An effective branch-and-bound algorithm for the maximum s -bundle problem

Yi Zhou, Weibo Lin, Jin-Kao Hao, Mingyu Xiao, Yan Jin

Version of Record: https://www.sciencedirect.com/science/article/pii/S0377221721003957
Manuscript_6b82552e864e49da6da7544a2f2fb4df

An effective branch-and-bound algorithm for the maximum s-bundle problem

Yi Zhou [a], Weibo Lin [a], Jin-Kao Hao [b,*], Mingyu Xiao [a], Yan Jin [c]

[a] *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*

[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c] *School of Computer Science, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China*

Abstract

An s-bundle (where s is a positive integer) is a connected graph, the vertex connectivity of which is at least $n - s$, where n is the number of vertices in the graph. As a relaxation of the classical clique model, the s-bundle is relevant for representing cohesive groups with an emphasis on the connectivity of members; thus, it is of great practical importance. In this work, we investigate the fundamental problem of finding the maximum s-bundle from a given graph and present an effective branch-and-bound algorithm for solving this NP-hard problem. The proposed algorithm is distinguished owing to its new multi-branching rules, graph coloring-based bounding technique, and reduction rules using structural information. The experiments indicate that the algorithm outperforms the best-known approaches on a wide range of well-known benchmark graphs for different s values. In particular, compared with the popular Russian Doll Search algorithm, the proposed algorithm almost doubles the success rate of solving large social networks in an hour when $s = 5$.

Keywords: Branch and bound; combinatorial optimization; graph theory; clique relaxation; s-bundle.

1 Introduction

A clique of an undirected graph is a subset of vertices that induces a complete subgraph, that is, the vertices are pairwise adjacent. Cliques are among

* Corresponding author
Email address: jin-kao.kao@univ-angers.fr (Jin-Kao Hao).

the most widely used models for representing cohesive groups (also called communities and clusters) in various applications such as wireless communication networks [7], bio-informatics networks [5], and social network analysis [8]. However, in many other situations, the clique model becomes too restrictive owing to its pairwise connectivity requirement. For instance, in social networks, members of a community are not necessarily connected pairwise. Hence, several *relaxed clique models* have been introduced to formulate cohesive groups and communities in real networks [4,16]. In general, a *relaxed clique* represents a dense graph that is close to a clique. Existing examples of relaxed cliques include s-plex [4,28], *s-defective clique* [21], and *quasi-clique* [15] and *s-club* [2,14] which are defined by relaxing the vertex degree, edge number, edge density, and pairwise distance of vertices, respectively.

In this work, we are interested in the *s-bundle*, which is another relaxed clique model that mainly concerns the connectivity of a subgraph [17]. To define the concept of the *s-bundle*, we first recall the notion of a *vertex cut*. A vertex cut of a connected graph G is a subset of vertices in which, upon removal of these vertices and their incident edges from G, the remaining graph becomes a disconnected or trivial graph (a 1-vertex graph). An s-bundle is a graph of n vertices, in which any vertex cut of G has a size of at least $n - s$.

Similar to the maximum clique problem, which is to determine a largest clique in a given graph [15,21,25,29], the *maximum s-bundle problem* studied in this work is used to find a largest s-bundle of a given graph.

Notably, the maximum s-bundle problem is different from the *maximum s-block problem* [17] (or *maximum s-vertex connected component problem* [24]), which is to find a largest induced subgraph for which all the vertex cuts of this subgraph are larger than s. Indeed, the maximum s-block problem can be solved in polynomial time [24], contrary to the maximum s-bundle problem for which no polynomial algorithm exists unless P=NP [9]. Nevertheless, practically effective algorithms for the maximum s-bundle problem are of great importance because the problem naturally arises in many network applications, especially those with a particular emphasis on the connectivity of members—for example, extracting large robustly connected subgroups from social networks [17,22].

To the best of our knowledge, there are only two algorithms for this problem in the literature. The first approach [9] is based on the general Russian doll search (RDS) framework [23], and it is designed to solve maximum relaxed clique problems that satisfy the *hereditary property* [17]. The other is a flow-based integer linear program that solves the f-vertex-connectivity clique problem [22]. Notably, both approaches are general methods since they are designed to find the maximum relaxed cliques or subgraphs (maximum s-bundles are a special case). Consequently, these methods do not explore the problem-

specific properties related to s-bundles. This is in sharp contrasts to many dedicated exact algorithms for several maximum relaxed clique problems, including the maximum s-plex problem [26,4], maximum quasi-clique problem [15], and maximum s-club problem [20,14], and the large body of solution algorithms for the conventional maximum clique problem [25].

We conclude that the research on solving the maximum s-bundle problem is still in its infancy, and effective algorithms to address this important problem need to be developed. We partially fill this gap by introducing a dedicated exact algorithm that surpasses the state-of-the-art approaches for solving the maximum s-bundle problem. The main contributions of this work are summarized as follows.

- We show several properties of the maximum s-bundle problem and propose an effective branch-and-bound algorithm. Specifically, we introduce novel multi-branching rules that dramatically reduce the number of branches in the branch-and-bound search tree. We also propose upper-bounding techniques using the k-color vertex partition and vertex degrees. New reduction rules based on useful problem properties have also been devised.
- We show that when the algorithm is applied to a wide range of benchmark instances covering random graphs, 80 hard 2^{nd} DIMACS graphs, and 43 popular social networks, it closes a number of instances that cannot be solved by any existing method. Moreover, a speedup of at least one order of magnitude was observed compared to the existing algorithms.
- Our algorithm is made publicly available thus allowing researchers and practitioners to solve various applications.

The remainder of this paper is organized as follows. In the next section, we introduce useful notations and important properties of the s-bundle. In Section 3, we elaborate on our branch-and-bound algorithm, including its branching, bounding, and reduction rules. The computational results and experimental analyses are presented in Section 4, followed by our conclusions and future research.

2 Notations and problem properties

2.1 Notations

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. For a vertex subset $S \subseteq V$, $G[S]$ denotes the *graph induced by* S in G.

For a vertex $v \in V$, $N_G(v) = \{u : \{u, v\} \in E\}$ denotes the *set of neighbor*

vertices of v, and $N_G[v] = N_G(v) \cup \{v\}$ is the *set of closed neighbor vertices* of v. For two vertices u and v in V, the *distance between u and v* in G, $dist_G(v, u)$, is the length of the shortest path between the two vertices.

A *vertex cut* of $G = (V, E)$ is a set S of vertices such that $G[V \setminus S]$ is disconnected or trivial (i.e., a 1-vertex graph). The *graph connectivity* of G, denoted by $\kappa(G)$, is the size of the minimum vertex cut in G. A graph G is an *s-bundle* if $\kappa(G) \geq |V| - s$, where s is a positive integer. It is easy to observe that a 1-bundle is a clique.

If G is a connected graph, the *vertex connectivity* of two non-adjacent vertices u and v, denoted by $\kappa_G(u, v)$, is the size of the minimum set $S \subset V$ such that u and v are disconnected in $G[V \setminus S]$. According to Menger's theorem [13], $\kappa_G(u, v)$ is equal to the maximum number of vertex-disjoint paths connecting u and v in a connected graph G. Therefore, a graph $G = (V, E)$ is an s-bundle if either $|V| \leq s$ or G is connected, and there are at least $|V| - s$ vertex-disjoint paths connecting any two non-adjacent vertices in G.

Given a graph G and a positive integer s, the *maximum s-bundle problem* studied in this work is to find a largest s-bundle in G.

2.2 Important problem properties

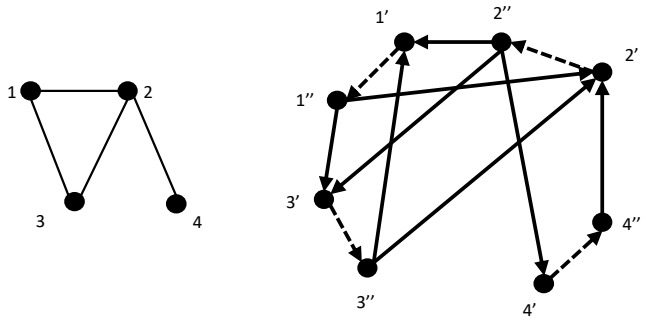
Property 1. Given an undirected graph $G = (V, E)$ with $|V| = n$ vertices and m edges, it takes $O(n^3 m)$ time to decide whether G is an s-bundle [9].

Determining whether graph G is an s-bundle is an important task in our proposed algorithm. Hence, we present a detailed procedure here. By definition, if $n \leq s$, G is an s-bundle. If $n > s$, we determine if G is connected, which can be completed in time $O(m)$ by breadth-first search or depth-first search. Then, if $n > s$ and G is not connected, G is not an s-bundle. However, if $n > s$ and G is connected, we must further check whether $\kappa(G) \geq n - s$ to make the decision.

For this, we first build a directed flow graph $H = (U, A)$, where each arc in A is associated with a capacity.

- For each vertex $u \in V$, make two copies u' and u'' and create an arc (u', u'') in H.
- For each edge $\{u, v\} \in E$, create two arcs (u'', v') and (v'', u') in H.

The capacity of every arc in H is 1. Clearly, there are $2n$ vertices and $n + 2m$ arcs in H. An example of building a flow graph is shown in Figure 1.



(a) A connected graph G.

(b) Directed flow graph H. The capacity of every arc is 1.

Fig. 1. An example of building directed flow graph

Then, the vertex connectivity of two non-adjacent vertices u and v in G, $\kappa_G(u, v)$, is equivalent to the maximum u'', v'-flow in H. The graph connectivity of G is the smallest vertex connectivity of all pairs of non-adjacent vertices in G.

As highlighted in [9], it is not necessary to compute the exact $\kappa_G(u, v)$ for each pair of non-adjacent vertices u and v in G. By definition, the connected graph G is an s-bundle if and only if the vertex connectivity is not smaller than $n - s$. In other words, G is an s-bundle if we can send $n - s$ flows from u'' to v' in H for any two non-adjacent vertices $u, v \in V$. According to network flow theory, one can send $n - s$ flows from u'' to v' if and only if there are $n - s$ augmenting paths between u'' and v'. Because an augmenting path can be found via the breadth-first search in $O(m)$ time and there are quadratic pairs of non-adjacent vertices, the running time of deciding whether G is an s-bundle is bounded by $O(n^3 m)$.

Property 2. Let $G = (V, E)$ be an s-bundle. Then, for any $S \subseteq V$, $G[S]$ is still an s-bundle [17].

In graph theory, the hereditary property [17] is a graph property; if it holds for G, then it also holds for all the induced subgraphs of G. Hence, the above statement indicates that the s-bundle is hereditary [27].

Property 3. Let $G = (V, E)$ be an s-bundle. The following statements hold:

- If $s = 1$, then for any two vertices u and v in V, $dist_G(u, v) = 1$.
- If $s \geq 2$ and $|V| > s$, then for any two vertices u and v in V, $dist_G(u, v) \leq \lfloor \frac{s-2}{|V|-s} \rfloor + 2$.

Proof. The first statement holds because a 1-bundle is a clique. We consider the second statement under the condition $|V| > s \geq 2$. By definition, G is connected when G is an s-bundle and $|V| > s$. Let u and v be two arbitrary

non-adjacent vertices in G. By Menger's theorem [13] there are at least $|V| - s$ vertex-disjoint paths between u and v. Each path goes through at least $dist_G(u, v) - 1$ vertices without counting u and v. We denote $d = dist_G(u, v)$. By summing the number of vertices in all paths, we obtain $(|V|-s)(d-1)+2 \leq |V|$. As d is an integer, $d \leq \lfloor \frac{s-2}{|V|-s} \rfloor + 2$. □

Property 4. Let $G = (V, E)$ be an s-bundle. Then, for any vertex $v \in V$, $|N_G(v)| \geq |V| - s$.

Proof. Assume that $G = (V, E)$ is an s-bundle, but there is a vertex $v \in V$ such that $|N_G(v)| < |V| - s$. Clearly, $N_G(v)$ is a vertex cut as $G[V \setminus N_G(v)]$ is a disconnected graph or a trivial graph (with only one vertex v). We arrive at a contradiction as $\kappa(G) \geq |V| - s$. □

Note that the reverse of Property 4 does not hold. Consider a graph $G = (V, E)$ with $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 2\}, \{3, 4\}\}$. For each vertex v in G, $|N_G(v)| \geq 4-3$. However, G is clearly not a 3-bundle. Indeed, if for any $v \in V$, $|N_G(v)| \geq |V| - s$, then G is called an *s-plex*. Property 4 generally states that an s-bundle is an s-plex.

In the next section, we present our branch-and-bound algorithm for solving the maximum s-bundle problem.

3 A novel branch-and-bound algorithm

3.1 General framework

The proposed algorithm for the maximum s-bundle problem is based on the general branch-and-bound search method (see Algorithm 1). The core of the algorithm is a recursive function named MSB-Rec. MSB-Rec accepts four arguments: input graph G, input integer s, *current set* S, and *candidate set* C. Generally, MSB-Rec(G, s, S, C) examines all the s-bundles that must include vertices of S and probably include vertices of C and identifies the largest.

For notational brevity, we use G' to denote $G' = G[S \cup C]$. Note that MSB-Rec(G, s, S, C) searches only the solutions in G'. In the algorithm, we use S^* to record the best solution, that is, the largest s-bundle yet found. Initially, MSB-Rec$(G, s, \emptyset, C)$ is called and S^* is fixed as the empty set.

In lines 7–8, MSB-Rec checks whether $G'[S]$ forms an s-bundle by Property 1. The search stops if $G'[S]$ is not an s-bundle because any further current search with S would be fruitless (Property 2). In lines 9–10, S^* is updated if a larger

s-bundle is found. Lines 11–12 show the bounding phase and the remaining lines in the box illustrate the branching and reduction phases. Indeed, MSB-Rec is a depth-first tree search algorithm that implements three types of rules: *branching*, *bounding*, and *reduction*.

Branching. Branching rules allow the systematic exploration of the search space. From the search tree perspective, the branching rules guide the generation of sub-branches, each of which represents a part of the search space. The branching rules must be *complete*, i.e., an optimal solution must be obtained from at least one of the sub-branches. For example, the pseudo codes in the box of Algorithm 1 show the implementation of the traditional *binary branching rule*.

Branching Rule 1 (binary branching rule)**.** Let u be the vertex from C of the minimum degree in G', i.e., $u = \text{argmin}_{v \in C} |N_{G'}(v)|$. Then, two branches are generated.

- In the first branch, u is moved from C to S.
- In the second branch, u is removed from C.

It is undisputed that this branching rule is complete. In Section 3.2, we present the improved branching rules that globally lead to fewer branches.

Bounding and Reduction. The bounding and reduction rules aim to prune subspaces where no optimal solution exists.

Following the bounding rules, MSB-Rec estimates the upper bound of the best-known solution in $G'[C]$ to determine whether the current search continues or stops (pruning). If the upper bound of the largest s-bundles in $G'[C]$ plus $|S|$ is not better than $|S^*|$, the search can be stopped. For example, the size of C is a valid upper bound. In Section 3.3, we introduce two bounding techniques, the *color-bound* that establishes connections between the size of s-bundles and the graph color number, and the simpler *degree-bound*. This procedure further reduces the size of the candidate set by removing vertices that cannot form an s-bundle with the new candidate set of vertices that cannot belong to any optimal solution. In the algorithm, the reduction procedure is only triggered when a new candidate set must be built for the recursive call of MSB-Rec. Our reduction rules use the different s-bundle graph properties that are presented in Section 3.4.

Note that the reduction and bounding procedures aim to improve the performance of the algorithm. The algorithm may still find an optimal solution if these procedures are not applied, but probably in a prohibitively long time.

Algorithm 1: The skeleton of the branch-and-bound algorithm with a simple binary branching rule.

```
1  MSB-Skeleton(G = (V, E), s)
2  begin
3      S* ← ∅
4      MSB-Rec(G, s, ∅, V)
5  MSB-Rec(G, s, S, C)
6  begin
       /* Let G' = G[S ∪ C]. Check if G'[S] is an s-bundle      */
7      if G'[S] is not an s-bundle then
8          return
9      if |S| ≥ |S*| then
10         S* ← S
       /* Upper-bounding the current search                     */
11     if Bound(G'[C], s) + |S| ≤ |S*| then
12         return
       /* Generate subbranches by branching rules. The codes in
          the box shows the traditional binary branching rule.  */
       u ← argmin_{v∈C} |N_{G'}(v)|
       MSB-Rec(G, s, S, C \ {u})
       /* Reduce fruitless vertices */
13     C' ← Reduction(G, s, S ∪ {u}, C \ {u})
       MSB-Rec(G, s, S ∪ {u}, C')
```

3.2 The multi-branching rules

In this section, we propose the multi-branching rules for MSB. Given the current set S and candidate set C, we consider the branching rules when $G'[S]$ is an s-bundle (otherwise, the search stops). We use u_p to denote the vertex of the minimum degree in G', i.e., $u_p = \text{argmin}_{v \in S \cup C} |N_{G'}(v)|$. Indeed, the multi-branching rules combine three individual branching rules with one used only when the algorithm must branch. Specifically, the first branching rule is applied when $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \in S$, and the second rule is used when $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \notin S$, and the third rule is triggered when $|N_{G'}(u_p)| \geq |S \cup C| - s$.

3.2.1 $|N_{G'}(u_p)| < |S \cup C| - s$ *and* $u_p \in S$

We now deal with the case $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \in S$. Unlike the binary branching rule, which only considers two possibilities according to whether a vertex from the candidate set is in the solution, the first multi-branching rule considers the possibilities of a subset of vertices from C, i.e., $C \setminus N_{G'}(u_p)$, and generates at least two new branches.

First, we show that $C \setminus N_{G'}(u_p)$ is a non-empty set if $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \in S$. Let n_o and n_i denote the size of $C \setminus N_{G'}(u_p)$ and the size of $S \setminus N_{G'}[u_p]$, respectively. Clearly, $n_o + n_i$ is the size of the non-neighbor vertices of u_p in G', i.e., $|(S \cup C) \setminus N_{G'}[u_p]|$. Assuming $n_o = |C \setminus N_{G'}(u_p)| = 0$, then $n_i + 0 = |(S \cup C) \setminus N_{G'}[u_p]| = |S \setminus N_{G'}[u_p]| \geq s$, i.e., $|S \cap N_{G'}(u_p)| < |S| - s$, which contradicts the condition that $G'[S]$ is an s-bundle (because u_p in $G'[S]$, violates Property 4).

Then, we can formalize the first multi-branching rule.

Branching Rule 2. If $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \in S$, without loss of generality, we assume that $C \setminus N_{G'}(u_p)$ is arbitrarily ordered as $v_1, ..., v_{n_o}$ ($n_o > 0$). Then, we generate $t + 1$ new branches, where $t = s - 1 - n_i$.

- In the first branch, v_1 is removed from C.
- For $i \in \{2, \ldots, t\}$, in the i^{th} branch, $v_1, \ldots, v_{i-1}$ are moved from C to S and v_i is deleted from C.
- In the $(t+1)^{th}$ branch, $v_1, \ldots, v_t$ are moved from C to S and $\{v_{t+1}, \ldots, v_{n_o}\}$ are deleted from C.

Example of Branching Rule 2. Consider the graph in Figure 2 with $s = 4$. $G[S]$ is an s-bundle. The vertex of the minimum degree in G', u_p, is vertex 1. Then $S \setminus N_{G'}[u_p] = \{3\}$ and $C \setminus N_{G'}(u_p) = \{5, 6, 7, 8\}$, i.e., $n_i = 1$ and $n_o = 4$. Assume that the vertices of $C \setminus N_{G'}(u_p)$ are ordered as $5, 6, 7, 8$. Then, using Branching Rule 2, we generate 3 new branches as $t = 4 - 1 - 1 = 2$.

(1) In the first branch, we call MSB-Rec with new current set $S = \{1, 2, 3, 4\}$ and candidate set $C = \{6, 7, 8, 9\}$.
(2) In the second branch, we call MSB-Rec with new current set $S = \{1, 2, 3, 4, 5\}$ and candidate set $C = \{7, 8, 9\}$.
(3) In the third branch, we call MSB-Rec with new current set $S = \{1, 2, 3, 4, 5, 6\}$ and candidate set $C = \{9\}$.

For illustrative purposes, we also show the partial branch-and-bound tree after applying Branching Rule 2 in Figure. 3 and the partial search tree generated by the binary branching rule in Figure 4. Note that in the branches represented by grey nodes in Figure 4, $G'[S]$ is not an s-plex (a graph where each vertex is adjacent to all but s vertices), and thus cannot be an s-bundle. In both

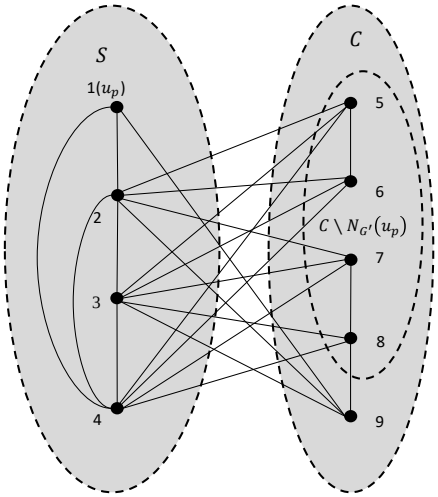


Fig. 2. An example of G', $s = 4$, S and C are marked by shadow.

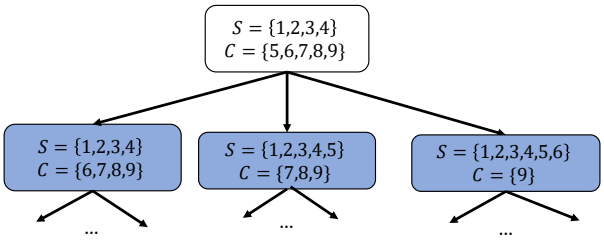


Fig. 3. Partial branch-and-bound tree by Branching Rule 2 (multi-branching rule) for graph G'.

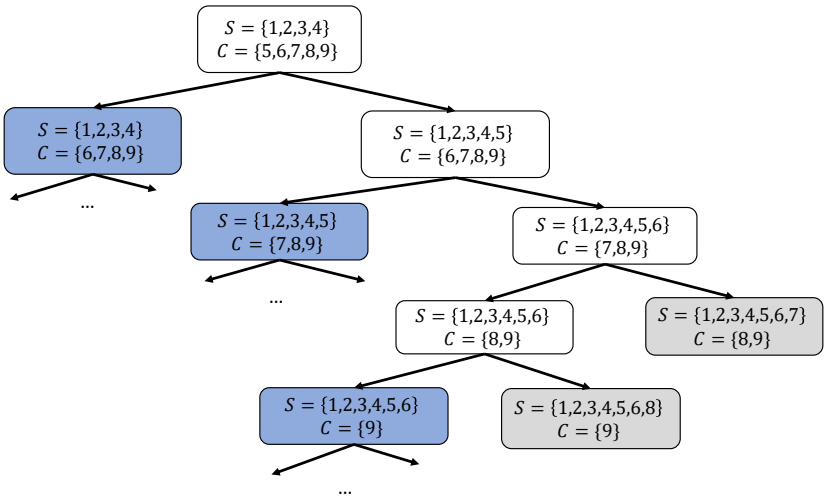


Fig. 4. Partial branch-and-bound tree by Branching Rule 1 (binary branching rule) for graph G'.

search trees, only the branches represented by the blue nodes can be further branched. Clearly, by Branching Rule 2, fewer branches are generated.

Completeness of Branching Rule 2.

Lemma 1. Given the current set S, the candidate set C. Let u_p be the vertex of the minimum degree in G'. If $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \in S$, assume that $G'[S^{opt}]$ and $S \subseteq S^{opt}$ form a maximum s-bundle of G'. Then, S^{opt} can be found from one of the branches generated by Branching Rule 2.

Proof. First, as $G'[S^{opt}]$ is an s-bundle and $u_p \in S^{opt}$, by Property 4, there are at most $s-1$ vertices in S^{opt} that are not adjacent to u_p, excluding u_p itself. By definition, there are n_i vertices in $S \subseteq S^{opt}$ that are not adjacent to u_p. Thus, S^{opt} includes at most $s-1-n_i$ vertices from $C \setminus N_{G'}(u_p)$. Differently put, at most t vertices of $v_1, ..., v_{n_o}$ belong to S^{opt}. Note that t is strictly smaller than n_o; otherwise, $|(S \cup C) \setminus N_{G'}[u_p]| = n_i + n_o \leq n_i + s - 1 - n_i = s - 1$, which contradicts the condition that $|N_{G'}(u_p)| < |S \cup C| - s$.

Let us consider Branching Rule 2 and the possibilities of the vertices $v_1, ..., v_{n_o}$. First, assuming that $v_1, ..., v_t \in S^{opt}$ and $v_{t+1}, \ldots, v_{n_o} \notin S^{opt}$, this case is covered in the $(t+1)^{th}$ branch. Second, assuming that $v_1 \notin S^{opt}$, S^{opt} must be found in the first branch because only v_1 is removed from the candidate set in the first branch. Finally, suppose that $v_1 \in S^{opt}$, let i be the first index such that $v_1, ..., v_i \in S^{opt}$ and $v_{i+1} \notin S^{opt}$. Then, S^{opt} can be found in the $(i+1)^{th}$ branch because in this branch, $v_1, ..., v_i$ are moved into the solution set and v_{i+1} is removed for further consideration. In summary, Branching Rule 2 covers all the cases. □

3.2.2 $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \notin S$

We now present the second multi-branching rule to deal with the case where $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \notin S$.

Branching Rule 3. If $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \notin S$, i.e., $u_p \in C$, we generate $t+2$ branches, where $t = s - 1 - n_i$.

- First, u_p is moved from C to S. Then, we resort to Branching Rule 2 to generate the first $t+1$ branches.
- In the $(t+2)^{th}$ branch, u_p is removed from C.

Completeness of Branching Rule 3. The completeness of this rule can be verified as follows: Assume that S^{opt} is an optimal solution of G' and $S \subseteq S^{opt}$, it suffices to consider two different cases: $u_p \in S^{opt}$ and $u_p \notin S^{opt}$. If $u_p \in S^{opt}$, then S^{opt} must be found in the first $t+1$ branches by Lemma 1. If $u_p \notin S^{opt}$, S^{opt} must be found in the $(t+2)^{th}$ branch because u_p has been removed from the candidate set in this branch.

3.2.3 $|N_{G'}(u_p)| \geq |S \cup C| - s$

If $|N_{G'}(u_p)| \geq |S \cup C| - s$, then G' itself is an s-plex. In this case, the above rules cannot be applied because $C \setminus N_{G'}(u_p)$ may be empty. We then use the binary branching rule.

Branching Rule 4. If $|N_{G'}(u_p)| \geq |S \cup C| - s$, let u be the vertex from C of the minimum degree in G', that is, $u = \text{argmin}_{v \in C} |N_{G'}(v)|$. (Note that u may be different from u_p because $u_p \in S \cup C$ but $u \in C$.) Then, two branches are generated.

- In the first branch, u is moved from C to S.
- In the second branch, u is removed from C.

It is experimentally observed that if G' is an s-plex, then G' is an s-bundle with high probability. Thus, in the implementation, we verify whether G' is an s-bundle before applying Branching Rule 4. If G' is an s-bundle, the current search stops when the entire G' is a maximum s-bundle.

It is clear that for all the conditions regarding $|N_{G'}(u_p)| < |S \cup C| - s$ or its negation, one of the three branching rules must be used. Each of these branching rules is complete. Hence, we can safely state that no optimal solution is missed by the multi-branching rules.

3.3 Bounding rules

To bound the size of the maximum s-bundle in the current branch, we provide two types of upper bounds, that is, *color-bound* and *degree-bound*. The *color-bound* is defined as follows:.

Lemma 2 (Color-bound)**.** If a graph $G'[C]$ can be k-colored, that is, C is partitioned into k independent sets (also known as color classes) $\mathcal{P} = \{C_1, \ldots, C_k\}$, then $\sum_{i=1}^{k} \min\{|C_i|, s\}$ is the upper bound of the size of the maximum s-bundle in $G'[C]$. $\sum_{i=1}^{k} \min\{|C_i|, s\}$ is termed the *color-bound* of $G'[C]$.

Proof. Suppose that $G'[F]$ is a maximum s-bundle in $G'[C]$. Let $F_i = F \cap C_i$. By Property 2, $G'[F_i]$ is still an s-bundle. Conversely, because C_i is an independent set, the size of the maximum s-bundle in graph $G'[C_i]$ is $\min\{|C_i|, s\}$. Hence, $F_i \leq \min\{|C_i|, s\}$. Therefore, $|F| = \sum_{i=1}^{k} |F_i| \leq \sum_{i=1}^{k} \min\{|C_i|, s\}$. □

The color-bound is clearly tighter than or at least equal to a trivial cardinality bound, i.e., $|C|$. When $s = 1$, the color-bound is the same as the number of

colors k. Thus, Lemma 2 generalizes that the chromatic number of a graph is an upper bound of the clique size of the graph [18,25].

It is also clear that the smaller the number of colors, the tighter the final color-bound. To trade off between the computational time and the quality of the bound, we adopt the fast greedy coloring heuristic presented in [19]. This heuristic, denoted by $ColorBound(G'[C], s)$, includes the following steps:

(1) Initialize the color number k as 1.
(2) Open an empty color class C_k and build C_k iteratively. For each iteration, find a vertex in C that is not adjacent to any vertices in C_k, then move the vertex to C_k, and repeat the above operations until such a vertex vanishes. In our implementation, the vertices in C are evaluated in the lexicographic order of their labels.
(3) Remove vertices of C_k from C. If C is not empty, increase k by 1 and return to Step 2. Otherwise, $\sum_{i=1}^{k} \min\{|C_i|, s\}$ is the color-bound of the given graph $G'[C]$.

The time complexity of $ColorBound(G'[C], s)$ is $O(|C|^3)$. However, in practice it is fast, especially when the bit-parallel technique [19] is employed to accelerate Step 2.

The other bound, i.e. the degree-bound, is defined as follows.

Lemma 3 (Degree-bound)**.** Given a graph G', let $\Delta_{G'}(C) = \max_{u \in C} |N_{G'}(u)|$ be the maximum degree of the vertex in C. Then, $\Delta_{G'}(C) + s$ is an upper bound of the size of the maximum s-bundle in G', known as *degree-bound*.

The degree-bound is directly from Property 4, but it is formalized for the maximum s-bundle problem for the first time.

3.4 Reduction techniques

The reduction rules aim to shrink the candidate set C and thus improve the search efficiency and runtime of the algorithm. We show three reduction rules based on the structural properties of s-bundles. Given the current set S, we define its *saturated subset* $P_S = \{u \in S : |N_{G'}(u) \cap S| = |S| - s\}$.

Reduction Rule 1. Given a vertex $v \in C$, if v satisfies any of the following conditions, then v is not in any s-bundle that contains S.

- $|N_{G'}(v) \cap S| < |S| + 1 - s$
- $P_S \setminus N_{G'}(v) \neq \emptyset$

Proof. The first statement clearly holds due to Property 4. For the second statement, assume that $u \in P_S$ and $u \notin N_{G'}(v)$. Then, $|N_{G'}(u) \cap (S \cup \{v\})| = |S| - s - 1$, indicating that $S \cup \{v\}$ cannot by Property 4form an s-bundle in G'. □

During the search, $|S^*|$ is a lower bound of the maximum size of s-bundles, i.e., the best-known solution found so far. We investigated more reduction rules with $|S^*|$.

Reduction Rule 2. Given a vertex $v \in C$ and a lower bound $|S^*|$, if $|N_{G'}(v)| + s \leq |S^*|$, then v is not in any s-bundle that contains S and is larger than $|S^*|$.

Proof. $|N_{G'}(v)| + s$ is an upper bound of the size of a solution containing v in G'. When we look for a solution larger than $|S^*|$, v can be removed if the upper bound is smaller than or equal to $|S^*|$. □

Reduction Rule 3. Given a vertex $v \in C$ and a lower bound $|S^*|$, the following reduction rules are applied:

- When $s = 1$, if there exists $u \in S$ such that $dist_{G'}(u, v) > 1$, then v is not in any s-bundle which contains S.
- When $|S^*| \geq s \geq 2$, if there exists $u \in S$ such that $dist_{G'}(u, v) > \lfloor \frac{s-2}{|S^*|+1-s} \rfloor + 2$, then v is not in any s-bundle which contains S and is larger than $|S^*|$.

Proof. These reduction rules are direct applications of Property 3. The first rule is straightforward. We prove the second statement by contradiction. Assume that v is in the maximum s-bundle $G'[S^{opt}]$, $S \subseteq S^{opt}$, and $|S^{opt}| \geq |S^*|+1$. From Property 3, we have $dist_{G'}(u, v) \leq \lfloor \frac{s-2}{|S^{opt}|-s} \rfloor + 2 \leq \lfloor \frac{s-2}{|S^*|+1-s} \rfloor + 2$, which is a contradiction. □

3.5 The entire MSB algorithm

Combining the above elements, we obtain the branch-and-bound-based MSB algorithm shown in Algorithm 2.

The input graph $G = (V, E)$ is represented by both its adjacent matrix and adjacent list. We maintain vertex sets S and C in the recursive procedure using the *linear heap* data structure [6]. The linear heap supports $O(1)$-time insertion and deletion, but has a space cost of $\Theta(|V|)$. In addition, we maintain two auxiliary vectors $n_{G'}[1, ..., |V|]$ and $n_S[1, ..., |V|]$ such that $n_{G'}(u) = |N_{G'}(u)|$ and $n_S(v) = |N_{G'}(u) \cap S|$. The two vectors are updated when sets S and C change.

MSB uses the greedy heuristic, *ColorBound* described in Section 3.3, to compute the color-bound in line 12. We adopted the implementation techniques introduced in [19]. The degree-bound is not explicitly used in MSB because Reduction Rule 2 ensures that the degree bound is never smaller than $|S^*|$.

Lines 15–37 implement the multi-branching rules. Specifically, lines 19–28 implement Branching Rule 2 and are executed when $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \in S$, lines 15–28 implement Branching Rule 3 and are executed when $|N_{G'}(u_p)| < |S \cup C| - s$ and $u_p \notin S$. Note that set $C \setminus N_{G'}(u_p)$ is obtained by iterating over C. Finally, lines 30–37 implement Branching Rule 4 and are executed when $|N_{G'}(u_p)| \geq |S \cup C| - s$.

The reduction procedure, *Reduction*, is shown in Algorithm 3. The procedure is called at lines 24, 27 and 36 in Algorithm 2 when a new branch is generated. In Algorithm 3, Reduction Rule 1 corresponds to lines 5 and 6, and Reduction Rule 2 corresponds to lines 7 and 8. In the implementation, $|N_{G'}(v) \cap S|$ is obtained from $n_S(v)$, $P_S \setminus N_{G'}(v)$ is computed by enumerating P_S, and $|N_{G'}(v)|$ is obtained from $n_{G'}(v)$. Thus, the time complexity is $O(|S||C|)$. To use Reduction Rule 3, we first run a breadth-first search starting from a randomly selected vertex $u \in S$ to compute the shortest distance from u to every vertex in C. Then, the vertices not meeting the distance conditions in Reduction Rule 3 are removed from C. The running time of *Reduction* is bounded by $O(|S||C| + |E|)$.

3.6 Relations with the RDS algorithm

The RDS algorithm for the maximum s-bundle problem presented in [9] is a leading-edge algorithm. It follows the general Russian Doll Search framework [23] and uses a basic branch-and-bound algorithm with a binary branching rule as its main subroutine. Given a graph of n vertices, the RDS algorithm first sorts the vertices normally by degeneracy order. Assume that the vertices are sorted as $v_1, \ldots, v_n$, where n is the number of vertices in G. Then, the RDS algorithm runs n rounds of the underlying branch-and-bound algorithm. In the i^{th} round, where i decreases from n to 1, the RDS algorithm solves the maximum s-bundle problem in a subgraph induced by $\{v_{i+1}, \ldots, v_n\}$ with the restriction that v_i must be in the solution. Consequently, the best over all the n solutions collected from the n rounds is the optimal solution. An advantage of this framework is that the optimal values found in the $(i+1)^{th}$,...,n^{th} rounds can serve as a bound for the i^{th} round. For example, in the i^{th} branch-and-bound search, when v_j $(j > i)$ is moved to the sets S and $C \subseteq \{v_{j+1}, \ldots, v_n\}$, the solution that was previously found in the j^{th} round provides an upper bound.

Algorithm 2: The entire MSB algorithm for the maximum s-bundle problem

```
1  MSB(G = (V, E), s)
2  begin
3      S ← {u ∈ V : N_G(u) = |V| − 1}
4      S* ← |S|
5      MSB-REC(G, s, S, V \ S)
6  MSB-Rec(G, s, S, C)
7  begin
       /* Let G' = G[S ∪ C]                                          */
8      if G'[S] is not an s-bundle then
9          return
10     if |S| ≥ |S*| then
11         S* ← S
12     if ColorBound(G'[C], s) + |S| ≤ |S*| then
13         return
14     u_p ← argmin_{v∈S∪C} |N_{G'}(v)|
15     if |N_{G'}(u_p)| < |S ∪ C| − s then
16         if u_p ∉ S then
17             MSB-Rec(G, s, S, C \ {u_p})
18             S ← S ∪ {u_p}, C ← C \ {u_p}
19         Denote n_i = |S \ N_{G'}[u_p]|, n_o = |C \ N_{G'}(u_p)| and t = s − 1 − n_i. Also,
             assume C \ N_{G'}(u_p) = {v_1, ..., v_{n_o}}.
20         for i ∈ 1, ..., t + 1 do
21             if i = 1 then
22                 MSB-Rec(G, s, S, C \ {v_1})
23             else if 2 ≤ i ≤ t then
24                 C' ← Reduction(G, s, S ∪ {v_1, ..., v_{i−1}}, C \ {v_1, ..., v_i})
25                 MSB-Rec(G, s, S ∪ {v_1, ..., v_{i−1}}, C')
26             else
27                 C' ← Reduction(G, s, S ∪ {v_1, ..., v_t}, C \ {v_1, ..., v_{n_o}})
28                 MSB-Rec(G, s, S ∪ {v_1, ..., v_t}, C')
29     else
30         if G' is an s-bundle then
31             if |S ∪ C| > |S*| then
32                 S* ← S ∪ C
33                 return
34         u ← argmin_{v∈C} |N_{G'}(v)|
35         MSB-Rec(G, s, S, C \ {u})
36         C' ← Reduction(G, s, S ∪ {u}, C \ {u})
37         MSB-Rec(G, s, S ∪ {u}, C')
```

Algorithm 3: The reduction procedure

```
1  Reduction(G = (V, E), s, S, C)
2  begin
       /* Let G' = G[S ∪ C]                                          */
3      Extract P_S from current S.
4      for v ∈ C do
5          if |N_G'(v) ∩ S| < |S| + 1 − s or P_S \ N_G'(v) ≠ ∅ then
6              C ← C \ {v}
7          else if |N_G'(v)| + s ≤ |S*| then
8              C ← C \ {v}
9      if |S*| ≥ s ≥ 2 then
10         Randomly select a vertex u ∈ S and compute dist_G'(u, v) for
             v ∈ C by BFS search.
11         for v ∈ C that dist_G'(u, v) > ⌊(s−2)/(|S*|+1−s)⌋ + 2 do
12             C ← C \ {v}
13     return C
```

Although both RDS and MSB rely on branch-and-bound, the branching, reduction, and bounding rules in the two algorithms are intrinsically different. First, MSB uses multi-branching rules, whereas the branching rule used in RDS is equivalent to the basic binary branching rule. Second, MSB exploits some novel structural properties of the maximum s-bundles like the diameter and color-bound properties, i.e., Property 3 and Lemma 2. Lastly, MSB solves non-overlapping subproblems via branch-and-bound, while RDS solves a series of overlapped subproblems and uses the results of solved subproblems for pruning branches.

4 Computational experiments

This section is dedicated to a computational assessment of the proposed MSB algorithm. We show the experimental results obtained on benchmark instances and comparisons with the best-performing exact algorithms in the literature.

4.1 Experiments setup

The proposed MSB algorithm was programmed in C++ [1] and were compiled by g++ with the optimization option '-O2'. The experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8 GHz and 8 GB RAM) running CentOS 6.5. When we solved the DIMACS machine benchmarking program dfmax.c [2] without compilation optimization flag, the runtime on our machine was 0.40, 2.50, and 9.55 seconds for graphs r300.5, r400.5, and r500.5, respectively. All experiments were carried out with four popular benchmark sets: Erdös-Rényi random graphs, 2^{nd} DIMACS graphs, SNAP, and 10^{th} DIMACS networks. These benchmark sets are commonly used for testing clique and relaxed clique algorithms [10,12,14].

4.2 Computational results and comparisons

As our references we used the RDS algorithm of [9], the flow-based ILP model solved by the MIP solver, CPLEX of [22], and the basic branch-and-bound algorithm BB. Note that BB is almost the same as MSB, except for using only the binary branching rule for branching.

For the RDS algorithm of [9], we faithfully re-implemented it and verified that the results of our implementation generally matched those reported in [9]. For the flow-based linear model of [22], we ran CPLEX (Version 12.71). Whenever possible, the implementation of BB shares the same data structures and subroutines as the MSB algorithm. To ensure a fair comparison we ran all compared algorithms on the computer described above

4.2.1 Random graphs

For a graph $G(n, p)$ of the Erdös-Rényi random benchmark, an edge with a unified edge probability $p \in [0, 1]$ exists between any pair of n vertices. We investigated the performance of the aforementioned algorithms on 180 random graphs with $n = 50$ (20 graphs for each $p \in \{0.1, 0.2, \ldots, 0.9\}$). In Figure 5, we present the comparative average time to solve the graphs of different densities when s equals $2, \ldots, 5$. The horizontal axis gives the edge density of G, and the vertical axis shows the average time in seconds.

[1] The source code of our MSB algorithms and the codes of the reference algorithms (see below) are publicly available at `https://github.com/joey001/max-s-bundle.git`.

[2] http://archive.dimacs.rutgers.edu/pub/dsj/clique/

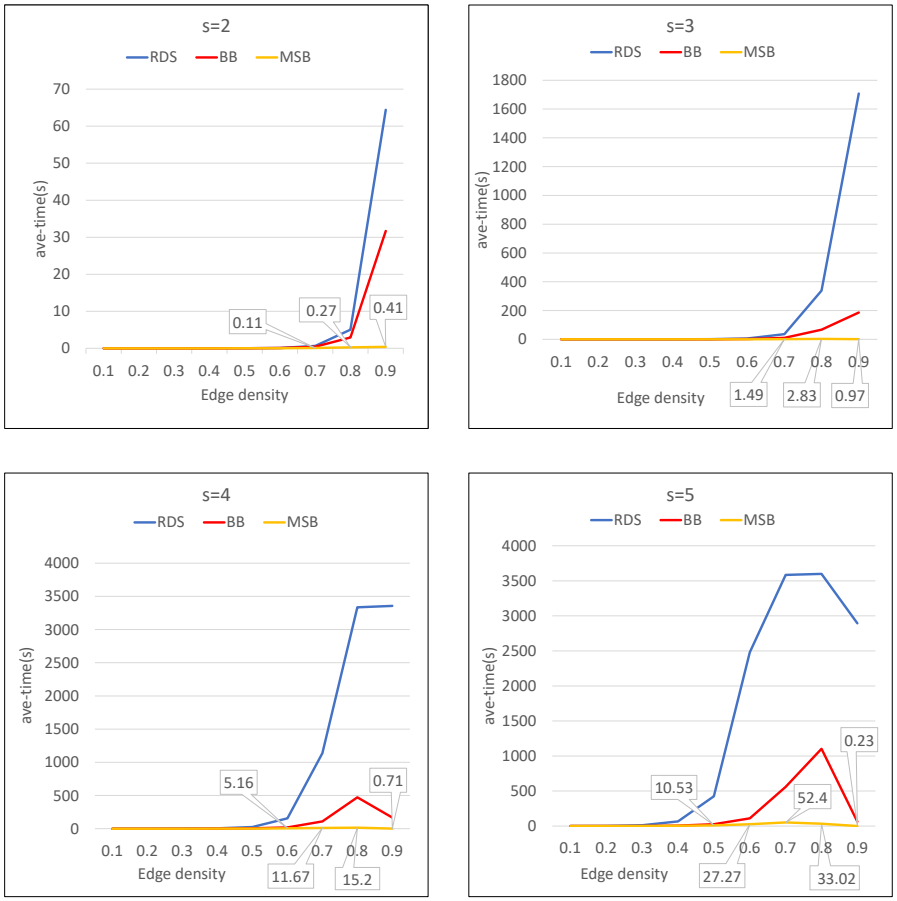


Fig. 5. Different edge densities against average computation times for random graphs with 50 vertices.

However, CPLEX failed to solve any of these random graphs for any s in $\{2, 3, 4, 5\}$ within an hour. We further observe that, for our benchmark graphs with at least 100 vertices, CPLEX failed to load the ILP model because of memory corruption. To some extent, this situation is not surprising given that there are $O(n^4)$ variables and constraints for a graph of n vertices following the flow-based model in [22]. Thus, we omitted CPLEX in the comparative study.

Regarding computation time, one notices that the average time used by MSB was the lowest among the three compared algorithms. BB performs better than RDS when the edge density is larger than 0.7 for all s values. Moreover, we observe a significant gap between the two algorithms on the dense random graphs for each s. Hence, MSB is always the best performing algorithm for such instances.

4.2.2 The 2nd DIMACS graphs

All 80 graphs of the 2^{nd} DIMACS Implementation Challenge Benchmark (the 2^{nd} DIMACS) [3] were tested. These benchmark instances include both small graphs (up to 50 vertices and 1,000 edges) and large graphs (up to 4,000 vertices and 5,000,000 edges) that encompass real-world problems (e.g., coding theory, fault diagnosis, and the Steiner triple problem) and random graphs. In Tables 1-4, we present the computational results of all solved graphs for each $s \in \{2, 3, 4, 5\}$ and omit the instances that cannot in an hour be solved to optimality by any of the three algorithms. The first column shows the basic information of each graph (name, number of vertices, and edges). Column $|S^{opt}|$ provides the optimal value. For each algorithm, we show the time consumption and the number of branches in the search tree. For the RDS algorithm, we show the summation of the number of branches over all rounds of branch-and-bound. If the runtime is shorter than an hour, the corresponding algorithm obtains the an optimal solution in one hour. The last two rows in each table show the number of solved graphs, "#SOLVED", and the average runtime for solving these instances, "AVE-TIME". Note that the runtime for unsolvable benchmark instances is always 3600.01 seconds (indicated in italic format). The minimum time consumption and the maximum number of solved instances are highlighted in boldface. "0" indicates that the corresponding instance was solved in less than 0.01 seconds. We have made the following comments on the results:

For $s = 2$, RDS solved 19 instances within 3600 seconds while MSB and BB solved 15 of these 19 instances. Although MSB solved fewer graphs than RDS, it required less computation time than RDS for 6 instances (indicated in boldface). For $s = 3$, MSB and BB still solved fewer graphs than RDS. However, the situation changes for $s = 4$ and $s = 5$ because MSB and BB solved 10 instances against 9 instances for RDS. Moreover, MSB consumed much less time than RDS. MSB dominates the other algorithms, particularly on the *c-fat* graphs. Finally, in Tables 3 and 4, MSB generated much fewer branches than BB, with a reduction greater than 50%. We provide more information about the rules' impact on the number of branches in Section 4.3.

4.2.3 SNAP and the 10^{th} DIMACS networks

The Stanford Large Network Dataset Collection(SNAP) provides many large-scale social and information networks [11], including graphs retrieved from social networks, communication networks, citation networks, etc. The 10^{th} DIMACS Implementation Challenge Benchmark (the 10^{th} DIMACS) contains artificial and real-world graphs from different applications [3]. Experiments

[3] http://archive.dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/

Table 1
Computational results for the 2^{nd} DIMACS graphs when $s = 2$.

graph			$\lvert S^{opt}\rvert$	RDS		BB		MSB	
name	#vtx	#edge	time(s)	#branches	time(s)	#branches	time(s)	#branches	
brock200_2.clq	200	9876	13	**140.09**	23204771	701.61	279680379	258	40950215
c-fat200-1.clq	200	1534	12	0	3757	0	8070	0	345
c-fat200-2.clq	200	3235	24	0	2221	0	4539	0	402
c-fat200-5.clq	200	8473	58	0.02	2565	0.02	8990	**0.01**	1825
c-fat500-1.clq	500	4459	14	0.07	19844	0.06	35815	**0**	723
c-fat500-10.clq	500	46627	126	0.3	10372	0.3	46696	**0.12**	7383
c-fat500-2.clq	500	9139	26	0.05	10462	0.09	33859	**0.01**	1023
c-fat500-5.clq	500	23191	64	0.05	6381	0.11	31151	**0.03**	2260
hamming6-2.clq	64	1824	32	**0.03**	1182	705.25	310576331	10.55	2963698
hamming6-4.clq	64	704	6	**0**	4569	0.04	58258	0.04	23197
hamming8-2.clq	256	31616	128	**73.48**	23389	*3600.01*	119378085	*3600.01*	133759989
hamming8-4.clq	256	20864	16	**187.86**	8101239	*3600.01*	753853474	*3600.01*	347806429
johnson8-2-4.clq	28	210	5	0	1465	0.01	11857	0	6916
johnson8-4-4.clq	70	1855	14	**0.43**	43441	122.54	108627984	54.3	17124673
keller4.clq	171	9435	15	**3442.72**	156769519	*3600.01*	1460375865	*3600.01*	367595746
MANN_a9.clq	45	918	26	13.55	331829	256.99	74773734	**1.01**	179429
p_hat300-1.clq	300	10933	10	**5.51**	1428704	44.85	29476779	22.79	3447325
p_hat500-1.clq	500	31569	12	**143.48**	26425229	1785.94	541199388	750.25	54381068
p_hat700-1.clq	700	60999	13	**926.55**	134143326	*3600.01*	725683758	*3600.01*	148853164
#SOLVED				19		15		15	
AVE-TIME				2806.6		2970.2		2938.7	

Table 2
Computational results for the 2^{nd} DIMACS graphs when $s = 3$.

graph			$\lvert S^{opt}\rvert$	RDS		BB		MSB	
name	#vtx	#edge		time(s)	#branches	time(s)	#branches	time(s)	#branches
c-fat200-1.clq	200	1534	12	0.12	90123	0.02	16283	**0**	841
c-fat200-2.clq	200	3235	24	0.04	17829	0.01	7845	**0**	490
c-fat200-5.clq	200	8473	58	0.06	6010	0.07	44287	**0.02**	2606
c-fat500-1.clq	500	4459	14	2.63	1003177	0.09	50296	**0.01**	1396
c-fat500-10.clq	500	46627	126	0.51	19549	1	249791	**0.16**	9578
c-fat500-2.clq	500	9139	26	0.63	192141	0.19	89770	**0.02**	1935
c-fat500-5.clq	500	23191	64	0.23	31711	0.29	86616	**0.05**	2522
hamming6-2.clq	64	1824	32	**18.45**	253714	*3600.01*	929298635	*3600.01*	598357118
hamming6-4.clq	64	704	8	**0.11**	31337	0.54	718935	0.68	319781
johnson8-2-4.clq	28	210	8	0.07	11209	0.05	47967	**0.02**	27630
johnson8-4-4.clq	70	1855	18	**891.25**	13720679	*3600.01*	2404085611	*3600.01*	762776066
MANN_a9.clq	45	918	36	3.69	27632	2.22	610143	**0.02**	13031
p_hat300-1.clq	300	10933	12	**622.36**	81407111	*3600.01*	1819070399	2654.45	291775936
#SOLVED				13		10		11	
AVE-TIME				3034.2		3207.8		3205.9	

were conducted on 43 representative graphs that were also used in [9]. To deal with very large graphs, the so-called *peel* procedure [1] was applied to each graph during the pre-processing step. It recursively removes vertices of less than $\omega(G) - s$ degrees, where $\omega(G)$ is the best-known lower bound of the clique number of G. Clearly, this procedure reduces the graph size without removing any optimal solutions. The reduced graphs were used to test all three compared algorithms.

For each $s \in \{2, 3, 4, 5\}$, we show the number of solved graphs and the average time-consumption for solving the graphs in Table 5. For all s, the MSB and BB solved more instances in a shorter average time. Remarkably, when $s = 5$,

Table 3
Computational results for the 2^{nd} DIMACS graphs when $s = 4$.

graph			$\|S^{opt}\|$	RDS		BB		MSB	
name	#vtx	#edge		time(s)	#branches	time(s)	#branches	time(s)	#branches
c-fat200-1.clq	200	1534	12	6.49	5074105	0.08	36864	**0.01**	3324
c-fat200-2.clq	200	3235	24	0.79	385727	0.02	24555	**0**	1034
c-fat200-5.clq	200	8473	58	0.54	31892	0.54	297958	**0.05**	4320
c-fat500-1.clq	500	4459	14	202.6	124087559	0.28	99517	**0.03**	5201
c-fat500-10.clq	500	46627	126	2.99	88184	5.85	1586216	**0.27**	11322
c-fat500-2.clq	500	9139	26	27.31	10443398	0.55	333375	**0.08**	5539
c-fat500-5.clq	500	23191	64	2.27	486119	1.07	435989	**0.08**	3755
hamming6-4.clq	64	704	10	**2.29**	374894	2.91	5103105	5.67	2488713
johnson8-2-4.clq	28	210	9	1.92	163360	0.63	346597	**0.47**	183688
MANN_a9.clq	45	918	36	*3600.01*	2705586	1214.03	55729412	4.57	878911
#SOLVED				9		10		10	
AVE-TIME				3198.0		3232.4		3222.5	

Table 4
Computational results for the 2^{nd} DIMACS graphs when $s = 5$.

graph			$\|S^{opt}\|$	RDS		BB		MSB	
name	#vtx	#edge		time(s)	#branches	time(s)	#branches	time(s)	#branches
c-fat200-1.clq	200	1534	12	230.07	332799550	0.62	90247	**0.11**	20830
c-fat200-2.clq	200	3235	24	11.37	9697151	0.17	129575	**0.03**	3778
c-fat200-5.clq	200	8473	58	5.04	300289	3.64	2040620	**0.24**	9079
c-fat500-1.clq	500	4459	8	*3600.01*	3451363703	1.01	276611	**0.44**	42861
c-fat500-10.clq	500	46627	126	30.39	657809	33.41	11354380	**0.99**	15756
c-fat500-2.clq	500	9139	26	1201.87	604112363	3.7	1757377	**0.39**	29615
c-fat500-5.clq	500	23191	64	31.67	7823971	6.85	2922154	**0.28**	9338
hamming6-4.clq	64	704	12	48.14	3659296	**20.35**	22304439	36.67	13885967
johnson8-2-4.clq	28	210	12	9.06	266073	0.39	252089	**0.3**	154757
MANN_a9.clq	45	918	45	0.79	1020	0	1	0	2
#SOLVED				9		10		10	
AVE-TIME				3214.6		3184.8		3164.1	

Table 5
The statistics of solving the 10^{th} DIMACS and SNAP graphs in an hour when $s =$ 2, 3, 4 and 5.

statistics	$s = 2$			$s = 3$		
	RDS	BB	MSB	RDS	BB	MSB
#SOLVED	**42**	40	40	31	37	**38**
AVE-TIME	**206.3**	358.5	358.0	1143.6	611.6	**462.1**
statistics	$s = 4$			$s = 5$		
	RDS	BB	MSB	RDS	BB	MSB
#SOLVED	26	35	**37**	18	32	**36**
AVE-TIME	1461.5	866.1	**576.4**	2131.5	1011.8	**643.1**

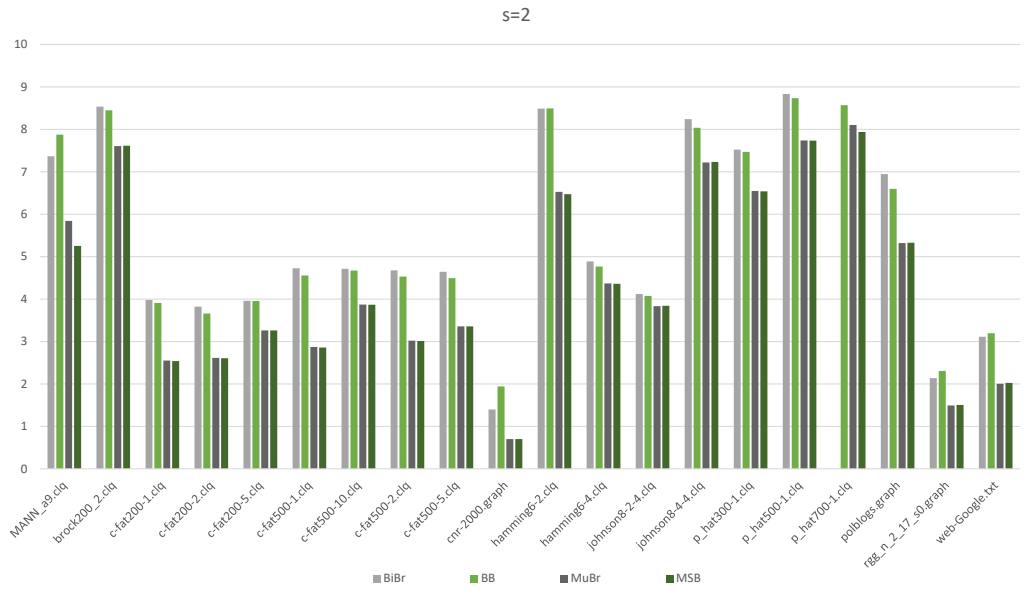


Fig. 6. Number of branches (log scale) generated by MSB and three variants for $s = 2$.

the MSB solved 36 graphs, which doubled the number of graphs solved by RDS.

Overall, MSB outperforms RDS and BB when $s = 4, 5$, and RDS performs better than MSB for $s = 2$ in terms of computational time. For cases where RDS runs faster than MSB, such as brock200_2.clq, hamming6-2.clq, and wiki-Vote.txt (see Appendix), RDS also produces fewer branches than MSB. As pointed out in Section 3.6, RDS uses the results of previously solved subproblems for bounding. Apparently, this bounding technique renders RDS very efficient for pruning unnecessary branches for small s values, e.g., $s = 2$.

4.3 An analysis of multi-branching and color based bounding

Multi-branching rules and color-based bounding techniques are MSB's two main features. For a better understanding of these algorithmic components, we present more comparisons among BB, MSB, and two new algorithmic variants: BiBr and MuBr. BiBr and MuBr are also branch-and-bound algorithms, but they use different combinations of branching and bounding rules.

- BiBr: A branch-and-bound variant which uses the Binary Branching rule but ignores the color based bounding technique.
- MuBr: A branch-and-bound variant which uses the Multi-Branching rules, but ignores the color based bounding technique.

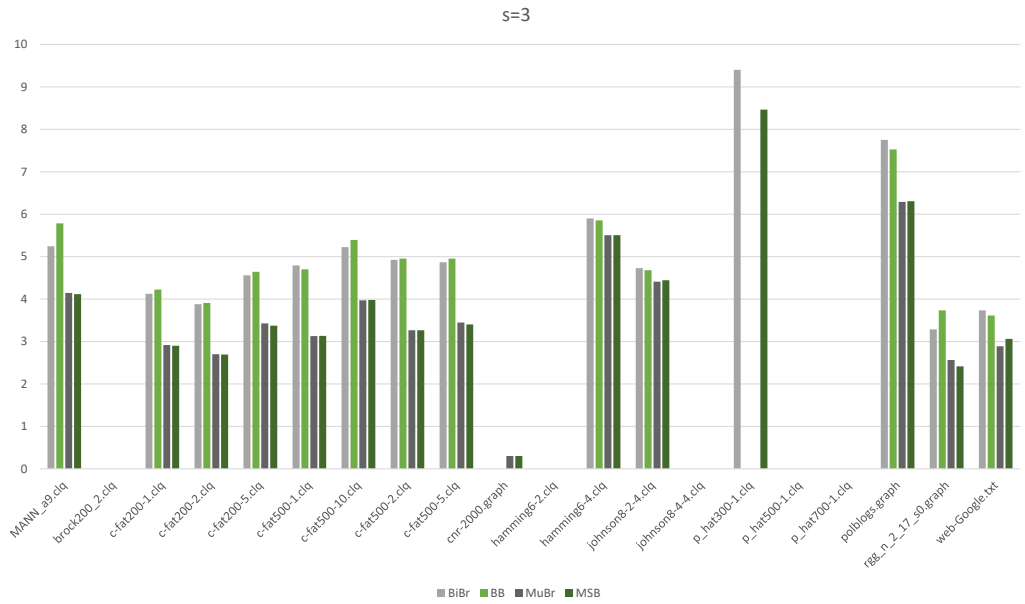


Fig. 7. Number of branches (log scale) generated by MSB and three variants for $s = 3$.

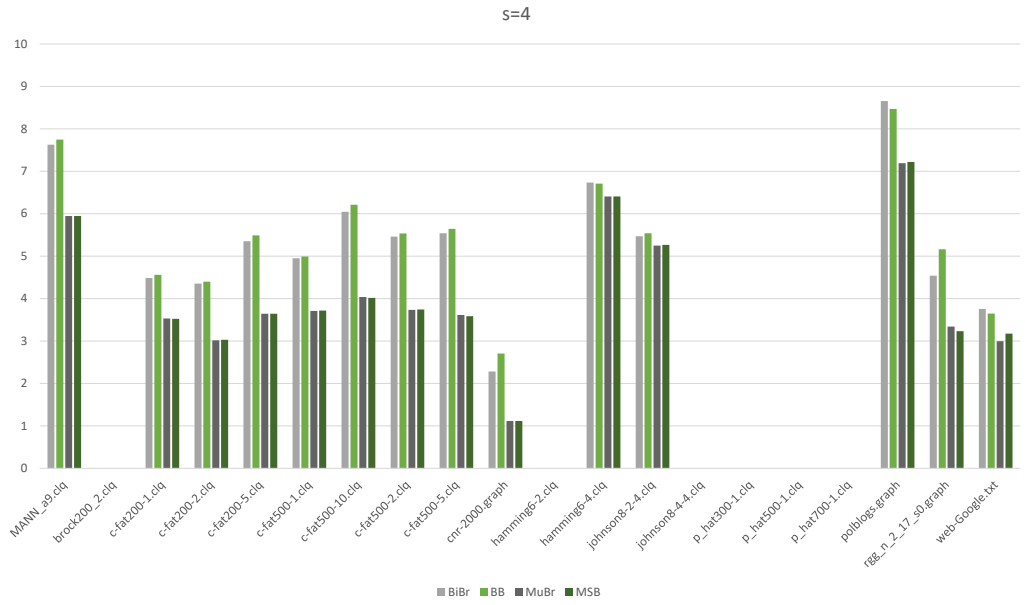


Fig. 8. Number of branches (log scale) generated by MSB and three variants for $s = 4$.

Notably, BB uses the binary branching rule and color-based bounding, whereas MSB uses multi-branching rules and color-based bounding. Therefore, BiBr and MuBr are branch-and-bound algorithms that do not compute color-bound. We ran these algorithms on the 19 graphs shown in Table 1 and 4 graphs from the set of the SNAP and 10^{th} DIMACS networks: cnr-2000.graph, polblogs.graph, rgg_n_2_17_s0.graph, and web-Google.txt. These graphs have dif-

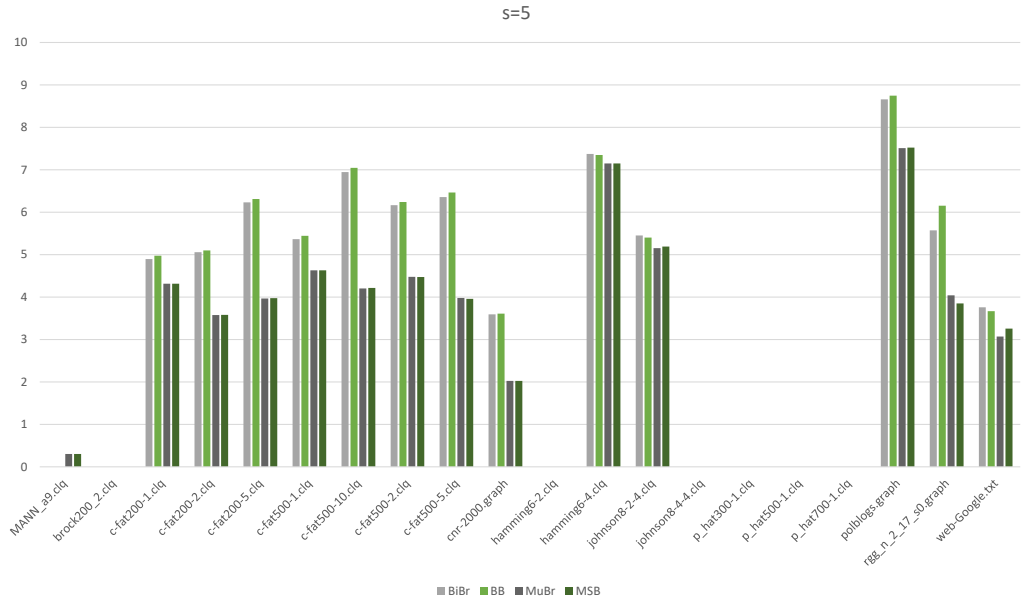


Fig. 9. Number of branches (log scale) generated by MSB and three variants when $s = 5$.

ferent structures and the only known common feature is that they can be solved by RDS within one hour for $s = 2$. We used the same experimental protocol with a cut-off time of one hour for all four algorithms. We show the total number of branches (on a log scale) generated by each algorithm in Figures 6 to 9. If, per example, an algorithm cannot produce the an optimal solution within one hour, the corresponding number of branches is omitted in the figures.

From Figures 6 to 9, it is clear that the number of branches for MuBr and MSB is dramatically lower than that of BiBr and BB for almost all the instances. For example, for the c-fat family and four SNAP and 10^{th} DIMACS, MuBr, and MSB reduce the number of branches compared to BiBr and BB for all s values by one order of magnitude. This indicates that the multi-branching technique is the main driving force for reducing the size of the search tree. For $s = 3, 4, 5$, the efficacy of the color-bound is not significant and for $s = 2$, the algorithms with the color-bound (BB and MSB) generally showed a slight reduction over the algorithms without this technique (BiBr and MuBr). Nevertheless, our experiments showed that the color-bound tends to be tighter than the degree-bound when the input graph is dense and s is small. For example, if we compare the degree-bound (column DegB), color-bound (column ColB), and LP relaxation (column LP) before the exact search, but after applying the *peel* procedure in Table 6, one observes a clear trend that the color-bound is tighter than the degree-bound when s is smaller. As we mentioned, because the LP formulation of [22] suffers from an excessive number of variables and constraints we only obtained the solution of the LP relaxation for a few small graphs. From the current results, there is no evidence that the bound via LP

relaxation is stronger than the other two bounds. Nevertheless, it would be interesting to develop a compact LP formulation or tighten the color-bound with a larger s for this problem.

Table 6
A comparison among the color-bound, degree-bound and LP relaxation before the exact search (at the root node of the search tree).

Graph	s=2			s=3			s=4			s=5		
	DegB	ColB	LP	DegB	ColB	LP	DegB	ColB	LP	DegB	ColB	LP
MANN_a9.clq	43	**33**	43.9	**44**	45	44.3	45	45	**44.8**	46	45	45
brock200_2.clq	116	**72**	-	**117**	105	-	**118**	136	-	**119**	164	-
c-fat200-1.clq	**19**	24	-	**20**	36	-	**21**	48	-	**22**	60	-
c-fat200-2.clq	**36**	46	-	**37**	68	-	**38**	90	-	**39**	112	-
c-fat200-5.clq	**88**	134	-	**89**	200	-	**90**	200	-	**91**	200	-
c-fat500-1.clq	**22**	28	-	**23**	42	-	**24**	56	-	**25**	70	-
c-fat500-10.clq	**190**	252	-	**191**	376	-	**192**	500	-	**193**	500	-
c-fat500-2.clq	**40**	52	-	**41**	78	-	**42**	104	-	**43**	130	-
c-fat500-5.clq	**97**	128	-	**98**	190	-	**99**	252	-	**100**	314	-
cnr-2000.graph	87	**86**	86	88	**86**	86	92	89	**88.8**	**93**	170	-
hamming6-2.clq	**59**	64	61.6	**60**	64	62.1	**61**	64	62.6	**62**	64	63.1
hamming6-4.clq	24	**16**	-	25	**24**	-	**26**	32	-	**27**	40	-
hamming8-2.clq	**249**	256	-	**250**	256	-	**251**	256	-	**252**	256	-
hamming8-4.clq	165	**64**	-	166	**96**	-	167	**128**	-	168	**160**	-
johnson8-2-4.clq	17	**12**	23.5	18	18	23.9	**19**	22	24.3	**20**	25	24.8
johnson8-4-4.clq	55	**40**	-	**56**	60	-	**57**	68	-	**58**	70	-
keller4.clq	126	**74**	-	127	**101**	-	128	**124**	-	**129**	140	-
p_hat300-1.clq	134	**54**	-	135	**79**	-	136	**103**	-	137	**125**	-
p_hat500-1.clq	206	**80**	-	207	**118**	-	208	**156**	-	209	**194**	-
p_hat700-1.clq	288	**103**	-	289	**153**	-	290	**203**	-	291	**251**	-
polblogs.graph	218	**55**	-	227	**79**	-	241	**100**	-	255	**119**	-
rgg_n_2_17_s0.graph	**19**	29	23.5	**20**	45	-	**29**	59	-	**33**	75	-
web-Google.txt	**56**	83	-	**82**	92	-	**86**	100	-	**87**	107	-

5 Conclusions and Perspectives

The concept of the s-bundle is an important member of the family of relaxed clique models. We presented an effective branch-and-bound algorithm named MSB to solve the NP-hard maximum s-bundle problem. With the general branch-and-bound method, the proposed algorithm integrates problem-specific techniques including multi-branching rules, color-based bounding techniques, and structural reduction rules.

We performed extensive experiments on various benchmark sets and compared the performance of our algorithm with state-of-the-art methods. The computational results show the highly competitive performance of the proposed MSB algorithm, especially for $s = 4$ and 5. Thus, this work enriches the toolkit for effectively solving the maximum s-bundle problem. We also performed additional experiments to investigate the efficacy of the proposed multi-branching and color-based bounding techniques for reducing the size of the search tree.

From an algorithmic perspective, this work invites us to investigate similar

ideas to design effective algorithms for other relaxed clique problems. Given that the s-bundle model can formulate several practical problems, as discussed in the introduction, the proposed algorithm can hopefully be used to better solve these problems, and to this, the availability of the source code of our algorithm will significantly contribute. Finally, it is known that the RDS algorithm of [9] is applicable to the maximum s-bundle problem and also the maximum weight s-bundle problem. Similarly, the MSB algorithm is easily adapted to the weight version of the maximum s-bundle problem by simply removing (or changing) the color-bound and retaining the other rules unchanged.

Acknowledgements

We are grateful to the reviewers for their insightful comments and suggestions, which helped us to improve the presentation of the paper significantly. This work was partially supported by the National Natural Science Foundation of China under Grant No. 61802049 and the Scientific Research Fund of UESTC under No. ZYGX2018KYQD210.

References

[1] J. Abello, M. Resende, S. Sudarsky, Massive quasi-clique detection, LATIN 2002: Theoretical Informatics (2002) 598–612.

[2] M. T. Almeida, F. D. Carvalho, An analytical comparison of the LP relaxations of integer models for the k-club problem, European Journal of Operational Research 232 (3) (2014) 489–498.

[3] D. A. Bader, H. Meyerhenke, P. Sanders, D. Wagner (eds.), Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings, vol. 588 of Contemporary Mathematics, American Mathematical Society, 2013 (2013).

[4] B. Balasundaram, S. Butenko, I. V. Hicks, Clique relaxations in social network analysis: The maximum k-plex problem, Operations Research 59 (1) (2011) 133–142.

[5] S. Butenko, W. E. Wilhelm, Clique-detection models in computational biochemistry and genomics, European Journal of Operational Research 173 (1) (2006) 1–17.

[6] L. Chang, L. Qin, Linear heap data structures, in: Cohesive Subgraph Computation over Large Sparse Graphs, Springer, 2018, pp. 9–20.

[7] Y. Chen, A. Liestman, J. Liu, Clustering algorithms for ad hoc wireless networks, Ad Hoc and Sensor Networks 28 (2004) 76.

[8] M. G. Everett, S. P. Borgatti, Analyzing clique overlap, Connections 21 (1) (1998) 49–61.

[9] T. Gschwind, S. Irnich, I. Podlinski, Maximum weight relaxed cliques and russian doll search revisited, Discrete Applied Mathematics 234 (2018) 131–138.

[10] H. Jiang, C. Li, F. Manyà, An exact algorithm for the maximum weight clique problem in large graphs, in: S. P. Singh, S. Markovitch (eds.), Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA, AAAI Press, 2017.

[11] J. Leskovec, A. Krevl, SNAP Datasets: Stanford large network dataset collection, `http://snap.stanford.edu/data` (Jun. 2014).

[12] C. McCreesh, P. Prosser, Reducing the branching in a branch and bound algorithm for the maximum clique problem, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2014.

[13] K. Menger, Zur allgemeinen kurventheorie, Fundamenta Mathematicae 10 (1) (1927) 96–115.

[14] F. M. Pajouh, B. Balasundaram, I. V. Hicks, On the 2-club polytope of graphs, Operations Research 64 (6) (2016) 1466–1481.

[15] F. M. Pajouh, Z. Miao, B. Balasundaram, A branch-and-bound approach for maximum quasi-cliques, Annals of Operations Research 216 (1) (2014) 145–161.

[16] J. Pattillo, N. Youssef, S. Butenko, Clique relaxation models in social network analysis, Handbook of Optimization in Complex Networks (2012) 143–162.

[17] J. Pattillo, N. Youssef, S. Butenko, On clique relaxation models in network analysis, European Journal of Operational Research 226 (1) (2013) 9–18.

[18] P. Prosser, Exact algorithms for maximum clique: A computational study, Algorithms 5 (4) (2012) 545–587.

[19] P. San Segundo, D. Rodríguez-Losada, A. Jiménez, An exact bit-parallel algorithm for the maximum clique problem, Computers & Operations Research 38 (2) (2011) 571–581.

[20] S. Shahinpour, S. Butenko, Algorithms for the maximum k-club problem in graphs, Journal of Combinatorial Optimization 26 (3) (2013) 520–554.

[21] S. Trukhanov, C. Balasubramaniam, B. Balasundaram, S. Butenko, Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations, Computational Optimization and Applications 56 (1) (2013) 113–130.

[22] A. Veremyev, O. A. Prokopyev, V. Boginski, E. L. Pasiliao, Finding maximum subgraphs with relatively large vertex connectivity, European Journal of Operational Research 239 (2) (2014) 349–362.

[23] G. Verfaillie, M. Lemaître, T. Schiex, Russian doll search for solving constraint optimization problems, in: W. J. Clancey, D. S. Weld (eds.), Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1, AAAI Press / The MIT Press, 1996.

[24] D. Wen, L. Qin, Y. Zhang, L. Chang, L. Chen, Enumerating k-vertex connected components in large graphs, in: 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019, IEEE, 2019.

[25] Q. Wu, J.-K. Hao, A review on algorithms for maximum clique problems, European Journal of Operational Research 242 (3) (2015) 693–709.

[26] M. Xiao, W. Lin, Y. Dai, Y. Zeng, A fast algorithm to compute maximum k-plexes in social network analysis, in: S. P. Singh, S. Markovitch (eds.), Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA, AAAI Press, 2017.

[27] M. Yannakakis, Node- and edge-deletion np-complete problems, in: R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, A. V. Aho (eds.), Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA, ACM, 1978.

[28] Y. Zhou, J.-K. Hao, Frequency-driven tabu search for the maximum s-plex problem, Computers & Operations Research 86 (2017) 65–78.

[29] Y. Zhou, A. Rossi, J.-K. Hao, Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs, European Journal of Operational Research 269 (3) (2018) 834–843.