FixCon: A Generic Solver for Fixed-Cardinality Subgraph Problems[*]

Christian Komusiewicz[†] Frank Sommer[‡]

Abstract

In fixed-cardinality optimization problems in graphs, we are given a graph $G = (V, E)$, an objective function f, and an integer k and search for a set $S \subseteq V$ of k vertices that maximizes $f(G[S])$ where $G[S]$ is the subgraph of G induced by S. We implement an enumeration-based algorithm for solving fixed-cardinality optimization problems when $G[S]$ needs to be connected. To avoid enumerating all connected subgraphs of order k, we present several generic pruning rules and a generic heuristic for computing a lower bound for the objective value. We perform an experimental analysis of the performance of the algorithm and the usefulness of the pruning rules for eight example problems in which one aims to find dense, sparse, or degree-constrained connected subgraphs, respectively. Our experiments show that, when this generic solver is combined with problem-specific pruning rules, our algorithm is competitive with out-of-the-box ILP formulations for these problems.

1 Introduction

In this work, we engineer an algorithm for the following generic NP-hard graph problem.

> Connected Fixed-Cardinality Optimization (CFCO)
> **Input:** An undirected simple graph $G = (V, E)$, an objective function $f\colon \mathcal{G} \to \mathbb{R}$, and an integer k.
> **Task:** Find a set $S \subseteq V$ of size k such that $G[S]$ is connected and S maximizes $f(G[S])$ under these conditions.

Herein, $\mathcal{G}$ is the set of all undirected simple graphs and $G[S] \coloneqq (S, \{\{u, v\} \in E \mid u, v \in S\})$ denotes the *subgraph induced by S*. Throughout this work, we assume that f depends only on the isomorphism class of $G[S]$. In other words, for any two isomorphic graphs H and H' we have $f(H) = f(H')$.

CFCO has applications for example in computational biology [1] and facility layout problems [6]. Due to the generic nature of CFCO it contains Clique as a special case and thus it is NP-hard. On the positive side, CFCO can be solved in $O((e(\Delta-1))^k \cdot |V|^{O(1)} \cdot T_f(k))$ time where $T_f(k)$ is the time needed for evaluating f on graphs of order k and Δ is the maximum degree of G [10].

The algorithm achieving this running time is quite simple: enumerate all connected induced subgraphs of order at most k and evaluate f for those subgraphs $G[S]$ of order *exactly* k, keeping the best subgraph $G[S]$ and outputting S after the enumeration has finished. An implementation of this algorithm was developed for the special case when the task is to decide whether G contains an order-k connected μ-clique, where a graph with k vertices is a μ-clique if it has at least $\mu \cdot \binom{k}{2}$ edges [11]. As a proof of concept, this implementation showed that enumeration of connected subgraphs can be a useful algorithmic approach for special cases of CFCO. A major drawback of this approach, however, is that for each new problem one has to provide a new implementation and develop a new set of reduction and pruning rules. This makes it unlikely that these algorithms will find widespread use in real-world applications.

In contrast, integer programming and SAT solving has proved to be an extremely useful tool for solving NP-hard problems. One reason for this is that SAT and ILP solvers have been engineered over decades and, as a result, can solve many real-world instances very quickly. Another reason is that, due to the generic nature of SAT and ILP, one may formulate many combinatorial optimization problems rather easily as a SAT or ILP problem.

In this work, we aim to lift enumeration-based algorithms from specific to generic applications. We develop the algorithmic tool *FixCon* in which the user needs to program only the objective function f. The user may furthermore provide some properties of the objective function that will then be exploited by generic pruning rules that restrict the search space for the enumeration algorithm. The pruning rules assume only the correctness of the provided properties and otherwise treat f as a black box. In addition, the user may implement problem-specific pruning rules. Since the theoretical running time guarantees for CFCO are good

[*]To appear in *Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX '20)*, Salt Lake City, USA, January 2020. © SIAM.

[†]Philipps-Universität Marburg, Germany, komusiewicz@informatik.uni-marburg.de

[‡]Philipps-Universität Marburg, Germany, fsommer@informatik.uni-marburg.de

Supported by the DFG, project MAGZ (KO 3669/4-1).

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

only in the case of small k [4, 10], we focus on engineering *FixCon* for $k \leq 20$.

Our contribution. We implement three variants of the generic enumeration-based algorithm of Komusiewicz and Sorge [10]. We then provide three generic pruning rules that help decreasing the size of the search tree used by the enumeration algorithm. To develop these rules, we note and make use of two properties of the objective function f: vertex-addition bounds (that limit the change of the objective value when a vertex is added to the graph) and edge-monotonicity (which means that adding an edge to a graph does not decrease its objective value). In addition, we decrease the search space size by identifying vertices with the same neighborhoods in the input graph (called *twins*) and by identifying cases in which a current subgraph cannot be extended to one that gives a better solution than the current one. The latter two rules are valid for any function f that assigns the same value to graphs from the same isomorphism class. Moreover, we provide generic heuristics that provide *FixCon* with lower bounds for the value of the optimal solution. Finally, for some problems, we provide new problem-specific reduction rules.

We analyze the algorithms and the effect of the generic and problem-specific pruning rules for eight example problems and compare them with ILP formulations for these problems. In a nutshell, we show that the best version of our algorithm can solve the majority of the benchmark instances within 600 seconds per instance. Moreover, we outperform the ILP formulations in terms of number of solved instances.

Our source code is available at `https://www.uni-marburg.de/en/fb12/research-groups/algorith/software/fixcon`.

Related work. Bruglieri et al. [3] give a systematic survey of special cases of Fixed-Cardinality Optimization. Another generic algorithm for FCO relies on the random separation technique [4]. We did not choose this technique as basis for our implementation since it has worse running time guarantees and since it seems hard to exploit properties of the objective functions during the algorithm. Maxwell et al. [13] propose an algorithm to enumerate *all* connected induced subgraphs H that for a given "hereditary" objective function f and threshold t, fulfill $f(H) > t$. We do not compare with Maxwell et al. [13] since the enumeration problem is much harder: our algorithm may stop immediately after finding one optimal solution; in the enumeration problem, one must output all solutions and thus continue the search. From another perspective, our implementation is more general, as f does not need to be hereditary in *FixCon*.

Notation. We consider undirected graphs $G = (V, E)$, and denote by $E(G)$ the edge set of G and by $V(G)$ its vertex set. For a vertex v, $N_G(v) := \{u \mid \{u, v\} \in E\}$ denotes the *open neighborhood* of v, and $N_G[v] := N(v) \cup \{v\}$ denotes the *closed neighborhood* of v. For a set $S \subseteq V$, $N_G(S) := \bigcup_{v \in S} N(v) \setminus S$ denotes the *open neighborhood of* S; $N_G[S] := N(S) \cup S$ denotes the *closed neighborhood of* S. If there is no danger of confusion, we skip the subscript G. For a vertex $v \in V$, $G - v := G[V \setminus \{v\}]$ denotes the graph G without vertex v. For two vertices u and v in a graph G, $\text{dist}_G(u, v)$ denotes the length of a shortest path from u to v in G. For two graphs G and H, we let $H \cong G$ denote that H and G are isomorphic. A graph is *r-regular* if every vertex has degree r.

2 Example Problems

In this section, we describe eight example objective functions that we will consider as input in the CFCO instances.

Dense subgraph problems. The first two problems have applications in finding dense cohesive subgraphs. In the first, we aim to maximize the number of edges in the subgraph.

Densest Subgraph:

$$f(H) := |E(H)|.$$

In the second problem we aim to find a graph in which the minimum degree is large.

Max-Min-Degree Subgraph:

$$f(H) := \min_{v \in V(H)} \{|N_H(v)|\}.$$

For both problems, cliques of order k give the best objective values for all graphs of order k. A further problem in this direction would be to minimize the diameter of the induced subgraphs. We did not include this problem in the comparison since for $k \leq 20$, finding solutions with diameter 2 is trivial in our instances and thus the problem is essentially only to decide whether there is a clique on k vertices.

Sparse subgraph problems. The next four problems are concerned with finding sparse subgraphs. The first is essentially the opposite of Max-Min-Degree, that is, we aim to find a subgraph with a minimal maximum degree; we formulate it as a maximization problem.

Min-Max-Degree Subgraph:

$$f(H) := -\max_{v \in V(H)} \{|N_H(v)|\}.$$

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

The next problems are essentially recognition problems, formulated as maximization problems. We may find induced subgraphs that are trees by solving the following problem.

Acyclic Subgraph:

$$f(H) \coloneqq \begin{cases} 1 & H \text{ is a tree,} \\ 0 & \text{otherwise.} \end{cases}$$

In a similar fashion, we may look for connected induced subgraphs that contain no triangle.

Triangle-Free Subgraph:

$$f(H) \coloneqq \begin{cases} 1 & H \text{ contains no triangle,} \\ 0 & \text{otherwise.} \end{cases}$$

The final problem from this group is to search for a subgraph that has a large diameter.

Maximum-Diameter Subgraph:

$$f(H) \coloneqq \max_{u,v \in V(H)} \operatorname{dist}_H(u,v).$$

Degree-constrained subgraph problems. In the final two problems we consider two variants of restricting vertex degrees. Again, these problems are basically decision problems, formulated as maximization problems.

r-Regular Subgraph:

$$f(H) \coloneqq \begin{cases} 1 & H \text{ is } r\text{-regular,} \\ 0 & \text{otherwise.} \end{cases}$$

In the experiments, we use $r = 3$ for even k and $r = 4$ for odd k.

(a, b)-Degree-Constrained Subgraph:

$$f(H) \coloneqq \begin{cases} 1 & H \text{ has minimum degree at least } a \\ & \text{ and maximum degree at most } b, \\ 0 & \text{otherwise.} \end{cases}$$

We set $a = 3$ and $b = 5$ in the experiments.

3 Experimental Setup

Each experiment was performed on a single thread of an Intel(R) Xeon(R) Silver 4116 CPU with 2.1 GHz, 24 CPUs and 128 GB RAM running Python 3.6.8. We used igraph with the python-igraph interface (`http://igraph.org/python/`) as the graph data structure. We performed experiments on 40 real-world networks from Konect [12], the DIMACS Challenge on Graph Clustering and Partitioning [2], and the Network Repository [15]; 10 networks are sparse and small (less than 500 vertices), 10 are sparse and have medium size (500 – 5 000 vertices), 10 are sparse and large (between 5 000 and 500 000 vertices), and 10 are dense. The instance names and some of their properties are shown in Table 1 in the appendix. To also study instances with different structure, we built 20 random instances in the $G_{n,p}$ model with $n \in \{100, 200, \dots, 1000\}$ and $p \in \{0.1, 0.2\}$. We set a timeout of 600 seconds minutes per instance. The time for reading the input is not included in the running time, for the ILP we do include the time for passing the initial set of constraints. For each graph, we built an instance for each of the eight problems and each $k \in \{4, \dots, 20\}$. We call k *small* if $k \le 10$ and *large* if $11 \le k \le 20$.

We add improvements to the algorithms and evaluate their effect one by one. In any section, we compare the version with the new improvement and all previous improvements to the variant that has all previous improvements but not the new one.

Algorithm 1 The *Simple* algorithm; the initial call is *Simple*$(\{v\}, N(v))$.

```
1: procedure Simple($C, X$)
2:     if $|C| = k$ then
3:         store $C$ if $f(G[C])$ is maximal
4:         return
5:     while $X \neq \emptyset$ do
6:         $u \coloneqq$ arbitrary vertex of $X$        ▷ Create branch that adds $u$ to $C$
7:         $X \coloneqq X \setminus \{u\}$
8:         $X' \coloneqq X \cup (N(u) \setminus N[C])$
9:         Simple($C \cup \{u\}, X'$)
10:    return
```

Algorithm 2 The *Pivot* algorithm; the initial call is *Pivot*$(\{v\}, \{\}, \{\})$.

```
1: procedure Pivot($P, Q, F$)
2:     if $|P \cup Q| = k$ then
3:         store $P \cup Q$ if $f(G[P \cup Q])$ is maximal
4:         return
5:     while $P \neq \emptyset$ do
6:         $p \coloneqq$ choose element of $P$
7:         for each $z \in N(p) \setminus (P \cup Q \cup F)$ do
8:             Pivot($P \cup \{z\}, Q, F$)
9:             $F \coloneqq F \cup \{z\}$
10:        $P \coloneqq P \setminus \{p\}$
11:        $Q \coloneqq Q \cup \{p\}$
12:    return
```

4 Enumeration Algorithms

We consider three different algorithms for enumerating connected induced subgraphs of order k. In all of them, there is one main algorithm loop (Algorithm 1

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

and Algorithm 2) which considers, in some order, each vertex $v \in V(G)$, enumerates all connected induced subgraphs of order k containing v, and then removes v from G.

All three algorithms are search tree algorithms in which each node of the search tree is associated with a set C of vertices that induces a connected graph, called a *connected set* in the following. In each search tree node with $|C| < k$, we branch into the possibilities to add vertices of $N(C)$. The most straightforward algorithm following this paradigm was used as a basic routine in the FANMOD tool for network motif detection [16, 17]. We refer to this algorithm as *Simple*. The algorithm maintains a set X during the search, which is the set of vertices in $N(C)$ that may still be added to C. For each vertex $u \in X$, we create a branch where we add u to the connected set C. Afterwards, all vertices of $N(u) \setminus N[C]$ are added to X. After this branch, u is removed from X, as we already considered the possibility to add it to the current subgraph.

The two other algorithms that we use are called *Pivot* [9] and *Kavosh* [7], the latter has also been used in network motif detection. Both follow a similar approach with a small difference. In both algorithms the connected set C is partitioned into a set P which can have further neighbors and a set Q which can have no further neighbors. Further, for each search tree node there is a set F of forbidden vertices which will not be added to C. Note, that the set of forbidden vertices in *Simple* is implicit defined by $N[C] \setminus X$.

In *Pivot*, for each vertex $p \in P$ we create for each vertex $w \in N(P) \setminus (P \cup Q \cup F)$ a branch where we add w to the connected set C. After returning from this branch, w is added to the set of forbidden vertices F. After considering all such vertices w, vertex p is moved to the set Q which is not allowed to have further neighbors.

In *Kavosh*, we do not add single vertices of $N(P)$ in the branching but instead create one branch for each subset of size at most $k - |P|$ of $N(P) \setminus (P \cup Q \cup F)$. Hence, the search tree in *Kavosh* has, on average, smaller *depth* and larger *breadth* than the search trees of *Simple* and *Pivot*.

All three algorithms were implemented by the authors in previous work [8]; we use this previous implementation with one difference: instead of using recursion, we now use arrays each with up to k pointers on their positions to replace the recursion stack. This proved to be substantially faster than the previous implementation. The pseudocode of *Simple* and *Pivot* is shown in Algorithm 1 and 2, respectively; it is adapted to fixed-cardinality optimization since instead of outputting the connected set C, we evaluate $f(G[C])$ and save C if it gives the current-best objective value.

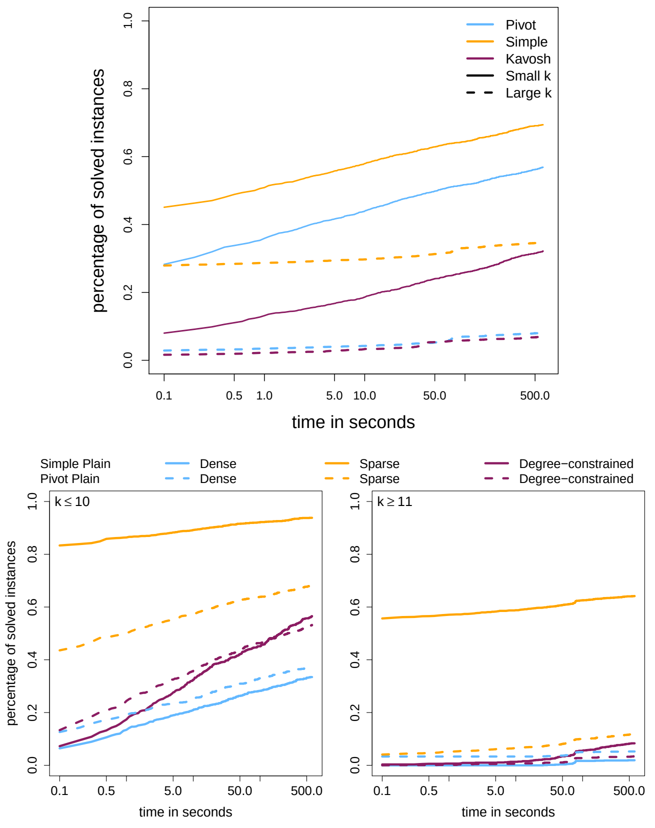

Figure 1: Top: Comparison of the plain version of the three algorithms. Bottom: Comparison of the running times of the plain version of *Simple* and *Pivot* for the three problem categories.

In the plain version of our implementation, we use the following easy way to prune the search space: we stop the search once a solution that is obviously optimal has been found. To this end, users may provide *FixCon* with a *global* upper bound for the value of f on graphs of order k. For many problems such a bound is easy to determine. For DENSEST SUBGRAPH the global upper bound is $\binom{k}{2}$, for MAX-MIN-DEGREE SUBGRAPH it is $k - 1$. For MIN-MAX-DEGREE SUBGRAPH, the global upper bound is -2, as a connected graph on at least four vertices has at least one vertex that has two or more neighbors. For MAXIMUM-DIAMETER SUBGRAPH, the global upper bound is $k - 1$, this bound is met only by the path on k vertices. Finally, for the remaining problems which all describe graph properties, the best objective value is 1 which can be seen directly from the definition.

The performance of the plain version of our algorithms is shown in Figure 1. The top part visualizes the running times for the three enumeration algorithms.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

Kavosh is substantially slower than *Pivot* which is again substantially slower than *Simple*. In all further versions of the algorithms that we tested, *Kavosh* was slower than the other two algorithms. This is due to the structure of the enumeration tree in *Kavosh* which has many leaves (corresponding to subgraphs of order k) and few inner nodes (corresponding to subgraphs of order less than k). Since the further improvements are rooted in the idea to prune the search tree at some inner node, *Kavosh* benefits less from these improvements. Hence, to improve the presentation, *Kavosh* is excluded from further experiments. The bottom part of Figure 1 shows the performance of the plain versions of *Simple* and *Pivot* separated by categories. *Pivot* is only faster in the dense category. The main observations are that the sparse problems are the easiest for the plain algorithm version, followed by the degree-constrained problems, and the dense problems which are the hardest. *Simple* solved 69 % of the instances with small k and 34% of the instances with large k. Thus, as expected, the instances with large k are much harder than those with small k.

5 Pruning Rules

The plain algorithm only prunes the search tree if the graph contains a solution that meets the global upper bound which is often not the case. Thus, we propose three additional pruning rules that help to further decrease the number of search tree nodes. In these pruning rules, we use the current connected set C to obtain an upper bound on the value of $f(G[S])$ for all $S \supset C$. In the formulation of the pruning rules, we denote $G[C]$ by H_C for brevity.

Vertex-based Upper Bounds. To allow for a computation of an upper bound, we consider the following property of objective functions f.

Definition 1. *An objective function f is* vertex-addition-bounded by value x, *if for every graph H and all graphs H^* that are obtained by adding some vertex to H and making this vertex adjacent to some subset of $V(H)$, we have $f(H^*) \leq f(H) + x$.*

We may now use the following rule.

Pruning Rule 1. (*Vertex-addition Rule*) *Let C be the current connected set, let $|C| = k - \ell$, let f be vertex-addition-bounded by x, and let z denote the objective value of the current best solution. If $f(H_C) + \ell \cdot x \leq z$, then return to the parent node in the search tree.*

We can use the *Vertex-addition Rule (VAR)* for all eight problems. For example, the objective function for Triangle-Free Subgraph is vertex-addition-bounded by 0, since adding a new vertex to H_C can only introduce new triangles and never destroys existing triangles. More generally, when the aim is to find an induced subgraph fulfilling some hereditary property, then we may use an objective function that is vertex-addition-bounded by 0. Hence, Acyclic Subgraph is also vertex-addition-bounded by 0.

For Min-Max-Degree Subgraph the vertex-addition bound is 0 as adding a vertex does not decrease the maximum degree. For Maximum-Diameter Subgraph the vertex-addition bound is 1, as adding a vertex may increase the diameter by at most 1. For the dense category, the vertex-addition bound is $k-1$ for Densest Subgraph and 1 for Max-Min-Degree-Subgraph.

Finally, for the two problems from the degree-constrained category, we can adapt the definition of f to distinguish two cases for graphs that do not fulfill the property. For example, if a subgraph H_C contains a vertex with degree at least $r + 1$, then H_C is not the subgraph of an r-regular graph. Similarly, if H_C contains a vertex with degree at least $b + 1$, then H_C is not the subgraph of a graph fulfilling the constraints of (a, b)-Degree-Constrained Subgraph. By setting the objective value for such graphs to be $-\infty$ and setting the vertex-addition bound to 1, we can model this observation rather easily.

r-Regular Subgraph:

$$f(H) := \begin{cases} 1 & H \text{ is } r\text{-regular}, \\ -\infty & H \text{ has a vertex } v \text{ with } |N(v)| > r, \\ 0 & \text{otherwise.} \end{cases}$$

(a, b)-Degree-Constrained Subgraph:

$$f(H) := \begin{cases} 1 & H \text{ has minimum degree at least } a \\ & \text{ and maximum degree at most } b, \\ -\infty & H \text{ has a vertex } v \text{ with } |N(v)| > b, \\ 0 & \text{otherwise.} \end{cases}$$

The results are shown in Figure 2. Both algorithms benefit from a substantial speed-up when using the *VAR*. Hence, *VAR* is enabled in all following variants. The effect of *VAR* for *Pivot* and *Simple* for the three categories is shown in the bottom part of Figure 2. *Pivot* is now faster than *Simple* in all categories for small and large k, now solving almost all instances of the sparse category. The comparison of the plain version of *Simple* and *Pivot* with the *VAR* is shown in the top part of Figure 2. For small k and the dense and the degree-constrained category, *VAR* gives a speedup factor of more than 10. For instances of the degree-constrained category and large k, all instances solved by the plain version of *Simple* are now solved within 0.1 seconds by *Pivot* with

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

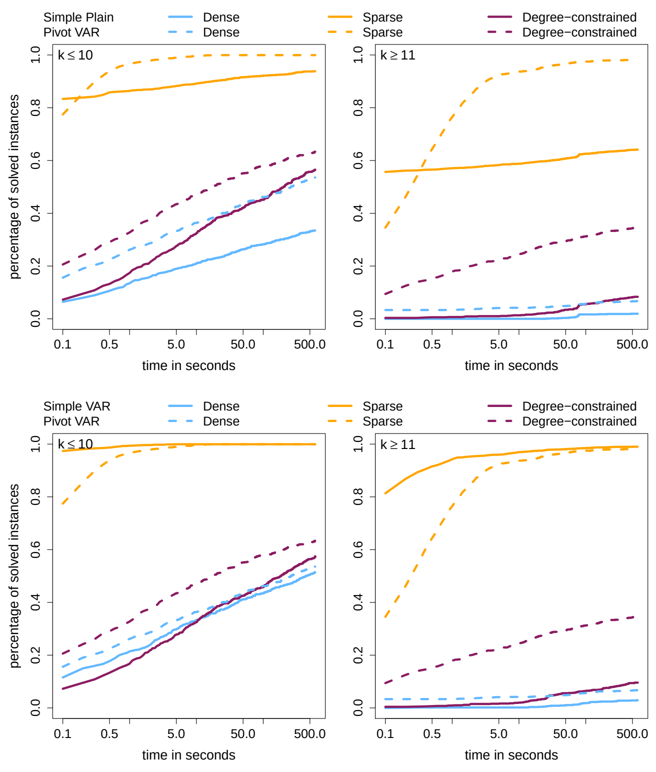

Figure 2: Top: Comparison of the plain version of *Simple* and *Pivot* with *VAR*. Bottom: Comparison of *Simple* and *Pivot* with *VAR*.

VAR. For instances of the dense category with large k, *VAR* has almost no effect as the algorithm enumerates too many dense subgraphs of order roughly $k/2$ which are not pruned by *VAR*.

Summarizing, with *VAR* enabled, *Pivot* solved 79 % of the instances for small k and 59 % of the instances for large k.

Monotonicity. We call an objective function f *edge-monotone* if adding an edge to a graph H does not decrease the objective value $f(H)$. For Densest Subgraph the function f is edge-monotone, since adding an edge increases the objective value by 1. Similarly, f is edge-monotone in Max-Min-Degree Subgraph. The other six objective functions are not edge-monotone. We use edge-monotonicity as follows to prune the search tree.

Pruning Rule 2. (*Clique Join Rule*) *Let C be the current connected set, let $\ell = k - |C|$, and let z denote the current best objective value. Let H_{C^*} be the graph obtained from H_C by adding an ℓ-vertex clique K and making it adjacent to all vertices of $P \subseteq C$. If $f(H_{C^*}) \leq z$, then return to the parent node in the search tree.*

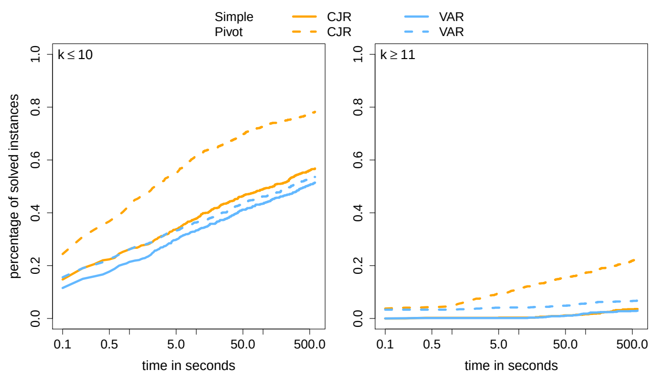

Figure 3: Comparison of *Pivot* and *Simple* with *VAR* and *CJR*, respectively.

The correctness of the *Clique Join Rule (CJR)* is obvious, since all connected induced subgraphs found in the subtree rooted at the current node are a subgraph of H_{C^*}. Consequently, they cannot achieve a better objective value than z.

The upper bound provided by the *VAR* is never better than the one provided by the *CJR*. However, the vertex-addition bound is faster to compute since we do not need to add a clique to the current subgraph H_C. Hence, we apply first the *VAR* and then the *CJR*.

Figure 3 shows the effect of the *CJR*. *Pivot* with the *CJR* is roughly 100 times faster than *Pivot* with only the *VAR*, the previously fastest for this category. For *Simple*, the speed-up is much smaller. We conclude that any further variants of the program should include some variant of the *CJR* when f is edge-monotone.

Universal Graphs. In contrast to the two rules above, the following pruning rule puts no restrictions on the objective function f. Such rules are highly desirable but it is intuitively clear that it is hard to prune the search tree when we have no knowledge about f.

Consider the connected set C of size $k - \ell$ at a node in the search tree and assume that z is the current best objective value. As in the other rules, if H_C can not be expanded to a connected subgraph H_{C^*} of order k such that $f(H_{C^*}) > z$, we can discard the connected set C and hence return to the parent of the current node. To test this condition without any knowledge of f, we simply try each possibility to expand H_C to a connected subgraph of order k. This can be done as follows.

Pruning Rule 3. (*Universal Graph Rule*) *For each (up to isomorphism) subgraph J on ℓ vertices, try each possibility of adding edges between $V(J)$ and P such that the resulting graph H_{C^*} is connected. If $f(H_{C^*}) \leq z$ for all resulting graphs H_{C^*}, then return to the parent*

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

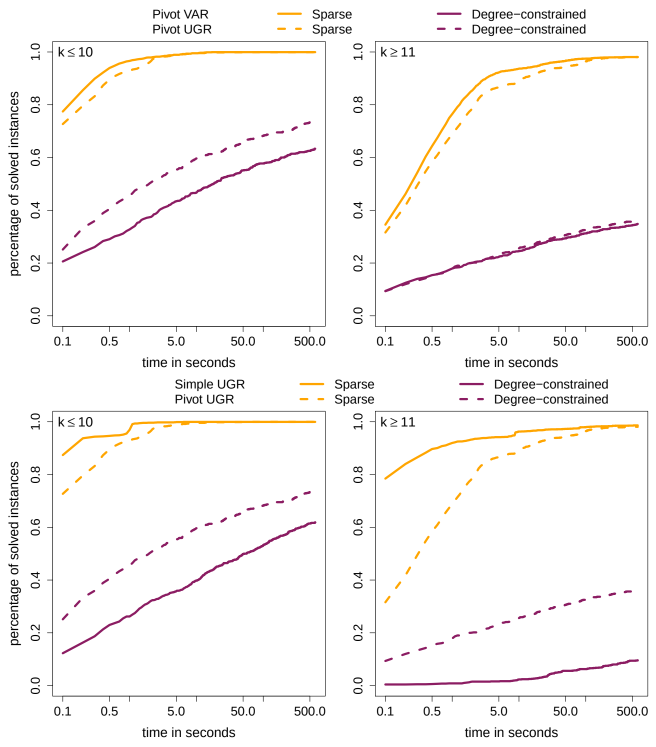

Figure 4: Top: Comparison of *Pivot* with the *UGR* and *Pivot* with *VAR*. Bottom: Comparison of *Simple* and *Pivot* with the *UGR*.

of the current node. Otherwise, abort the pruning rule after encountering the first H_{C^*} *with* $f(H_{C^*}) > z$.

The *Universal Graph Rule (UGR)* is obviously correct but it is not clear when this rule will lead to an improvement in running time as the number of graphs and edge additions to consider grows exponentially in ℓ and the number of vertices of P. To this end, we compute the number of graphs that we generate.

Let $I(n)$ be the number of graphs (up to isomorphism) of order n. In the *UGR* we have to evaluate $y = I(\ell) \cdot 2^{\ell \cdot |P|}$ graphs. Since y can be extremely large, we apply the universal graph rule only for $\ell \leq 3$ where $I(\ell) = 2^{\ell-1}$. In addition, we compare y with an estimate of the number of search tree nodes in the subtree rooted at the current node corresponding to C.

For this estimation, we consider the size of $Y := N(P) \setminus F$ to obtain an estimate on the number of vertices we will add to the current connected set C. More precisely, we apply the *UGR* if $2^{|P| \cdot \ell + \ell - 1} < |Y|^\ell$.

The effect of the *UGR* for *Pivot* is shown in the top part of Figure 4; the dense category is excluded from the experiments, since the *UGR* is never better than the *CJR*. For the problems of the sparse category this rule had a small negative effect. This is due the fact that *Pivot* with the *VAR* already solved almost all instances in that category. For problems of the degree-constrained category and large k, the rule had almost no effect, but for small k the rule gives a speed-up of factor 20 compared with *Pivot* with the *VAR*. Hence, we recommend to use the *UGR* for objective functions that are not edge-monotone. The running time comparison of *Simple* and *Pivot* is shown in the bottom part of Figure 4. *Simple* is slightly faster than *Pivot* in the sparse category but overall both variants solve the same number of instances in the sparse category. In the degree-constrained category, *Pivot* is much faster. For the future, it seems promising to fine-tune the *UGR*, for example by applying the rule if $2^{|P| \cdot \ell + \ell - 1} < c_u \cdot |Y|^\ell$ where c_u is a constant whose optimal value may be experimentally determined.

Summarizing, with the *CJR* for the dense category and the *UGR* for the other categories, *Pivot* solved 88 % of the instances for small k and 63 % of the instances for large k. *Pivot* benefits more from both rules than *Simple* because the set P is smaller for *Pivot*. Thus, it could be promising to consider further enumeration strategies which often lead to small P.

6 Neighborhood-Based Data Reduction and Branching

To further decrease the running times, we consider the relations between the neighborhoods of vertices in G.

Twin Sets. The first idea is to avoid enumerating too many graphs that are isomorphic by identifying vertices that have the same neighborhood. For this, we use the following definition. Two vertices u and v are called *true twins* if $N[u] = N[v]$ and *false twins* if $N(u) = N(v)$. If one of both cases occurs, u and v are called *twins*. If there are two vertices u and v with $N[u] = N[v]$, then there exists no vertex w with $N(u) = N(w)$, and vice versa. As a consequence, being twins is an equivalence relation. A maximal set of twins is called a *twin set*.

Before starting the enumeration algorithm, we compute all twin sets and apply the following two reduction rules.

Reduction Rule 1. *Let* $\mathcal{C}$ *be a set of true twins of* G *with* $|\mathcal{C}| > k$. *Remove* $|\mathcal{C}| - k$ *vertices of* $|\mathcal{C}|$ *from* G.

Reduction Rule 2. *Let* $\mathcal{I}$ *be a set of false twins of* G *with* $|\mathcal{I}| \geq k$. *Remove* $|\mathcal{I}| - k + 1$ *vertices of* $|\mathcal{I}|$ *from* G.

The correctness of the rules follows from the fact that each solution containing exactly k vertices can contain at most k vertices of a true twin set and at most $k - 1$ vertices of a false twin set.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

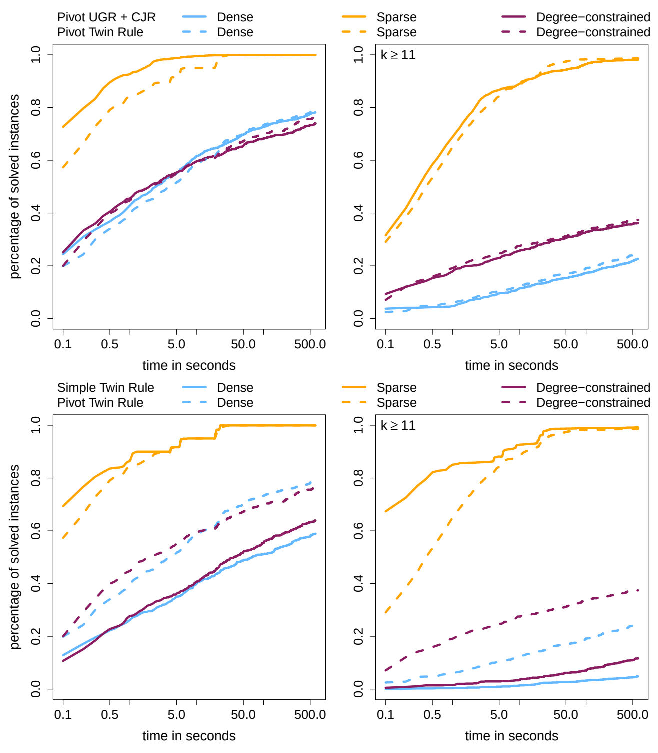

Figure 5: Top: Comparison of *Pivot* with the *Twin Rule* and *Pivot* with *UGR* and *CJR*. Bottom: Comparison of *Simple* and *Pivot* with the *Twin Rule*.

During the enumeration, we use twin sets as follows.

PRUNING RULE 4. (*Twin Rule*) *Let C be the connected set at the current node, let F denote the current set of forbidden vertices and let v be a vertex such that the algorithm considered each solution extending $C \cup \{v\}$. Let $\mathcal{C}$ denote the twin set of v. Add all vertices of $\mathcal{C} \setminus C$ to F.*

The correctness can be seen as follows: Let z be the current best objective value. Assume there exists another connected set C' with $C \cup \{u\} \subseteq C'$ and $u \in \mathcal{C} \setminus C$ such that $f(H_{C'}) > z$. Consider the set $C'' := (C' \setminus \{u\}) \cup \{v\}$. Since u and v are twins, $H_{C'} \cong H_{C''}$. Since $v \in C''$, the algorithm already considered the connected set C''. Hence, $f(H_{C''}) \leq z$, a contradiction to the assumption that $f(H_{C'}) > z$.

The effect of the *Twin Rule* for *Pivot* is shown in the top part of Figure 5. For running times below ten seconds, *Pivot* with *Twin Rule* is slower than *Pivot* with the *CJR* and *UGR*. This is due to the time which is needed to calculate the twin sets. In the sparse category, the *Twin Rule* gave no improvement since almost all instances can be solved within the time limit. For the

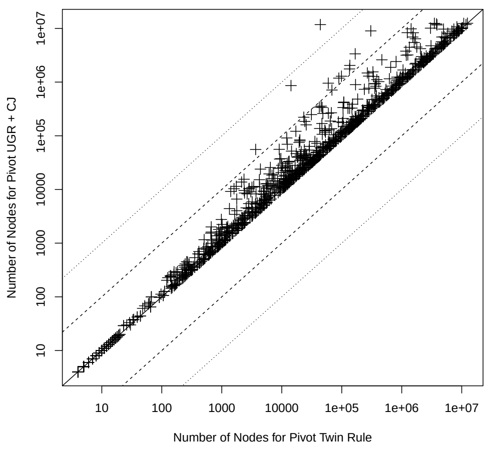

Figure 6: Comparison of the number of search tree nodes of *Pivot* with the *Twin Rule* and *Pivot* with *UGR* and *CJR*.

dense and degree-constrained category, the *Twin Rule* gives a running time improvement of roughly a factor two for the harder instances. Moreover, as shown in Figure 6, the *Twin Rule* may decrease the size of the search tree tremendously in some cases. Since the running time overhead incurred by the rule is not too high, we enable the twin rule for all three categories.

The bottom part of Figure 5 compares *Simple* and *Pivot* with the *Twin Rule*. In the sparse category, *Simple* is competitive. In the dense and degree-constrained category, for small k, *Pivot* is more than ten times faster than *Simple* and for large k, *Pivot* solves each instance that was solved by *Simple* within the time limit in less than one second.

Neighborhood Relation for Edge-Monotonicity In the following, we provide an improvement of the Twin Rule for edge-monotone objective functions f.

PRUNING RULE 5. (*Neighborhood Inclusion Rule*) *Let C be the connected set at the current node and let F denote the current set of forbidden vertices and let v be a vertex such that the algorithm considered each solution extending $C \cup \{v\}$. Add all vertices $u \notin C$ with $N(u) \subseteq N[v]$ to F.*

The correctness of the *Neighborhood Inclusion Rule* (*NIR*) can be seen as follows: Let z be the current best objective value. Assume there exists a connected set C' with $C \cup \{u\} \subseteq C'$, $u \notin C$ and $N(u) \subseteq N[v]$ such that $f(H_{C'}) > z$. Since $N(u) \subseteq N[v]$, $H_{C'}$ is a subgraph of $H_{C''}$ where $C'' := (C' \setminus \{u\}) \cup \{v\}$. Since f is edge-monotone, $f(H_{C'}) \leq f(H_{C''})$. Hence, $f(H_{C'}) \leq z$, a contradiction to the fact that $f(H_{C'}) > z$.

Figure 7 shows the effect of the *NIR*. Similar to the

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

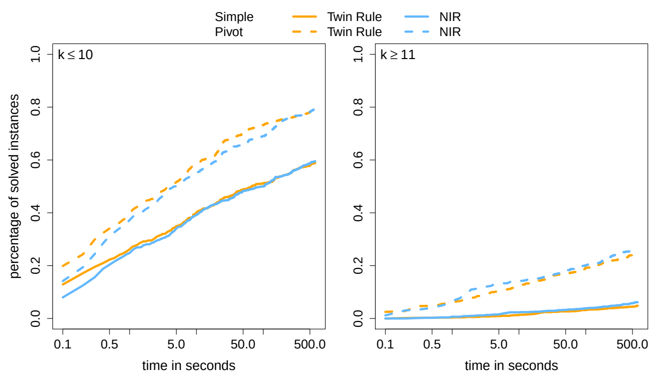

Figure 7: Comparison of *Simple* and *Pivot* with and without the *NIR*.

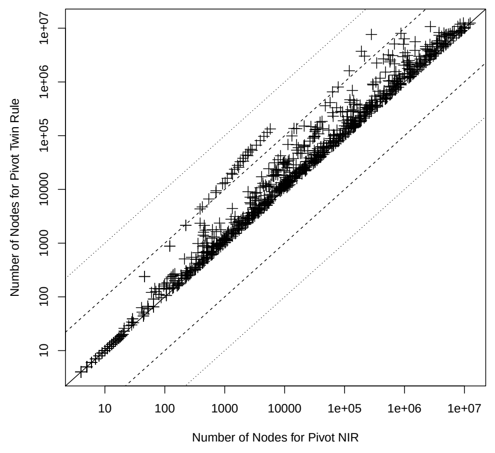

Figure 8: Comparison of number of search tree nodes of *Pivot* with the *Twin Rule* and *Pivot* with the *NIR*.

Twin Rule, for running times below 10 seconds, *Pivot* with the *NIR* is slower than *Pivot* without it, which is again due to the time needed to compute the relations. For small k, the *NIR* does not improve *Pivot*. For large k, *Pivot* with *NIR* is slightly faster than *Pivot* with *Twin Rule*. Moreover, as shown in Figure 8, there are again some cases in which the number of search tree nodes is decreased tremendously. Hence, we enable this rule in the following. For *Simple* and small k the speedup is much better, but *Pivot* remains roughly 20 times faster than *Simple*.

Summarizing, with the *NIR* for the dense category and the *Twin Rule* for the other categories, *Pivot* solved 88 % of the instances for small k and 65 % of the instances for large k.

7 Heuristic Lower Bounds

We also implemented three randomized heuristics to compute good initial solutions. These provide a lower bound for the objective function f which will be useful to prune the search tree. The heuristics work as follows: We have a probability measure P_0 on $V(G)$. Choose a start vertex v_1 with probability $P_0(v_1)$. For a connected set C of size $\ell < k$ we have a probability measure P_ℓ on $N(C)$, and choose a vertex v_ℓ with probability $P_\ell(v_\ell)$.

In the first heuristic, we let P_i be the uniform distribution. This heuristic is aimed to be good for objective functions where we have little knowledge.

In the second heuristic, we set $P_0(v) := |N(v)|/(2|E|)$ for each $v \in V(G)$. Furthermore, for each $1 \leq i < k$ and for each $v \in N(C)$, where C is the current connected set, we set: $P_i(v) := |N(v) \cap C|/R$, where $R := \sum_{v \in N(C)} N(v) \cap C$. This heuristic is aimed to be suitable for edge-monotone objective functions.

In the third heuristic, we set $P_0(v) := (\Delta - |N(v)|)/T$, where $T := \sum_{v \in V(G)}(\Delta - |N(v)|)$ for each $v \in V(G)$. Furthermore, for each $1 \leq i < k$ and for each $v \in N(C)$, we set: $P_i(v) := (\Delta - |N(v)|)/M$, where $M := \sum_{v \in N(C)}(\Delta - |N(v)|)$. This heuristic is aimed to be suitable for problems in the sparse category.

Since instances with larger k and larger n are harder, we apply the heuristic more often, the larger n and k are. More precisely, each heuristic is applied $\log(n) \cdot k$ times. We have chosen a linear dependence on k since the subgraph size in our experiments is at most 20 and a logarithmic dependence on n since we consider networks with up to 500 000 vertices. We refer to each application as a *trial*. Let z denote the objective value of the current best solution.

While iteratively building a random subgraph in some trial, we check whether the current subgraph can still lead to a solution with objective value better than z by applying the *VAR*. If this is not the case, then we discard the current subgraph and start with the next trial.

Since the variance of the best solution found by these heuristics is huge, after each trial of each heuristic we perform local search to improve the solution quality of the found connected set C: In the local search, we consider all possibilities of obtaining a better connected set C^* by swapping a vertex $v \in C$ with a vertex $u \in V \setminus C$ as follows. For each $v \in C$, we compute the connected components $C_1, \ldots, C_\ell$ of $H_C - v$. Afterwards, we compute their common neighborhood $\mathcal{N} := (N(C_1) \cap \ldots \cap N(C_\ell)) \setminus C$ in $G - C$. We then compute the vertex $u \in \mathcal{N}$ for which the connected set $C^* := C \cup \{u\} \setminus \{v\}$ obtained by removing v and adding u has a maximal value $f(H_{C^*})$. If $f(H_{C^*}) > z$ set $C \leftarrow C^*$, $z \leftarrow f(H_{C^*})$ and continue, otherwise we abort the trial.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

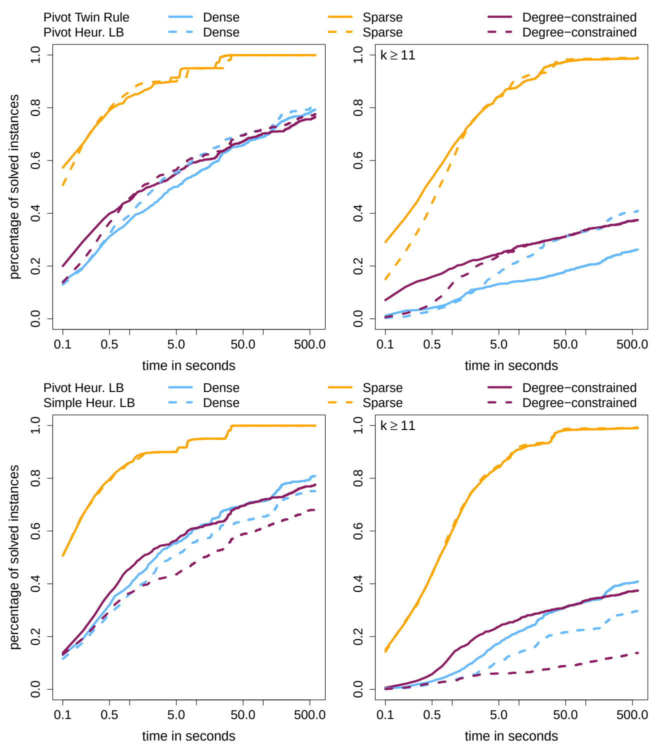

Figure 9: Top: Comparison of *Pivot* with and without heuristic lower bounds. Bottom: Comparison of *Simple* and *Pivot* with heuristic lower bounds.

The effect of the computation of the heuristic lower bounds for *Pivot* is shown in the top part of Figure 9. For the small instances, we can observe a small negative effect that is caused by the additional time needed to compute the lower bounds. For the large instances, we either observe no effect or a small positive effect with one exception: for the dense category and large k, the rule gives a speed-up factor of almost 100 and almost doubles the number of instances that can be solved within the time limit. The comparison of *Pivot* and *Simple* with heuristic lower bounds is shown in the bottom of Figure 9. *Simple* is only competitive with *Pivot* in the sparse category.

With the greedy heuristics and all previous rules, *Pivot* solved 89 % of the instances for small k and 69 % of the instances for large k, outperforming *Simple* also in this variant.

8 Problem-Specific Pruning Rules

To allow for further improvement of the algorithms, we extend *FixCon* to include problem-specific pruning rules that, in addition, use not only H_C as input but also the graph G. We design such rules for the two harder categories: the dense category and the degree-constrained category.

First, we describe the pruning rules applied for Densest Subgraph. The first rule is an adaptation of a known upper bound on the number of edges of any order-k graph containing the connected set C [11, 14]. We adapt this bound to our setting as follows. Let C be the current connected set, let S be any order-k solution extending C, and let $\ell := k - |C|$ denote the number of vertices to add. We partition the edges of $G[S]$ into three subsets: the edges between vertices of C, whose number is denoted by $m(C)$ and already known, the edges between vertices of C and $S \setminus C$, whose number will be denoted by $m(C, S \setminus C)$, and the edges between vertices of $S \setminus C$, whose number will be denoted by $m(S \setminus C)$. We compute upper bounds on $m(C, S \setminus C)$ and $m(S \setminus C)$.

For bounding $m(C, S \setminus C)$, we exploit that no neighbors of vertices in $C \setminus P$ may be added. We thus first compute the set $Y := V \setminus (C \cup N(C \setminus P) \cup F)$ of vertices that may still be added to C. For each vertex $y \in Y$, we compute $\deg_P(y) := |N(y) \cap P|$ and $\deg_{V \setminus C}(y) := |N(y) \cap V \setminus C|$. Now $m(C, S \setminus C) + m(S \setminus C) \leq b_0(C, P)$ where

$$b_0(C, P) := \sum_{i=1}^{\ell} (\deg_P(y_i) + \min(\deg_{V \setminus C}(y_i), \ell - 1)/2)$$

where $\{y_1, \ldots, y_\ell\} \subseteq Y$ is the set of vertices that maximizes this sum. The second summand is divided by 2 since these edges are double counted in the sum. Moreover, if $b_0(C, P)$ is not integral, we may round down. Thus, we obtain the first improved pruning rule.

Pruning Rule 6. *Let C be the current connected set and let z denote the current-best objective value. If $m(C) + \lfloor b_0(C, P) \rfloor \leq z$, then return to the parent node.*

We now further refine this bound by partitioning the set of edges between vertices in $S \setminus C$ even further. To this end, let $S_1 := (S \cap N(P)) \setminus C)$ denote the vertices of $S \setminus C$ that are neighbors of P, and $S_2 := S \setminus (C \cup S_1)$ denote the remaining vertices. We partition the edges in $S \setminus C$ into those edges inside S_1, those edges between S_1 and S_2 and those edges inside S_2. Moreover, we consider all possible sizes ℓ' of S_2; due to the connectivity constraint we have $|S_1| \geq 1$ and thus $0 \leq \ell' < \ell$. For each ℓ', we compute an upper bound of $m(S, \ell')$, defined as the maximum number of edges we may achieve by having exactly ℓ' vertices in S_2. The upper bound for $m(S)$ is then the maximum of $m(S, \ell')$ over all ℓ'. Thus, in the following, we describe how to compute bounds for $m(S, \ell')$. For fixed ℓ', we denote the numbers of the edge sets in the partition by $m(S_1, \ell')$, $m(S_1, S_2, \ell')$ and $m(S_2, \ell')$, respectively.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

The number $m(S_2, \ell')$ is at most $\binom{\ell'}{2}$ since in the best case, the vertices of S_2 form a clique. To bound $m(S_1, \ell')$ we compute $\deg_{N(P)}(v) := |N(v) \cap N(P)|$ for all vertices in $Y \cap N(P)$. To bound $m(S_1, S_2, \ell')$, we compute $\deg_{V \setminus N[C]}(v) := |N(v) \setminus N[C]|$ for vertices in $Y \cap N(P)$. We now obtain the bound $m(C, S \setminus C, \ell') + m(S_1, \ell') + m(S_1, S_2, \ell') + m(S_2, \ell') \leq b_1(C, P, \ell')$ where

$$b_1(C, P, \ell') := \sum_{i=1}^{\ell-\ell'} \Big(\deg_P(v_i) + \min(\deg_{N(P)}(v_i), \ell - \ell' - 1)/2 + \min(\deg_{V \setminus N[C]}(v_i), \ell') \Big)$$

where $v_1, \ldots, v_{\ell-\ell'}$ are the vertices of $Y \cap N(P)$ maximizing the sum.

Pruning Rule 7. *Let C be the current connected set and let z denote the current-best objective value. If $m(C) + \max_{0 \leq \ell' < \ell}(\lfloor b_1(C, P, \ell') \rfloor + \binom{\ell'}{2}) \leq z$, then return to the parent node.*

The top part of Figure 10 shows the effect of both pruning rules. The previously known upper bound gives a considerable improvement for small k and large k. Using in addition the new upper bound gives a rather small improvement for large k and has a negligible effect for small k.

For Max-Min-Degree-Subgraph, the pruning rules are simpler.

Pruning Rule 8. *Let C be the current connected set and let z denote the current-best objective value. If*

1. *C contains a vertex with degree at most z in G, or*
2. *C contains a vertex v with at most $z - \deg_{H_C}(v)$ neighbors in Y, or*
3. *$z > \ell - 1$ and $N(P) \cap Y$ contains less than ℓ vertices v with $|N(v) \cap P| + \min(\ell - 1, |N(v) \setminus C|) > z$, or*
4. *$N(P) \cap Y$ contains no vertex with $|N(v) \cap P| + \min(\ell - 1, |N(v) \setminus C|) > z$*

then return to the parent of the current search tree node.

The first two cases of the pruning rule are obviously correct. For the third case, observe that if $z > \ell - 1$, only vertices with at least one neighbor in P may be added. Moreover, a vertex may only be added if his number of neighbors in P plus the number of neighbors in the remaining part of S exceeds z. If there are less than ℓ candidates to add, then there is no solution extending C. For the last case, observe that we need to add at least one vertex of $N(P) \cap Y$. If there is no suitable candidate, then we can discard the current connected set.

Next, we describe pruning rules for r-Regular Subgraph and (a, b)-Degree-Constrained Subgraph. We will only describe the rules for the more general (a, b)-Degree-Constrained Subgraph since we use the same rules for r-Regular Subgraph by setting $a = r$ and $b = r$.

In the following, for each vertex of P, we let $\text{demand}(v) := a - \deg_C(v)$ denote the *demand* of v, that is, the number of edges with one endpoint being v that we need to add in order to fulfill the degree constraint of v. Here, $\deg_C(v)$ is the degree of the vertex in H_C. Recall that $Q = C \setminus P$ denotes the set of vertices for which we may not add further neighbors.

Pruning Rule 9. *Let C be the current connected set. Return to the parent node if*

1. *Q contains a vertex of degree less than a, or*
2. *C contains a vertex v with $\text{demand}(v) > k - |C|$, or*
3. *C contains a vertex v that has degree less than a in G.*

This rule is correct since in each case the vertex v cannot have degree at least a in H_S for a subgraph S with minimal degree a.

The following rule is correct, since in the cases described by the rule, every addition of a vertex to C creates at least one vertex of degree at least $b + 1$.

Pruning Rule 10. *Let C be the current connected set. If $|C| < k$ and if all vertices of P have degree b in H_C, then return to the parent node.*

In the next case, the idea is to count how many edges need to be added in order to increase the degree of every vertex of H_C above the threshold a. This number is compared with an upper bound on the number of edges that we may obtain by adding $k - |C|$ new vertices whose degree may not exceed b.

Pruning Rule 11. *Let C be the current connected set. Let $V_{<a}$ denote the set of vertices in H_C that have degree less than a. If $\sum_{v \in V_{<a}} \text{demand}(v) > (k - |C|) \cdot b$, then return to the parent node.*

We now take a closer look at the vertices of $V(G) \setminus C$ that we may add to C. More precisely, we exploit the following observation: we may not add any vertex u that has degree less than a in G or any vertex u that has a neighbor w in C which has degree b in H_C. Thus, we may define for each vertex $v \in C$, a set of *possible neighbors* $N^*_G(v, C) := \{u \in V(G) \setminus C \mid \deg_G(u) \geq a$ and there is no $w \in C \cap N(u)$ with $\deg_C(w) = b\}$.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

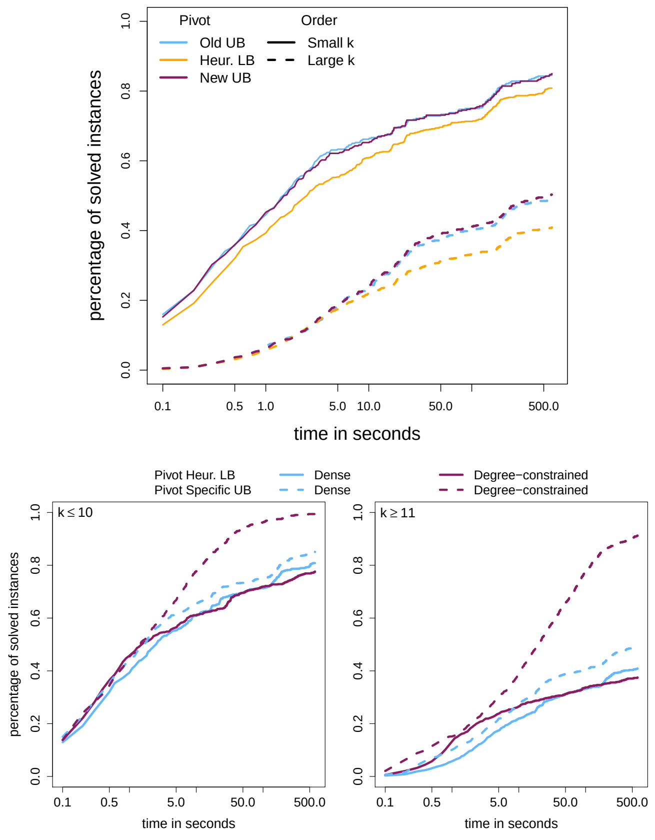

Figure 10: Comparison of *Pivot* with and without problem-specific pruning rules. Top: The two rules for Densest Subgraph. Bottom: Comparison for the dense and degree-constrained category.

Pruning Rule 12. *Let C be the current connected set. If C contains a vertex v such that* $\text{demand}(v) > |N^*_G(v,C)|$, *then return to the parent node.*

We now look at pairs of vertices u and v in C for which we still need to add neighbors. We exploit the following observation: the fewer the number of common neighbors of u and v in $V(G)\setminus C$, the larger is the number of vertices that we must add. If this number is too large, then C cannot be extended to a solution.

Pruning Rule 13. *Let C be the current connected set.*

1) If C contains two vertices u and v such that $\text{demand}(v) + \text{demand}(u) - \min(b - \deg_C(v), b - \deg_C(u), |N^*_G(v,C) \cap N^*_G(u,C)|) > k - |C|$, *or*

2) if C contains three vertices u, v, and w such that $N^*_G(w,C)$ is disjoint from $N^*_G(u,C)$ and from $N^*_G(v,C)$ and $\text{demand}(v) + \text{demand}(u) + \text{demand}(w) - \min(b - \deg_C(v), b - \deg_C(u), |N^*_G(v,C) \cap N^*_G(u,C)|) > k - |C|$, *then return to the parent node.*

The correctness of the first part of the rule can be seen as follows: In order to fulfill the degree constants for v and u, we need to add $\text{demand}(v) + \text{demand}(u)$ edges with an endpoint in u or v. Vertices that are common neighbors of u and v add two such edges, all other vertices add only one such edge. Hence, we may write the number of vertices to add as the number of neighbors of u plus the number of neighbors of v minus the number of common neighbors of u and v. Since the number of common neighbors of u and v that we may add is at most $\min(b - \deg_C(v), b - \deg_C(u), |N^*_G(v,C)| \cap N^*_G(u,C)|)$, the left hand side thus gives a lower bound on the number of vertices that we need to add to increase the degree of u and v sufficiently. If this lower bound exceeds $k - |C|$, the number of vertices that we may still add, then there is no connected set C' that extends C.

For the second part, we simply consider a third vertex w if the set of possible neighbors of w is disjoint from those of v and u. The arguments for the correctness are completely analogous. Since the second part of the rule is costly in terms of running time (we have to consider all triples of vertices in C), we apply this part only if $\text{demand}(v) + \text{demand}(w) - \min(b - \deg_C(v), b - \deg_C(u), |N^*_G(v,C) \cap N^*_G(u,C)|) > (k - |C|)/2$. The rationale behind this condition is that it is unlikely that the second part of the rule will be successful if the condition is not met.

The final rule extends Rule 13 to larger subsets of vertices in C. Instead of trying all possibilities of such sets, we greedily compute a set of vertices in C for which we still need to add neighbors and which have disjoint possible neighborhoods.

Pruning Rule 14. *Let C be the current connected set. If C has a subset A of vertices such that for all $u, v \in A$ we have $N^*_G(v,C) \cap N^*_G(u,C) = \emptyset$ and* $\sum_{v \in A} \text{demand}(v) \geq k - |C|$, *then return to the parent node.*

As stated above, the set A is computed greedily. More precisely, we consider the vertices of C in some order and add the first vertex v in $C \setminus A$ with $\text{demand}(v) > 0$ whose possible neighborhood is disjoint from all possible neighborhoods of A.

The overall effect of these pruning rules is shown in the bottom of Figure 10. There is a substantial speed-up for the dense category, particularly for large k, and a tremendous speed-up for the degree-constrained category for small and large k.

9 A Comparison with ILP formulations

To compare *FixCon* with some competitor, we developed ILP formulations for all eight problems. As ILP solver, we used Gurobi version 8.01 with the Python interface.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

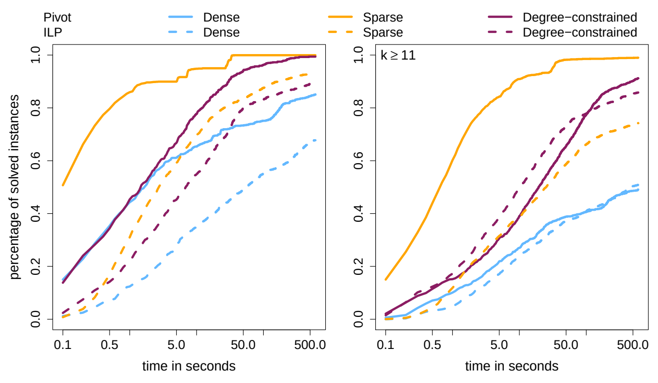

Figure 11: Comparison of the ILP with the version of *Pivot* containing all improvements.

In all eight formulations we have binary variables x_v for each vertex $v \in V$ and the constraint $\sum_{v \in V} x_v = k$ which ensures that k vertices are selected. As proposed by Althaus et al. [1] we use lazy constraints to ensure the connectivity of the solution: If at some node of the search tree, we have a disconnected solution S, then we add the following constraint for each connected component C of $G[S]$ in a callback [1]

$$\sum_{v \in C} x_v - \sum_{v \in N(C)} x_v \leq |C| - 1.$$

Figure 11 compares the ILP formulations with the best variants of *Pivot*.

Overall, *Pivot* solves 96% of the instances with small k and 84% of the instances with large k, whereas the ILP solves 85% of the small and 70% of the large instances.

For small k, *Pivot* is significantly faster than the ILP in all three categories. In the degree-constrained category for small k and in the sparse category for all k, *Pivot* solves almost all instances within the time limit. The biggest difference can be observed for the sparse category. This can be somewhat expected since *FixCon* maintains the connectivity constraint during the search, whereas for the ILP, the connectivity constraint is somewhat contrary to the fact that sparse graphs are preferred by the objective functions.

For the dense category and small k, *Pivot* is roughly 20 times faster than the ILP. For the dense category and the degree-constrained category and large k, the ILP and *Pivot* are competitive. A further comparison is shown in Table 1 (in the appendix); the table shows for each real-world network the largest k such that all eight CFCO problems could be solved within ten minutes.

Overall, we conclude that *FixCon* with *Pivot* is competitive with off-the-shelf ILP formulations of the considered CFCO problems when $k \leq 20$ and problem-specific pruning rules are employed.

10 Conclusion

We have demonstrated the usefulness of subgraph enumeration for the generic Connected Fixed-Cardinality Optimization problem. For using the generic version of *FixCon* without problem-specific pruning rules a user only needs to implement the objective function f and provide some properties of f. This version can be used as a baseline for comparison with special purpose algorithms or as a base implementation that can be improved by adding problem-specific pruning rules. In the latter setting, *FixCon* is competitive with standard ILP formulations for the problems under consideration.

There are many avenues to pursue in future research. First, we aim to further improve the generic part of *FixCon*, for example by extending the *Twin Rule* which exploits symmetry in the neighborhood to cases where vertices have almost the same neighborhood. Second, we aim to collect further example problems and to further improve the problem-specific rules for the problems considered in this work. Third, we aim to extend the algorithms to allow richer graph models; for example, by allowing vertex and edge weights, vertex and edge colors, or directed edges. One could also aim to address problems where the objective function depends not only on the subgraph itself but also on the rest of the graph. This could be used to find, for example, connected dominating sets or connected vertex covers of the graph or to solve core-periphery subgraph problems. Finally, one could extend *FixCon* to enumeration problems, where one wants to output all optimal solutions or to counting problems, where one wants to output their number.

It is also open to translate other generic subgraph problem representations into *FixCon* objective functions. For example, one could aim to automatically translate first-order-logic formulas into *FixCon* objective functions and, in particular, extract useful properties such as edge-monotonicity directly from the formulas.

Another interesting line of research is to examine how parallelization can be used to speed up *FixCon*, as it was done for example for clique enumeration [5].

[1]We also tried using one constraint for each $c \in C$ as Althaus et al. [1]; in preliminary experiments this gave slightly worse results.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

References

[1] E. Althaus, M. Blumenstock, A. Disterhoft, A. Hildebrandt, and M. Krupp, *Algorithms for the maximum weight connected k-induced subgraph problem*, in Proceedings of the 8th International Conference on Combinatorial Optimization and Applications (COCOA '14), vol. 8881 of Lecture Notes in Computer Science, Springer, 2014, pp. 268–282.

[2] D. A. Bader, A. Kappes, H. Meyerhenke, P. Sanders, C. Schulz, and D. Wagner, *Benchmarking for graph clustering and partitioning*, in Encyclopedia of Social Network Analysis and Mining, Springer, 2014, pp. 73–82.

[3] M. Bruglieri, M. Ehrgott, H. W. Hamacher, and F. Maffioli, *An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints*, Discrete Applied Mathematics, 154 (2006), pp. 1344–1357.

[4] L. Cai, S. M. Chan, and S. O. Chan, *Random separation: A new method for solving fixed-cardinality optimization problems*, in Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC '06), vol. 4169 of Lecture Notes in Computer Science, Springer, 2006, pp. 239–250.

[5] E. Coppa, I. Finocchi, and R. L. Garcia, *Counting cliques in parallel without a cluster: Engineering a fork/join algorithm for shared-memory platforms*, Information Sciences, 496 (2019), pp. 553–571.

[6] M. Fischetti, H. W. Hamacher, K. Jørnsten, and F. Maffioli, *Weighted* k*-cardinality trees: Complexity and polyhedral structure*, Networks, 24 (1994), pp. 11–21.

[7] Z. R. M. Kashani, H. Ahrabian, E. Elahi, A. Nowzari-Dalini, E. S. Ansari, S. Asadi, S. Mohammadi, F. Schreiber, and A. Masoudi-Nejad, *Kavosh: a new algorithm for finding network motifs*, BMC Bioinformatics, 10 (2009), p. 318.

[8] C. Komusiewicz and F. Sommer, *Enumerating connected induced subgraphs: Improved delay and experimental comparison*, in Proceedings of the 45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '19), vol. 11376 of Lecture Notes in Computer Science, Springer, 2019, pp. 272–284.

[9] C. Komusiewicz and M. Sorge, *Finding dense subgraphs of sparse graphs*, in Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC '12), vol. 7535 of Lecture Notes in Computer Science, Springer, 2012, pp. 242–251.

[10] C. Komusiewicz and M. Sorge, *An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems*, Discrete Applied Mathematics, 193 (2015), pp. 145–161.

[11] C. Komusiewicz, M. Sorge, and K. Stahl, *Finding connected subgraphs of fixed minimum density: Implementation and experiments*, in Proceedings of the 14th International Symposium on Experimental Algorithms (SEA '15), vol. 9125 of Lecture Notes in Computer Science, Springer, 2015, pp. 82–93.

[12] J. Kunegis, *KONECT: the Koblenz network collection*, in Proceedings of the 22nd International World Wide Web Conference (WWW '13), International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 1343–1350.

[13] S. Maxwell, M. R. Chance, and M. Koyutürk, *Efficiently enumerating all connected induced subgraphs of a large molecular network*, in Proceedings of the First International Conference on Algorithms for Computational Biology (AlCoB '14), vol. 8542 of Lecture Notes in Computer Science, Springer, 2014, pp. 171–182.

[14] F. M. Pajouh, Z. Miao, and B. Balasundaram, *A branch-and-bound approach for maximum quasi-cliques*, Annals OR, 216 (2014), pp. 145–161.

[15] R. A. Rossi and N. K. Ahmed, *The network data repository with interactive graph analytics and visualization*, in Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI '15), AAAI Press, 2015, pp. 4292–4293, `http://networkrepository.com`.

[16] S. Wernicke, *A faster algorithm for detecting network motifs*, in Proceedings of the 5th International Workshop on Algorithms in Bioinformatics (WABI '05), vol. 3692 of Lecture Notes in Computer Science, Springer, 2005, pp. 165–177.

[17] S. Wernicke and F. Rasche, *FANMOD: a tool for fast network motif detection*, Bioinformatics, 22 (2006), pp. 1152–1153.

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited

A Further Experimental Results

Table 1: The table shows for each instance the largest value k^* such that all eight problems could be solved for all $k \leq k^*$. *Pivot* and *Simple* are the respective algorithms with all improvements up to the heuristic lower bound. *Pivot+* has additionally the problem-specific lower bounds and described in Section 8. A value of N/A for the ILP means that for all k, there is at least one problem which the ILP did not solve for this graph.

Size	Name	$\|V\|$	$\|E\|$	*Pivot*	*Simple*	*Pivot+*	*ILP*
Small	moreno-zebra	27	111	**20**	**20**	**20**	**20**
	ucidata-zachary	34	78	15	14	**15**	9
	contiguous-usa	49	107	12	12	18	**20**
	dolphins	62	159	9	9	**18**	15
	ca-sandi-auths	86	124	11	12	**18**	15
	adjnoun_adjacency	112	425	6	6	12	**17**
	arenas-jazz	198	2 742	6	6	**20**	16
	inf-USAir97	332	2 126	6	5	**17**	10
	ca-netscience	379	914	8	8	**19**	13
	bio-celegans	453	2 025	6	5	**13**	11
Medium	bio-diseasome	516	1 188	7	6	**20**	12
	soc-wiki-Vote	889	2 914	6	5	12	**15**
	arenas-email	1 133	5 451	6	5	13	**16**
	inf-euroroad	1 174	1 417	10	10	13	**18**
	bio-yeast	1 458	1 948	7	7	12	**14**
	ca-CSphd	1 882	1 740	9	10	**13**	10
	soc-hamsterster	2 426	16 630	6	5	**20**	12
	inf-openflights	2 939	15 677	6	5	**16**	8
	ca-GrQc	4 158	13 422	6	5	**20**	12
	inf-power	4 941	6 594	8	8	17	**18**
Large	soc-advogato	6 541	39 432	6	4	**11**	6
	bio-dmela	7 393	25 569	6	4	**8**	5
	ca-HepPh	11 204	117 619	6	5	**20**	13
	ca-AstroPh	17 903	196 972	6	5	**17**	N/A
	soc-brightkite	56 739	212 945	5	4	**15**	N/A
	coAuthorsCiteseer	227 320	814 134	5	5	**20**	N/A
	coAuthorsDBLP	299 067	977 676	5	5	**16**	N/A
	soc-twitter-follows	404 719	713 319	4	4	**5**	N/A
	coPapersCiteseer	434 102	16 036 720	6	5	**20**	N/A
	coPapersDBLP	540 486	15 245 729	5	5	**13**	N/A
Dense	bn-cat-mixed-species_brain_1	65	730	7	6	**14**	11
	robot24c1_mat5	404	14 261	12	6	11	**13**
	econ-beause	507	39 428	6	5	20	6
	bn-mouse_retina_1	1 076	577,350	6	5	**16**	12
	comsol	1 500	48 119	6	5	**20**	N/A
	bn-fly-drosophila_medulla_1	1 781	8 911	6	4	**10**	5
	heart2	2 339	340 229	8	6	**20**	9
	econ-orani678	2 529	86 768	6	6	**11**	N/A
	psmigr_1	3 140	410 781	6	5	**17**	16
	bn-human-BNU_1_0025890_session_1	177 584	15 669 037	5	5	**9**	N/A

Copyright © 2020 by SIAM
Unauthorized reproduction of this article is prohibited