Proc. 8th SEA, 2009

Algorithms and Experiments for Clique Relaxations—Finding Maximum s-Plexes

Hannes Moser*, Rolf Niedermeier, and Manuel Sorge**

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{hannes.moser,rolf.niedermeier,manuel.sorge}@uni-jena.de

Abstract. We propose new practical algorithms to find degree-relaxed variants of cliques called s-plexes. An s-plex denotes a vertex subset in a graph inducing a subgraph where every vertex has edges to all but at most s vertices in the s-plex. Cliques are 1-plexes. In analogy to the special case of finding maximum-cardinality cliques, finding maximum-cardinality s-plexes is NP-hard. Complementing previous work, we develop combinatorial, exact algorithms, which are strongly based on methods from parameterized algorithmics. The experiments with our freely available implementation indicate the competitiveness of our approach, for many real-world graphs outperforming the previously used methods.

1 Introduction

Finding maximum-cardinality cliques in graphs now for a long time is a major challenge for algorithmic graph theory and corresponding algorithm engineering efforts (cf. DIMACS challenge [5]). The corresponding Maximum Clique problem is NP-hard and neither effective approximation nor parameterized approaches exist that allow for efficient algorithms with provable performance bounds. Hence, the use of heuristic approaches always has been an important tool for practical solutions of Maximum Clique. The concept of cliques, however, has been criticized for its overly restrictive nature asking for *complete* subgraphs. A more relaxed concept of a dense subgraph has been introduced by Seidman and Foster [14] with the notion of s-plexes. A 1-plex is the same as a clique. For $s \geq 2$, an *s-plex* of a graph $G = (V, E)$ is a vertex set $S \subseteq V$ such that in the induced subgraph $G[S]$ every vertex has degree at least $|S| - s$. Unfortunately, finding maximum-cardinality s-plexes turns out to be computationally basically as hard as clique detection is [2, 8]. Thus, recently the development of practical (heuristic) algorithms for s-plex detection has received quite some interest [2, 9, 15]. In this work, we contribute novel tools for the efficient detection of maximum-cardinality s-plexes. Other than previous work [2, 9, 15] (where [15] deals with s-plex *enumeration*), our algorithms draw on methods from parameterized algorithmics [10].

The Maximum s-Plex problem for an integer $s \geq 1$ is defined as follows.

* Supported by the DFG, project AREG, NI 369/9.
** Supported by the DFG, project PIAF, NI 369/4.

Input: A graph $G = (V, E)$ and a nonnegative integer k.
Question: Is there an s-plex $S \subseteq V$ of size at least k?

Clearly, in our experiments we actually choose to maximize the value of k. Recent work on clique finding has exploited the close connection (indeed, duality) between Maximum Clique and the Minimum Vertex Cover problem [1, 4]. We follow the same spirit here and make use of the duality between Maximum s-Plex and the Minimum d-Bounded-Degree Deletion problem (d-BDD for short). The latter problem is defined as follows.

Input: A graph $G = (V, E)$ and a nonnegative integer k.
Question: Is there a vertex set $S \subseteq V$ of size at most k whose deletion makes $G[V \setminus S]$ a graph of maximum degree d?

Clearly, we are interested in minimizing the value k. The point now is that an n-vertex graph has an s-plex of size k iff its complement graph has a solution set for d-BDD of size $n - k$ with $d := s - 1$. We exploit this close connection by making use of fixed-parameter tractability results for d-BDD [6, 8] and adding some new ones.

Our contributions. On the theoretical side, we provide an improved depth-bounded search tree for 1-BDD (the search tree has size $O(2.31^k)$ instead of previously $O(3^k)$ [8]) and an algorithm for 1-BDD based on iterative compression (exponential factor 2^k). Note that, by duality, these algorithms can be used for finding 2-plexes. Moreover, we present several very effective heuristics (still yielding optimal solution sets) which help to significantly boost the performance of the underlying fixed-parameter algorithms in applications. We perform a number of computational studies, comparing with previous work [2, 9] on exact solutions for s-plex finding which mainly rely on integer linear programming and branch-and-bound. For several real-world graphs, we mostly achieved speedups by orders of magnitude when compared to the previous work. Concerning some dense synthetic instances, we are most of the time slightly slower than approaches based on integer linear programming.

Preliminaries. In this paper, all graphs are simple and undirected. For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, we write $G[S]$ to denote the graph induced by S in G, that is, $G[S] := (S, \{e \in E \mid e \subseteq S\})$. For a vertex $v \in V$, we write $G - v$ instead of $G[V \setminus \{v\}]$ and for a vertex set $S \subseteq V$ we write $G - S$ instead of $G[V \setminus S]$. We define $N(v) := \{u \in V \mid \{u, v\} \in E\}$, $N[v] := N(v) \cup \{v\}$; the *degree* of a vertex v is $|N(v)|$. If every vertex in G has degree at most d, then we say that G has *maximum degree* d. A vertex set $S \subseteq V$ is a *d-bdd-set* if $G - S$ has maximum degree d.

A parameterized problem is *fixed-parameter tractable* if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function depending only on the parameter k, not on the input size n [10]. We also employ search trees for our fixed-parameter algorithms. Search tree algorithms work in a recursive manner. The number of recursion calls is the number of nodes in the according tree. This number is governed by linear recurrences with constant coefficients. These

can be solved by standard mathematical methods [10]. If the algorithm solves a problem instance of size s and calls itself recursively for problem instances of sizes $s - d_1, \ldots, s - d_i$, then $(d_1, \ldots, d_i)$ is called the *branching vector* of this recursion. It corresponds to the recurrence $T_s = T_{s-d_1} + \cdots + T_{s-d_i}$ for the asymptotic size T_s of the overall search tree.

Due to the lack of space, some details are deferred to a full version of the paper.

2 Algorithms

Before coming to some new (mostly fixed-parameter) algorithms, we start with surveying algorithmic approaches that have been developed so far.

Known Approaches. Balasundaram et al. [2] presented a 0/1 integer linear program for Maximum s-Plex, generalizing a known formulation for the special case Maximum Clique. In addition, they carried out a polyhedral study of the problem and discussed a branch-and-cut implementation as the basis of computational tests. In follow-up work, McClosky and Hicks [9] described combinatorial algorithms for Maximum s-Plex, both of heuristic (without provable guarantees on the solution quality) and exact nature. Their heuristic algorithms are based on certain upper and lower bounds for vertex coloring and their exact algorithms are based on adapting known algorithms for Maximum Clique.

As mentioned before and already undertaken for the special cases of Maximum Clique and Minimum Vertex Cover (cf. [1, 4]), an alternative route to solving Maximum s-Plex is to do a “detour” via d-BDD in the complement graph. This is our approach, which, thus, can also be seen as work on d-BDD. Concerning d-BDD, Nishimura et al. [11] presented a depth-bounded search tree yielding a solving algorithm running in $O((d+k)^{k+3} \cdot k + n \cdot (d+k))$ time. Subsequently, an improved simple search tree algorithm running in $O((d+2)^k \cdot (d+k)^2 + n \cdot (d+k))$ time was described [8]. Finally, very recently, an intricate combinatorial data reduction algorithm has been developed [6]. More specifically, it was shown that Minimum d-Bounded-Degree Deletion with a solution set of size k possesses a problem kernel[1] containing at most $(d^3 + 4d^2 + 6d + 4) \cdot k$ vertices, which is computable in $O(n^{5/2} \cdot m + n^3)$ time.

Concerning implementations and experimental work, only the investigations of Balasundaram et al. [2], McClosky and Hicks [9], and Wu and Pei [15] have been accompanied by computational studies. Hence, it is one of the goals of our work to study the practical potential of the new approaches that are based on combinatorial algorithms that avoid polyhedral methods.

Our main algorithm uses a bounded search tree and polynomial-time data reduction rules interleaving with the search tree. In general, the branching strategy of the search tree algorithm chooses a vertex v of degree at least $d+1$, and

[1] Intuitively, a problem kernel is an equivalent problem instance whose size can be upper-bounded by a function independent of the size of the original input instance but only depending on the parameter k (see [10] for details).

then branches into the subcases of deleting v and every possibility of deleting all but d neighbors of v. In this case we say that the strategy "branches on v and $N(v)$". In practice, it is favorable to delete many vertices in each branching step, that is, v should be a vertex of high degree. Most parts of the subsequent descriptions of new algorithmic approaches refer to this.

Conditional application of BDD-Rule. By preliminary experiments, we found out that the direct application of the aforementioned problem kernel of at most $O(d^3 \cdot k)$ vertices is only effective for very few real-world graphs. Therefore, we turned our attention to use the corresponding data reduction rule (called *BDD-rule*) as an interleaving step in a search tree approach. However, applying the rule in every search tree node is not practical. We only apply it in a search tree node if there is a high probability that it will successfully reduce the graph.

Guided branching. The aforementioned problem kernel is based on a $(d+2)$-approximate solution[2] X (hence, $|X| \leq (d+2) \cdot k$). With this size bound on X, by applying the BDD-rule, the size bound for the reduced graph can be derived. This means that the interleaving of this kernel with the search tree algorithm can only be effective if X is small compared to $V \setminus X$ (more precisely, if $|V \setminus X| > (d+1)^2 \cdot |X|$). That is why it is beneficial when the branching strategy tends to branch on vertices in X (thereby deleting more vertices in X) such that after few branching steps X gets small enough. However, in order to decrease the size of X more efficiently, it can be useful to branch on v and only a subset of $N(v)$. To this end, among the vertices of maximum degree, the vertex v to branch on is chosen such that $|N[v] \cap X|$ is maximized and the algorithm only branches on v and $N(v) \cap X$. Since $|X|$ is an upper bound on the size of an optimal solution, this branching strategy can also help in speeding up the search process (by using this upper bound in the search tree to detect branches that cannot lead to a minimum solution) even if interleaving with the BDD-rule is not effective.

Edge-count rule. The *edge-count rule* tests whether the given d-BDD instance is a no-instance. The rule counts how many edges can be deleted from the graph $G = (V, E)$ by at most k vertex deletions based on the vertex degree distribution of the graph. If the number of such edges is too small, then the graph cannot be turned into a graph with maximum degree d by at most k vertex deletions. The number of edges m' that can be deleted by at most k vertex deletions is computed as follows: sort the vertices of G by non-decreasing degree and sum up the degrees of the first k vertices in that order. Then, test whether $m - m' > \frac{d \cdot n}{2}$. If so, then (G, k) is a no-instance. Due to its simplicity, this rule can be implemented to run very efficiently.

Improved search tree for $d = 1$. For the practically relevant special case $d = 1$, we give a more refined branching strategy with an improved search tree size of $O(2.31^k)$. We refrain from conceivable further asymptotic improvements

[2] This $(d+2)$-approximate solution can be computed by greedily finding a maximal collection of vertex-disjoint copies of stars with $(d+1)$ leaves.

(which appear likely when using even further refined branching strategies) in order to keep the algorithm easy to implement and efficient by avoiding the overhead incurred by more complicated strategies.

We start with considering a vertex v of degree $t > 1$. Clearly, v either needs to be deleted or all but one of its neighbors to achieve maximum degree one. Let $N(v) = \{u_1, \ldots, u_t\}$. If not deleting v, branch into the following $t + 1$ subcases:
1. Delete $N(v)$.
2. For each $u_i \in N(v)$, $1 \leq i \leq t$, delete $(N(v) \setminus \{u_i\}) \cup (N(u_i) \setminus \{v\})$.

The correctness of this branching can be seen as follows. First, clearly in each subcase v eventually gets maximum degree one. Second, the branching covers all possibilities how v can be made a maximum-degree-one vertex: one can keep at most one vertex from $N(v)$, the rest has to be deleted. If u_i is the neighbor that shall not be deleted, then clearly all vertices from $N(v) \setminus \{u_i\}$ have to be deleted (otherwise, v would have degree greater than one) and all neighbors of u_i except for v (that is, $(N(u_i) \setminus \{v\})$ have to be deleted (otherwise, u_i would have degree greater than one). Finally, the case of deleting all of $N(v)$ also needs to be considered since, otherwise, one would overlook the situation that all of v's neighbors have to be deleted for reasons lying outside the neighborhood of v. One obtains a branching into $t+2$ cases with the corresponding branching vector

$$(1, t, t - 1 + |N(u_1) \setminus N[v]|, \ldots, t - 1 + |N(u_t) \setminus N[v]|).$$

It is not hard to check[3] that the worst-case branching vector occurs for $t = 2$ and $|N(u_1) \setminus N[v]| = |N(u_2) \setminus N[v]| = 1$, meaning $(1, 2, 2, 2)$ with the branching number 2.31. In analogy to the general result [8], this gives the following.

Theorem 1. Minimum 1-Bounded-Degree Deletion *is solvable in* $O(2.31^k \cdot k^2 + kn)$ *time.*

Theorem 1 is a pure worst-case result. In the implementation, it is clearly favorable to first branch on high-degree vertices (large t-values), making the approach typically much more efficient than the theoretical bound predicts. Without proof, we mention in passing that 1-BDD can be also solved in $O(2^k \cdot k^{5/2} + n + m)$ time using the technique of iterative compression; however, here we focus on the more practical search tree algorithm as described above.

3 Implementation, Algorithmic Tricks, and Experiments

Implementation. Our implementation is written in the functional programming language Objective Caml[4]. A reason for this choice was that we could make use of a purely functional graph data structure. This data structure makes the implementation of a search-tree based algorithm much easier, since we do

[3] We omit some details here; basically, one can argue that for $t = 2$ cases where $|N(u_1) \setminus N[v]| = 0$ are actually easier (often avoiding branching at all) and $t > 2$ gives branching vectors with smaller branching numbers.

[4] See `http://caml.inria.fr/`

```
Algorithm: bddsolve (G, X, k)
Input: A graph G = (V, E), a d-bdd-set X for G, and an integer k ≥ 0.
Output: A minimum-size d-bdd-set S for G with |S| ≤ k, or "no-instance".
 1 S ← ∅
 2 repeat
 3     Remove each vertex v from G for which ∀_{w∈N[v]} deg(w) ≤ d.
 4     while ∃v ∈ V : deg(v) > d + k                          ▷ High-degree rule
 5         G ← G − v; X ← X \ {v}; S ← S ∪ {v}; k := k − 1.
 6     while ∃v ∈ V : v has at least d + 1 degree-1 neighbors  ▷ Degree-1 rule
 7         G ← G − v; X ← X \ {v}; S ← S ∪ {v}; k := k − 1.
 8     if |N(X)| > (d + 1) · |X| then
 9         call BDD-rule to obtain vertex sets A and B [6]
10         G ← G − (A ∪ B); X ← X \ A; S ← S ∪ A; k ← k − |A|
11 until none of the rules applies.
12 if k < 0 then return "no-instance"
13 l := greedily computed lower bound of the size of a minimum d-bdd-set.
14 if k < l or edge-rule tells "no-instance" then return "no-instance"
15 if maximum degree of G is d then return S
16 Among all max.-deg. vertices, choose a vertex v where |N[v] ∩ X| is maximum.
17 if |N(v) ∩ X| > d then                           ▷ Branch on v and N(v) ∩ X
18     call bddsolve (G − v, X \ {v}, k − 1)
19     for all size (|N(v) ∩ X| − d)-subsets C ⊆ N(v) ∩ X do
20         call bddsolve (G \ C, X \ C, k − |C|)
21 else branch analogously to lines 18–20 on v and N(v).
22 if all recursive calls of bddsolve returned "no-instance" then
23     return "no-instance"
24 else return S ∪ S′, where S′ is a smallest set returned by the bddsolve calls.
```

Fig. 1: Pseudocode of the basic algorithm to compute a minimum d-bdd-set.

not have to care about undoing changes to the data structure that were applied in other search tree branches. Moreover, it is a stated (and usually achieved) goal of the Objective Caml developers that Objective Caml code runs at most twice as slow as code generated by a decent C compiler. This speed difference is not a major factor for our considerations, since we are interested in the relative performance of algorithms. Moreover, since we are dealing with exponential-time algorithms, algorithmic improvements usually lead to time savings that cannot be bounded by any constant factor, so this effect seems small in comparison.

Our implementation is open source and it is freely available.[5] In Figure 1, we give the pseudocode of the basic search tree algorithm to compute a minimum d-bdd-set of size at most k for a graph. The data reduction rules in lines 3–7

[5] `http://theinf1.informatik.uni-jena.de/splex/`

[6] The BDD-rule [6] returns two disjoint vertex sets A and B such that there exists a minimum-cardinality d-bdd-set S with $A \subseteq S$ and $S \cap B = \emptyset$.

remove parts of the graph that can be omitted from further consideration (line 3), high-degree vertices (lines 4–5), and some neighbors of degree-1 vertices (lines 6-7). The simple correctness proofs for these rules are omitted here. Note that the rules not only have to delete vertices from the graph G, but also from the d-bdd-set X (see "guided branching" in Section 2), in order to preserve the invariant that X is a d-bdd-set for G. Concerning the BDD-rule (lines 8–10), we changed the condition from $|V \setminus X| > (d+1)^2 \cdot |X|$ (which guarantees success of the BDD-rule application, see Section 2) to $|N(X)| > (d+1) \cdot |X|$ (which makes the success of the BDD-rule probable in practice, even if the condition $|V \setminus X| > (d+1)^2 \cdot |X|$ is not met). In lines 12–15 we perform several tests whether the instance resulting by the application of the data reduction rules is a no-instance. In line 15 we test whether the instance has already bounded degree d. Then, in line 16 the algorithm selects a vertex to branch on. The branching is then performed in lines 18–21. Then, in lines 22–24 the algorithm either returns that the input instance is a no-instance or returns the best solution that it has found.

Algorithmic Tricks. Concerning the initial $(d + 2)$-approximate solution X needed for the guided branching, it turns out that a greedy solution, computed by simply taking a vertex of highest degree into the solution until the remaining graph has bounded degree d, very often is smaller than a $(d + 2)$-approximate solution, although this method does not provably guarantee an approximation factor of $d + 2$. Such a greedy solution is also computed at the beginning of the computation (before invoking the search tree algorithm), and its size is taken as the initial value of k. Note that our implementation contains many algorithmic tweaks that are not covered by the basic description in Figure 1. For instance, the effect of the guided branching can be improved by recomputing X from time to time in the course of the branching process. Moreover, it improves performance significantly if one updates the value of k if a branch has found a solution that is smaller than the initial k. For $d = 1$, we implemented the improved branching described in Section 2 instead of the branching shown in Figure 1.

In the following, we comment about some particularities of our search tree implementation. One of the most important issues was the computation of the complement graph, which has to be performed before executing the **bddsolve** algorithm (Figure 1). For sparse graphs, the complement graph is dense and, surprisingly, in practice the amount of time and memory to compute it exceeds often the time and memory needed for finding a maximum s-plex. Therefore, we implemented a wrapper that simulates a complement graph, rather than actually computing it. This wrapper, of course, is theoretically slower than the original graph data structure, since the data structure calls have to be translated by the wrapper. However, in practice, this method turns out to be almost always much more efficient than computing the complement graph directly.

For the graphs we considered, it turned out that applying the data reduction rules (see lines 3–10 in Figure 1) in every search tree node yields the best results. In particular, the degree-one rule and the high-degree rule are mostly very effective. To be able to apply these rules more quickly, it seems to be reasonable to implement a data structure that provides fast access to vertices with a particular

degree. However, this results in an increase of memory usage, and since the data structure has to be updated very frequently, many operations take more time. For instance, the deletion of a vertex, which is one of the most frequently called routines, needs about twice the time in our experiments. Moreover, we noticed an increased garbage collection overhead. Summarizing, such a data structure slows down the algorithm; surprisingly, for the degree-one and the high-degree rule a simple sweep over all vertices gives a faster implementation.

Experiments. All experiments were run on AMD Athlon 64 3700+ machines with 2.2 GHz, 1 M L2 cache, and 3 GB main memory running under the Debian GNU/Linux 4.0 operating system with the Objective Caml 3.09.2 compiler. The experiments of Balasundaram et al. [2] were performed on Dell Precision PWS690 machines with a 2.66 GHz Xeon Processor, 3 GB main memory, implemented using ILOG CPLEX 10.0. The processor speeds are comparable, so we compare the running times directly without applying a correction factor. The experiments of McClosky and Hicks [9] were run on a 2.2 GHz Dual-Core AMD Opteron processor with 3 GB main memory. We assume that their implementation uses one core only and compare the running times directly. Note that for both papers [2, 9] the corresponding source code is not publicly available.

Balasundaram et al. [2] performed experiments with two groups of graphs. One group can be characterized as social networks, which are derived from real-world data. The second group of graphs contains various graphs using the *Sanchis generator* [13] and clique instances from the second DIMACS challenge [5]. Balasundaram et al. [2] used an integer linear programming formulation combined with branch & cut methods. One of their exact algorithms, called BC(MIS), generates cuts based on a greedily computed independent set. For the real-world graphs, they use a variant called IPBC, which iterates over all vertices and searches an s-plex only in the vicinity of each iterated vertex (using the BC(MIS) approach). In the following, we compare our approach with the BC(MIS) and IPBC algorithms and also with the exact algorithm "OsterPlex" by McClosky and Hicks [9], which is an adapted version of an algorithm for finding maximum-cardinality cliques by Östergård [12]. The experiments of McClosky and Hicks [9] cover almost all social networks and the instances from the DIMACS challenge.

Social Networks. This group contains a set of Erdős collaboration networks [7] (ERDŐS graphs), collaboration networks in computational geometry [3] (GEOM graphs), and text-mining networks based on Reuters news [3] (DAYS graphs). Due to space constraints, we omit the DAYS graphs; our results for ERDŐS and GEOM graphs also hold for the DAYS graphs in the qualitative sense.

ERDŐS graphs: Each vertex in an Erdős graph represents a scientist, and two vertices are adjacent if the corresponding scientists have published together. The graphs, obtained from [7], are named "ERDOS-x-y", where x represents the last two digits of the year for which the network was constructed, and y the maximum distance from each vertex to Erdős in the graph. As Balasundaram et al. [2] and McClosky and Hicks [9], we consider $x \in \{97, 98, 99\}$ and $y \in \{1, 2\}$.

Table 1: Running time and number of search tree nodes for ERDŐS and GEOM graphs compared with the running times of the IPBC and OsterPlex algorithm. Note that the OsterPlex experiments [9] were aborted after one hour.

s	graph	IPBC	OsterPlex	search tree algorithm			
				no guided branching		guided branching	
		seconds [2]	seconds [9]	seconds	nodes	seconds	nodes
2	ERDOS-97-1	2.9	0	0.9	179	0.3	311
	ERDOS-97-2	2123	1253	12.7	187	8.6	502
	ERDOS-98-1	2.2	0	1.1	201	0.4	358
	ERDOS-98-2	2251	1514	33.1	181	9.8	398
	ERDOS-99-1	4.2	0	1.2	212	0.4	357
	ERDOS-99-2	2442	1757	44.1	194	11.0	414
3	ERDOS-97-1	7.2	19	25.5	118620	0.7	10295
	ERDOS-97-2	32773	$\geq$ 3600	620	596753	14.6	54695
	ERDOS-98-1	17.8	20	11.7	51965	1.0	13637
	ERDOS-98-2	45448	$\geq$ 3600	762	694455	26.3	120605
	ERDOS-99-1	15.6	21	12.6	56704	1.7	28753
	ERDOS-99-2	40164	$\geq$ 3600	1425	969064	36.8	132981
2	GEOM-0	12147	397	5.2	0	5.2	0
	GEOM-1	946	1118	0.3	20	0.3	20
	GEOM-2	487	1145	0.2	17	0.1	32
3	GEOM-0	20948	$\geq$ 3600	5.2	0	5.2	0
	GEOM-1	1027	$\geq$ 3600	1.0	5065	0.4	887
	GEOM-2	489	$\geq$ 3600	0.2	1225	0.1	3

GEOM graphs: Each vertex represents an author in computational geometry. For each pair of authors the number of joint publications is available. Given a threshold t, two authors are adjacent if they have more than t joint publications. The graphs, obtained from [3], are named "GEOM-t", where $t \in \{0, 1, 2\}$.

We compared the IPBC algorithm [2] and the OsterPlex algorithm [9] with our methods. We discovered experimentally that the guided branching has a strong effect on the running time for these instances, while the BDD-rule and the edge-count rule had only minuscule effects. Therefore, we performed experiments with and without guided branching. The resulting running times for the ERDŐS and GEOM graphs are given in Table 1. For the ERDŐS graphs, our method without guided branching outperforms the approach of Balasundaram et al. [2] by one or two orders of magnitude. With guided branching, the running time is improved by three orders of magnitude. To our surprise, the BDD-rule (almost) does not apply at all. The reason is that X (see "guided branching" in Section 2) is rather big, and we apply the high-degree rule first (see Figure 1), which reduces the graph so effectively that the condition for applying the BDD-rule is (almost) never met. When switching off the high-degree rule, almost all reduction is then performed by the BDD-rule. The OsterPlex algorithm [9] is mostly faster than the IPBC algorithm [2], and for some instances it has running times comparable

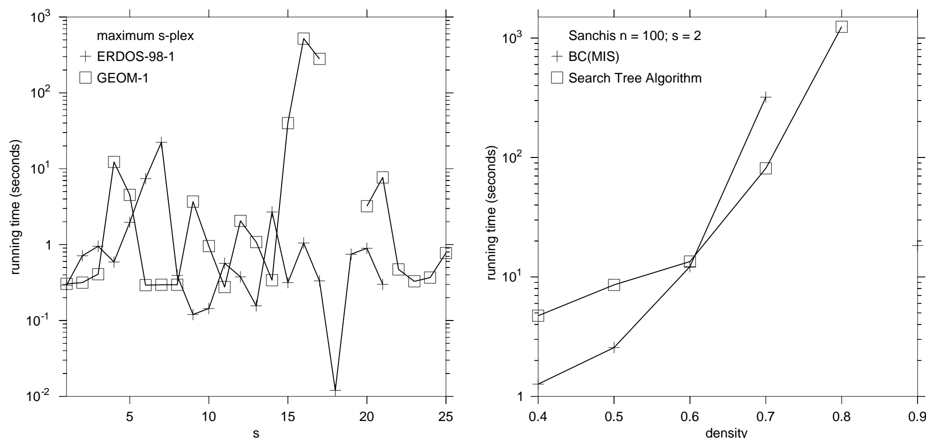

Fig. 2: Left: running times of our approach for $1 \leq s \leq 25$ on the ERDOS-98-1 and the GEOM-1 graph. Missing data points are due to exceeded running time limit of 60 minutes. Right: running times of our approach (search tree algorithm) compared with the running times of the BC(MIS) approach [2].

to our approach with guided branching, while in general it is about two orders of magnitude slower than our approach.

For the GEOM graphs, we observe similar speedups of up to three orders of magnitude (see Table 1). Interestingly, for some instances our approach does not branch at all; it immediately finds a solution using the data reduction rules. Since the data reduction rules are very effective and few branchings take place, the effect of the guided branching is not as pronounced as for the ERDŐS graphs.

Since the preceding experiments indicate that the running time of our approach does not increase too much with increasing s (recall that $s = d + 1$), we performed experiments on two of the real-world graphs for $1 \leq s \leq 25$. The results are shown in Figure 2. For most values of s, the instances can be solved within some seconds, only some take around one hour or more. Interestingly, there is a peak of the running time around $s = 19$. We conclude that our approach seems to be able to find maximum s-plexes for a wide range of the parameter s for these types of graph.

Sanchis and DIMACS Graphs. The Sanchis generator [13] produces graphs with known maximum clique size with a specified number of vertices n and edges m, and a construction parameter r. As Balasundaram et al. [2], we fixed the maximum clique size at $\lceil n/5 \rceil$, and the construction parameter to $\lfloor 0.75(n/c-1) \rfloor$. The number of edges is determined by the density d, that is, we compute the number of edges as $m := \lfloor dn(n-1)/2 \rfloor$. We performed experiments for $n \in \{100, 200\}$ and $d \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

Balasundaram et al. [2] used Sanchis graphs to study how the efficiency of their methods depends on the number of graph vertices, the density of the graph, and on the value s defining s-plexes. Their methods perform best on sparse

graphs, and become less effective on dense graphs. Likewise, small graphs can be solved quickly, while larger graphs become more difficult to solve. Balasundaram et al. [2] performed experiments with the BC(MIS) algorithm for $s \in \{1, 2\}$. They observed that the case $s = 2$ is generally more difficult to solve than $s = 1$.

We observe the same general behavior as for the BC(MIS) algorithm, that is, dense Sanchis graphs are harder to solve than sparse ones, and graphs with many vertices are harder to solve than graphs with few vertices. We can observe that, especially on sparse instances, our approach is about one order of magnitude slower than the BC(MIS) algorithm (see Figure 2). However, the available data seems to indicate that the running time of our approach increases not as quickly with increasing density as the BC(MIS) algorithm does, but this needs to be checked more carefully with further experimentation, also including higher values of s. For Sanchis graphs with more vertices, there are too few instances where the BC(MIS) algorithm gave exact results within a three-hour running time limit in order to do a similar comparison, and likewise our approach did mostly not terminate within that time.

Finally, we briefly report about our findings concerning instances from the DIMACS challenge. Here, we compare with the BC(MIS) algorithm [2] and the OsterPlex algorithm [9]. Summarizing, out of the 32 considered instances we could solve 23 instances for $s = 1$ and 14 instances for $s = 2$, while BC(MIS) could solve 20 instances for $s = 1$ and 16 instances for $s = 2$ within a running time limit of three hours. For the instances that neither BC(MIS) nor our approach could solve exactly within three hours, we observe that our lower/upper bounds seem to be worse than the ones computed by BC(MIS). Compared to the OsterPlex algorithm, we could solve within one hour all but five instances for $s = 2$, which OsterPlex can solve within that time. Summarizing, BC(MIS) and OsterPlex are at least as good as our approach for these instances. In this respect, it would be interesting to study whether the OsterPlex and the BC(MIS) algorithms could be efficiently combined with ours.

4 Conclusion and Outlook

In some analogy to previous work on maximum-cardinality clique finding [1, 4], we demonstrated that an exact combinatorial approach provides competitive algorithms for finding maximum-cardinality s-plexes. Clearly, due to the NP-hardness of the problem, there are limitations concerning the range of practical feasibility. On the one hand, we believe that there is still some room for further tuning our algorithms and implementations (which in future work also should be compared with other approaches in an experimental study that is based on the *same* platform); on the other hand, we think that at some point more restrictions such as the one of "isolation" (see [8]) have to be imposed in order to gain practical algorithms. Our focus was on finding s-plexes of maximum size; studies concerning efficient approximation algorithms are left open.

References

[1] F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41 (3):411–430, 2007.

[2] B. Balasundaram, S. Butenko, I. V. Hicks, and S. Sachdeva. Clique relaxations in social network analysis: The maximum k-plex problem. URL `http://iem.okstate.edu/baski/files/kplex4web.pdf`. Manuscript, February 2008.

[3] V. Batagelj and A. Mrvar. Pajek datasets, 2006. URL `http://vlado.fmf.uni-lj.si/pub/networks/data/`. Accessed January 2009.

[4] E. J. Chesler et al. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nat. Genet.*, 37(3):233–242, 2005.

[5] DIMACS. Maximum clique, graph coloring, and satisfiability. Second DIMACS implementation challenge, 1995. URL `http://dimacs.rutgers.edu/Challenges/`. Accessed November 2008.

[6] M. R. Fellows, J. Guo, H. Moser, and R. Niedermeier. A generalization of Nemhauser and Trotter's local optimization theorem. In *Proc. 26th STACS*, pages 409–420. IBFI Dagstuhl, Germany, 2009.

[7] J. Grossman, P. Ion, and R. D. Castro. The Erdős number project, 2007. URL `http://www.oakland.edu/enp/`. Accessed January 2009.

[8] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for enumerating dense subgraphs. In *Proc. 13th COCOON*, volume 4598 of *LNCS*, pages 140–150. Springer, 2007.

[9] B. McClosky and I. V. Hicks. Combinatorial algorithms for the maximum k-plex problem. URL `http://www.caam.rice.edu/~bjm4/CombiOptPaper.pdf`. Manuscript, January 2009.

[10] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[11] N. Nishimura, P. Ragde, and D. M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of Vertex Cover. *Discrete Appl. Math.*, 152(1–3):229–245, 2005.

[12] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120(1-3):197–207, 2002.

[13] L. A. Sanchis and A. Jagota. Some experimental and theoretical results on test case generators for the maximum clique problem. *INFORMS J. Comput.*, 8(2):103–117, 1996.

[14] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978.

[15] B. Wu and X. Pei. A parallel algorithm for enumerating all the maximal k-plexes. In *Emerging Technologies in Knowledge Discovery and Data Mining*, volume 4819 of *LNAI*, pages 476–483. Springer, 2007.