom-up manner, and then combined this algorithm with a top-down approach to obtain exact results [38]. Subsequently, Liu et al. [40] enhanced the precision of these approximation algorithms using the bottom-up methods. Nonetheless, MSBP is NP-hard unless $P=NP$ [37, 44], and existing algorithms for solving problems related to s-blocks cannot be directly adapted to solve MSBP.

Other cohesive structures. Instead of vertex connectivity, the s-edge-connected component (or s-ECC) is another connectivity-related model based on edge connectivity. In this model, an s-ECC is defined as a maximal subgraph that remains connected after the removal of any $s-1$

[11]The k-core of a graph G is the maximal induced subgraph g of G such that every vertex $u\in V(g)$ has degree $d(u,g)\geq k$.
[12]The k-truss of a graph G is the maximal subgraph g within G where every edge in g participates in at least $k-2$ triangles.

edges from the graph [9, 13, 61, 62, 64]. In addition, there are many other cohesive subgraphs, including but not limited to s-plex [11, 12, 25, 33, 34, 52, 55, 65], k-core [5, 16], k-truss [17, 31, 50], γ-quasi-clique [35, 45, 58, 63], s-defective clique [8, 15, 20, 26], and densest subgraph [42, 56]. Among these subgraph models, we note that the problems of finding *maximum* subgraph models are also NP-hard . For instance, the SOTA time complexity for the maximum s-plex problem is $O^*((\delta d)^{s+1}\alpha_s^\delta + \min\{\alpha_s^n, n^{2s-2}\})$ [12], for the maximum s-defective clique problem is α_s^n [18], and for the maximum γ-quasi-clique problem is $O^*(\beta_s^n)$ [58]. Note that $1 < \alpha_s < \beta_s < \lambda_s < 2$. Nonetheless, they are not applicable to the MSBP due to the distinct nature of the problems. Further, cohesive subgraphs have also been widely studied in other types of graphs, including bipartite graphs [14, 21, 41, 59, 59, 60], directed graphs [27], temporal graphs [6], and uncertain graphs [19]. For a comprehensive overview of the research on cohesive subgraphs, see the excellent books and surveys , e.g., [10, 23, 24, 32, 36].

8 Conclusion

In this paper, we advanced the state-of-the-art for MSBP in terms of both theoretical time complexity and practical performance. Specifically, we first developed a new branch-and-bound algorithm, SymBD, based on the concept of local vertex connectivity. We proved that SymBD improves the time complexity to $O^*(\lambda_s^n)$ where λ_s is strictly less than 2. To further boost the practical efficiency, we also proposed a novel heuristic method and two upper-bound-based reductions. Extensive empirical studies on 664 graphs demonstrated the practical superiority of SymBD over the existing algorithms. In the future, it is interesting to explore the parallel version of SymBD.

Acknowledgments

The research is supported in part by the National Natural Science Foundation of China (Grant No. 62102117), by the Shenzhen Science and Technology Program (Grant No. GXWD20231129111306002), and by the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2023A1515011188). This research is also supported by the Shenzhen Science and Technology Program under Grant No. GXWD20220817124827001 and No. JCYJ20210324132406016. This research is also supported in part by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 1 Award (RG77/21)). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

References

[1] Technical Report and Source Code. https://github.com/liubufan1998/Maximum-sbundle-computation/blob/master/Technical%20Report.pdf.

[2] James Abello, Mauricio GC Resende, and Sandra Sudarsky. 2002. Massive quasi-clique detection. In *Proceedings of the Latin American Symposium on Theoretical Informatics (LATIN)*. 598–612.

[3] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. 2016. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems* 55 (2016), 278–288.

[4] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. 2012. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proceedings of the VLDB Endowment* 5, 6 (2012), 574–585.

[5] Vladimir Batagelj and Matjaž Zaveršnik. 2003. An $O(m)$ algorithm for cores decomposition of networks. *CoRR* cs.DS/0310049 (2003).

[6] Matthias Bentert, Anne-Sophie Himmel, Hendrik Molter, Marco Morik, Rolf Niedermeier, and René Saitenmacher. 2019. Listing all maximal k-plexes in temporal graphs. *ACM J. Exp. Algorithmics* 24, 1 (2019), 1.13:1–1.13:27.

[7] Lijun Chang. 2019. Efficient maximum clique computation over large sparse graphs. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (SIGKDD)*. 529–538.

[8] Lijun Chang. 2023. Efficient maximum k-defective clique computation with improved time complexity. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 3 (2023), 1–26.

[9] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. 459–474.

[10] Lijun Chang and Lu Qin. 2018. *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer.

[11] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient maximum k-plex computation over large sparse graphs. *Proceedings of the VLDB Endowment* 16, 2 (2022), 127–139.

[12] Lijun Chang and Kai Yao. 2024. Maximum k-plex computation: Theory and practice. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 1 (2024), 1–26.

[13] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Xuemin Lin, Chengfei Liu, and Weifa Liang. 2013. Efficiently computing k-edge connected components via graph decomposition. In *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. 205–216.

[14] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2021. Efficient exact algorithms for maximum balanced biclique search in bipartite graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. 248–260.

[15] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. 2021. Computing Maximum k-Defective Cliques in Massive Graphs. *Computers & Operations Research* 127 (2021), 105131.

[16] James Cheng, Yiping Ke, Shumo Chu, and M Tamer Özsu. 2011. Efficient core decomposition in massive networks. In *Proceedings of the IEEE International Conference Data Engineering (ICDE)*. 51–62.

[17] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report* 16, 3.1 (2008).

[18] Qiangqiang Dai, Ronghua Li, Donghang Cui, and Guoren Wang. 2024. Theoretically and Practically Efficient Maximum Defective Clique Search. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 4 (2024), 1–27.

[19] 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 *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. 2034–2047.

[20] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, and Guoren Wang. 2023. Maximal defective clique enumeration. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 1 (2023), 1–26.

[21] Qiangqiang Dai, Rong-Hua Li, Xiaowei Ye, Meihao Liao, Weipeng Zhang, and Guoren Wang. 2023. Hereditary cohesive subgraphs enumeration on bipartite graphs: The power of pivot-based approaches. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 2 (2023), 1–26.

[22] Shimon Even and R Endre Tarjan. 1975. Network flow and testing graph connectivity. *SIAM journal on computing* 4, 4 (1975), 507–518.

[23] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29 (2020), 353–392.

[24] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions. In *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. 2829–2838.

[25] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. 2018. An exact algorithm for maximum k-plexes in massive graphs.. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1449–1455.

[26] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. 2022. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 10174–10183.

[27] Shuohao Gao, Kaiqiang Yu, Shengxin Liu, Cheng Long, and Zelong Qiu. 2024. On searching maximum directed (k, ℓ)-plex. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2570–2583.

[28] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. 2018. Maximum weight relaxed cliques and Russian Doll Search revisited. *Discrete Applied Mathematics* 234 (2018), 131–138.

[29] Guimu Guo, Da Yan, Lyuheng Yuan, Jalal Khalil, Cheng Long, Zhe Jiang, and Yang Zhou. 2022. Maximal directed quasi-clique mining. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 1900–1913.

[30] Shan Hu, Yi Zhou, Mingyu Xiao, Zhang-Hua Fu, and Zhipeng Lü. 2023. Listing maximal k-relaxed-vertex connected components from large graphs. *Information Sciences* 620 (2023), 67–83.

[31] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the ACM on Management of Data (SIGMOD)*. 1311–1322.

[32] Xin Huang, Laks V. S. Lakshmanan, and Jianliang Xu. 2019. *Community Search over Big Graphs*. Morgan & Claypool Publishers.

[33] Hua Jiang, Fusheng Xu, Zhifei Zheng, Bowen Wang, and Wei Zhou. 2023. A refined upper bound and inprocessing for the maximum k-plex problem. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 5613–5621.

[34] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. 2021. A new upper bound based on vertex partitioning for the maximum k-plex problem.. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1689–1696.

[35] Jalal Khalil, Da Yan, Guimu Guo, and Lyuheng Yuan. 2022. Parallel mining of large maximal quasi-cliques. *The VLDB Journal* 31, 4 (2022), 649–674.

[36] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. A Survey of Algorithms for Dense Subgraph Discovery. *Managing and Mining Graph Data* (2010), 303–336.

[37] John M. Lewis and Mihalis Yannakakis. 1980. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. System Sci.* 20, 2 (1980), 219–230.

[38] Yuan Li, Guoren Wang, Yuhai Zhao, Feida Zhu, and Yubao Wu. 2020. Towards k-vertex connected component discovery from large networks. *World Wide Web* 23 (2020), 799–830.

[39] Yuan Li, Yuhai Zhao, Guoren Wang, Feida Zhu, Yubao Wu, and Shengle Shi. 2017. Effective k-vertex connected component detection in large-scale networks. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. 404–421.

[40] Haoyu Liu, Yongcai Wang, Xiaojia Xu, and Deying Li. 2024. Bottom-up k-Vertex connected component enumeration by multiple expansion. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 3000–3012.

[41] Wensheng Luo, Kenli Li, Xu Zhou, Yunjun Gao, and Keqin Li. 2022. Maximum biplex search over bipartite graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 898–910.

[42] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. On directed densest subgraph discovery. *ACM Transactions on Database Systems (TODS)* 46, 4 (2021), 1–45.

[43] David W. Matula. 1978. k-blocks and ultrablocks in graphs. *Journal of Combinatorial Theory, Series B* 24, 1 (1978), 1–13.

[44] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2013. On clique relaxation models in network analysis. *European Journal of Operational Research* 226, 1 (2013), 9–18.

[45] Jian Pei, Daxin Jiang, and Aidong Zhang. 2005. On mining cross-graph quasi-cliques. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (SIGKDD)*. 228–238.

[46] Stephen B. Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology* 6, 1 (1978), 139–154.

[47] Robert S. Sinkovits, James Moody, B. Tolga Oztan, and Douglas R. White. 2016. Fast determination of structurally cohesive subgroups in large networks. *Journal of Computational Science* 17 (2016), 62–72.

[48] Alexander Veremyev, Oleg A. Prokopyev, Vladimir Boginski, and Eduardo L. Pasiliao. 2014. Finding maximum subgraphs with relatively large vertex connectivity. *European Journal of Operational Research* 239, 2 (2014), 349–362.

[49] Lutz Volkmann. 2008. On local connectivity of graphs. *Applied Mathematics Letters* 21, 1 (2008), 63–66.

[50] Jia Wang and James Cheng. 2012. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment* 5, 9 (2012), 812–823.

[51] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-clique listing: An edge-oriented branching strategy. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 1 (2024), 1–26.

[52] Zhengren Wang, Yi Zhou, Chunyu Luo, and Mingyu Xiao. 2023. A fast maximum k-plex algorithm parameterized by the degeneracy gap. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 5648–5656.

[53] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Ling Chen. 2019. Enumerating k-vertex connected components in large graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 52–63.

[54] Jingen Xiang, Cong Guo, and Ashraf Aboulnaga. 2013. Scalable maximum clique computation using MapReduce. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 74–85.

[55] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. 2017. A fast algorithm to compute maximum k-plexes in social network analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 919–925.

[56] Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao. 2024. Efficient and effective algorithms for densest subgraph discovery and maintenance. *The VLDB Journal* (2024).

[57] 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.

[58] Kaiqiang Yu and Cheng Long. 2023. Fast maximal quasi-clique enumeration: A pruning and branching co-design approach. *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)* 1, 3 (2023), 1–26.

[59] Kaiqiang Yu and Cheng Long. 2023. Maximum k-biplex search on bipartite graphs: A symmetric-BK branching approach. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 1 (2023), 1–26.

[60] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient algorithms for maximal k-biplex enumeration. In *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. 860–873.

[61] Ting Yu, Mengchi Liu, Zujie Ren, and Ji Zhang. 2023. A fast approximate method for k-edge connected component detection in graphs with high accuracy. *Information Sciences* 633 (2023), 384–409.

[62] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. I/O efficient ECC graph decomposition via graph reduction. *The VLDB Journal* 26, 2 (2017), 275–300.

[63] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. 2006. Coherent closed quasi-clique discovery from large dense graph databases. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (SIGKDD)*. 797–802.

[64] Rui Zhou, Chengfei Liu, Jeffrey Xu Yu, Weifa Liang, Baichen Chen, and Jianxin Li. 2012. Finding maximal k-edge-connected subgraphs from a large graph. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 480–491.

[65] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. 2021. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 12453–12460.

[66] Yi Zhou, Weibo Lin, Jin-Kao Hao, Mingyu Xiao, and Yan Jin. 2022. An effective branch-and-bound algorithm for the maximum s-bundle problem. *European Journal of Operational Research* 297, 1 (2022), 27–39.

A Omitted Details for Upper Bounds

Next, we will prove the correctness of **UB2** and **UB3** in sequence.

The correctness proof of **UB2**. The proof is mainly based on Lemma 4.1. Assume that the pair of vertices u and v has the smallest $\kappa(G[S \cup C], u, v)$ among all the pairs of vertices in S. If there exists a graph g containing S whose size exceeds $\kappa(G[S \cup C], u, v) + s$, then we have $\kappa(g, u, v) + s < |V(g)|$, thereby preventing g from being a qualified s-bundle according to Lemma 4.1. □

The correctness proof of **UB3** is as follows.

The correctness proof of **UB3**. The proof is due to the fact that any partition Π_i (for $i \geq 1$) can contribute at most $\min\{s - \delta_i, |\Pi_i|\}$ vertices to S, since otherwise the expanded S would violate the s-bundle requirement. More precisely, without loss of generality, suppose that $s - \delta_i < |\Pi_i|$. We try to include a set of vertices B into S where $|B| > s - \delta_i$ vertices. Then, for the corresponding vertex pair vp_i, we have $\kappa(S \cup B, u, v) < |S \cup B| - s$. According to Lemma 4.1, in this case, $S \cup B$ is no longer a qualified s-bundle. □

For the CP-bound (**UB3**), we observe that multiple feasible partitioning methods exist, meaning that a vertex can be included in different partitions. These different partitioning methods affect the quality of the upper bound we derive. To make the CP-bound tighter, we observe that if a partition Π_i already contains $s - \delta_i$ vertices, adding more vertices to Π_i does not decrease the value of $\min\{s - \delta_i, |\Pi_i|\}$; however, it might decrease the value of $\min\{s - \delta_j, |\Pi_j|\}$ for Π_j, as the number of vertices in Π_j might be decreased. Motivated by this, we propose a refined CP-bound using a specific partitioning method. First, we sort all vertex pairs in S based on their values of $s - \delta_i$ in non-decreasing order. Then, for each vertex v in C, we sequentially attempt to add v to the partitions based on the defined ordering. That is, when v encounters the **first** vertex pair vp_i with which it belongs to the Q_i of vp_i, we include v in the partition Π_i associated with vp_i. In such a way, we try to include as many vertices as possible into the partition Π_i with a small value of $s - \delta_i$, potentially resulting in a tighter upper bound.

B Proof of Theorem 4.4

Proof. We analyze the time complexity of SymBD-L and SymBD-H from two perspectives: 1) the time spent in each individual recursion, as well as 2) the total number of recursions. First, each recursion of SymBD-L and SymBD-H runs in polynomial time $O(|V(g)|^{2.5}|E(g)|)$. This is because the local vertex connectivity of any two vertices in $g = G[S \cup C]$ can be computed in $O(|V(g)|^{0.5}|E(g)|)$ by using the maximum flow algorithm in [22], and we need to perform at most $O(|V(g)|^2)$ times of maximum flow computations.

Next, we analyze the number of recursions of SymBD-L and SymBD-H. Note that the case of SymBD-H can be readily reduced to the case of SymBD-L, so we only provide a proof for SymBD-L and omit the proof for SymBD-H. Let $T(n)$ be the largest number of recursions where $n = |C|$. We have the following three scenarios.

Scenario 1: $u \in S$ and $v \in S$. In this case, we remove i vertices from C in $B_i (1 \leq i \leq x)$ and y vertices from C in B_{x+1} in the worst-case. Hence, we have

$$T_1(n) \leq \sum_{i=1}^{x} T_1(n - i) + T_1(n - y), \tag{14}$$

As discussed earlier, we have $x \leq s-2$ and $x < y$. It is easy to verify that we reach the maximum of $T_1(n)$ when $x = s-2$ and $y = s-1$. We thus have $T_1(n) \leq \sum_{i=1}^{s-2} T_1(n-i) + T_1(n-(s-1))$. By solving this linear recurrence, the worst-case running time is $O(|V|^{2.5}|E|\lambda_s^n)$ where λ_s is the largest positive real root of $x^s - 2x^{s-1} + 1 = 0$. The first few solutions to the equation are $\gamma_1 = 1$, $\gamma_2 = 1$, $\gamma_3 = 1.618$, $\gamma_4 = 1.839$ and $\gamma_5 = 1.928$ when $s = 1, 2, 3, 4$, and 5, respectively.

Scenario 2: $u \in S$ and $v \in C$ (or $v \in S$ and $u \in C$, the analysis is symmetric). We remove i vertices from C in $B_i (1 \leq i \leq x)$ and y vertices from C in B_{x+1} in the worst-case. Hence, we have

$$T_2(n) \leq \sum_{i=1}^{x} T_2(n-i) + T_2(n-y), \tag{15}$$

As discussed earlier, we have $x \leq s-1$ and $x < y$. It is easy to verify that we reach the maximum of $T_2(n)$ when $x = s-1$ and $y = s$. Thus, the worst-case running time is $O(|V|^{2.5}|E|\lambda_s^n)$ where λ_s is the largest positive real root of $x^{s+1} - 2x^s + 1 = 0$. The first few solutions to the equation are $\gamma_1 = 1$, $\gamma_2 = 1.618$, $\gamma_3 = 1.839$, $\gamma_4 = 1.928$ and $\gamma_5 = 1.966$ when $s = 1, 2, 3, 4$, and 5, respectively.

Scenario 3: $u \in C$ and $v \in C$. In this scenario, we remove i vertices from C in $B_i (1 \leq i \leq x)$ and y vertices from C in B_{x+1} in the worst-case. Hence, we have

$$T_3(n) \leq \sum_{i=1}^{x} T_3(n-i) + T_3(n-y), \tag{16}$$

Similarly, we have $x \leq s$ and $x < y$. It is easy to verify that we reach the maximum of $T_3(n)$ when $x = s$ and $y = s+1$. Thus the worst-case running time is $O(|V|^{2.5}|E|\lambda_s^n)$ where λ_s is the largest positive real root of $x^{s+2} - 2x^{s+1} + 1 = 0$. The first few solutions to the equation are $\gamma_1 = 1.618$, $\gamma_2 = 1.839$, $\gamma_3 = 1.928$, $\gamma_4 = 1.966$ and $\gamma_5 = 1.984$ when $s = 1, 2, 3, 4$, and 5, respectively.

Combing the time complexities of all the three scenarios, the proof of Theorem 4.4 is thus complete. □

C Additional Experiments

Overview. The additional experimental results reported here include: the number of solved instances on the three collections of graphs; the running times of SymBD and the competitor on the 40 represented graphs; running times of different branching methods; running times of our proposed heuristic method (i.e., DegenPro); detailed information about our heuristic solution; and running times of our proposed upper bounds (i.e., **UB2** and **UB3**).

Efficiency of Our Method. The number of solved instances on the three collections of graphs is shown in Figures 8-10. Overall, our proposed SymBD outperforms both MSB and CMBK in terms of the number of instances solved across nearly all values of s on datasets 2nd DIMACS graphs, 10th DIMACS graphs, and real-world graphs, with the exception of $s = 2$ in the 2nd DIMACS. The superiority of our algorithm is more pronounced on 10th DIMACS graphs and real-world graphs. Specifically, SymBD can solve more instances within 1 second than MSB and CMBK can in 3 hours, respectively. Additionally, we observe that CMBK solves more instances than MSB at smaller values of s; however, as the value of s increases, the performance of CMBK is gradually surpassed by MSB. For example, in the 10th DIMACS with $s = 4$, CMBK resolves 19 instances in 3 hours, which is 4 more than MSB; yet, when s increases to 8, MSB surpasses CMBK by solving an additional 10 instances.

Effectiveness of Branching Methods. The experimental results of the three branching methods are presented in Tables 19-31, where '-L' and '-Bin' represent SymBD-L and BinSymBD, respectively. Regarding the evaluation of different branching methods on 40 graphs, we observe that the proposed SymBD has the best empirical performance for the majority of the values of s, and this advantage is

more pronounced on the real-world graphs. For instance, when $s = 2$, SymBD achieves the shortest running time on 26 out of the 40 graphs, while SymBD-L and BinSymBD attain the optimal runtime on only 13 and 9 instances, respectively. As s is increased to 10, SymBD exhibits the minimum running time on 23 graphs, 16 of which are from the real-world graph dataset.

Effectiveness of Heuristic Methods. Tables 32-44 present the ablation studies of our heuristic method, where '-Heu' denotes Algorithm 3 without the heuristic method DegenPro. We observe that DegenPro exhibits excellent performance on the real-world graphs, while its performance is relatively average on the 10th DIMACS dataset. This difference may be attributed to the fact that we can perform efficient graph reductions based on the heuristic solutions obtained from DegenPro for the real-world graphs, whereas the graph reductions effects are less obvious on the 10th DIMACS graphs.

To gain a deeper understanding of DegenPro, we show more detailed information in Tables 59-72 with the size of heuristic solution, the preprocessing time, and the size of reduced graph. In these tables, we denote the size of the obtained heuristic solution as lb, the size of the maximum s-bundle as *real*, the time spent on the preprocessing stage as *time*, the number of vertices of the refined graph as n', and the number of edges of the refined graph as m'. We have the following observations. First, for $s = 5$ and $s = 10$, we can see that there are 53 out of the 80 instances such that the gap between the heuristic solution and the optimum solution is within 1. This demonstrates the effectiveness of our proposed heuristic approach DegenPro. Second, the preprocessing step is particularly effective for real-world graphs. For instance, with $s = 5$, the number of vertices from real-world graphs is reduced to less than 1K, and the number of edges is reduced to less than 1M. For the 2nd and 10th DIMACS graphs, the impact of preprocessing is relatively less obvious, especially for the former. The main reason is that the 2nd DIMACS graphs tend to be denser than real-world graphs, while the maximum s-bundles in the 10th DIMACS graphs are generally smaller. These characteristics implies that the preprocessing step is particularly challenging for these graphs.

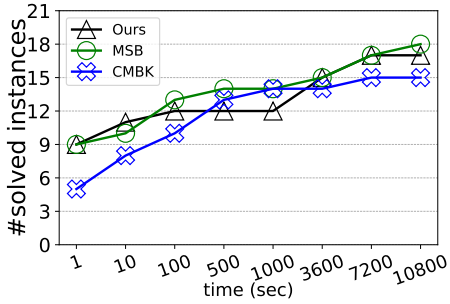
Effectiveness of Upper Bounds. To validate the effectiveness of our proposed upper bounding rules **RR2** and **RR3**, we compare with a variant '-UB', which is SymBD that only utilizes the upper bounding rule proposed in [66], i.e., SymBD without our proposed **UB2** and **UB3**. The detailed results are presented in Tables 45-58. Overall, the upper bounds we have proposed enables SymBD to achieve better practical performance on the majority of the graphs. For example, when $s = 5$ and 10, respectively, we have the following observations. First, we can see that SymBD achieves the shortest running time in 22 out of the 40 graphs when s is set to 5, and the advantage of SymBD expands when s is increased to 10. In particular, on G27 when $s = 10$, SymBD runs 4× faster than the competitor. Second, our proposed upper bounding methods are more effective on the 10th DIMACS and real-world datasets than on the 2nd DIMACS dataset. This is because the graphs in the 2nd DIMACS dataset are relatively dense, which constrains the ability of our proposed upper bounding methods to identify branches that can be pruned effectively.

Table 6. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 2$

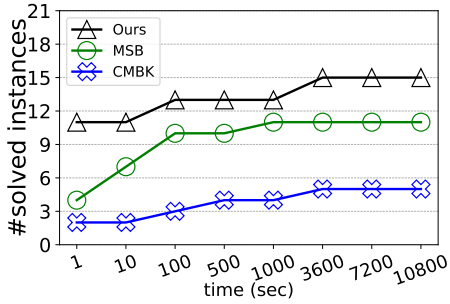
ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	1454.82	385.93	**360.73**	G21	**313.49**	OOT	OOT
G2	**0.00**	0.01	0.06	G22	**0.62**	OOT	OOT
G3	**0.00**	0.02	0.14	G23	**0.12**	OOT	OOT
G4	**0.00**	0.06	0.74	G24	**0.06**	OOT	OOT
G5	**0.00**	0.10	1.13	G25	OOT	OOT	OOT
G6	**0.01**	0.23	7.24	G26	**59.30**	OOT	OOT
G7	**3.25**	15.05	OOT	G27	**7289.75**	OOT	OOT
G8	**0.06**	0.07	0.09	G28	**4.99**	OOT	OOT
G9	**25.16**	85.20	210.90	G29	**119.69**	OOT	OOT
G10	2719.72	37.33	**19.51**	G30	OOT	OOT	OOT
G11	**5.89**	OOT	OOT	G31	OOT	OOT	OOT
G12	**1.33**	OOT	OOT	G32	**4.92**	OOT	OOT
G13	**18.77**	OOT	OOT	G33	**9.04**	OOT	OOT
G14	**0.94**	OOT	OOT	G34	**9.03**	OOT	OOT
G15	**3.71**	OOT	OOT	G35	**7.48**	OOT	OOT
G16	**0.78**	OOT	OOT	G36	**10.00**	OOT	OOT
G17	**0.48**	OOT	OOT	G37	OOT	OOT	OOT
G18	**0.11**	OOT	OOT	G38	**2.65**	OOT	OOT
G19	**2.84**	OOT	OOT	G39	**27.11**	OOT	OOT
G20	**0.25**	OOT	OOT	G40	**1596.32**	OOT	OOT

Table 7. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 3$

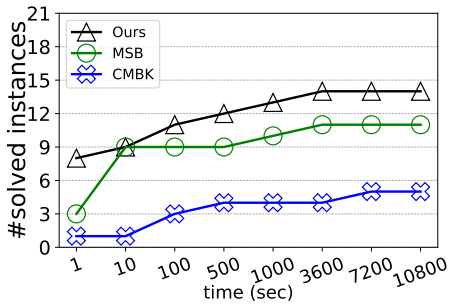
ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	**9450.59**	OOT	OOT	G21	**916.53**	OOT	OOT
G2	**0.00**	0.02	0.20	G22	**0.64**	OOT	OOT
G3	**0.00**	0.02	1.59	G23	**0.12**	OOT	OOT
G4	**0.00**	0.07	1.24	G24	**0.07**	OOT	OOT
G5	**0.00**	0.12	6.52	G25	OOT	OOT	OOT
G6	**0.00**	0.24	264.37	G26	**49.33**	OOT	OOT
G7	**496.34**	OOT	OOT	G27	**3249.62**	OOT	OOT
G8	**0.08**	1.09	1.94	G28	**4.69**	OOT	OOT
G9	**333.57**	10093.37	OOT	G29	**493.67**	OOT	OOT
G10	2590.33	3575.22	**2378.83**	G30	**2483.41**	OOT	OOT
G11	**38.19**	OOT	OOT	G31	**5243.18**	OOT	OOT
G12	**1.41**	OOT	OOT	G32	**4.06**	OOT	OOT
G13	**10.63**	OOT	OOT	G33	**38.49**	OOT	OOT
G14	**0.69**	OOT	OOT	G34	**38.45**	OOT	OOT
G15	**2.59**	OOT	OOT	G35	**7.38**	OOT	OOT
G16	**1.55**	OOT	OOT	G36	**6.91**	OOT	OOT
G17	**0.51**	OOT	OOT	G37	OOT	OOT	OOT
G18	**0.12**	OOT	OOT	G38	**2.67**	OOT	OOT
G19	**1.62**	OOT	OOT	G39	**26.12**	OOT	OOT
G20	**0.27**	OOT	OOT	G40	**1104.34**	OOT	OOT



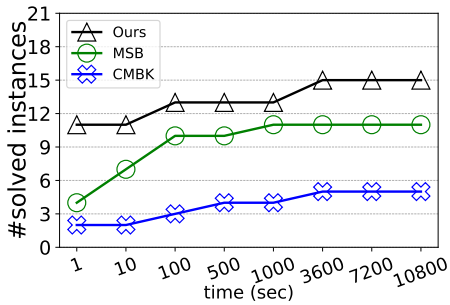
(a) $s = 2$



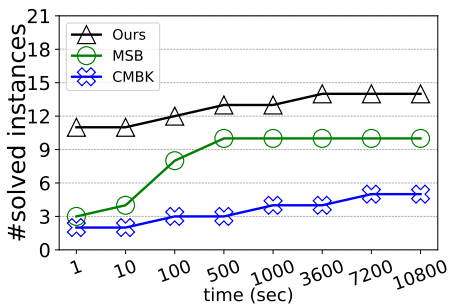
(b) $s = 4$



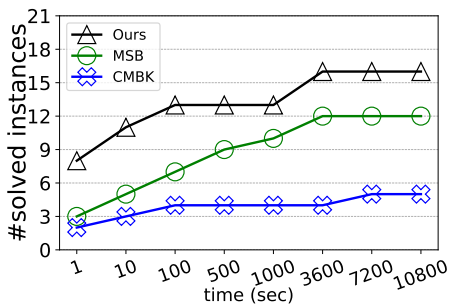
(c) $s = 6$



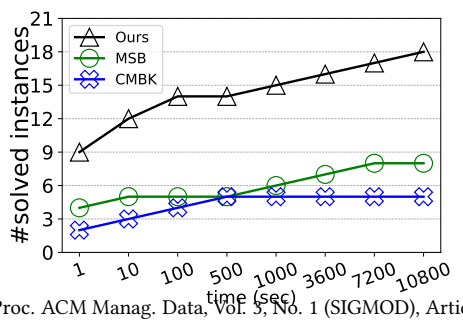
(d) $s = 7$



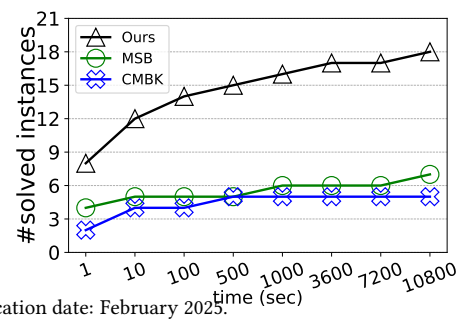
(e) $s = 8$



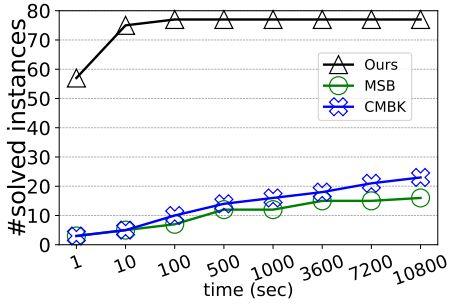
(f) $s = 9$



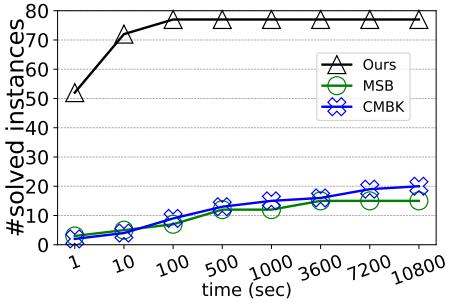
(g) $s = 11$



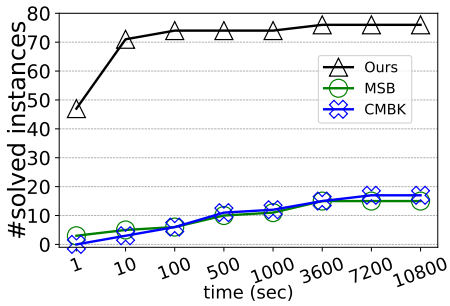
(h) $s = 12$



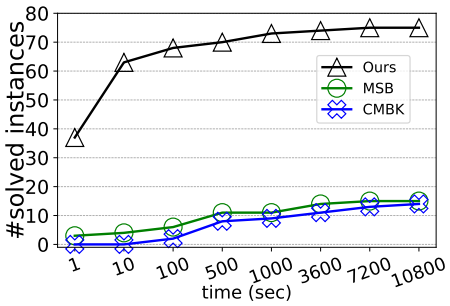
(a) $s = 2$



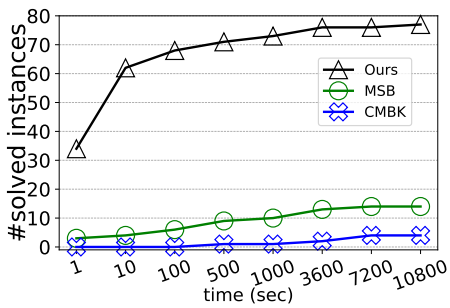
(b) $s = 4$



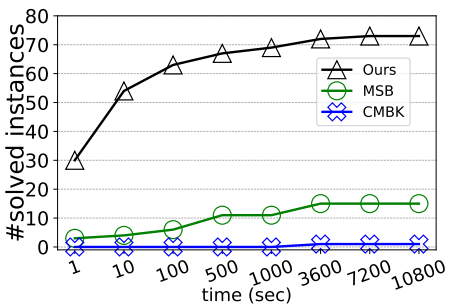
(c) $s = 6$



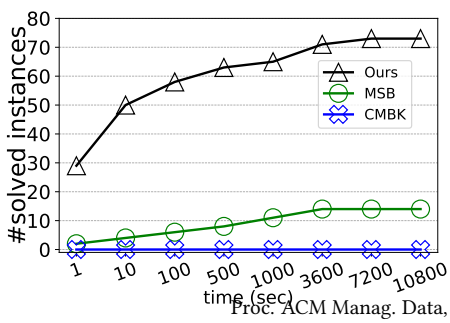
(d) $s = 7$



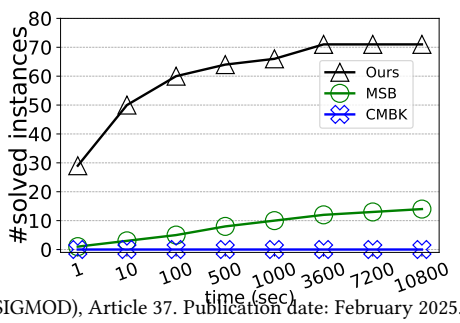
(e) $s = 8$



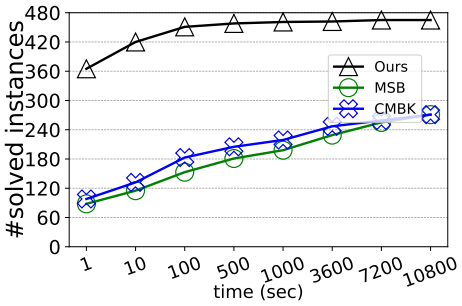
(f) $s = 9$



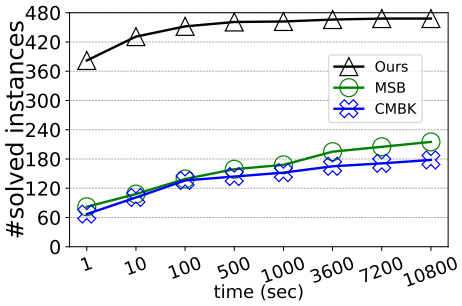
(g) $s = 11$



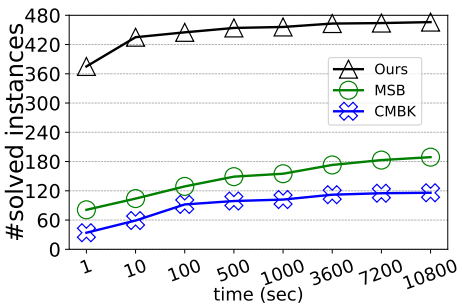
(h) $s = 12$



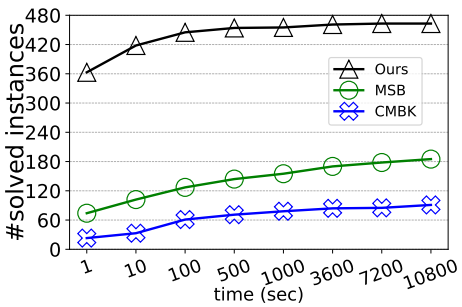
(a) $s = 2$



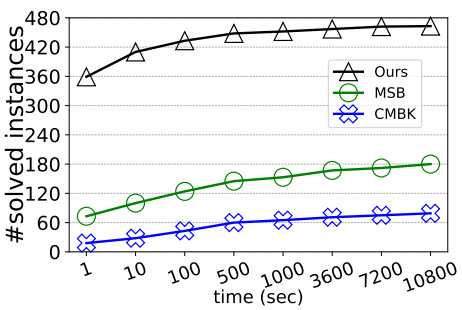
(b) $s = 4$



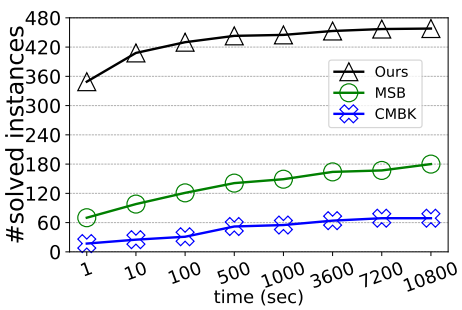
(c) $s = 6$



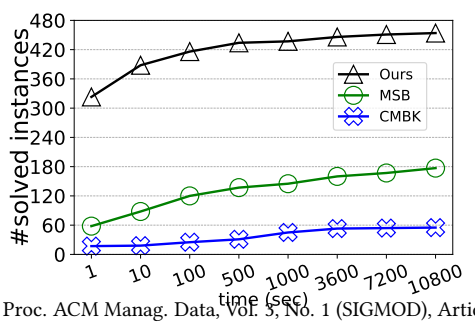
(d) $s = 7$



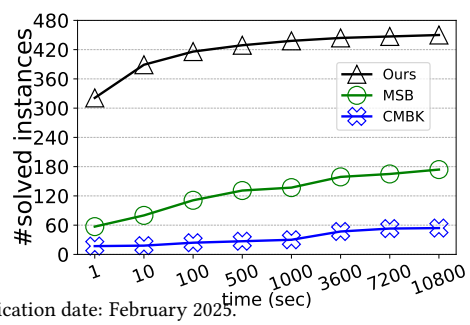
(e) $s = 8$



(f) $s = 9$



(g) $s = 11$



(h) $s = 12$

Table 8. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 4$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**211.44**	OOT	OOT
G2	**0.00**	0.03	1.78	G22	**0.68**	OOT	OOT
G3	**0.00**	0.02	62.46	G23	**0.31**	OOT	OOT
G4	**0.00**	0.10	8.37	G24	**0.16**	OOT	OOT
G5	**0.01**	0.20	286.96	G25	OOT	OOT	OOT
G6	**0.15**	0.30	OOT	G26	**33.43**	OOT	OOT
G7	**325.96**	OOT	OOT	G27	**1479.74**	OOT	OOT
G8	**0.10**	7.53	31.18	G28	**3.34**	OOT	OOT
G9	**2709.17**	OOT	OOT	G29	**518.81**	OOT	OOT
G10	**3026.86**	OOT	OOT	G30	**4442.52**	OOT	OOT
G11	**560.54**	OOT	OOT	G31	**649.01**	OOT	OOT
G12	**1.69**	OOT	OOT	G32	**4.53**	OOT	OOT
G13	**11.71**	OOT	OOT	G33	**7.47**	OOT	OOT
G14	**0.42**	OOT	OOT	G34	**7.76**	OOT	OOT
G15	**167.75**	OOT	OOT	G35	**7.24**	OOT	OOT
G16	**1814.31**	OOT	OOT	G36	**6.37**	OOT	OOT
G17	**1.26**	OOT	OOT	G37	OOT	OOT	OOT
G18	**0.33**	OOT	OOT	G38	**2.95**	OOT	OOT
G19	**2.75**	OOT	OOT	G39	**23.30**	OOT	OOT
G20	**0.34**	OOT	OOT	G40	**1932.74**	OOT	OOT

Table 9. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 6$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**12.05**	OOT	OOT
G2	**0.03**	0.88	42.68	G22	**1.22**	OOT	OOT
G3	**0.01**	0.33	OOT	G23	**0.31**	OOT	OOT
G4	**0.06**	2.89	399.34	G24	**0.19**	OOT	OOT
G5	**0.02**	3.56	OOT	G25	OOT	OOT	OOT
G6	**0.58**	2.15	OOT	G26	**44.29**	OOT	OOT
G7	**936.93**	1676.07	OOT	G27	**1403.37**	OOT	OOT
G8	**0.23**	504.08	5307.94	G28	**3.22**	OOT	OOT
G9	OOT	OOT	OOT	G29	**300.39**	OOT	OOT
G10	OOT	OOT	OOT	G30	**4999.00**	OOT	OOT
G11	OOT	OOT	OOT	G31	**317.44**	OOT	OOT
G12	**0.51**	OOT	OOT	G32	**5.07**	OOT	OOT
G13	**3.79**	OOT	OOT	G33	**7.99**	OOT	OOT
G14	**3.66**	OOT	OOT	G34	**7.86**	OOT	OOT
G15	**1.53**	OOT	OOT	G35	**6.70**	OOT	OOT
G16	**4.55**	OOT	OOT	G36	**6.44**	OOT	OOT
G17	**1.55**	OOT	OOT	G37	**4414.43**	OOT	OOT
G18	**3.13**	OOT	OOT	G38	**20.23**	OOT	OOT
G19	**1.47**	OOT	OOT	G39	**25.17**	OOT	OOT
G20	**0.30**	OOT	OOT	G40	**13.91**	OOT	OOT

Table 10. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 7$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**12.45**	OOT	OOT
G2	**0.02**	0.95	123.82	G22	**1.20**	OOT	OOT
G3	**0.01**	2.12	OOT	G23	**0.36**	OOT	OOT
G4	**0.07**	9.97	1248.25	G24	**0.23**	OOT	OOT
G5	**0.01**	24.02	OOT	G25	OOT	OOT	OOT
G6	**0.74**	13.07	OOT	G26	**34.20**	OOT	OOT
G7	0.04	0.04	**0.01**	G27	**1530.58**	OOT	OOT
G8	**0.06**	980.57	OOT	G28	**4.22**	OOT	OOT
G9	OOT	OOT	OOT	G29	**287.79**	OOT	OOT
G10	OOT	OOT	OOT	G30	**1690.16**	OOT	OOT
G11	OOT	OOT	OOT	G31	**216.96**	OOT	OOT
G12	**0.63**	OOT	OOT	G32	**5.20**	OOT	OOT
G13	**4.12**	OOT	OOT	G33	**22.01**	OOT	OOT
G14	**2.27**	OOT	OOT	G34	**22.05**	OOT	OOT
G15	**7.04**	OOT	OOT	G35	**6.85**	OOT	OOT
G16	**663.32**	OOT	OOT	G36	**7.84**	OOT	OOT
G17	**11.30**	OOT	OOT	G37	**5568.08**	OOT	OOT
G18	**4.51**	OOT	OOT	G38	**18.29**	OOT	OOT
G19	**3.70**	OOT	OOT	G39	**21.84**	OOT	OOT
G20	**0.32**	OOT	OOT	G40	**14.94**	OOT	OOT

Table 11. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 8$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**11.38**	OOT	OOT
G2	**0.02**	0.56	534.48	G22	**4.84**	OOT	OOT
G3	**0.17**	30.32	OOT	G23	**0.28**	OOT	OOT
G4	**0.08**	10.45	4166.10	G24	**0.32**	OOT	OOT
G5	**0.41**	290.78	OOT	G25	OOT	OOT	OOT
G6	**0.71**	93.65	OOT	G26	**18.93**	OOT	OOT
G7	0.03	0.04	**0.01**	G27	**578.26**	OOT	OOT
G8	**0.37**	OOT	OOT	G28	**5.54**	OOT	OOT
G9	OOT	OOT	OOT	G29	**2514.25**	OOT	OOT
G10	OOT	OOT	OOT	G30	**365.05**	OOT	OOT
G11	**2051.30**	OOT	OOT	G31	**212.33**	OOT	OOT
G12	**0.48**	OOT	OOT	G32	**3.07**	OOT	OOT
G13	**12.89**	OOT	OOT	G33	**23.06**	OOT	OOT
G14	**2.82**	OOT	OOT	G34	**23.13**	OOT	OOT
G15	**9.11**	OOT	OOT	G35	**7.15**	OOT	OOT
G16	**6.97**	OOT	OOT	G36	**6.33**	OOT	OOT
G17	**4.00**	OOT	OOT	G37	**4619.60**	OOT	OOT
G18	**8.63**	OOT	OOT	G38	**21.14**	OOT	OOT
G19	**1.40**	OOT	OOT	G39	**20.98**	OOT	OOT
G20	**0.32**	OOT	OOT	G40	**23.93**	OOT	OOT

Table 12. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 9$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**9.51**	OOT	OOT
G2	**18.58**	77.32	5859.53	G22	**2.80**	OOT	OOT
G3	**0.19**	161.00	OOT	G23	**0.43**	OOT	OOT
G4	**0.08**	5.90	OOT	G24	**0.57**	OOT	OOT
G5	**0.70**	1621.41	OOT	G25	OOT	OOT	OOT
G6	**1.03**	674.54	OOT	G26	**20.04**	OOT	OOT
G7	0.03	0.03	**0.01**	G27	**794.10**	OOT	OOT
G8	**2.63**	OOT	OOT	G28	**5.52**	OOT	OOT
G9	**3277.60**	OOT	OOT	G29	**1203.13**	OOT	OOT
G10	OOT	OOT	OOT	G30	**163.61**	OOT	OOT
G11	OOT	OOT	OOT	G31	**165.77**	OOT	OOT
G12	**2613.93**	OOT	OOT	G32	**3.49**	OOT	OOT
G13	OOT	OOT	OOT	G33	**53.30**	OOT	OOT
G14	**2.10**	OOT	OOT	G34	**53.58**	OOT	OOT
G15	**7.13**	OOT	OOT	G35	**6.89**	OOT	OOT
G16	**4.83**	OOT	OOT	G36	**6.64**	OOT	OOT
G17	**21.72**	OOT	OOT	G37	**3167.43**	OOT	OOT
G18	**1.23**	OOT	OOT	G38	**9.23**	OOT	OOT
G19	**1.52**	OOT	OOT	G39	**26.64**	OOT	OOT
G20	**0.33**	OOT	OOT	G40	**21.73**	OOT	OOT

Table 13. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 10$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**6.83**	OOT	OOT
G2	**38.29**	631.25	OOT	G22	**1.93**	OOT	OOT
G3	**0.31**	730.05	OOT	G23	**0.36**	OOT	OOT
G4	**428.75**	1699.24	OOT	G24	**0.63**	OOT	OOT
G5	**0.82**	8452.45	OOT	G25	**3699.51**	OOT	OOT
G6	**1.22**	4823.27	OOT	G26	**6.99**	OOT	OOT
G7	0.03	0.04	**0.01**	G27	**1217.38**	OOT	OOT
G8	**0.09**	OOT	OOT	G28	**3.98**	OOT	OOT
G9	OOT	OOT	OOT	G29	OOT	OOT	OOT
G10	OOT	OOT	OOT	G30	**141.32**	OOT	OOT
G11	OOT	OOT	OOT	G31	**129.83**	OOT	OOT
G12	**998.43**	OOT	OOT	G32	**3.43**	OOT	OOT
G13	**6529.59**	OOT	OOT	G33	**53.77**	OOT	OOT
G14	**2.74**	OOT	OOT	G34	**53.69**	OOT	OOT
G15	**9.14**	OOT	OOT	G35	**7.08**	OOT	OOT
G16	**6.31**	OOT	OOT	G36	**6.81**	OOT	OOT
G17	**4.93**	OOT	OOT	G37	**3088.77**	OOT	OOT
G18	**4.50**	OOT	OOT	G38	**8.77**	OOT	OOT
G19	**6.68**	OOT	OOT	G39	**24.95**	OOT	OOT
G20	**0.36**	OOT	OOT	G40	**20.15**	OOT	OOT

Table 14. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 11$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**5.01**	OOT	OOT
G2	**9.34**	6291.68	OOT	G22	OOT	OOT	OOT
G3	**0.27**	2872.73	OOT	G23	**1.14**	OOT	OOT
G4	**1046.56**	OOT	OOT	G24	**0.60**	OOT	OOT
G5	**0.91**	OOT	OOT	G25	**687.31**	OOT	OOT
G6	**1.32**	OOT	OOT	G26	**8.30**	OOT	OOT
G7	0.03	0.03	**0.01**	G27	**1640.61**	OOT	OOT
G8	**0.04**	OOT	OOT	G28	**2.81**	OOT	OOT
G9	OOT	OOT	OOT	G29	**317.23**	OOT	OOT
G10	OOT	OOT	OOT	G30	**168.19**	OOT	OOT
G11	OOT	OOT	OOT	G31	**115.82**	OOT	OOT
G12	**1754.41**	OOT	OOT	G32	**4.64**	OOT	OOT
G13	**2803.32**	OOT	OOT	G33	**51.39**	OOT	OOT
G14	**1.97**	OOT	OOT	G34	**51.82**	OOT	OOT
G15	**7.01**	OOT	OOT	G35	**7.15**	OOT	OOT
G16	**4.53**	OOT	OOT	G36	**6.29**	OOT	OOT
G17	**9.92**	OOT	OOT	G37	**3723.98**	OOT	OOT
G18	**1.53**	OOT	OOT	G38	**25.45**	OOT	OOT
G19	**7.08**	OOT	OOT	G39	**32.78**	OOT	OOT
G20	**0.39**	OOT	OOT	G40	**29.30**	OOT	OOT

Table 15. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 12$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**23.88**	OOT	OOT
G2	**1.36**	OOT	OOT	G22	OOT	OOT	OOT
G3	**0.35**	10487.13	OOT	G23	**1.16**	OOT	OOT
G4	**297.94**	OOT	OOT	G24	**0.98**	OOT	OOT
G5	**0.98**	OOT	OOT	G25	**929.47**	OOT	OOT
G6	**1.38**	OOT	OOT	G26	**7.79**	OOT	OOT
G7	**0.00**	0.04	**0.00**	G27	**1910.62**	OOT	OOT
G8	**0.02**	OOT	OOT	G28	**2.42**	OOT	OOT
G9	OOT	OOT	OOT	G29	**69.83**	OOT	OOT
G10	OOT	OOT	OOT	G30	**71.08**	OOT	OOT
G11	OOT	OOT	OOT	G31	**113.55**	OOT	OOT
G12	**552.56**	OOT	OOT	G32	**3.74**	OOT	OOT
G13	OOT	OOT	OOT	G33	**46.57**	OOT	OOT
G14	**2.90**	OOT	OOT	G34	**46.55**	OOT	OOT
G15	**7.54**	OOT	OOT	G35	**7.13**	OOT	OOT
G16	**4.80**	OOT	OOT	G36	**7.85**	OOT	OOT
G17	**6.76**	OOT	OOT	G37	**4923.81**	OOT	OOT
G18	**1.45**	OOT	OOT	G38	**156.21**	OOT	OOT
G19	**6.64**	OOT	OOT	G39	**35.15**	OOT	OOT
G20	**0.40**	OOT	OOT	G40	**35.43**	OOT	OOT

Table 16. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 13$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**104.48**	OOT	OOT
G2	**0.04**	OOT	OOT	G22	OOT	OOT	OOT
G3	**0.33**	8844.61	OOT	G23	**0.89**	OOT	OOT
G4	**52.57**	OOT	OOT	G24	**2.25**	OOT	OOT
G5	**0.85**	OOT	OOT	G25	**157.56**	OOT	OOT
G6	**1.35**	OOT	OOT	G26	**8.15**	OOT	OOT
G7	**0.00**	0.04	0.00	G27	**2240.02**	OOT	OOT
G8	**0.06**	OOT	OOT	G28	**2.24**	OOT	OOT
G9	**64.45**	OOT	OOT	G29	**63.11**	OOT	OOT
G10	OOT	OOT	OOT	G30	**64.52**	OOT	OOT
G11	OOT	OOT	OOT	G31	**110.85**	OOT	OOT
G12	**211.93**	OOT	OOT	G32	**3.87**	OOT	OOT
G13	OOT	OOT	OOT	G33	**219.37**	OOT	OOT
G14	**2.33**	OOT	OOT	G34	**218.22**	OOT	OOT
G15	**7.17**	OOT	OOT	G35	**7.09**	OOT	OOT
G16	**4.73**	OOT	OOT	G36	**6.53**	OOT	OOT
G17	**8.56**	OOT	OOT	G37	**6172.27**	OOT	OOT
G18	**3.63**	OOT	OOT	G38	**810.85**	OOT	OOT
G19	**8.18**	OOT	OOT	G39	**42.47**	OOT	OOT
G20	**0.28**	OOT	OOT	G40	**44.00**	OOT	OOT

Table 17. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 14$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**65.98**	OOT	OOT
G2	**0.03**	OOT	OOT	G22	OOT	OOT	OOT
G3	**0.34**	5977.76	OOT	G23	**1.08**	OOT	OOT
G4	**5.36**	OOT	OOT	G24	**4.72**	OOT	OOT
G5	**0.91**	OOT	OOT	G25	**53.86**	OOT	OOT
G6	**1.45**	OOT	OOT	G26	**6.54**	OOT	OOT
G7	**0.00**	0.04	**0.00**	G27	OOT	OOT	OOT
G8	**0.00**	OOT	OOT	G28	**2.74**	OOT	OOT
G9	OOT	OOT	OOT	G29	**2209.82**	OOT	OOT
G10	OOT	OOT	OOT	G30	**62.34**	OOT	OOT
G11	OOT	OOT	OOT	G31	**111.62**	OOT	OOT
G12	**2099.03**	OOT	OOT	G32	**4.34**	OOT	OOT
G13	**4066.04**	OOT	OOT	G33	**179.04**	OOT	OOT
G14	**3.01**	OOT	OOT	G34	**178.80**	OOT	OOT
G15	**7.43**	OOT	OOT	G35	**117.44**	OOT	OOT
G16	**5.67**	OOT	OOT	G36	**87.82**	OOT	OOT
G17	**17.46**	OOT	OOT	G37	**7132.81**	OOT	OOT
G18	**1.93**	OOT	OOT	G38	OOT	OOT	OOT
G19	**1.43**	OOT	OOT	G39	**50.13**	OOT	OOT
G20	**0.31**	OOT	OOT	G40	**32.52**	OOT	OOT

Table 18. Running time in seconds of SymBD and state-of-the-arts on 40 graphs with $s = 15$

ID	Ours	MSB	CMBK	ID	Ours	MSB	CMBK
G1	OOT	OOT	OOT	G21	**2474.88**	OOT	OOT
G2	**0.03**	OOT	OOT	G22	OOT	OOT	OOT
G3	OOT	OOT	OOT	G23	**13.91**	OOT	OOT
G4	**0.20**	OOT	OOT	G24	**7.55**	OOT	OOT
G5	**1.15**	OOT	OOT	G25	**41.09**	OOT	OOT
G6	**1.52**	OOT	OOT	G26	**5.23**	OOT	OOT
G7	**0.00**	0.04	**0.00**	G27	OOT	OOT	OOT
G8	**0.01**	3403.13	OOT	G28	**2.57**	OOT	OOT
G9	**1492.15**	9857.90	OOT	G29	**765.76**	OOT	OOT
G10	OOT	OOT	OOT	G30	**60.17**	OOT	OOT
G11	OOT	OOT	OOT	G31	**103.52**	OOT	OOT
G12	**364.10**	OOT	OOT	G32	**3.97**	OOT	OOT
G13	**661.23**	OOT	OOT	G33	**981.96**	OOT	OOT
G14	**6.89**	OOT	OOT	G34	**985.32**	OOT	OOT
G15	**13.48**	OOT	OOT	G35	**102.41**	OOT	OOT
G16	**5.18**	OOT	OOT	G36	**101.41**	OOT	OOT
G17	**11.43**	OOT	OOT	G37	**7817.05**	OOT	OOT
G18	**1.84**	OOT	OOT	G38	OOT	OOT	OOT
G19	**7.63**	OOT	OOT	G39	**55.22**	OOT	OOT
G20	**0.30**	OOT	OOT	G40	**45.91**	OOT	OOT

Table 19. Running time in seconds of different branching methods on 40 graphs with $s = 2$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	1454.82	**1327.54**	2386.88	G21	**313.49**	313.78	317.82
G2	**0.00**	**0.00**	**0.00**	G22	**0.62**	0.66	0.90
G3	**0.00**	**0.00**	**0.00**	G23	**0.12**	0.14	**0.12**
G4	**0.00**	**0.00**	**0.00**	G24	**0.06**	**0.06**	0.07
G5	**0.00**	**0.00**	**0.00**	G25	OOT	OOT	OOT
G6	**0.01**	**0.01**	**0.01**	G26	**59.30**	92.96	100.13
G7	**3.25**	15.23	14.00	G27	**7289.75**	7576.91	8205.54
G8	**0.06**	0.08	0.08	G28	4.99	**4.92**	5.08
G9	**25.16**	55.86	91.87	G29	**119.69**	1090.72	185.69
G10	2719.72	**2618.39**	2963.06	G30	OOT	OOT	OOT
G11	**5.89**	7.33	6.26	G31	OOT	OOT	OOT
G12	**1.33**	1.46	1.35	G32	**4.92**	5.84	5.15
G13	18.77	**16.68**	19.58	G33	**9.04**	10.53	12.19
G14	**0.94**	0.98	1.03	G34	**9.03**	10.83	12.18
G15	3.71	**3.56**	4.17	G35	7.48	7.40	**7.39**
G16	**0.78**	0.79	0.80	G36	10.00	**7.50**	8.11
G17	**0.48**	0.56	0.62	G37	OOT	OOT	OOT
G18	**0.11**	0.13	0.12	G38	**2.65**	4.84	4.03
G19	**2.84**	3.45	3.06	G39	27.11	26.96	**23.05**
G20	**0.25**	**0.25**	0.31	G40	1596.32	2630.40	**1302.73**

Table 20. Running time in seconds of different branching methods on 40 graphs with $s = 3$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	**9450.59**	OOT	OOT	G21	**916.53**	OOT	OOT
G2	**0.00**	**0.00**	**0.00**	G22	0.64	**0.62**	0.71
G3	**0.00**	**0.00**	**0.00**	G23	**0.12**	0.14	0.13
G4	**0.00**	**0.00**	**0.00**	G24	0.07	**0.06**	**0.06**
G5	**0.00**	**0.00**	**0.00**	G25	OOT	OOT	OOT
G6	**0.00**	**0.00**	**0.00**	G26	**49.33**	1818.79	336.36
G7	**496.34**	6320.10	6964.65	G27	**3249.62**	OOT	OOT
G8	**0.08**	0.25	0.18	G28	4.69	**4.68**	5.12
G9	**333.57**	3985.15	3858.78	G29	**493.67**	OOT	4044.40
G10	**2590.33**	OOT	OOT	G30	**2483.41**	OOT	OOT
G11	38.19	**37.07**	38.70	G31	**5243.18**	OOT	OOT
G12	**1.41**	1.52	1.43	G32	**4.06**	4.19	4.67
G13	**10.63**	12.06	10.96	G33	**38.49**	618.38	175.90
G14	0.69	**0.59**	0.67	G34	**38.45**	610.39	178.80
G15	2.59	2.66	**2.41**	G35	7.38	**7.13**	7.55
G16	1.55	1.68	**1.54**	G36	**6.91**	7.44	7.62
G17	**0.51**	0.55	**0.51**	G37	OOT	OOT	OOT
G18	0.12	0.13	**0.11**	G38	**2.67**	43.54	5.44
G19	1.62	1.45	**1.43**	G39	26.12	**23.89**	26.22
G20	**0.27**	0.30	0.28	G40	**1104.34**	OOT	OOT

Table 21. Running time in seconds of different branching methods on 40 graphs with $s = 4$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**211.44**	OOT	6314.06
G2	**0.00**	**0.00**	**0.00**	G22	**0.68**	0.75	0.75
G3	**0.00**	**0.00**	**0.00**	G23	0.31	**0.30**	0.34
G4	**0.00**	**0.00**	**0.00**	G24	**0.16**	0.19	0.19
G5	**0.01**	**0.01**	**0.01**	G25	OOT	OOT	OOT
G6	**0.15**	0.18	0.20	G26	**33.43**	OOT	305.68
G7	**325.96**	OOT	4013.84	G27	**1479.74**	OOT	OOT
G8	**0.10**	1.13	0.23	G28	**3.34**	3.67	3.35
G9	**2709.17**	OOT	OOT	G29	**518.81**	OOT	3460.07
G10	**3026.86**	OOT	OOT	G30	**4442.52**	OOT	OOT
G11	**560.54**	954.48	617.63	G31	**649.01**	OOT	9144.86
G12	1.69	**1.51**	1.61	G32	4.53	5.46	**3.83**
G13	**11.71**	11.72	11.77	G33	**7.47**	1302.18	26.19
G14	0.42	0.40	**0.32**	G34	**7.76**	1196.15	27.08
G15	167.75	OOT	**164.50**	G35	7.24	7.05	**6.63**
G16	1814.31	OOT	**380.74**	G36	**6.37**	7.23	7.26
G17	1.26	1.16	**1.05**	G37	OOT	OOT	OOT
G18	0.33	0.26	**0.25**	G38	**2.95**	443.29	5.84
G19	**2.75**	2.87	2.97	G39	23.30	**22.09**	25.54
G20	0.34	**0.23**	0.32	G40	**1932.74**	OOT	9682.03

Table 22. Running time in seconds of different branching methods on 40 graphs with $s = 6$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**12.05**	OOT	180.41
G2	0.03	**0.02**	0.03	G22	**1.22**	1.78	1.28
G3	**0.01**	**0.01**	**0.01**	G23	0.31	0.34	**0.29**
G4	0.06	**0.05**	**0.05**	G24	0.19	OOT	**0.15**
G5	0.02	**0.01**	0.02	G25	OOT	OOT	OOT
G6	0.58	**0.51**	0.58	G26	**44.29**	OOT	279.31
G7	**936.93**	1178.50	1210.11	G27	**1403.37**	OOT	OOT
G8	**0.23**	10.08	0.43	G28	3.22	**2.74**	2.93
G9	OOT	OOT	OOT	G29	**300.39**	OOT	588.72
G10	OOT	OOT	OOT	G30	**4999.00**	OOT	OOT
G11	OOT	OOT	OOT	G31	**317.44**	OOT	1928.44
G12	**0.51**	0.59	0.62	G32	5.07	8.81	**4.17**
G13	**3.79**	**3.79**	3.91	G33	**7.99**	OOT	24.06
G14	**3.66**	OOT	3.73	G34	**7.86**	OOT	23.91
G15	1.53	1.26	**1.22**	G35	**6.70**	6.89	7.24
G16	**4.55**	OOT	4.57	G36	**6.44**	6.80	6.81
G17	**1.55**	2.53	1.67	G37	**4414.43**	OOT	OOT
G18	3.13	OOT	**2.97**	G38	**20.23**	OOT	372.08
G19	**1.47**	1.51	1.54	G39	25.17	**21.64**	23.14
G20	**0.30**	0.32	0.43	G40	13.91	11.15	**8.04**

Table 23. Running time in seconds of different branching methods on 40 graphs with $s = 7$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**12.45**	OOT	235.30
G2	**0.02**	0.03	**0.02**	G22	1.20	**1.14**	1.28
G3	**0.01**	0.02	**0.01**	G23	0.36	OOT	**0.35**
G4	**0.07**	0.08	0.08	G24	0.23	OOT	**0.20**
G5	**0.01**	**0.01**	**0.01**	G25	OOT	OOT	OOT
G6	0.74	0.86	**0.57**	G26	**34.20**	OOT	143.62
G7	0.04	0.03	**0.02**	G27	**1530.58**	OOT	OOT
G8	**0.06**	4.24	0.10	G28	4.22	4.07	**3.55**
G9	OOT	OOT	OOT	G29	**287.79**	OOT	387.28
G10	OOT	OOT	OOT	G30	**1690.16**	OOT	OOT
G11	OOT	OOT	OOT	G31	**216.96**	OOT	775.84
G12	0.63	**0.57**	0.59	G32	5.20	5.89	**3.94**
G13	4.12	3.88	**3.77**	G33	**22.01**	OOT	64.68
G14	2.27	OOT	**2.64**	G34	**22.05**	OOT	67.71
G15	7.04	OOT	**6.65**	G35	**6.85**	7.58	7.48
G16	663.32	OOT	**279.80**	G36	7.84	**6.08**	6.60
G17	11.30	OOT	**11.22**	G37	**5568.08**	OOT	OOT
G18	**4.51**	OOT	4.53	G38	**18.29**	OOT	605.22
G19	**3.70**	OOT	4.53	G39	**21.84**	24.86	26.94
G20	0.32	**0.28**	0.33	G40	14.94	**10.23**	15.36

Table 24. Running time in seconds of different branching methods on 40 graphs with $s = 8$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**11.38**	OOT	270.05
G2	**0.02**	**0.02**	**0.02**	G22	**4.84**	OOT	18.40
G3	0.17	**0.14**	**0.14**	G23	**0.28**	43.55	0.37
G4	0.08	0.08	**0.06**	G24	0.32	OOT	**0.26**
G5	0.41	**0.36**	0.37	G25	OOT	OOT	OOT
G6	**0.71**	1.05	0.93	G26	**18.93**	OOT	60.34
G7	**0.03**	**0.03**	**0.03**	G27	**578.26**	OOT	OOT
G8	**0.37**	29.94	0.96	G28	**5.54**	5.84	5.63
G9	OOT	OOT	OOT	G29	**2514.25**	OOT	2903.12
G10	OOT	OOT	OOT	G30	**365.05**	OOT	1877.44
G11	**2051.30**	4999.82	2275.09	G31	**212.33**	OOT	572.20
G12	**0.48**	0.60	0.58	G32	**3.07**	3.72	3.63
G13	12.89	12.99	**12.88**	G33	**23.06**	OOT	65.43
G14	2.82	OOT	**2.75**	G34	**23.13**	OOT	65.07
G15	**9.11**	OOT	9.78	G35	7.15	**6.92**	7.00
G16	**6.97**	OOT	7.13	G36	**6.33**	6.62	7.10
G17	**4.00**	OOT	4.02	G37	**4619.60**	OOT	OOT
G18	8.63	OOT	**7.35**	G38	**21.14**	OOT	940.33
G19	**1.40**	OOT	1.65	G39	20.98	**20.77**	23.78
G20	0.32	0.27	**0.26**	G40	23.93	17.25	**16.57**

Table 25. Running time in seconds of different branching methods on 40 graphs with $s = 9$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**9.51**	OOT	244.60
G2	18.58	OOT	**17.77**	G22	**2.80**	OOT	20.25
G3	**0.19**	0.24	0.24	G23	0.43	OOT	**0.41**
G4	0.08	**0.06**	0.08	G24	**0.57**	OOT	0.61
G5	0.70	**0.53**	0.70	G25	OOT	OOT	OOT
G6	1.03	2.00	**0.89**	G26	**20.04**	OOT	63.17
G7	0.03	0.03	**0.02**	G27	**794.10**	OOT	OOT
G8	**2.63**	4990.30	8.00	G28	5.52	**5.45**	5.86
G9	**3277.60**	OOT	OOT	G29	**1203.13**	OOT	1209.01
G10	OOT	OOT	OOT	G30	**163.61**	OOT	522.95
G11	OOT	OOT	OOT	G31	**165.77**	OOT	292.87
G12	2613.93	OOT	**1649.66**	G32	**3.49**	4.28	3.69
G13	OOT	OOT	OOT	G33	**53.30**	OOT	233.30
G14	**2.10**	OOT	2.25	G34	**53.58**	OOT	226.04
G15	7.13	OOT	**6.73**	G35	6.89	6.86	**6.54**
G16	4.83	OOT	**4.62**	G36	6.64	**6.48**	9.67
G17	21.72	OOT	**19.51**	G37	**3167.43**	OOT	OOT
G18	**1.23**	OOT	1.32	G38	**9.23**	OOT	26.89
G19	**1.52**	OOT	1.61	G39	26.64	**22.83**	27.80
G20	0.33	**0.32**	0.33	G40	21.73	**12.09**	17.18

Table 26. Running time in seconds of different branching methods on 40 graphs with $s = 10$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**6.83**	OOT	169.65
G2	38.29	OOT	**13.20**	G22	**1.93**	33.18	3.05
G3	**0.31**	**0.31**	**0.31**	G23	**0.36**	OOT	0.43
G4	428.75	OOT	**413.65**	G24	**0.63**	OOT	0.69
G5	**0.82**	**0.82**	**0.82**	G25	**3699.51**	OOT	OOT
G6	1.22	1.12	**0.95**	G26	**6.99**	OOT	20.04
G7	0.03	0.03	**0.02**	G27	**1217.38**	OOT	OOT
G8	**0.09**	4.94	0.33	G28	3.98	**3.87**	4.03
G9	OOT	OOT	OOT	G29	OOT	OOT	OOT
G10	OOT	OOT	OOT	G30	**141.32**	OOT	341.23
G11	OOT	OOT	OOT	G31	**129.83**	OOT	178.56
G12	**998.43**	OOT	1556.71	G32	3.43	**3.33**	3.47
G13	**6529.59**	OOT	OOT	G33	**53.77**	OOT	234.95
G14	**2.74**	OOT	2.87	G34	**53.69**	OOT	235.32
G15	9.14	OOT	**9.05**	G35	7.08	7.72	**6.64**
G16	6.31	OOT	**6.03**	G36	**6.81**	8.74	7.37
G17	4.93	OOT	**4.72**	G37	**3088.77**	OOT	OOT
G18	**4.50**	OOT	5.85	G38	**8.77**	OOT	27.98
G19	6.68	OOT	**6.65**	G39	**24.95**	29.03	28.49
G20	0.36	0.35	**0.34**	G40	**20.15**	25.52	24.74

Table 27. Running time in seconds of different branching methods on 40 graphs with $s = 11$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**5.01**	OOT	86.65
G2	9.34	OOT	**5.87**	G22	OOT	OOT	OOT
G3	**0.27**	0.36	0.36	G23	1.14	OOT	**1.05**
G4	1046.56	OOT	**341.29**	G24	0.6	OOT	**0.52**
G5	**0.91**	0.93	0.93	G25	**687.31**	OOT	7437.35
G6	**1.32**	1.51	1.41	G26	**8.30**	OOT	24.48
G7	**0.03**	**0.03**	**0.03**	G27	**1640.61**	OOT	OOT
G8	**0.04**	0.98	0.09	G28	2.81	**2.18**	2.29
G9	OOT	OOT	OOT	G29	**317.23**	OOT	321.78
G10	OOT	OOT	OOT	G30	**168.19**	OOT	438.06
G11	OOT	OOT	OOT	G31	**115.82**	OOT	125.24
G12	1754.41	OOT	**976.73**	G32	4.64	3.53	**3.44**
G13	**2803.32**	OOT	5167.70	G33	**51.39**	OOT	244.45
G14	**1.97**	OOT	2.27	G34	**51.82**	OOT	244.67
G15	**7.01**	OOT	7.18	G35	**7.15**	OOT	815.70
G16	**4.53**	OOT	4.91	G36	**6.29**	6.86	6.95
G17	9.92	OOT	**7.99**	G37	**3723.98**	OOT	OOT
G18	**1.53**	OOT	1.62	G38	**25.45**	OOT	612.40
G19	**7.08**	OOT	**7.08**	G39	32.78	**30.49**	33.95
G20	0.39	**0.27**	0.38	G40	29.3	**25.12**	29.24

Table 28. Running time in seconds of different branching methods on 40 graphs with $s = 12$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**23.88**	OOT	OOT
G2	**1.36**	OOT	2.74	G22	OOT	OOT	OOT
G3	**0.35**	0.42	9.05	G23	1.16	OOT	**1.12**
G4	297.94	OOT	**165.66**	G24	**0.98**	OOT	68.95
G5	0.98	**0.77**	0.96	G25	**929.47**	OOT	3837.86
G6	1.38	1.54	**1.19**	G26	**7.79**	OOT	27.72
G7	**0.00**	**0.00**	**0.00**	G27	**1910.62**	OOT	OOT
G8	**0.02**	0.06	**0.02**	G28	2.42	2.61	**2.06**
G9	OOT	OOT	OOT	G29	**69.83**	1183.57	83.59
G10	OOT	OOT	OOT	G30	**71.08**	OOT	81.90
G11	OOT	OOT	OOT	G31	**113.55**	OOT	115.98
G12	**552.56**	OOT	771.98	G32	**3.74**	4.10	3.90
G13	OOT	OOT	**3216.19**	G33	**46.57**	OOT	249.58
G14	2.90	OOT	**2.78**	G34	**46.55**	OOT	245.12
G15	7.54	OOT	**7.05**	G35	**7.13**	OOT	57.10
G16	**4.80**	OOT	5.67	G36	**7.85**	OOT	99.83
G17	**6.76**	OOT	8.62	G37	**4923.81**	OOT	OOT
G18	**1.45**	OOT	1.53	G38	**156.21**	OOT	OOT
G19	**6.64**	OOT	7.76	G39	35.15	**34.29**	34.49
G20	0.40	0.36	**0.30**	G40	35.43	**32.01**	34.77

Table 29. Running time in seconds of different branching methods on 40 graphs with $s = 13$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**104.48**	OOT	OOT
G2	**0.04**	OOT	0.53	G22	OOT	OOT	OOT
G3	**0.33**	0.43	5.60	G23	**0.89**	OOT	0.94
G4	**52.57**	OOT	93.17	G24	**2.25**	OOT	568.53
G5	**0.85**	1.12	43.45	G25	**157.56**	OOT	862.38
G6	1.35	1.45	**1.34**	G26	**8.15**	OOT	32.39
G7	**0.00**	**0.00**	**0.00**	G27	**2240.02**	OOT	OOT
G8	**0.06**	121.99	0.13	G28	**2.24**	OOT	2.49
G9	**64.45**	OOT	1275.43	G29	**63.11**	140.79	65.76
G10	OOT	OOT	OOT	G30	64.52	OOT	**63.79**
G11	OOT	OOT	OOT	G31	**110.85**	OOT	137.06
G12	211.93	OOT	**206.43**	G32	**3.87**	4.27	4.58
G13	OOT	OOT	OOT	G33	**219.37**	OOT	2391.08
G14	**2.33**	OOT	6.16	G34	**218.22**	OOT	2361.13
G15	**7.17**	OOT	7.72	G35	7.09	**6.75**	6.91
G16	**4.73**	OOT	5.87	G36	**6.53**	7.45	9.04
G17	**8.56**	OOT	8.64	G37	**6172.27**	OOT	OOT
G18	**3.63**	OOT	4.28	G38	**810.85**	OOT	OOT
G19	8.18	OOT	**7.11**	G39	**42.47**	43.47	43.59
G20	**0.28**	0.36	0.33	G40	44.00	31.30	**15.20**

Table 30. Running time in seconds of different branching methods on 40 graphs with $s = 14$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**65.98**	OOT	OOT
G2	**0.03**	OOT	0.15	G22	OOT	OOT	OOT
G3	0.34	**0.33**	2.90	G23	1.08	OOT	**1.03**
G4	**5.36**	OOT	43.76	G24	**4.72**	OOT	OOT
G5	**0.91**	1.17	27.49	G25	**53.86**	OOT	135.74
G6	**1.45**	1.57	**1.45**	G26	**6.54**	OOT	30.88
G7	**0.00**	**0.00**	**0.00**	G27	OOT	OOT	OOT
G8	**0.00**	**0.00**	**0.00**	G28	2.74	OOT	**2.20**
G9	OOT	OOT	OOT	G29	2209.82	**151.4**	OOT
G10	OOT	OOT	OOT	G30	62.34	OOT	**57.22**
G11	OOT	OOT	OOT	G31	**111.62**	OOT	123.97
G12	2099.03	OOT	**665.82**	G32	4.34	4.59	**3.99**
G13	**4066.04**	OOT	OOT	G33	**179.04**	OOT	2678.96
G14	**3.01**	OOT	43.01	G34	**178.80**	OOT	2651.70
G15	**7.43**	OOT	16.46	G35	**117.44**	OOT	OOT
G16	**5.67**	OOT	49.56	G36	**87.82**	OOT	OOT
G17	**17.46**	OOT	20.81	G37	**7132.81**	OOT	OOT
G18	**1.93**	OOT	3.01	G38	OOT	OOT	OOT
G19	**1.43**	OOT	1.58	G39	50.13	47.67	**44.94**
G20	0.31	0.38	**0.30**	G40	32.52	**29.19**	42.29

Table 31. Running time in seconds of different branching methods on 40 graphs with $s = 15$

ID	Ours	-L	-Bin	ID	Ours	-L	-Bin
G1	OOT	OOT	OOT	G21	**2474.88**	OOT	OOT
G2	**0.03**	OOT	0.04	G22	OOT	OOT	OOT
G3	OOT	OOT	OOT	G23	**13.91**	OOT	913.49
G4	**0.20**	OOT	10.33	G24	**7.55**	OOT	OOT
G5	1.15	**1.01**	14.08	G25	**41.09**	OOT	111.87
G6	**1.52**	1.66	1.72	G26	**5.23**	OOT	21.41
G7	**0.00**	**0.00**	**0.00**	G27	OOT	OOT	OOT
G8	**0.01**	**0.01**	**0.01**	G28	**2.57**	OOT	2553.85
G9	**1492.15**	OOT	4547.55	G29	765.76	**325.13**	OOT
G10	OOT	OOT	OOT	G30	**60.17**	OOT	61.57
G11	OOT	OOT	OOT	G31	**103.52**	OOT	125.94
G12	**364.10**	OOT	951.36	G32	**3.97**	21.08	4.28
G13	**661.23**	OOT	1790.11	G33	**981.96**	OOT	OOT
G14	**6.89**	OOT	254.07	G34	**985.32**	OOT	OOT
G15	**13.48**	OOT	305.49	G35	**102.41**	OOT	541.33
G16	**5.18**	OOT	30.54	G36	**101.41**	OOT	383.04
G17	**11.43**	OOT	16.91	G37	**7817.05**	OOT	OOT
G18	**1.84**	OOT	6.75	G38	OOT	OOT	OOT
G19	7.63	OOT	**7.50**	G39	55.22	**54.88**	55.47
G20	**0.30**	0.38	0.39	G40	45.91	45.11	**20.23**

Table 32. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 2$

ID	Ours	-Heu	ID	Ours	-Heu
G1	**1454.82**	1457.43	G21	**313.49**	OOT
G2	**0.00**	**0.00**	G22	**0.62**	2.26
G3	**0.00**	**0.00**	G23	**0.12**	0.21
G4	**0.00**	**0.00**	G24	**0.06**	0.11
G5	**0.00**	**0.00**	G25	OOT	OOT
G6	0.01	**0.00**	G26	**59.30**	62.41
G7	**3.25**	3.29	G27	**7289.75**	9429.78
G8	**0.06**	0.08	G28	4.99	**2.76**
G9	**25.16**	26.78	G29	**119.69**	649.77
G10	**2719.72**	2740.47	G30	OOT	OOT
G11	5.89	**3.18**	G31	OOT	OOT
G12	1.33	**0.41**	G32	**4.92**	8.88
G13	18.77	**3.28**	G33	**9.04**	66.71
G14	0.94	**0.55**	G34	**9.03**	66.52
G15	3.71	**1.96**	G35	**7.48**	11.62
G16	**0.78**	1.34	G36	**10.00**	11.98
G17	**0.48**	0.99	G37	OOT	OOT
G18	**0.11**	0.19	G38	**2.65**	630.95
G19	2.84	**2.22**	G39	**27.11**	OOT
G20	**0.25**	0.46	G40	1596.32	**1453.09**

Table 33. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 3$

ID	Ours	-Heu	ID	Ours	-Heu
G1	**9450.59**	9585.53	G21	**916.53**	10344.82
G2	**0.00**	**0.00**	G22	**0.64**	5.90
G3	**0.00**	**0.00**	G23	**0.12**	0.21
G4	**0.00**	**0.00**	G24	**0.07**	0.11
G5	**0.00**	**0.00**	G25	OOT	OOT
G6	**0.00**	**0.00**	G26	**49.33**	52.26
G7	**496.34**	498.42	G27	**3249.62**	4151.27
G8	**0.08**	0.09	G28	4.69	**2.81**
G9	**333.57**	338.20	G29	**493.67**	1121.03
G10	**2590.33**	2646.86	G30	**2483.41**	OOT
G11	38.19	**33.24**	G31	**5243.18**	OOT
G12	1.41	**0.50**	G32	**4.06**	6.66
G13	10.63	**3.26**	G33	**38.49**	79.53
G14	0.69	**0.53**	G34	**38.45**	79.86
G15	2.59	**2.09**	G35	**7.38**	12.08
G16	1.55	**1.29**	G36	**6.91**	11.60
G17	**0.51**	0.88	G37	OOT	OOT
G18	**0.12**	0.19	G38	**2.67**	306.55
G19	**1.62**	2.24	G39	**26.12**	OOT
G20	**0.27**	0.41	G40	**1104.34**	1472.80

Table 34. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 4$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**211.44**	OOT
G2	**0.00**	**0.00**	G22	**0.68**	5.15
G3	**0.00**	**0.00**	G23	0.31	**0.24**
G4	**0.00**	**0.00**	G24	0.16	**0.13**
G5	0.01	**0.00**	G25	OOT	OOT
G6	**0.15**	0.20	G26	33.43	**32.72**
G7	**325.96**	327.84	G27	**1479.74**	1964.81
G8	**0.10**	0.12	G28	3.34	**2.94**
G9	2709.17	**2699.70**	G29	**518.81**	580.31
G10	**3026.86**	3071.14	G30	**4442.52**	OOT
G11	**560.54**	567.76	G31	**649.01**	OOT
G12	1.69	**0.34**	G32	**4.53**	6.19
G13	11.71	**3.03**	G33	**7.47**	34.82
G14	**0.42**	5.80	G34	**7.76**	34.90
G15	**167.75**	200.20	G35	**7.24**	11.59
G16	**1814.31**	1850.14	G36	**6.37**	17.19
G17	1.26	**0.82**	G37	OOT	OOT
G18	0.33	**0.20**	G38	**2.95**	231.31
G19	2.75	**2.10**	G39	**23.30**	OOT
G20	**0.34**	0.49	G40	**1932.74**	2256.83

Table 35. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 6$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**12.05**	3332.01
G2	0.03	**0.02**	G22	**1.22**	197.35
G3	**0.01**	**0.01**	G23	0.31	**0.23**
G4	0.06	**0.05**	G24	0.19	**0.18**
G5	0.02	**0.01**	G25	OOT	OOT
G6	0.58	**0.57**	G26	**44.29**	50.69
G7	936.93	**932.62**	G27	**1403.37**	1597.90
G8	**0.23**	0.88	G28	**3.22**	19.71
G9	OOT	OOT	G29	**300.39**	334.78
G10	OOT	OOT	G30	**4999.00**	OOT
G11	OOT	OOT	G31	**317.44**	OOT
G12	0.51	**0.47**	G32	**5.07**	6.62
G13	3.79	**3.02**	G33	**7.99**	55.26
G14	3.66	**2.74**	G34	**7.86**	55.88
G15	**1.53**	8.67	G35	**6.70**	11.12
G16	4.55	**2.83**	G36	**6.44**	10.92
G17	1.55	**0.82**	G37	**4414.43**	6053.71
G18	3.13	**3.09**	G38	**20.23**	259.55
G19	**1.47**	2.44	G39	**25.17**	OOT
G20	**0.30**	0.61	G40	**13.91**	58.72

Table 36. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 7$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**12.45**	3050.29
G2	**0.02**	**0.02**	G22	**1.20**	149.96
G3	**0.01**	**0.01**	G23	0.36	**0.25**
G4	0.07	**0.05**	G24	0.23	**0.14**
G5	**0.01**	**0.01**	G25	OOT	OOT
G6	**0.74**	0.75	G26	**34.20**	35.11
G7	**0.04**	0.52	G27	**1530.58**	1843.27
G8	**0.06**	0.11	G28	**4.22**	19.60
G9	OOT	OOT	G29	**287.79**	329.86
G10	OOT	OOT	G30	**1690.16**	OOT
G11	OOT	OOT	G31	**216.96**	OOT
G12	0.63	**0.50**	G32	5.20	**3.83**
G13	4.12	**3.11**	G33	**22.01**	84.81
G14	2.27	**1.62**	G34	**22.05**	84.87
G15	7.04	**4.79**	G35	**6.85**	11.83
G16	663.32	**612.03**	G36	**7.84**	11.08
G17	11.30	**10.48**	G37	**5568.08**	7337.52
G18	4.51	**4.31**	G38	**18.29**	261.13
G19	3.70	**2.57**	G39	**21.84**	OOT
G20	**0.32**	0.53	G40	**14.94**	195.49

Table 37. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 8$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**11.38**	3368.29
G2	**0.02**	**0.02**	G22	**4.84**	6075.37
G3	0.17	**0.14**	G23	0.28	**0.25**
G4	0.08	**0.06**	G24	0.32	**0.13**
G5	0.41	**0.29**	G25	OOT	OOT
G6	**0.71**	0.72	G26	**18.93**	19.17
G7	**0.03**	0.52	G27	**578.26**	695.65
G8	**0.37**	5.11	G28	**5.54**	30.74
G9	OOT	OOT	G29	**2514.25**	2943.94
G10	OOT	OOT	G30	**365.05**	OOT
G11	**2051.30**	2080.10	G31	**212.33**	OOT
G12	**0.48**	**0.48**	G32	**3.07**	3.86
G13	12.89	**10.24**	G33	**23.06**	107.18
G14	2.82	**2.14**	G34	**23.13**	106.73
G15	9.11	**7.20**	G35	**7.15**	10.89
G16	6.97	**5.88**	G36	**6.33**	11.03
G17	4.00	**3.47**	G37	**4619.60**	5521.90
G18	**8.63**	9.63	G38	**21.14**	431.52
G19	**1.40**	5.56	G39	**20.98**	OOT
G20	**0.32**	0.51	G40	**23.93**	248.08

Table 38. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 9$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**9.51**	3264.59
G2	18.58	**18.48**	G22	**2.80**	OOT
G3	**0.19**	0.25	G23	0.43	**0.30**
G4	0.08	**0.05**	G24	0.57	**0.42**
G5	**0.70**	**0.70**	G25	OOT	OOT
G6	1.03	**1.02**	G26	**20.04**	24.32
G7	**0.03**	0.52	G27	**794.10**	1036.86
G8	2.63	**2.58**	G28	**5.52**	55.30
G9	3277.60	**3274.77**	G29	**1203.13**	1442.71
G10	OOT	OOT	G30	**163.61**	OOT
G11	OOT	OOT	G31	**165.77**	OOT
G12	**2613.93**	2751.61	G32	**3.49**	4.14
G13	OOT	OOT	G33	**53.30**	151.50
G14	2.10	**1.64**	G34	**53.58**	153.16
G15	7.13	**5.11**	G35	**6.89**	10.02
G16	4.83	**3.22**	G36	**6.64**	10.72
G17	21.72	**17.62**	G37	**3167.43**	4015.16
G18	1.23	**0.88**	G38	**9.23**	1271.10
G19	**1.52**	5.45	G39	**26.64**	OOT
G20	**0.33**	0.52	G40	**21.73**	317.70

Table 39. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 10$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**6.83**	3324.34
G2	**38.29**	38.69	G22	**1.93**	OOT
G3	**0.31**	**0.31**	G23	0.36	**0.26**
G4	**428.75**	439.40	G24	0.63	**0.42**
G5	0.82	**0.64**	G25	**3699.51**	3901.88
G6	**1.22**	1.24	G26	**6.99**	7.60
G7	**0.03**	0.52	G27	**1217.38**	1457.95
G8	**0.09**	1.03	G28	**3.98**	36.78
G9	OOT	OOT	G29	OOT	OOT
G10	OOT	OOT	G30	**141.32**	OOT
G11	OOT	OOT	G31	**129.83**	OOT
G12	**998.43**	1082.71	G32	**3.43**	5.09
G13	6529.59	**6513.98**	G33	**53.77**	177.61
G14	2.74	**2.46**	G34	**53.69**	176.08
G15	9.14	**6.94**	G35	**7.08**	10.72
G16	6.31	**4.88**	G36	**6.81**	11.22
G17	4.93	**3.90**	G37	**3088.77**	4391.75
G18	4.50	**4.34**	G38	**8.77**	3489.89
G19	6.68	**6.11**	G39	**24.95**	OOT
G20	**0.36**	0.59	G40	**20.15**	249.55

Table 40. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 11$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**5.01**	3336.49
G2	**9.34**	9.43	G22	OOT	OOT
G3	**0.27**	0.35	G23	1.14	**0.78**
G4	**1046.56**	1060.38	G24	0.6	**0.40**
G5	**0.91**	0.92	G25	**687.31**	780.56
G6	1.32	**1.07**	G26	**8.30**	8.72
G7	**0.03**	0.52	G27	**1640.61**	1946.01
G8	**0.04**	0.47	G28	**2.81**	44.54
G9	OOT	OOT	G29	**317.23**	415.84
G10	OOT	OOT	G30	**168.19**	OOT
G11	OOT	OOT	G31	**115.82**	OOT
G12	**1754.41**	1888.05	G32	**4.64**	7.83
G13	**2803.32**	2795.84	G33	**51.39**	408.37
G14	1.97	**1.43**	G34	**51.82**	410.34
G15	7.01	**6.12**	G35	**7.15**	11.48
G16	4.53	**3.38**	G36	**6.29**	10.71
G17	**9.92**	9.99	G37	**3723.98**	5469.50
G18	**1.53**	1.66	G38	**25.45**	9740.39
G19	7.08	**5.97**	G39	**32.78**	OOT
G20	**0.39**	0.53	G40	**29.3**	423.06

Table 41. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 12$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**23.88**	3271.42
G2	1.36	**1.17**	G22	OOT	OOT
G3	**0.35**	0.39	G23	1.16	**0.69**
G4	**297.94**	318.00	G24	0.98	**0.76**
G5	0.98	**0.96**	G25	**929.47**	1027.13
G6	**1.38**	**1.38**	G26	**7.79**	9.65
G7	**0.00**	0.52	G27	**1910.62**	2308.76
G8	**0.02**	0.16	G28	**2.42**	81.06
G9	OOT	OOT	G29	**69.83**	144.33
G10	OOT	OOT	G30	**71.08**	OOT
G11	OOT	OOT	G31	**113.55**	OOT
G12	**552.56**	578.98	G32	**3.74**	7.76
G13	OOT	OOT	G33	**46.57**	371.62
G14	2.90	**1.58**	G34	**46.55**	369.22
G15	7.54	**4.76**	G35	**7.13**	11.69
G16	4.80	**3.23**	G36	**7.85**	12.18
G17	6.76	**6.42**	G37	**4923.81**	6352.21
G18	1.45	**1.22**	G38	**156.21**	OOT
G19	**6.64**	10.91	G39	**35.15**	OOT
G20	**0.40**	0.54	G40	**35.43**	446.62

Table 42. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 13$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**104.48**	OOT
G2	**0.04**	0.05	G22	OOT	OOT
G3	0.33	**0.30**	G23	0.89	**0.67**
G4	**52.57**	52.94	G24	2.25	**1.98**
G5	0.85	**0.82**	G25	**157.56**	240.85
G6	1.35	**1.34**	G26	**8.15**	9.52
G7	**0.00**	0.41	G27	**2240.02**	2569.43
G8	**0.06**	0.07	G28	**2.24**	189.53
G9	**64.45**	65.35	G29	**63.11**	134.77
G10	OOT	OOT	G30	**64.52**	OOT
G11	OOT	OOT	G31	**110.85**	OOT
G12	**211.93**	230.14	G32	**3.87**	15.31
G13	OOT	OOT	G33	**219.37**	641.80
G14	2.33	**1.54**	G34	**218.22**	641.02
G15	7.17	**4.82**	G35	**7.09**	25.77
G16	4.73	**3.26**	G36	**6.53**	26.91
G17	8.56	**7.68**	G37	**6172.27**	7828.57
G18	**3.63**	3.80	G38	**810.85**	OOT
G19	**8.18**	10.25	G39	**42.47**	OOT
G20	**0.28**	0.49	G40	**44.00**	568.82

Table 43. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 14$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**65.98**	OOT
G2	**0.03**	154.47	G22	OOT	OOT
G3	0.34	**0.29**	G23	1.08	**0.74**
G4	**5.36**	5.38	G24	4.72	**4.57**
G5	0.91	**0.81**	G25	**53.86**	161.63
G6	**1.45**	1.46	G26	**6.54**	8.08
G7	**0.00**	0.41	G27	OOT	OOT
G8	**0.00**	0.04	G28	**2.74**	473.84
G9	OOT	OOT	G29	**2209.82**	2261.00
G10	OOT	OOT	G30	**62.34**	OOT
G11	OOT	OOT	G31	**111.62**	OOT
G12	**2099.03**	2293.45	G32	**4.34**	16.50
G13	**4066.04**	4124.45	G33	**179.04**	1550.97
G14	3.01	**2.20**	G34	**178.80**	1552.28
G15	7.43	**6.44**	G35	**117.44**	124.89
G16	5.67	**3.86**	G36	**87.82**	92.13
G17	17.46	**17.26**	G37	**7132.81**	8986.52
G18	1.93	**1.78**	G38	OOT	OOT
G19	**1.43**	13.76	G39	**50.13**	OOT
G20	**0.31**	0.45	G40	**32.52**	732.47

Table 44. Running time in seconds of verifying our proposed heuristic method on 40 graphs with $s = 15$

ID	Ours	-Heu	ID	Ours	-Heu
G1	OOT	OOT	G21	**2474.88**	OOT
G2	**0.03**	44.56	G22	OOT	OOT
G3	OOT	OOT	G23	13.91	**13.43**
G4	0.20	**0.19**	G24	7.55	**7.28**
G5	1.15	**0.78**	G25	**41.09**	190.21
G6	**1.52**	1.72	G26	**5.23**	6.92
G7	**0.00**	0.52	G27	OOT	OOT
G8	**0.01**	0.02	G28	**2.57**	1206.65
G9	1492.15	**1483.69**	G29	**765.76**	854.69
G10	OOT	OOT	G30	**60.17**	OOT
G11	OOT	OOT	G31	**103.52**	OOT
G12	**364.10**	395.36	G32	**3.97**	36.38
G13	**661.23**	664.84	G33	**981.96**	2029.18
G14	6.89	**5.83**	G34	**985.32**	2027.83
G15	13.48	**10.78**	G35	**102.41**	106.40
G16	5.18	**3.82**	G36	**101.41**	106.10
G17	11.43	**9.60**	G37	**7817.05**	9829.89
G18	1.84	**1.48**	G38	OOT	OOT
G19	**7.63**	10.92	G39	**55.22**	OOT
G20	**0.30**	0.43	G40	**45.91**	877.76

Table 45. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 2$

ID	Ours	-UB	ID	Ours	-UB
G1	**1454.82**	1458.62	G21	**313.49**	315.46
G2	**0.00**	**0.00**	G22	**0.62**	0.67
G3	**0.00**	**0.00**	G23	**0.12**	**0.12**
G4	**0.00**	**0.00**	G24	**0.06**	**0.06**
G5	**0.00**	**0.00**	G25	OOT	OOT
G6	**0.01**	**0.01**	G26	**59.30**	61.27
G7	3.25	**3.12**	G27	7289.75	**7273.99**
G8	**0.06**	0.07	G28	**4.99**	5.27
G9	25.16	**24.15**	G29	**119.69**	121.64
G10	**2719.72**	2737.13	G30	OOT	OOT
G11	**5.89**	5.91	G31	OOT	OOT
G12	**1.33**	1.36	G32	4.92	**4.05**
G13	18.77	**18.36**	G33	**9.04**	9.17
G14	0.94	**0.79**	G34	**9.03**	9.11
G15	3.71	**3.28**	G35	7.48	**7.03**
G16	**0.78**	**0.78**	G36	10.00	**6.83**
G17	**0.48**	0.50	G37	OOT	OOT
G18	**0.11**	0.12	G38	**2.65**	2.69
G19	**2.84**	3.22	G39	27.11	**26.59**
G20	**0.25**	0.26	G40	**1596.32**	1975.11

Table 46. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 3$

ID	Ours	-UB	ID	Ours	-UB
G1	**9450.59**	9479.49	G21	**916.53**	920.61
G2	**0.00**	**0.00**	G22	0.64	**0.61**
G3	**0.00**	**0.00**	G23	**0.12**	0.13
G4	**0.00**	**0.00**	G24	0.07	**0.06**
G5	**0.00**	**0.00**	G25	OOT	OOT
G6	**0.00**	**0.00**	G26	49.33	**48.94**
G7	496.34	**426.21**	G27	**3249.62**	3255.81
G8	0.08	**0.07**	G28	**4.69**	4.96
G9	333.57	**312.05**	G29	493.67	**471.67**
G10	**2590.33**	2616.28	G30	2483.41	**2389.15**
G11	**38.19**	43.04	G31	**5243.18**	5254.51
G12	**1.41**	**1.41**	G32	**4.06**	4.56
G13	**10.63**	10.75	G33	**38.49**	38.70
G14	0.69	**0.68**	G34	**38.45**	38.68
G15	**2.59**	**2.59**	G35	**7.38**	7.63
G16	1.55	**1.47**	G36	**6.91**	7.20
G17	**0.51**	0.55	G37	OOT	OOT
G18	**0.12**	0.13	G38	**2.67**	2.68
G19	1.62	**1.48**	G39	**26.12**	27.95
G20	**0.27**	0.30	G40	**1104.34**	1233.96

Table 47. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 4$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**211.44**	212.34
G2	**0.00**	**0.00**	G22	**0.68**	0.77
G3	**0.00**	**0.00**	G23	0.31	**0.30**
G4	**0.00**	**0.00**	G24	**0.16**	0.18
G5	**0.01**	**0.01**	G25	OOT	OOT
G6	**0.15**	0.20	G26	33.43	**32.72**
G7	325.96	**247.49**	G27	1479.74	**1458.49**
G8	**0.10**	**0.10**	G28	**3.34**	3.40
G9	2709.17	**2505.71**	G29	518.81	**448.04**
G10	**3026.86**	3046.25	G30	4442.52	**4070.11**
G11	**560.54**	606.02	G31	649.01	**626.54**
G12	1.69	**1.64**	G32	4.53	**4.01**
G13	11.71	**11.60**	G33	**7.47**	7.77
G14	0.42	**0.39**	G34	7.76	**7.47**
G15	167.75	**164.18**	G35	7.24	**6.89**
G16	**1814.31**	1900.23	G36	**6.37**	6.85
G17	1.26	**1.03**	G37	OOT	OOT
G18	0.33	**0.32**	G38	2.95	**2.71**
G19	**2.75**	2.76	G39	23.30	**23.26**
G20	0.34	**0.26**	G40	**1932.74**	2000.91

Table 48. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 5$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**10.87**	11.08
G2	**0.02**	0.05	G22	1.86	**1.67**
G3	**0.00**	**0.00**	G23	**0.23**	0.29
G4	**0.03**	0.14	G24	**0.17**	0.19
G5	0.02	**0.01**	G25	OOT	OOT
G6	0.40	**0.39**	G26	31.16	**30.17**
G7	61.17	**38.62**	G27	**767.88**	778.12
G8	0.12	**0.10**	G28	3.63	**3.61**
G9	891.54	**824.62**	G29	**215.02**	247.63
G10	**4787.32**	4860.88	G30	5324.32	**4589.67**
G11	**39.64**	39.92	G31	440.31	**427.07**
G12	**0.56**	0.57	G32	**3.90**	4.81
G13	**4.41**	4.44	G33	**20.93**	21.07
G14	**2.29**	2.58	G34	**21.11**	21.40
G15	8.85	**8.51**	G35	7.80	**6.98**
G16	**6.82**	7.81	G36	**6.72**	6.91
G17	**1.17**	1.20	G37	8112.97	**8104.03**
G18	0.33	**0.32**	G38	**4.46**	**4.46**
G19	**1.45**	1.57	G39	**20.68**	22.40
G20	0.30	**0.28**	G40	959.84	**889.86**

Table 49. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 6$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**12.05**	12.18
G2	**0.03**	0.04	G22	**1.22**	1.26
G3	**0.01**	**0.01**	G23	0.31	**0.29**
G4	**0.06**	0.23	G24	**0.19**	**0.19**
G5	0.02	**0.01**	G25	OOT	OOT
G6	**0.58**	0.59	G26	**44.29**	47.43
G7	936.93	**791.14**	G27	**1403.37**	1634.68
G8	0.23	**0.20**	G28	3.22	**2.75**
G9	OOT	OOT	G29	**300.39**	1597.49
G10	OOT	OOT	G30	4999.00	**4526.86**
G11	OOT	OOT	G31	317.44	**307.57**
G12	**0.51**	0.65	G32	5.07	**3.43**
G13	**3.79**	3.80	G33	7.99	**7.84**
G14	**3.66**	**3.66**	G34	**7.86**	8.01
G15	1.53	**1.26**	G35	6.70	**6.67**
G16	**4.55**	4.81	G36	**6.44**	8.25
G17	1.55	**1.51**	G37	**4414.43**	4549.44
G18	**3.13**	3.78	G38	20.23	**19.83**
G19	**1.47**	1.53	G39	25.17	**22.54**
G20	0.30	**0.29**	G40	13.91	**9.19**

Table 50. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 7$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**12.45**	12.68
G2	**0.02**	0.03	G22	**1.20**	1.33
G3	**0.01**	**0.01**	G23	**0.36**	**0.36**
G4	**0.07**	0.11	G24	0.23	**0.20**
G5	**0.01**	**0.01**	G25	OOT	OOT
G6	0.74	**0.58**	G26	**34.20**	38.88
G7	0.04	**0.02**	G27	**1530.58**	2231.75
G8	0.06	**0.05**	G28	4.22	**3.54**
G9	OOT	OOT	G29	**287.79**	7467.08
G10	OOT	OOT	G30	**1690.16**	1813.48
G11	OOT	OOT	G31	**216.96**	223.46
G12	0.63	**0.48**	G32	5.20	**4.30**
G13	4.12	**3.77**	G33	**22.01**	23.08
G14	**2.27**	2.65	G34	**22.05**	23.46
G15	7.04	**6.85**	G35	**6.85**	7.24
G16	**663.32**	680.43	G36	7.84	**7.22**
G17	11.30	**10.99**	G37	**5568.08**	6020.65
G18	4.51	**4.40**	G38	18.29	**18.16**
G19	**3.70**	4.02	G39	21.84	**21.37**
G20	**0.32**	0.34	G40	14.94	**14.37**

Table 51. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 8$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**11.38**	12.71
G2	**0.02**	0.04	G22	4.84	**4.80**
G3	**0.17**	9.29	G23	**0.28**	0.41
G4	**0.08**	0.14	G24	0.32	**0.24**
G5	**0.41**	83.67	G25	OOT	OOT
G6	**0.71**	0.94	G26	**18.93**	22.35
G7	**0.03**	**0.03**	G27	**578.26**	1039.29
G8	0.37	**0.26**	G28	5.54	**5.49**
G9	OOT	OOT	G29	**2514.25**	OOT
G10	OOT	OOT	G30	**365.05**	457.12
G11	**2051.30**	2078.74	G31	**212.33**	223.46
G12	**0.48**	0.59	G32	**3.07**	3.67
G13	**12.89**	14.07	G33	**23.06**	25.79
G14	2.82	**2.59**	G34	**23.13**	25.51
G15	9.11	**9.05**	G35	7.15	**7.02**
G16	**6.97**	7.04	G36	**6.33**	6.34
G17	4.00	**3.90**	G37	**4619.60**	5210.41
G18	**8.63**	9.48	G38	21.14	**20.53**
G19	**1.40**	1.49	G39	**20.98**	22.36
G20	0.32	**0.31**	G40	23.93	**18.15**

Table 52. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 9$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**9.51**	10.98
G2	**18.58**	**18.58**	G22	2.80	**2.70**
G3	**0.19**	7.48	G23	0.43	**0.32**
G4	**0.08**	0.26	G24	0.57	**0.49**
G5	**0.70**	156.62	G25	OOT	OOT
G6	1.03	**0.93**	G26	**20.04**	25.53
G7	0.03	**0.02**	G27	**794.10**	2473.15
G8	2.63	**1.96**	G28	**5.52**	5.82
G9	3277.60	**3051.29**	G29	**1203.13**	OOT
G10	OOT	OOT	G30	**163.61**	207.58
G11	OOT	OOT	G31	**165.77**	178.65
G12	2613.93	**2583.63**	G32	**3.49**	4.17
G13	OOT	OOT	G33	**53.30**	94.98
G14	2.10	**2.07**	G34	**53.58**	95.03
G15	**7.13**	7.27	G35	**6.89**	**6.89**
G16	4.83	**4.73**	G36	**6.64**	7.10
G17	21.72	**17.99**	G37	**3167.43**	3678.27
G18	1.23	**1.09**	G38	**9.23**	9.50
G19	**1.52**	1.63	G39	26.64	**26.18**
G20	0.33	**0.28**	G40	**21.73**	31.85

Table 53. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 10$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**6.83**	8.39
G2	**38.29**	38.49	G22	**1.93**	2.02
G3	**0.31**	7.03	G23	**0.36**	0.38
G4	**428.75**	432.69	G24	**0.63**	0.65
G5	**0.82**	81.45	G25	**3699.51**	3806.17
G6	1.22	**1.19**	G26	**6.99**	8.57
G7	**0.03**	**0.03**	G27	**1217.38**	5449.56
G8	0.09	**0.08**	G28	3.98	**3.85**
G9	OOT	OOT	G29	OOT	OOT
G10	OOT	OOT	G30	**141.32**	180.24
G11	OOT	OOT	G31	**129.83**	140.24
G12	**998.43**	1002.41	G32	**3.43**	3.85
G13	**6529.59**	6578.02	G33	**53.77**	109.03
G14	**2.74**	2.79	G34	**53.69**	108.26
G15	9.14	**9.07**	G35	**7.08**	7.28
G16	6.31	**6.12**	G36	6.81	**6.53**
G17	4.93	**4.70**	G37	**3088.77**	4564.73
G18	**4.50**	4.73	G38	8.77	**8.32**
G19	**6.68**	6.80	G39	**24.95**	27.51
G20	0.36	**0.34**	G40	**20.15**	21.98

Table 54. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 11$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**5.01**	6.47
G2	**9.34**	9.73	G22	OOT	OOT
G3	**0.27**	7.67	G23	1.14	**0.89**
G4	**1046.56**	1059.34	G24	**0.60**	0.64
G5	**0.91**	40.44	G25	**687.31**	701.66
G6	**1.32**	1.34	G26	**8.30**	12.30
G7	0.03	**0.02**	G27	**1640.61**	10482.39
G8	**0.04**	**0.04**	G28	2.81	**2.42**
G9	OOT	**8774.69**	G29	**317.23**	OOT
G10	OOT	OOT	G30	**168.19**	282.66
G11	OOT	OOT	G31	115.82	**110.94**
G12	**1754.41**	1800.73	G32	4.64	**3.83**
G13	**2803.32**	2820.74	G33	**51.39**	112.03
G14	**1.97**	2.12	G34	**51.82**	111.91
G15	**7.01**	7.03	G35	**7.15**	7.32
G16	**4.53**	4.78	G36	**6.29**	7.00
G17	**9.92**	10.14	G37	**3723.98**	8768.76
G18	**1.53**	1.72	G38	**25.45**	29.99
G19	**7.08**	7.27	G39	32.78	**29.81**
G20	**0.39**	0.75	G40	29.3	**28.02**

Table 55. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 12$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**23.88**	31.45
G2	**1.36**	1.57	G22	OOT	OOT
G3	**0.35**	131.57	G23	**1.16**	1.18
G4	**297.94**	304.56	G24	0.98	**0.97**
G5	**0.98**	29.87	G25	**929.47**	1005.84
G6	1.38	**1.33**	G26	**7.79**	12.90
G7	**0.00**	**0.00**	G27	**1910.62**	OOT
G8	**0.02**	**0.02**	G28	**2.42**	2.49
G9	OOT	OOT	G29	**69.83**	OOT
G10	OOT	OOT	G30	71.08	**67.79**
G11	OOT	OOT	G31	113.55	**107.58**
G12	552.56	**552.21**	G32	**3.74**	3.76
G13	OOT	OOT	G33	**46.57**	104.84
G14	2.90	**2.24**	G34	**46.55**	105.18
G15	7.54	**7.18**	G35	**7.13**	7.36
G16	**4.80**	**4.80**	G36	7.85	**7.22**
G17	**6.76**	6.91	G37	**4923.81**	OOT
G18	**1.45**	1.60	G38	**156.21**	210.81
G19	**6.64**	7.14	G39	35.15	**34.01**
G20	**0.40**	1.15	G40	35.43	**33.80**

Table 56. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 13$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**104.48**	224.72
G2	**0.04**	0.37	G22	OOT	OOT
G3	**0.33**	1899.11	G23	**0.89**	0.90
G4	**52.57**	63.56	G24	2.25	**2.17**
G5	**0.85**	3020.91	G25	**157.56**	164.60
G6	**1.35**	1.52	G26	**8.15**	12.94
G7	**0.00**	**0.00**	G27	**2240.02**	OOT
G8	**0.06**	**0.06**	G28	**2.24**	2.44
G9	64.45	**40.09**	G29	**63.11**	OOT
G10	OOT	OOT	G30	64.52	**63.29**
G11	OOT	OOT	G31	**110.85**	116.74
G12	**211.93**	226.09	G32	**3.87**	4.59
G13	OOT	OOT	G33	**219.37**	1093.14
G14	2.33	**2.14**	G34	**218.22**	1084.93
G15	**7.17**	7.42	G35	7.09	**6.90**
G16	**4.73**	4.87	G36	**6.53**	6.82
G17	**8.56**	8.90	G37	**6172.27**	OOT
G18	**3.63**	3.85	G38	**810.85**	3893.72
G19	8.18	**7.13**	G39	**42.47**	44.52
G20	**0.28**	1.46	G40	44.00	**35.29**

Table 57. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 14$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**65.98**	139.00
G2	**0.03**	0.16	G22	OOT	OOT
G3	**0.34**	8781.10	G23	**1.08**	1.01
G4	**5.36**	18.07	G24	**4.72**	4.73
G5	**0.91**	OOT	G25	53.86	**52.89**
G6	**1.45**	1.70	G26	**6.54**	10.19
G7	**0.00**	**0.00**	G27	OOT	OOT
G8	**0.00**	**0.00**	G28	2.74	**2.57**
G9	OOT	OOT	G29	**2209.82**	OOT
G10	OOT	OOT	G30	62.34	**55.73**
G11	OOT	OOT	G31	**111.62**	114.43
G12	**2099.03**	2144.39	G32	4.34	**3.89**
G13	4066.04	**4033.97**	G33	**179.04**	902.81
G14	3.01	**2.85**	G34	**178.80**	901.64
G15	**7.43**	7.52	G35	**117.44**	198.38
G16	5.67	**5.29**	G36	**87.82**	144.14
G17	17.46	**16.47**	G37	**7132.81**	OOT
G18	1.93	**1.85**	G38	OOT	OOT
G19	**1.43**	1.48	G39	50.13	**47.85**
G20	**0.31**	1.03	G40	**32.52**	35.01

Table 58. Running time in seconds of verifying our proposed upper bounds on 40 graphs with $s = 15$

ID	Ours	-UB	ID	Ours	-UB
G1	OOT	OOT	G21	**2474.88**	OOT
G2	**0.03**	0.05	G22	OOT	OOT
G3	OOT	OOT	G23	**13.91**	14.15
G4	**0.20**	8.16	G24	7.55	**7.51**
G5	**1.15**	OOT	G25	41.09	**38.27**
G6	**1.52**	2.04	G26	**5.23**	6.53
G7	**0.00**	**0.00**	G27	OOT	OOT
G8	**0.01**	**0.01**	G28	2.57	**2.45**
G9	1492.15	**450.98**	G29	**765.76**	6353.92
G10	OOT	OOT	G30	60.17	**58.63**
G11	OOT	OOT	G31	**103.52**	110.38
G12	364.10	**358.85**	G32	3.97	**3.93**
G13	**661.23**	670.33	G33	**981.96**	10040.68
G14	6.89	**6.51**	G34	**985.32**	9990.51
G15	13.48	**12.97**	G35	**102.41**	104.59
G16	**5.18**	5.56	G36	**101.41**	105.50
G17	11.43	**10.28**	G37	**7817.05**	OOT
G18	1.84	**1.79**	G38	OOT	OOT
G19	7.63	**7.06**	G39	55.22	**52.66**
G20	**0.30**	0.94	G40	45.91	**42.64**

Table 59. Detailed information of our heuristic solutions on 40 graphs with $s = 2$

ID	lb	real	time (s)	n'	m'
G1	**13**	13	0.05	200	9876
G2	**12**	12	0	90	729
G3	**24**	24	0	24	276
G4	**14**	14	0	140	1351
G5	**26**	26	0	260	4771
G6	**64**	64	0.01	128	5056
G7	**32**	32	0	64	1824
G8	**6**	6	0	64	480
G9	**14**	14	0	70	1855
G10	**10**	10	0.08	296	10703
G11	**8**	8	5.75	0	0
G12	**6**	6	1.29	0	0
G13	**6**	6	17.02	0	0
G14	**4**	4	0.87	0	0
G15	**4**	4	3.69	0	0
G16	**4**	4	0.67	0	0
G17	**5**	5	0.44	0	0
G18	**5**	5	0.09	0	0
G19	**5**	5	2.73	0	0
G20	**18**	18	0.23	0	0
G21	**19**	19	0.34	267	7961
G22	**6**	6	0.51	0	0
G23	**5**	5	0.06	0	0
G24	**5**	5	0.05	0	0
G25	274	OOT	8.17	380	69843
G26	**38**	38	0.46	264	11792
G27	**18**	18	0.63	798	35684
G28	**12**	12	4.92	0	0
G29	**402**	402	3.24	445	98338
G30	52	OOT	85.66	3146	190551
G31	57	OOT	145.45	2808	200325
G32	**31**	31	4.7	0	0
G33	**20**	20	0.44	253	7012
G34	**20**	20	0.41	253	7012
G35	**9**	9	7.41	0	0
G36	**9**	9	6.68	0	0
G37	62	OOT	7.94	657	73601
G38	**69**	69	0.81	185	8195
G39	**334**	334	26.69	0	0
G40	**872**	872	2.62	919	421245

Table 60. Detailed information of our heuristic solutions on 40 graphs with $s = 3$

ID	lb	real	time (s)	n'	m'
G1	14	OOT	0.05	200	9876
G2	**12**	12	0	200	1534
G3	**24**	24	0	200	3235
G4	**14**	14	0	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0	500	23191
G7	**32**	32	0	64	1824
G8	**8**	8	0	64	480
G9	**17**	18	0.01	70	1855
G10	12	OOT	0.08	296	10703
G11	**8**	8	5.85	4802000	42681372
G12	**6**	6	1.16	2073022	6219018
G13	**6**	7	9.58	16584963	49754842
G14	**5**	5	0.66	0	0
G15	**5**	5	2.49	0	0
G16	**5**	5	1.48	0	0
G17	**5**	5	0.48	72	132
G18	**5**	5	0.11	12	20
G19	**5**	6	1.59	2205	3499
G20	**18**	19	0.26	166	1505
G21	20	OOT	0.34	460	15432
G22	**8**	8	0.56	0	0
G23	**5**	6	0.11	4454	7393
G24	**5**	6	0.06	916	1491
G25	278	OOT	8.7	379	69537
G26	**42**	42	0.39	244	10546
G27	21	OOT	0.52	660	29604
G28	**13**	13	4.61	0	0
G29	**402**	402	3.23	848	179346
G30	58	OOT	83.08	1301	75086
G31	63	OOT	111.42	1342	94594
G32	**31**	32	4.03	79	1426
G33	**21**	21	0.36	387	12217
G34	**21**	21	0.34	387	12217
G35	**10**	10	6.51	0	0
G36	**10**	10	6.66	0	0
G37	67	OOT	7.62	550	59787
G38	**71**	71	0.59	185	8195
G39	**335**	335	22.26	0	0
G40	875	OOT	4.47	919	421245

Table 61. Detailed information of our heuristic solutions on 40 graphs with $s = 4$

ID	lb	real	time (s)	n'	m'
G1	16	OOT	0.04	200	9876
G2	**12**	12	0	200	1534
G3	**24**	24	0	200	3235
G4	**14**	14	0	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0	500	23191
G7	32	40	0	64	1824
G8	**10**	10	0	64	480
G9	20	OOT	0	70	1855
G10	13	OOT	0.06	300	10842
G11	**8**	8	6.42	4802000	42681372
G12	**8**	8	1.45	0	0
G13	**8**	8	11.04	0	0
G14	4	6	0.33	0	0
G15	**5**	6	3.57	12057441	18082179
G16	**5**	6	2.69	7122792	10680777
G17	**6**	6	1.09	72	132
G18	**6**	7	0.27	12	20
G19	**6**	7	2.67	2205	3499
G20	**19**	20	0.31	169	1568
G21	**23**	23	0.27	267	7961
G22	**10**	10	0.6	41	232
G23	**7**	7	0.24	0	0
G24	**6**	7	0.15	916	1491
G25	285	OOT	9.35	378	69219
G26	**45**	46	0.37	236	10046
G27	24	OOT	0.55	544	23568
G28	**14**	14	3.29	0	0
G29	**407**	407	4.02	444	97926
G30	62	OOT	79.26	1044	61662
G31	68	71	104.87	658	41785
G32	**32**	32	4.02	82	1539
G33	**24**	24	0.38	253	7012
G34	**24**	24	0.37	253	7012
G35	**11**	11	6.72	0	0
G36	**11**	11	6.22	0	0
G37	73	OOT	8.27	434	44757
G38	**73**	74	0.62	185	8195
G39	**337**	337	22.34	0	0
G40	**878**	878	6.72	908	411403

Table 62. Detailed information of our heuristic solutions on 40 graphs with $s = 5$

ID	lb	real	time (s)	n'	m'
G1	19	OOT	0.04	200	9876
G2	**12**	12	0.00	200	1K
G3	**24**	24	0.00	200	3K
G4	**14**	14	0.00	500	4K
G5	**26**	26	0.00	500	9K
G6	**64**	64	0.00	500	23K
G7	36	48	0.00	64	1K
G8	**12**	12	0.00	64	480
G9	25	28	0.00	70	1K
G10	16	OOT	0.06	296	10K
G11	**9**	10	4.68	4M	42M
G12	**9**	9	0.46	0	0
G13	**9**	9	4.18	0	0
G14	**6**	7	1.08	4M	6M
G15	**6**	7	4.07	12M	18M
G16	**6**	7	2.56	7M	10M
G17	**7**	7	1.02	72	132
G18	**7**	7	0.29	12	20
G19	**7**	8	1.44	2K	3K
G20	**20**	20	0.28	172	1K
G21	**26**	26	0.26	139	3K
G22	**11**	11	1.44	35	197
G23	**8**	8	0.15	0	0
G24	**7**	8	0.15	916	1K
G25	290	OOT	9.89	378	69K
G26	**48**	49	0.37	130	5K
G27	26	OOT	0.40	544	23K
G28	**15**	15	3.59	0	0
G29	410	412	3.53	444	97K
G30	66	OOT	78.55	805	49K
G31	73	76	113.83	490	31K
G32	**33**	34	3.83	85	1K
G33	**25**	26	0.30	387	12K
G34	**25**	26	0.37	387	12K
G35	**13**	13	7.27	0	0
G36	**13**	13	6.18	0	0
G37	77	OOT	6.70	390	39K
G38	**74**	75	0.55	202	9K
G39	**339**	339	20.44	0	0
G40	**881**	881	9.65	903	406K

Table 63. Detailed information of our heuristic solutions on 40 graphs with $s = 6$

ID	lb	real	time (s)	n'	m'
G1	21	OOT	0.04	200	9876
G2	**12**	12	0	200	1534
G3	**24**	24	0	200	3235
G4	**14**	14	0	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0.01	500	23191
G7	44	52	0.01	64	1824
G8	**13**	13	0	64	480
G9	29	OOT	0.01	70	1855
G10	17	OOT	0.07	300	10842
G11	11	OOT	2.43	4802000	42681372
G12	**10**	10	0.49	0	0
G13	**10**	10	3.48	0	0
G14	6	8	1.59	4588484	6879133
G15	5	8	1.16	0	0
G16	6	8	2.48	7122792	10680777
G17	**7**	8	1.47	5422578	5845273
G18	**7**	8	0.32	1837699	2062249
G19	**8**	9	1.44	2205	3499
G20	**21**	21	0.22	172	1621
G21	**28**	28	0.28	139	3592
G22	**13**	13	1.19	15	67
G23	**8**	9	0.28	4454	7393
G24	**8**	9	0.18	916	1491
G25	294	OOT	8.84	376	68577
G26	**51**	51	0.32	122	5384
G27	28	OOT	0.43	544	23568
G28	**16**	16	2.87	0	0
G29	**415**	415	4.94	443	97508
G30	70	OOT	72.28	627	38255
G31	78	80	101.83	306	19084
G32	**35**	36	3.66	53	1150
G33	**28**	28	0.42	253	7012
G34	**28**	28	0.43	253	7012
G35	**15**	15	6.45	0	0
G36	**15**	15	6.22	0	0
G37	82	OOT	5.91	337	32200
G38	**75**	76	0.54	286	16610
G39	**341**	341	20.93	0	0
G40	**884**	884	11.94	0	0

Table 64. Detailed information of our heuristic solutions on 40 graphs with $s = 7$

ID	lb	real	time (s)	n'	m'
G1	23	OOT	0.04	200	9876
G2	**13**	13	0	200	1534
G3	**24**	24	0	200	3235
G4	**14**	14	0	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0	500	23191
G7	33	64	0.02	0	0
G8	**16**	16	0	64	480
G9	32	OOT	0	70	1855
G10	19	OOT	0.06	300	10842
G11	12	OOT	2.33	4802000	42681372
G12	**11**	11	0.47	0	0
G13	**11**	11	3.92	0	0
G14	7	9	1.14	4588484	6879133
G15	**8**	9	3.69	12057441	18082179
G16	7	9	2.57	7122792	10680777
G17	**8**	9	1.29	5422578	5845273
G18	**8**	9	0.34	1837699	2062249
G19	8	10	3.23	9991507	12843104
G20	**22**	22	0.27	172	1623
G21	**30**	30	0.26	138	3569
G22	**15**	15	1.11	0	0
G23	**9**	10	0.27	4454	7393
G24	**9**	10	0.16	916	1491
G25	301	OOT	9.11	371	66990
G26	**53**	54	0.34	122	5384
G27	31	OOT	0.41	374	17042
G28	**17**	17	3.8	0	0
G29	**418**	418	6.41	442	97091
G30	76	OOT	73.58	323	20209
G31	**84**	84	107.5	275	16192
G32	**38**	39	4.09	50	1047
G33	**29**	30	0.31	387	12217
G34	**29**	30	0.32	387	12217
G35	**17**	17	6.72	0	0
G36	**17**	17	7.21	0	0
G37	84	OOT	7.14	337	32200
G38	**77**	78	0.66	280	16190
G39	**343**	343	21.33	0	0
G40	**885**	885	13.76	0	0

Table 65. Detailed information of our heuristic solutions on 40 graphs with $s = 8$

ID	lb	real	time (s)	n'	m'
G1	26	OOT	0.04	200	9876
G2	**14**	14	0	200	1534
G3	**24**	24	0	200	3235
G4	**15**	15	0.01	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0	500	23191
G7	33	64	0.02	0	0
G8	**16**	16	0	64	480
G9	36	OOT	0.01	70	1855
G10	21	OOT	0.06	300	10842
G11	**14**	14	2.46	4801992	42681324
G12	**12**	12	0.47	0	0
G13	**11**	12	5.16	16584963	49754842
G14	8	10	1.01	4588484	6879133
G15	8	10	3.32	12057441	18082179
G16	8	10	2.57	7122792	10680777
G17	**9**	10	1.41	5422578	5845273
G18	**9**	10	0.36	1837699	2062249
G19	**10**	11	1.37	2205	3499
G20	**23**	23	0.27	172	1624
G21	**32**	32	0.25	134	3473
G22	**16**	17	1.49	3065	37147
G23	**10**	11	0.24	4454	7393
G24	9	11	0.21	873219	1327171
G25	307	OOT	10.68	365	65084
G26	55	57	0.31	122	5384
G27	33	OOT	0.4	374	17042
G28	**18**	18	4.99	0	0
G29	**419**	419	3.83	443	97508
G30	**81**	82	63.92	292	17416
G31	85	87	93.09	286	17225
G32	**40**	41	2.92	50	1047
G33	**31**	32	0.38	386	12194
G34	**31**	32	0.31	386	12194
G35	**19**	19	6.64	0	0
G36	**19**	19	6.31	0	0
G37	84	OOT	6.94	374	37039
G38	**79**	80	0.67	199	13015
G39	**345**	345	20.28	0	0
G40	**887**	887	13.67	0	0

Table 66. Detailed information of our heuristic solutions on 40 graphs with $s = 9$

ID	lb	real	time (s)	n'	m'
G1	27	OOT	0.05	200	9876
G2	**15**	15	0	200	1534
G3	**24**	24	0	200	3235
G4	**16**	16	0.01	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0.01	500	23191
G7	33	64	0.02	0	0
G8	**17**	18	0	64	704
G9	41	OOT	0.01	70	1855
G10	23	OOT	0.08	300	10842
G11	14	OOT	2.43	4802000	42681372
G12	**12**	13	1.18	2073022	6219018
G13	12	OOT	4.9	16584963	49754842
G14	9	11	1.07	4588484	6879133
G15	9	11	3.29	12057441	18082179
G16	9	11	2.45	7122792	10680777
G17	**10**	11	1.43	5422578	5845273
G18	**10**	11	0.44	1837699	2062249
G19	**11**	12	1.35	2205	3499
G20	**23**	24	0.26	462	4024
G21	**34**	34	0.26	131	3398
G22	**18**	18	1.62	2462	31078
G23	**11**	12	0.35	4454	7393
G24	10	12	0.2	873219	1327171
G25	308	OOT	9.97	366	65395
G26	**59**	59	0.36	111	4642
G27	36	OOT	0.46	302	12947
G28	**19**	19	5.32	0	0
G29	**422**	422	4.37	442	97091
G30	84	86	58.65	280	16340
G31	88	90	105.38	275	16192
G32	**42**	43	2.8	50	1047
G33	**33**	33	0.31	385	12168
G34	**33**	33	0.42	385	12168
G35	**20**	20	6.84	0	0
G36	**20**	20	6.41	0	0
G37	87	OOT	7.41	352	34228
G38	**82**	82	0.75	117	5994
G39	345	347	19.35	426	88171
G40	**889**	889	18.97	0	0

Table 67. Detailed information of our heuristic solutions on 40 graphs with $s = 10$

ID	lb	real	time (s)	n'	m'
G1	30	OOT	0.05	200	9K
G2	**16**	16	0.00	200	1K
G3	**24**	24	0.00	200	3K
G4	**17**	17	0.01	500	4K
G5	**26**	26	0.00	500	9K
G6	**64**	64	0.01	500	23K
G7	33	64	0.02	0	0
G8	**20**	20	0.00	64	480
G9	44	OOT	0.02	70	1K
G10	24	OOT	0.08	300	10K
G11	16	OOT	2.53	4M	42M
G12	**13**	14	1.12	2M	6M
G13	13	OOT	4.99	16M	49M
G14	10	12	1.02	4M	6M
G15	10	12	4.23	12M	18M
G16	10	12	2.51	7M	10M
G17	10	12	1.64	7M	8M
G18	10	12	0.37	2M	2M
G19	11	13	3.28	9M	12M
G20	**24**	25	0.27	462	4K
G21	**36**	36	0.34	125	3K
G22	**20**	20	1.71	1K	24K
G23	**12**	13	0.29	4K	7K
G24	**11**	12	0.25	873K	1M
G25	309	OOT	12.13	367	65K
G26	**61**	62	0.30	111	4K
G27	38	OOT	0.43	302	12K
G28	**20**	20	3.59	0	0
G29	423	OOT	4.96	443	97K
G30	**88**	89	59.74	264	14K
G31	91	93	101.01	144	8K
G32	**44**	45	3.12	48	979
G33	**35**	35	0.44	383	12K
G34	**35**	35	0.38	383	12K
G35	**21**	21	7.02	123	1K
G36	**21**	21	6.71	123	1K
G37	92	OOT	6.29	306	28K
G38	**84**	84	0.73	117	5K
G39	346	349	22.84	426	88K
G40	**891**	891	19.91	0	0

Table 68. Detailed information of our heuristic solutions on 40 graphs with $s = 11$

ID	lb	real	time (s)	n'	m'
G1	32	OOT	0.04	200	9876
G2	**17**	17	0	200	1534
G3	**24**	24	0	200	3235
G4	**18**	18	0.01	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0.01	500	23191
G7	33	64	0.02	0	0
G8	**22**	22	0	64	480
G9	48	OOT	0.02	70	1855
G10	26	OOT	0.08	300	10894
G11	17	OOT	2.64	4801992	42681324
G12	**14**	15	0.98	2073022	6219018
G13	**14**	15	5.09	16584963	49754842
G14	11	13	1.29	4588484	6879133
G15	11	13	3.4	12057441	18082179
G16	11	13	2.5	7122792	10680777
G17	11	13	1.55	7733822	8156517
G18	11	13	0.43	2216688	2441238
G19	12	14	3.35	9991507	12843104
G20	**25**	26	0.31	462	4024
G21	**38**	38	0.32	123	3187
G22	20	OOT	1.47	67996	1081726
G23	12	14	0.39	1589938	2393299
G24	11	14	0.3	1087562	1541514
G25	319	321	11.03	351	60624
G26	**63**	64	0.35	111	4642
G27	40	OOT	0.45	302	12947
G28	**21**	21	2.15	300	2343
G29	**426**	427	4.91	442	97091
G30	**90**	91	60.14	264	14846
G31	93	96	103.96	144	8787
G32	**46**	46	4.31	48	979
G33	**37**	37	0.5	381	12063
G34	**37**	37	0.38	381	12063
G35	**22**	23	7.08	432	4984
G36	**23**	23	6.15	0	0
G37	94	OOT	6.62	306	28147
G38	**85**	86	0.93	199	13015
G39	346	351	23.04	426	88171
G40	**893**	893	28.12	0	0

Table 69. Detailed information of our heuristic solutions on 40 graphs with $s = 12$

ID	lb	real	time (s)	n'	m'
G1	34	OOT	0.04	200	9876
G2	**18**	18	0	200	1534
G3	**24**	24	0.01	200	3235
G4	**19**	19	0	500	4459
G5	**26**	26	0	500	9139
G6	**64**	64	0.01	500	23191
G7	**64**	64	0	0	0
G8	**24**	24	0	64	480
G9	48	OOT	0.01	70	1855
G10	27	OOT	0.06	300	10923
G11	19	OOT	2.47	4801992	42681324
G12	**15**	16	0.78	2073022	6219018
G13	**15**	16	5.76	16584963	49754842
G14	12	14	1.11	4588484	6879133
G15	12	14	3.58	12057441	18082179
G16	12	14	2.47	7122792	10680777
G17	12	14	1.64	7733822	8156517
G18	12	14	0.39	2216688	2441238
G19	13	15	3.18	9991507	12843104
G20	**26**	27	0.31	462	4024
G21	39	OOT	0.25	232	7067
G22	21	OOT	1.58	81744	1216446
G23	13	15	0.42	1589938	2393299
G24	12	15	0.24	1087562	1541514
G25	322	324	12.47	351	60624
G26	**66**	66	0.32	96	3716
G27	42	OOT	0.38	301	12917
G28	**22**	22	2.29	300	2343
G29	**429**	429	5.91	442	97091
G30	93	95	54.7	139	8143
G31	97	99	109.31	137	8091
G32	**48**	48	3.51	0	0
G33	**39**	39	0.41	378	11981
G34	**39**	39	0.36	378	11981
G35	**24**	24	7.09	285	3480
G36	**24**	24	6.9	285	3480
G37	95	OOT	6.44	320	29983
G38	86	OOT	0.77	221	15246
G39	348	353	20.66	426	88171
G40	**895**	895	23.07	0	0

Table 70. Detailed information of our heuristic solutions on 40 graphs with $s = 13$

ID	lb	real	time (s)	n'	m'
G1	36	OOT	0.04	200	9876
G2	**19**	19	0	200	1534
G3	**25**	25	0.02	200	3235
G4	**20**	20	0	500	4459
G5	**26**	26	0.02	500	9139
G6	**64**	64	0.01	500	23191
G7	**64**	64	0	0	0
G8	**25**	26	0	64	704
G9	**56**	56	0.02	70	1855
G10	29	OOT	0.07	300	10923
G11	19	OOT	2.66	4801992	42681324
G12	**16**	17	0.68	2073022	6219018
G13	16	OOT	5.15	16584963	49754842
G14	13	15	1.01	4588484	6879133
G15	13	15	3.6	12057441	18082179
G16	13	15	2.41	7122792	10680777
G17	13	15	1.61	7733822	8156517
G18	13	15	0.36	2216688	2441238
G19	14	16	3.13	9991507	12843104
G20	**27**	28	0.26	462	4024
G21	41	OOT	0.24	228	6953
G22	23	OOT	1.51	67996	1081726
G23	14	16	0.43	1589938	2393299
G24	13	15	0.23	1087562	1541514
G25	324	327	11.35	351	60624
G26	**68**	68	0.35	96	3716
G27	43	OOT	0.36	372	16983
G28	**23**	23	2.03	300	2343
G29	**430**	431	6.19	442	97091
G30	96	98	56.13	136	7852
G31	**99**	100	104.2	137	8091
G32	**48**	48	3.79	48	979
G33	**40**	40	0.37	450	15070
G34	**40**	40	0.39	450	15070
G35	**26**	26	6.86	208	2486
G36	**26**	26	6.41	208	2486
G37	97	OOT	6.08	320	29983
G38	87	OOT	0.68	254	18672
G39	348	355	19.67	426	88171
G40	**896**	896	33.66	0	0

Table 71. Detailed information of our heuristic solutions on 40 graphs with $s = 14$

ID	lb	real	time (s)	n'	m'
G1	38	OOT	0.06	200	9876
G2	**20**	20	0	200	1534
G3	**26**	26	0.03	200	3235
G4	**21**	21	0	500	4459
G5	**27**	27	0.06	500	9139
G6	**64**	64	0.01	500	23191
G7	**64**	64	0	0	0
G8	**28**	28	0	64	480
G9	56	OOT	0.03	70	1855
G10	31	OOT	0.08	300	10923
G11	20	OOT	2.49	4801992	42681324
G12	**17**	18	0.83	2073022	6219018
G13	17	OOT	5.06	16584963	49754842
G14	14	16	1.13	4588484	6879133
G15	14	16	3.67	12057441	18082179
G16	14	16	2.72	7122792	10680777
G17	14	16	1.57	7733822	8156517
G18	14	16	0.4	2216688	2441238
G19	**16**	16	1.35	2205	3499
G20	**28**	29	0.34	462	4024
G21	43	OOT	0.3	215	6564
G22	24	OOT	1.59	81744	1216446
G23	14	17	0.53	1957027	2760388
G24	14	OOT	0.24	1087562	1541514
G25	328	330	12.96	351	60622
G26	**70**	70	0.33	96	3716
G27	45	OOT	0.5	372	16983
G28	**23**	24	2.45	2109	13199
G29	431	OOT	8.12	442	97091
G30	98	100	50.87	136	7852
G31	**101**	102	103.38	137	8091
G32	**48**	48	4.11	114	2521
G33	**42**	42	0.49	447	14981
G34	**42**	42	0.52	447	14981
G35	26	OOT	9.09	1069	13282
G36	26	OOT	7.05	1069	13282
G37	99	OOT	6.8	320	29983
G38	81	OOT	1.66	523	39359
G39	350	357	20.8	426	88171
G40	**898**	898	41.6	0	0

Table 72. Detailed information of our heuristic solutions on 40 graphs with $s = 15$

ID	lb	real	time (s)	n'	m'
G1	40	OOT	0.06	200	9876
G2	**21**	21	0	200	1534
G3	27	OOT	0.04	200	3235
G4	**22**	22	0	500	4459
G5	**28**	28	0.1	500	9139
G6	**64**	64	0.01	500	23191
G7	**64**	64	0	0	0
G8	28	30	0	64	704
G9	56	60	0.03	70	1855
G10	32	OOT	0.09	300	10929
G11	21	OOT	2.41	4801992	42681324
G12	**18**	19	0.82	2073022	6219018
G13	**18**	19	5.05	16584963	49754842
G14	15	17	1.15	4588484	6879133
G15	15	17	3.98	12057441	18082179
G16	15	17	2.66	7122792	10680777
G17	15	17	1.62	7733822	8156517
G18	15	17	0.46	2216688	2441238
G19	**16**	17	3.24	9991507	12843104
G20	28	30	0.29	1981	15815
G21	44	OOT	0.37	381	13422
G22	26	OOT	1.5	57139	964631
G23	15	18	0.53	1957027	2760388
G24	15	OOT	0.3	1087562	1541514
G25	**332**	332	11.38	346	59010
G26	**72**	72	0.36	96	3716
G27	47	OOT	0.42	372	16983
G28	**24**	24	2.43	2109	13199
G29	432	OOT	8.88	442	97091
G30	**101**	101	50.86	134	7655
G31	102	104	101.55	144	8787
G32	**49**	49	3.79	122	2806
G33	43	OOT	0.41	568	19905
G34	43	OOT	0.54	568	19905
G35	**27**	28	6.52	1069	13282
G36	**27**	28	7.12	1069	13282
G37	101	OOT	5.51	320	29983
G38	83	OOT	1.64	517	38951
G39	350	359	19.06	426	88171
G40	**900**	900	19.63	0	0

Received July 2024; revised September 2024; accepted November 2024