Recent Advances in Practical Data Reduction

Faisal N. Abu-Khzam[1], Sebastian Lamm[2], Matthias Mnich[3], Alexander Noe[4], Christian Schulz[5](✉), and Darren Strash[6]

[1] Lebanese American University, Beirut, Lebanon
faisal.abukhzam@lau.edu.lb
[2] Karlsruhe Institute of Technologie, Karlsruhe, Germany
sebastian.lamm@kit.edu
[3] Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany
matthias.mnich@tuhh.de
[4] University of Vienna, Vienna, Austria
alexander.noe@univie.ac.at
[5] Heidelberg University, Heidelberg, Germany
christian.schulz@informatik.uni-heidelberg.de
[6] Hamilton College, New York, USA
dstrash@hamilton.edu

Abstract. Over the last two decades, significant advances have been made in the design and analysis of fixed-parameter algorithms for a wide variety of graph-theoretic problems. This has resulted in an algorithmic toolbox that is by now well-established. However, these theoretical algorithmic ideas have received very little attention from the practical perspective. We survey recent trends in data reduction engineering results for selected problems. Moreover, we describe concrete techniques that may be useful for future implementations in the area and give open problems and research questions.

Keywords: Data reduction · Kernelization · Fixed-parameter algorithms · Algorithm engineering

1 Introduction

Many important real-world optimization problems are NP-hard: it is believed that no polynomial time algorithm exists that always finds an optimal solution. However, many NP-hard problems have been shown to be fixed-parameter tractable (FPT): large inputs can be solved efficiently and provably optimally, as long as some problem parameter is small. Over the last two decades, significant advances have been made in the design and analysis of fixed-parameter algorithms for a wide variety of graph-theoretic problems. This has resulted in an algorithmic toolbox that is by now well-established. However, these theoretical algorithmic ideas have received very little attention from the practical perspective. Until recently, few fixed-parameter algorithms have been implemented and tested on real data sets, and their practical potential is far from understood. Traditionally, algorithms are designed using simple models of problems and machines. In turn, important results are provable, such as performance guarantees for all possible inputs. This often yields elegant solutions being adaptable to many applications with predictable performance for previously unknown inputs.

In contrast to algorithm theory, taking up and implementing an algorithm is part of application development. Unfortunately, transferring results from theory to practice is a slow process and sometimes the theoretically-best algorithms perform poorly in experiments. Hence, practitioners often do not read research papers from the theoretical algorithms community. This causes a growing gap between theory and practice: Realistic hardware with its parallelism, memory hierarchies, etc. is diverging from traditional machine models. This gap is also partially due to the fact that the research community working on algorithmic problems is fairly separated. On the one hand, there are "hard core" algorithms researchers that are focused mainly on theoretical work and rarely participate in conferences in application areas. On the other hand, researchers of application areas publish their work in conferences and journals of their respective fields, and often do not visit theory conferences. In contrast to algorithm theory, algorithm engineering uses an innovation cycle where algorithm design based on realistic models, theoretical analysis, efficient implementation, and careful experimental evaluation using real-world inputs closes gaps between theory and practice and leads to improved application code and reusable software libraries (see www.algorithm-engineering.de). This yields results that practitioners can rely on for their specific application.

On the one hand, experimental results can trigger new theoretical questions and suggest new properties of inputs that are relevant parameters to use in theoretical analysis. On the other hand, the rich toolbox of parameterized algorithm theory offers a rich set of algorithmic ideas that are challenging to implement and engineer in practical settings. By applying techniques from fixed-parameter algorithms in nontrivial ways, algorithms can be obtained that perform surprisingly well on real-world instances for NP-hard problems. The viability of this approach has been demonstrated in recent years through the Parameterized Algorithms and Computational Experiments Challenge (PACE) [28,54,55,58], in which teams compete to solve real-world inputs using ideas from parameterized algorithm design. Many researchers from all over the world have participated in that challenge. Moreover, the viability of this approach has recently been demonstrated by a wide range of papers. Since the engineering part in the area has recently gained some momentum, we survey recent results and techniques that have started to bridge the gap between theory and practice that is currently observed in the area.

Theoretical Context. All known exact and deterministic algorithms that solve NP-hard problems require time that is at least super-polynomial in the total size of the input. However, some problems can be solved by algorithms that run in time which is exponential only in the size of a fixed parameter while polynomial in the size of the input; those are called *fixed-parameter algorithms*. Here, the parameterized problem can be solved efficiently for small values of the fixed parameter. Formally, a parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the parameter of the problem. A parameterized problem L is *fixed-parameter tractable* if the question $(x,k) \in L$ can be decided by an algorithm in running time

$f(k) \cdot |x|^{\mathcal{O}(1)}$, where f is a computable function depending on k only. The corresponding complexity class is called FPT.

The *W hierarchy* [56] is an important hierarchy for the complexity of parameterized problems. A parameterized problem is in class $W[i]$, if we can transform every instance (x,k) to a decision circuit (a combinatorial circuit with only a single output gate) with *weft* at most i, such that the circuit outputs true if and only if $(x,k) \in L$. The weft of a combinatorial circuit is the maximum number of gates with more than two inputs on any path from input to output. Downey et al. [56] show that FPT $= W[0]$ and that $W[0] \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[poly]$.

Fixed-parameter tractability is closely related to data reduction and kernelization. *Data reduction rules*, or simply *reductions*, reduce the size of a graph while retaining the ability to compute an optimal solution. A graph on which a collection of data reduction rules have been exhaustively applied is called a *reduced* graph. In kernelization, the reduced graph is called a *kernel* $\mathcal{K}$. More formally, given an binary encoded instance $(x,k) \in \{0,1\}^* \times \mathbb{N}$ of some parameterized problem L, a *kernelization* for L produces an instance (x',k') in polynomial time that satisfies: $(x',k') \in L \Leftrightarrow (x,k) \in L$ and $|x'| + k' \leq f(k)$ where f is a computable function. Note that f only depends on the problem parameter k. So roughly speaking, kernelization can be thought of as a preprocessing routine that reduces a given problem instance to its “most difficult part”. The function f measures the kernel size. If $f(k) = \mathcal{O}(k^c)$ for some constant c then the kernel is called polynomial kernel, and we say the problem admits a polynomial kernel.

Many exact algorithms for parameterized problems combine these data reductions with *branching*. These algorithms are called *branch-and-reduce algorithms*. First, the algorithm aims to reduce the graph size by exhaustively applying reduction rules until there are no further data reductions possible or they are prohibitively expensive. Then, the algorithm picks an edge $e \in E$ (or a vertex $v \in V$, depending on the problem) and *branches* the problem into multiple subproblems, one subproblem for each potential state of e in regard to the problem. As an example, for the maximum cut problem or the multiterminal cut problem, branching creates two subproblems, one in which e is part of the cut and one in which e is not part of the cut. The branch-and-reduce algorithm then continues to apply reduction rules to both of these subproblems and continues branching when there are no further reductions possible. The branch-and-reduce algorithm returns the best result over all branches.

Organization. The rest of the paper is organized as follows. We first survey recent data reduction engineering results for selected NP-hard problems, and then for problems in P. We then describe concrete techniques that may be useful for future implementations in the area. Lastly, we give open problems and research questions.

2 Recent Advances for NP-Hard Problems

2.1 Maximum Independent Set and Minimum Vertex Cover

Given an undirected graph $G = (V,E)$, the goal of the *maximum independent set* (MIS) problem is to compute a set of vertices $I \subseteq V$ such that (1) no two vertices in I are adjacent to one another, (2) the set I has maximum cardinality among all such sets. The complement of an independent set I, $V \setminus I$, is called a *vertex cover*. The MIS problem and the complementary problem of finding a minimum vertex cover (MVC) are well-studied NP-hard optimization problems [75] that attract both researchers and practitioners alike. Furthermore, there is no polynomial time algorithm that approximates the MIS size within a factor $\mathcal{O}(n^{1-\varepsilon})$ for any constant $\varepsilon > 0$, unless P = NP [173]. Finally, MIS is $W[1]$-hard [56] when parameterized by solution size k. This makes it unlikely that the problem is fixed-parameter tractable in k [56]. On the other hand, MVC is fixed-parameter tractable in solution size k [56].

Exact Approaches. In recent years, the bridge between theoretically efficient algorithms and their practical applicability has been significantly reduced. In particular, the branch-and-reduce paradigm, i.e., branching algorithms that use a wide variety of reduction rules, have been (1) shown to achieve theoretical running times that are among the best for both MIS and MVC [69, 170], and (2) are able to solve large real-world networks in practice [5]. However, most often the approaches used in practice only use a small subset of the reduction rules that have been proposed to achieve good theoretical running times.

Abu-Khzam et al. [4] introduced and analyzed the crown reduction rule (and the usage of data reduction rules in this context in practice). Even though the crown rule is not as powerful as the linear programming (LP)-based rule [133] when considering the worst-case size of the resulting kernel, they experimentally verified that it often performs as well as the LP-based rule and is significantly faster in many cases. Furthermore, they show that the LP-based rule is most useful for fairly sparse graphs and should be avoided for dense graphs, as it yields little to no reduction in size.

Later, Akiba and Iwata [5] were the first to show the practicality of the branch-and-reduce paradigm for MVC (and MIS) compared to other state-of-the-art approaches like branch-and-bound and branch-and-cut. Their algorithm uses a wide spectrum of reduction rules that form the foundation of much subsequent work. This includes both conceptually simple reduction rules like degree-1 and degree-2 vertex folding [69], as well as more complicated but practically significant rules like unconfined [169] and an LP-based rule [95, 133]. Many of these reduction rules work by removing vertices that are part of some MIS. We illustrate this by briefly covering the degree-1 and degree-2 vertex fold reduction rules: (1) In the degree-1 reduction rule (see Fig. 1) one removes vertices v of degree one (and their neighbors), as they are always in at least one MIS. To see this, note that v or its neighbor w must be in some MIS I, otherwise $I \cup \{v\}$ is an independent set of larger cardinality. If w is in I, one can obtain an independent set of the same size by removing w from I and adding v instead. (2) For the degree-2 vertex fold (see Fig. 2) one removes vertices v with exactly two neighbors u and w that are not adjacent to each other. In this case a new vertex v' is inserted and connected to the union of the neighborhoods of u and w yielding a reduction of the graph size by two vertices. Finally, if v' is part of an MIS I' of the reduced graph, then $I = (I' \setminus \{v'\}) \cup \{u, w\}$ is an MIS of the original graph. Otherwise, $I = (I' \setminus \{v'\}) \cup \{v\}$ is an MIS of the original graph. Using their branch-and-reduce algorithm, Akiba and Iwata were able to solve a large variety of instances including social networks, web graphs and road networks. A

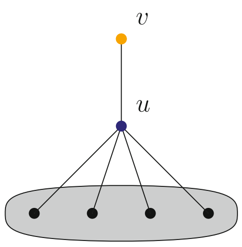

Fig. 1. Degree-1: Vertices v and u can be removed.

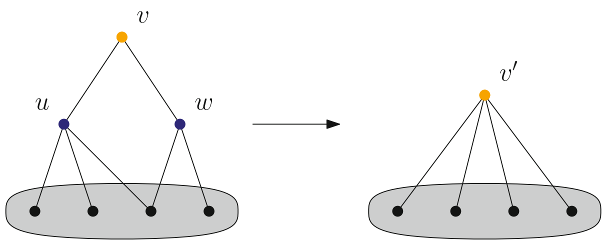

Fig. 2. Degree-2 vertex fold: Vertices v, u and w can be removed. In this case a new vertex v' is inserted.

similar approach that uses a quantum annealer to solve instances once they are small enough was recently presented by Pelofske et al. [140].

Although Akiba and Iwata [5] use a sophisticated set of reduction rules, Strash [155] showed that many of the more complicated rules are not necessary to compute an MIS in many large complex networks. Furthermore, the initial reduction rules applied to compute a reduced graph often have a bigger impact on performance, compared to further techniques used during the branch-and-reduce approach. Recently, Stallmann et al. [153] supported this idea by showing that networks G with a small normalized average degree ($\mathsf{nad}(G)$) can be efficiently handled by simple reduction rules. The $\mathsf{nad}(G)$ of a network G on n vertices is defined as the average degree of G normalized using a factor of $200/n$ if the average is larger than 20. Otherwise, if the average degree is at most 20, $\mathsf{nad}(G)$ is the same as the average degree of G. Additionally, the authors make use of the so-called degree spread t/b, where t is the degree at the 95th percentile and b at the 5th percentile. Based on these characteristics, the authors devise thresholds that indicate (1) if reductions should be used at all, (2) if more complex rules provide a significant benefit.

Open Problem 1. *What are graph characteristics and properties that determine the success of specific reduction rules?*

Recently, Hespe et al. [92] won the PACE Challenge 2019 vertex cover track by using a portfolio of exact approaches for MIS, MVC and maximum clique. In particular they use the reduction rules of Akiba and Iwata as an initial preprocessing step. Afterwards, an initial solution is computed using the state-of-the-art local search algorithm by Andrade et al. [7]. Finally, they switch between the branch-and-reduce algorithm of Akiba and Iwata [5] and the clique solver by Li et al. [119], which are applied to either the original graph or the graph resulting from the preprocessing step.

Heuristic Approaches. Reductions are also heavily used in many state-of-the-art heuristic approaches. Lamm et al. [114, 113 SPP, 150 SPP] use the same set of reductions originally used by Akiba and Iwata to develop an evolutionary algorithm that is able to compute high quality solutions for large graphs that are infeasible for branch-and-reduce. The authors use reductions for both preprocessing (to compute a kernel) and during the algorithm itself. In particular, they select vertices that are part of many highly fit individuals, which are independent sets, in their population. These vertices are then added to the resulting independent set, which includes removing them and their neighbors from the graph. Afterwards, reduction rules are applied and the evolutionary algorithm is called recursively on the resulting graph.

The idea of excluding a subset of vertices that are likely to be part of a high-quality independent set, is also explored by Gao et al. [73]. To select these vertices they perform multiple runs of a state-of-the-art local search algorithm (either NuMVC [33] or FastVC [34]). Vertices that are present in all resulting solutions are then added to the final solution and a new graph consisting of the remaining vertices and their corresponding edges is constructed. Afterwards, a final run of the local search on this graph is executed and its solution is combined with the previously removed vertices.

Dahlum et al. [51, 150 SPP] combine both simple exact reduction rules as well as inexact reductions with the ARW local search algorithm [7]. In particular, they remove cliques of up to size three (an exact reduction) and the top 1% high-degree vertices (an inexact reduction). The reasoning behind their inexact reduction is that high-degree vertices are not likely to be in a large independent set. Additionally, these vertices pose a significant bottleneck for local search. The authors also compare their algorithm against an algorithm that uses the data reduction rules of Akiba and Iwata as a preprocessing step. A similar preprocessing approach that only uses a subset of reduction rules is also presented by Cai et al. [37]. In particular, they use the degree-0, degree-1, degree-2 and domination rules.

Chang et al. [42] also make use of the idea of combining simple reduction rules that can be applied in (near-)linear time with an inexact reduction rule that removes high-degree vertices. For this purpose, they introduce the reducing-peeling framework that switches between the two types of reductions. Furthermore, they present a set of degree-2 path reductions that are special cases of the folding reduction. Combining these new rules with the degree-0, degree-1, dominance and an LP-based reduction rule, they propose an efficient preprocessing algorithm that is then combined with the ARW local search algorithm.

Open Problem 2. *Can one derive (near-)linear time special cases of the more complex reductions like the unconfined reduction that are not covered by existing reductions?*

In order to quickly achieve smaller reduced graphs than what is possible by using simple reduction rules, Hespe et al. [93, 150 SPP] provided the first shared-memory data reduction based on the rules of Akiba and Iwata. For this purpose they make use of both graph partitioning and parallel bipartite maximum matchings. The graph partitioning library KaHIP [148] is used to compute a partition of that graph which allows parallel execution of reduction rules that only need to check highly localized subgraphs, where bipartite maximum matchings are used to enable the parallel execution of the LP-based reduction rule. Furthermore, the authors present two speedup techniques for kernelization: (1) dependency checking that prunes applicability checks for certain reductions and (2) reduction tracking that stops their algorithm once the application of reduction rules only decreases the graph size by a negligible amount.

Open Problem 3. *Can the techniques used by Hespe et al. [93] be extended to a distributed memory setting? How can one efficiently apply reductions in distributed memory?*

Alsahafy and Chang [6] recently proposed an algorithm that combines the reducing-peeling framework with the exact clique solver MoMC by Li et al. [119]. Their algorithms splits reduction rules into two sets: ones that can be updated and applied incrementally (similar to Hespe et al. [93]), and ones that can not. Additionally, they continuously compute and maintain the connected components of the graph, which are then reduced individually. If a reduced component is small enough, it is then transformed into its complement and solved by MoMC. To ensure that components continue to get smaller, they use the same inexact reduction rule as Chang et al. [42] and then continue recursively on the resulting components. The authors also present a new exact reduction rule called the pyramid reduction.

Lastly, Lavallee et al. [118] evaluated a structural rounding approach for vertex cover. The main idea is to first edit a graph to a well-structured graph which can be solved more easily, and then apply a "lifting" algorithm to the partial solution to recover an approximation on the input network. Lavallee et al. find that their algorithm can outperform standard 2-approximation algorithms and that simpler lifting strategies are highly competitive with more sophisticated strategies.

Weighted MIS. Due to the significant practical results achieved for the unweighted case, there has been an increasing interest in generalizing these techniques for the weighted *maximum independent set* (WMIS) and *weighted minimum vertex cover* (WMVC) problems. For both problems, one is given an additional real-valued vertex weighting function $w : V \to \mathbb{R}^+$. In case of the WMIS problem one is then tasked with finding an independent set, such that the sum of the weights of its vertices is maximum among all possible independent sets. Analogously, for the WMVC one is tasked with finding a vertex cover of minimum weight.

Recently, Li et al. [120] used a set of four reduction rules during the initial construction phase of a local search algorithm. In particular, they use weighted reduction rules that are able to remove degree one and degree two vertices. They then use these reduction rules exhaustively in the beginning of their algorithm to obtain an improved initial solution. Their local search algorithm called NuMWVC is able to compute high quality solutions on a large variety of instances. This includes many instances commonly used for the unweighted case, which have been given vertex weights drawn from a uniform distribution. Since there are not many publicly available weighted instances, this is a common approach that is also used in other works [35, 77, 172, 115 SPP]

Wang et al. [165] also make use of reduction rules for vertices with degree at most 2 as a preprocessing step for a branch-and-bound solver. Furthermore they evaluate different degree-based heuristics for selecting branching vertices and use pruning based on the best solution found so far.

Lamm et al. [115 SPP] proposed a practically efficient branch-and-reduce algorithm for the WMIS problem that is able to solve a large number of real-world instances. For this purpose they develop a comprehensive set of practically efficient reduction rules. These include both generalizations of previous weighted and unweighted reduction rules, as well as two "meta reductions" which serve as a general framework for the other rules. They use these rules to build a branch-and-reduce algorithm that uses many of the approaches that worked well in the unweighted case. In particular, they use local searches to compute initial solution which can be used for pruning, treat connected components individually and make use of dependency checking. Finally, they show that their reduction rules can be used to improve the performance of other state-of-the-art algorithms

Zheng et al. [172] propose an exact and heuristic approach that both make use of reduction rules for vertices of degree at most 2. Their exact approach is a branch-and-reduce algorithm that applies these reduction rules recursively. However, the authors do not provide any details on the bounds or branching strategies used during the algorithm. Their heuristic approach is inspired by the reducing-peeling framework of Chang et al. [42]. Thus, it exhaustively applies their reduction rules and subsequently removes high-degree vertices to extend the space of possible reductions.

Gellner et al. [77] proposed new practically efficient variants of the struction rule by Ebenegger et al. [59]. The struction is a reduction that is able to be applied to arbitrary vertices in a graph, but comes at the cost of potentially increasing the overall number of vertices. Thus the authors propose three new variants of the struction that aim to limit the number of newly created vertices. Furthermore, they derive practically efficient special cases of their reduction rules and use them as a preprocessing step in the branch-and-reduce solver of Lamm et al. [115 SPP]. The algorithm is able to produce the smallest-known reduced graphs, solves more instances than previous exact approaches and has a running time that is comparable to heuristic algorithms.

Open Problem 4. *Can other problems also benefit from reductions that may temporarily increase the graph size? If so, how much of an increase should be allowed to remain practical?*

2.2 Finding and Enumerating Maximum Cliques

Given an undirected graph $G = (V, E)$, the goal of the *maximum clique* (MC) problem is to compute a set of vertices $C \subseteq V$ such that (1) all vertices in C are adjacent to one another, (2) the set C has maximum cardinality among all such sets. As mentioned in the previous section, MC solvers are often used in the context of independent sets. This is due to the fact that a clique of G is an independent set in the complement graph $\bar{G} = (V, \bar{E})$ with $\bar{E} = \{\{u, v\} \mid u, v \in V \wedge \{u, v\} \notin E\}$. Thus, one can leverage maximum clique algorithms for finding independent sets by computing the complement graph. Since many algorithms for finding maximum independent sets aim to perform well on sparse graphs, the resulting complement graphs that need to be handled by clique algorithms will often be dense. Fortunately MC has been more extensively studied for dense instances than for sparse instances. Like MIS and MVC, finding a maximum clique is also an NP-hard optimization problem [75]. Furthermore, unless P=NP, there is no polynomial time algorithm that approximates the MC size within a factor $\mathcal{O}(n^{1-\varepsilon})$ for any constant $\varepsilon > 0$ [173]. Finally, MC is $W[1]$-hard [56] parameterized by solution size k, making it unlikely that the problem is fixed-parameter tractable in k. However, it is fixed-parameter tractable under different parameterizations, e.g., when parameterized with the degeneracy of the graph [64]. All previous observations also hold for the *maximum clique enumeration* (MCE) problem of enumerating all maximum cliques in a graph.

Eblen et al. [60] presents a maximum clique solver (MCF) that adapts some of the reduction rules that have already been shown to work well for MVC and MIS. In particular, their algorithm begins by greedily computing a large clique C which is then used as a lower bound in order to remove vertices of degree less than $|C|-1$ [1]. Next, they use an adaptation of the degree-0 reduction rule previously used in MVC algorithms, as well as a rule based on heuristic colorings [160] to remove additional vertices. The authors also investigate the use of other reduction rules including an adaptation of the degree-1 reduction rule used in MVC algorithms. Finally, they compare applying reduction rules as a preprocessing method for a branch-and-bound solver against running them in a branch-and-reduce solver. Their experiments indicate that the branch-and-reduce approach performs better on real-world genome data.

Eblen et al. [60] then use the previous MCF solver to develop several approaches for the maximum clique enumeration (MCE) problem based on the algorithm by Bron and Kerbosch [30]. In particular, they develop two reduction rules based on MCF: First, they propose a reduction rule that uses MCF to compute a maximum clique cover and removes vertices not adjacent to this cover. Second, they propose a second data-driven preprocessing rule that computes so-called essential vertices, i.e., vertices that are present in every maximum clique. Vertices that are not adjacent to these vertices are subsequently removed from the graphs. Their experiments indicate that this rule works particularly well on large transcriptomic graphs, that often have a small set of essential vertices. However, its performance degrades for networks that do not have a small set of essential vertices, e.g., for uniform random graphs.

Open Problem 5. *Can one give similar data-driven reduction rules for other types of networks, e.g., social networks or road networks?*

Verma et al. [164] propose another type of reduction rule based on k-communities. A k-community is defined as a subgraph $G' = (V', E')$ where each edge $\{u, v\} \in E'$ connects vertices that have at least k common neighbors in G'. Subsequently, a subset of vertices $V' \subseteq V$ is called a k-community if there is a k-community with vertex set V' in G. Note, that a clique of size k is a $(k-t)$-community for any $t \in \{2, \ldots, k\}$. They then derive a reduction rule which computes a lower bound on the clique size based on maximum $(k-2)$-communities and prune vertices with a smaller degree. They then combine this reduction rule with the k-core based approach of Pardalos and Resende [1] and show that the resulting algorithm works well for handling large low-density graphs.

Chang [40, 41] notes that even though many real-world networks are usually sparse, MC has been more extensively studied for dense instances. Thus, the authors propose a branch-and-reduce algorithm that leverages the existing work on MC for dense instances by transforming an instance of MC over a sparse graph to instances of k-clique finding (KCF) over dense subgraphs. For this purpose, the authors iteratively compute small and dense subgraphs (so-called ego networks) that are then handled by a KCF solver. In order to reduce the size of the subgraphs that are handled by this solver, their algorithm uses a combination of well-known upper bounds and lightweight reduction rules. In particular, they use five reduction rules for KCF, most of which are targeted toward removing vertices of high degree. The authors also present a heuristic algorithm for MC, as well as a two stage approach for MCE that makes use of their exact algorithm to compute the size of the largest clique. Furthermore, they show that the reduction rules used for MC can also be adapted for MCE.

Weighted MC. Recently, Cai and Lin [36] proposed the first (and only) practical algorithm for the *(vertex-)weighted maximum clique* (WMC) problem that uses reduction rules. The WMC problem is a generalization of MC where one is given an additional real-valued vertex weighting function $w : V \rightarrow \mathbb{R}^+$. Subsequently, one is tasked with finding a clique, such that the sum of the weights of its vertices is maximal among all possible cliques. In order to solve WMC on large sparse graphs, Cai and Lin [36] interleave clique construction with reduction rules. To be more specific, they gradually add "beneficial" vertices to a clique using an approximation of the benefit of a vertex. This is done by computing the mean of a cost-efficient upper and lower bound for each vertex and then selecting vertices using a dynamic best from multiple selection [34]. Finally, if a new best clique is found, the graph is reduced using two reduction rules. Both rules make use of the fact that one is able to remove vertices where an upper bound on any maximum clique containing this vertex is smaller than the weight of the current best clique. For their rules, the authors then propose two different upper bounds that make use of the neighborhood of a vertex.

***k*-plexes.** A k-plex is a generalization of a clique where each vertex is allowed to have several missing connections, i.e., not every vertex has to be connected to all other vertices in the k-plex [151]. In particular, a k-plex is a subset $S \subseteq V$ such that the degree of every vertex in the induced subgraph $G[S]$ is at least $|S| - k$. Furthermore, $|S|$ is called the size of the k-plex and the *maximum k-plex problem* (MK) is that of finding a k-plex of maximum size.

Gao et al. [72] present multiple theoretical properties that allow the removal of vertices based on a lower bound on the maximum k-plex size. Based on these properties they propose four reduction procedures which are then used in a branch-and-reduce algorithm. In particular, they then use an extension of the algorithm by Jiang et al. [100] to compute an initial lower bound and then use this bound to exhaustively apply their linear-time vertex reduction and the more costly subgraph reduction rules for preprocessing. Afterwards they use different sets of reduction rules depending on the type of branch (selecting or discarding a vertex). The authors also present a type of targeted branching that aims to select vertices which lead to a larger reduction in size. The resulting algorithm is able to solve multiple previously infeasible real-world instances and is considerably faster than previous state-of-the-art solvers (e.g., [168]).

Open Problem 6. *Can targeted branching be used for other problems? For example, the most commonly used branching strategy for independent sets is degree-based and does not take any reduction rules into account.*

Conte et al. [46] investigated reduction rules for the problem of enumerating all maximum k-plexes. For this purpose, they introduce the concepts of coreness and cliqueness. Coreness states that vertices of a k-plex of size at least m must have a degree not smaller than $m-k$. Thus, vertices with a smaller degree can iteratively be removed, resulting in the computation of $(m-k)$-cores. Cliqueness states that every vertex of a k-plex of size at least m is part of a clique not smaller than $\lceil m/k \rceil$. Therefore, vertices with a degree less than $\lceil m/k \rceil$ can be removed from the graph. Furthermore, if one knows the size of the maximum clique ω the search space for the size of the maximum k-plex can be limited to $[\omega, \omega \cdot k]$. Based on these observations the authors then present an algorithm that begins by computing the size of a maximum clique. Afterwards a lower bound for the size of the maximum k-plex $p \in [\omega, \omega \cdot k]$ is guessed. If this guess turned out to be wrong (i.e., all k-plexes found are smaller than p), the interval bounds are updated and a new lower bound is guessed. Otherwise, all k-plexes with maximum size are returned. Their algorithm is able to reduce a large set of instances by up to 99% and achieves running times that are multiple orders of magnitude faster than previous approaches [14].

2.3 Maximum Cuts

The *max-cut* problem originates from important applications in physics and operations research [10]; therefore, it has long been the subject of engineering more and more sophisticated algorithms which solve large-scale instances arising in practice. In particular, max-cut is one of the few problems where engineers and practitioners alike are interested in finding optimal solutions (rather than just approximate ones). Formally, the max-cut problem takes as input an edge-weighted graph G and seeks a bipartition of the vertex set V of G into two disjoint parts, V_1 and V_2, which maximizes the weight of the edges which *cross* the bipartition, that is, edges whose one endpoint is in V_1 and the other endpoint is in V_2. The state of the art for max-cut though is that even after much effort, optimal solutions are still unknown for several benchmark instances. Those reasons are the key motivations for engineering effective, and efficient, kernelization rules. The objective is to reduce the given graph G to a new instance G' of smaller size, such that a maximum cut in G can be recovered efficiently from any maximum cut in G'. To the best of our knowledge, preprocessing rules with theoretical guarantees have been studied so far mainly for the unit-weight max-cut. That special case of max-cut, where all edges have the same (unit) weight, is still NP-hard. The goal is thus to find a bipartition (V_1, V_2) which maximizes the size of the cut, which is the number of edges with one endpoint in V_1 and the other endpoint in V_2. To measure the effectiveness of preprocessing rules for unit-weight max-cut, one introduces an integer parameter k. This parameter measures the difference between the size of the maximum cut, and the value

$m/2-(n-1)/4$, which is the well-known lower bound on the size of the maximum cut in any m-edge n-vertex graph, due to Edwards and Erdős [61,62]. There is a set of preprocessing rules, devised by Etscheid and Mnich [66 SPP] which compresses any m-edge n-vertex graph G in linear time to a graph G' on just $\mathscr{O}(k)$ vertices, while allowing to recover the maximum cut of G. This set of rules strengthened earlier work by Crowston et al. [47 SPP], and is moreover the asymptotically best possible. To understand the practical relevance of those rules, Ferizovic et al. [68 SPP] expanded and engineered them. They demonstrated their significant impact on benchmark data sets, including synthetic instances, and data sets from the VLSI and image segmentation application domains. Their experiments revealed that current state-of-the-art solvers can be sped up by up to multiple orders of magnitude when combined with their data reduction rules. On social and biological networks in particular, the preprocessing enabled them to solve four instances that were previously unsolved in a ten-hour time limit with state-of-the-art solvers; three of these instances are now solved in less than two seconds. It is possible to expand the work on preprocessing for unit-weight max-cut to instances with all positive weights. However, designing practically-efficient preprocessing rules for the general max-cut problem, which also provides theoretical guarantees on the kernel size, remains a challenge. Recent work in this direction was done by Lange et al. [116], who designed reduction rules for general max-cut. They showed the efficacy of their rules on instances from computer vision, biomedical image analysis and statistical physics, and for those instances managed to obtain substantial size reductions.

Open Problem 7. *Is it possible to engineer efficient reduction techniques for max-cut with general edge weights?*

2.4 Treewidth and Treedepth

Many NP-hard graph problems can be efficiently solved when the input graph is a tree. A tree decomposition maps vertices of a graph to vertices in a tree, which allows techniques for trees, especially dynamic programming, to be adapted to arbitrary graphs. However, the quality of the tree decomposition impacts the efficiency of such algorithms. *Treewidth* [146] is one measure of this quality, which has been extensively studied in parameterized algorithms literature, which we now describe.

Formally, a *tree decomposition* of a graph $G=(V,E)$ is a family of subsets $\mathscr{X} \subseteq 2^V \setminus \{\varnothing\}$ of V called bags, together with a tree $T=(\mathscr{X},F)$, such that

- $V=\cup_{X\in\mathscr{X}} X$,
- for all $\{u,v\}\in E$ there exists a bag $X\in\mathscr{X}$ such that $u,v\in X$, and
- for all $v\in V$, the bags $\mathscr{X}_v=\{X\in\mathscr{X}\mid v\in X\}$ containing v induce a connected subgraph $T[\mathscr{X}_v]$ (which is necessarily a subtree of T).

The *width* of a tree decomposition of G is one less than the cardinality of its largest bag, that is, $\max_{X\in\mathscr{X}}\{|X|\}-1$. The treewidth of G, denoted $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of G.

Unsurprisingly, computing $\mathrm{tw}(G)$ is NP-hard and deciding if $\mathrm{tw}(G)\le k$ for some positive integer k is NP-complete. This *treewidth* problem is a canonical problem with many theoretical and practical results in the literature. It is fixed-parameter tractable with running time $2^{\mathcal{O}(k^3)}n$ [21], implying it has a kernel exponential in k^3 [32]. The problem does not have a kernel size subexponential in k unless NP $\subseteq$ coNP/poly [22]. Hence, most work focuses on constructing tree decompositions of small width, either approximately [23], or exactly using methods such as positive-instance driven dynamic programming [156]. Both the first and second PACE Challenges had a treewidth track [55]. However, polynomial kernels exist for other parameters. Bodlaender et al. [25] give polynomial kernels of size $\mathcal{O}(\mathsf{fvs}(G)^4)$ and $\mathcal{O}(\mathsf{vc}(G)^3)$, where $\mathsf{fvs}(G)$ is the size of a minimum feedback vertex set and $\mathsf{vc}(G)$ the size of a minimum vertex cover of G, respectively. Their work is inspired by data reduction rules that are known to work well in practice (discussed below), and also includes new rules based on the notion of "clique seeing" paths. Jansen [98] improved the latter kernel to size $\mathcal{O}(\mathsf{vc}(G)^2)$ by introducing a new reduction rule to efficiently find independent sets whose elimination has a predictable effect on the treewidth. To the best of our knowledge, no experiments have been done with clique seeing paths or Jansen's reduction.

Open Problem 8. *Is the rule of Jansen [98] effective in practice?*

Much work has been done in making practical data reductions for the treewidth problem. In early work, Arnborg and Proskurowski [8] introduced reduction rules for recognizing and characterizing partial 3-trees. Bodlaender et al. [27] categorized these reductions into six types (islet, twig, series, triangle, buddy, and cube) and extended these rules, showing them to be highly effective at reducing graph size in practice [27]. Of note here are two variations of well-known reductions from other problems: simplicial vertices and twins of degree 3. They further give a reduction for *almost* simplicial vertices (vertices with all but one neighbor inducing a clique). On graphs with up to 3 032 vertices, the reductions quickly remove 77% of vertices on average, whereas the simplicial vertex reduction alone remove 51% of vertices on average. The worst performing instances had 30% of their vertices removed. Den van Eijkhof et al. [63] generalized many of these reduction rules. They not only introduce new weighted variants, but generalize most previous reductions with a "contraction" reduction rule, and further introduce a reduction for twins of higher degree.

Later, Bodlaender et al. [26] introduced the concept of a safe separator, which decomposes the graph into subgraphs that can be solved independently. It was already known that clique separators were safe [136]; however, they generalize the concept and introduce other easy-to-find separators. They further show that previous reduction rules are subsumed by their safe separator technique. In experiments, their reductions decomposed 33 out of 40 instances. When run as a preprocessing step, their technique speeds up an existing triangulation heuristic, sometimes by multiple orders of magnitude. However, it only gives modest speedups over preprocessing using existing reductions.

Open Problem 9. *How effective are existing treewidth reductions on large sparse graphs (e.g.,with millions of vertices) in practice?*

Open Problem 10. *Can heuristic methods be used to efficiently find safe separators in practice?*

A related concept exists for decompositions into rooted trees. A *treedepth decomposition* of a graph $G = (V,E)$ is a rooted forest F, together with an injective mapping

$\phi : V(G) \to V(F)$ such that, for each edge $(u,v) \in E$, one of $\phi(v)$ or $\phi(u)$ is an ancestor of the other. The treedepth of G, denoted by $\mathsf{td}(G)$, is the minimum height of any treedepth decomposition of G. The *treedepth* problem, deciding if $\mathsf{td}(G) \leq k$ for some positive integer k, is NP-complete [142].

Many similar results exist for the treewidth and treedepth problems. Reidl et al. [145] give a fixed-parameter tractable algorithm for treedepth k, with running time $2^{\mathcal{O}(k^2)}n$, implying the existence of a kernel of size exponential in k^2, and no subexponential kernel exists unless NP $\subseteq$ coNP/poly [22]. However, when parameterized on the vertex cover number $\mathsf{vc}(G)$, the problem has a kernel of size $\mathcal{O}(\mathsf{vc}(G)^3)$ [109], which is achieved through two simple reduction rules that also apply to treewidth: removing simplicial vertices and adding edges between vertices with at least k common neighbors.

However, as far as we are aware, there are significantly fewer experimental works with data reduction rules for treedepth. The 5th PACE Challenge in 2020 was dedicated to exact and heuristic solutions for treedepth. The winning solver by Trimble [161] did not employ any data reduction rules (instead using symmetry breaking together with a variety of lower bounding techniques); however, the second place solver by Korhonen [112] applies the simplicial vertex rule by Kobayashi and Tamaki [109] and a generalization of their common neighbor rule. Korhonen further introduces a new reduction rule based on the Schäffer's linear-time algorithm [149] for computing the treedepth of trees. This rule replaces a tree subgraph $G[T]$ having $|N(V \setminus T)| = 1$ with a subgraph of size $\mathsf{td}(G[T]^2)$. As far as we know there are no published results on the efficacy of these reduction rules. Of further interest is that this algorithm uses minimal separator enumeration. We conclude with the following open problems.

Open Problem 11. *How effective are the reductions of Kobayashi and Tamaki [109] and Korhonen [112] in practice?*

Open Problem 12. *Does the notion of a safe separator extend to the treedepth problem?*

2.5 Hitting Set

Given a set S along with a collection C of its subsets, the *hitting set* problem asks for a subset of S, of minimum cardinality, that has a non-empty intersection with each and every member of C. Hitting set is the dual of *set cover*, which seeks a minimum-cardinality subset of C whose union is S. If the elements of S and C are treated as red and blue vertices, respectively, of a bipartite graph, the equivalent graph theoretic problem is known as *red-blue dominating set* (RBDS).

Hitting set is NP-hard, and $W[2]$-hard when parameterized by the solution size [56]. It becomes fixed-parameter tractable when each member of C is of size bounded by a constant d. In this case the problem is often referred to as d-Hitting Set and it corresponds to RBDS restricted to (red-blue) graphs where each red vertex has at most d neighbors. The problem is also known to be fixed-parameter tractable when parameterized by $|C|$, but this particular parameter is expected to be large in practice. The most popular reductions for Hitting Set are due to Weihe [166]. They are simply based on removing any possible redundant elements from S and C. In this context, an element of S is redundant if all members of C that contain it contain another element; while a member of C is redundant if it is a superset of another member of C. The application of these two rules alone proved to be highly effective on large public transportation networks resulting in a huge reduction in size as pointed out recently by Bläsius et al. [15].

More sophisticated reduction algorithms appeared in the context of kernelization for *d-hitting set* [2, 129, 134]. The kernelization approach of Abu-Khzam [2] was adopted by Mellor et al. [125] and proved to be effective in the context of multiple drug selection for cancer therapy. Moreover, linear-time algorithms that can obtain a kernel of size $\mathcal{O}(k^d)$ were presented by van Bevern [162] and Fafianie and Kratsch [67]. Practical implementations of these algorithms have been addressed recently by van Bevern and Smirnov [163] where they were shown to be more efficient than the reduction procedure of Weihe [166] for small d (up to 5), but can result in more effective data reduction when combined with the reduction rules of Weihe [166].

2.6 Steiner Trees

Given an undirected graph with non-negative edge weights as well as a subset of the vertices (terminals), the *Steiner tree* problem is to find the lightest tree spanning the terminals. There has been a wide range of implementations tackling the Steiner tree problem. Data reductions have long been used for the problem, see, e.g., Polzin [141] or Daneshmand [52]. Daneshmand [52] in particular has shown already in 2004 that many Steiner tree problem instances can be solved by reduction- and heuristic-based approaches.

Recently there have been two implementation challenges, the 11th DIMACS Implementation Challenge in 2014 and the 3rd PACE Challenge [28] in 2018. Here, we focus on the most successful implementations of the 3rd PACE Challenge and the approaches that have been published afterwards. The PACE Challenge had three tracks overall – two exact tracks with one focusing on algorithms for problems with few terminals and one focusing on problems with low treewidth, as well as one heuristic track.

The implementation of Iwata and Shigemura [96] won the track with problems that have few terminals. Their algorithm is based on the dynamic programming formulation of Erickson-Monma-Veinott [65] which has a theoretical running time of $\mathcal{O}(3^t n + 2^t(n\log n + m))$ with t being the number of terminals. Iwata and Shigemura use a novel separator-based pruning technique to speed up their implementation (while keeping the worst-case bound of Erickson-Monma-Veinott). This technique allows them to prune a large number of entries in the dynamic programming table.

The track with problems that have low treewidth was won by SCIP-Jack [143, 144] due to Koch and Rehfeldt. This approach is based on the branch-and-cut principle and was already very successful during the 11th DIMACS Implementation Challenge. For the PACE Challenge, the authors use data reductions that typically reduce the number of edges in the problems by more than 90%. Many instances can already be completely solved by presolving. Moreover, on the remaining instances that can not be presolved, the authors use heuristics to find strong upper and lower bounds quickly. The authors find that in more than 90% of cases that the heuristic already finds the optimum solution on the instances that have not been presolved. Lastly, the branch-and-cut procedure is used to compute lower bounds and prove optimality. Later, the approach was improved [152] to run in distributed memory and thus, by using up to 43 000 cores, managed to solve additional previously unsolved instances or improved on the previously best known solution.

Open Problem 13. *Are there new reductions that have not yet been tried in practice that could help to solve more instances to optimality in practice?*

Open Problem 14. *Can existing reductions for the standard Steiner tree problem be transferred to the more general multi-level Steiner tree problem?*

2.7 Minimum Fill-In

The *minimum fill-in* problem is a critical problem that accelerates Gaussian elimination when solving sparse linear systems [147]. Given a matrix A representing the sparse linear system $Ax = b$, the goal is to find a permutation matrix P that minimizes the number of non-zeros introduced when factorizing $A = PAP^T$. Equivalently, treating A as the adjacency matrix of a graph $G = (V, E)$, we wish to minimize the number of edges introduced in an *elimination ordering*, defined as follows. An *elimination step* removes a vertex $v \in V$ and its incident edges, and adds edges between non-adjacent vertices in $N_G(v)$, producing an elimination graph G_v. An *elimination ordering* of G is a permutation $v_1 v_2 .. v_n$ of all the vertices in G, and the *fill-in* of the ordering is the number of edges introduced by eliminating vertices $v_1, v_2, \ldots, v_n$ in this order. The minimum fill-in is the smallest fill-in given by any elimination ordering. We are often interested in not just computing the minimum fill-in, but an elimination ordering that has minimum fill-in.

Not only is the minimum fill-in NP-hard to compute [171], no polynomial time approximation scheme exists for the problem unless P = NP [39]. However, the problem is fixed-parameter tractable [103], when the input parameter k is the minimum fill-in. The fastest known fixed-parameter algorithm for the problem is due to Fomin and Villanger [71], with running time $2^{\mathcal{O}(\sqrt{k}\log k)} + \mathcal{O}(k^2nm)$, where the additive $\mathcal{O}(k^2nm)$ term is the time to compute a kernel of $\mathcal{O}(k^3)$ vertices [102]. Note that this algorithm is subexponential in the minimum fill-in k and, moreover, is nearly optimal: Cao and Sandeep [39] showed that no algorithm with running time $2^{\mathcal{O}(k^{1/2-\delta})} \cdot n^{\mathcal{O}(1)}$ exists for any positive constant δ, assuming the exponential time hypothesis holds. The smallest known kernel for the problem is due to Natanzon et al. [132] has $2k^2 + 4k$ vertices. The reductions all have the same flavor and are derived for the equivalent problem of *chordal completion*: finding the minimum number of edges to add to the graph so that it is chordal. Kernelization is done by partitioning the vertices into two sets A and B where B induces a chordal graph and A contains vertices from every chordless cycle in G. The set A is formed by repeatedly finding chordless cycles in $G[B]$ via the MCS algorithm [157, 158] and moving a subset of their vertices to A until $G[B]$ is chordal. Then *essential edges* are added to the chordless cycles induced by A, which is the kernel.

In practice, the minimum fill-in problem is extremely hard to solve exactly. Indeed, in the 2nd PACE Challenge in 2017, the winning solver for the minimum fill-in problem only solved 54 out of 100 instances [55], when each instance is given a 30-min time limit. The top three submissions all used kernelization [102] together with dynamic programming over potential maximal cliques [29, 156]. The first place submission by Kobayashi and Tamaki used generalized variants of the data reduction rules of Bodlaender et al. [24], and the third place submission performed preprocessing adapted from the safe separator technique for treewidth [26] in addition to kernelization [102].

However, heuristics, including nested dissection [78] and minimum-degree ordering [159], work quite well in practice for real-world (typically sparse) graphs. Early researchers noted that indistinguishable vertices may be eliminated together, and therefore may be collapsed into a representative vertex while ordering [9, 57]. This reduction speeds up the minimum degree algorithm by more than a factor two in experiments [79]. Ost et al. [137 SPP] recently introduced new data reduction rules based on twins, simplicial vertices, and path compression, and experiments show that they are highly effective in practice when applied before running nested dissection. For road networks, when used as a preprocessing step with other inexact reductions, their techniques give speedups of between 1.79 and 6.37 over nested dissection while simultaneously reducing the fill-in. On social networks, their reductions yield speedups of between 1.72 and 3.92 on 19 out of 21 social networks tested, and the fill-in was reduced on all but one instance.

Open Problem 15. *How effective are the reductions by Ost et al. [137 SPP] when combined with other reductions [132]?*

Open Problem 16. *Is branch-and-reduce feasible for the minimum fill-in problem?*

2.8 Vertex Coloring

Given an unweighted, undirected simple graph $G = (V, E)$, the *q-coloring* problem asks if there exists an assignment of at most q colors to all vertices in V such that no two adjacent vertices have the same color (i.e., a *proper coloring*). The problem of finding the minimum number $\chi(G)$ of colors for which a proper coloring of G exists is known as the *chromatic number* problem.

These problems have received considerable attention by the parameterized algorithms community; however, somewhat surprisingly, there is a wide divide between theory and practice. In theory, a kernel parameterized on only the number of colors is unlikely: since graph coloring is NP-hard for $q = 3$ colors [74], this would give a constant-sized kernel, implying P = NP. Therefore, research has focused on other parameters.

When considering the treewidth $\mathsf{tw}(G)$ of the graph G, if G is given together with a tree decomposition of width $k \geq \mathsf{tw}(G)$, dynamic programming over the tree decomposition gives an algorithm solving q-coloring in time $q^k k^{\mathcal{O}(1)} n$ [49, Theorem 7.9]. Assuming the Strong Exponential Time Hypothesis (SETH) no algorithm of running time $\mathcal{O}(q-\varepsilon)^{\mathsf{tw}(G)}$ exists [122] for any $\varepsilon > 0$. Using the same technique, the chromatic number can be computed in time $k^{\mathcal{O}(k)} n$ [49, Theorem 7.10]. Since these algorithms are fixed-parameter algorithms, the result due to Cai et al. [32] implies kernels of size $q^k k^{\mathcal{O}(1)}$ and $k^{\mathcal{O}(k)}$ exist for q-coloring and chromatic number, respectively. Treewidth is often small for sparse graphs in practice; however, as far as we know, these techniques have not been tried in practice, leading to the following open problem.

Open Problem 17. *How effective is dynamic programming over a tree decomposition for q-coloring (or chromatic number) on sparse graphs in practice?*

Another parameter of interest is size of a minimum vertex cover. Recently, Jansen and Pieterse [99] gave a kernel parameterized on the number $q \geq 3$ of colors and the size k of a minimum vertex cover, having size $\mathcal{O}(k^{q-1} \log k)$ bits, which is optimal up to a factor of $k^{\mathcal{O}(1)}$ [97]. Their result also applies for a tighter parameter, when k is the size of the twin cover. Their technique uses constraint satisfaction with low-degree polynomials. However, in practice, sparse graphs often have a minimum vertex cover size that is linear in the number of vertices. Thus, to be useful in practice, the actual kernel would need to have significantly smaller size. However, to date no one has tested their method in practice, leading to our next open problem for q-coloring.

Open Problem 18. *How effective are the reductions of Jansen and Pieterse [99] in practice?*

The data reductions that have been implemented in practice are simple and without theoretical guarantees on the size of the reduced graph; however, they are also very effective on large sparse graphs. In particular, in experiments for a branch-and-cut algorithm, Mendéz-Díaz and Zabala [126] first preprocess the input graph by computing a large maximal clique K of k vertices, which is a lower bound on the chromatic number. They then iteratively remove each vertex v of degree at most $k-1$ (resulting in a k-core), which is possible since $\chi(G) = \chi(G - \{v\})$. They further give a rule to remove certain vertices with non-neighbors in K. In experiments on 63 graphs of up to 5 231 vertices from the second DIMACS Implementation Challenge[1], their data reductions reduced all graphs between 1–93%, working best on sparse instances. 36 of the 63 instances were reduced by at least 25%, and 21 instances were reduced by at least 50%. The largest percentage reduction was 93% for the `homer` instance, reducing from 561 to 38 vertices.

Verma et al. [164] extend this technique. They first compute lower and upper bounds for the chromatic number, and then iteratively apply the k-core reduction to heuristically color graphs for decreasing values of k. Their key contribution is beginning with an exact coloring of the k-core, which gives a better bound than an initial clique. With this technique they are able to exactly find the chromatic number for very large sparse graphs with up to millions of vertices, with running time varying from seconds to hours. In total they solve 33 of 53 instances from SNAP[2] and the tenth DIMACS Implementation Challenge[3]. Lin et al. [121] extended the low degree reduction to remove entire independent sets of vertices with low degree, which in some cases is orders of magnitude faster than the algorithm of Verma et al. [164]. However, they are not able to solve any additional instances.

We finally note that a crown reduction exists for the *dual coloring* problem, which asks if the graph has an $(n-k)$-coloring [70]. Crown reductions are particularly effective in practice for other problems, specifically the minimum vertex cover problem. In theory, for dual coloring, the crown reduction produces a kernel of size at

[1] http://archive.dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/.
[2] http://snap.stanford.edu/data.
[3] http://www.cc.gatech.edu/dimacs10/.

most $3k-3$ [70, Theorem 4.9]. As far as we are aware, no one has performed experiments with this reduction, leading to our final open problem for graph coloring.

Open Problem 19. *How effective is the crown reduction [70, Theorem 4.9] for graph coloring in practice?*

2.9 Cluster Editing

The *cluster editing* problem is as follows: given a graph $G=(V,E)$, transform it into a vertex-disjoint union of cliques by inserting and deleting a minimum number of edges,i.e., by making a minimum number of editions in the graph. The problem is also known as correlation clustering and has many applications, especially in computational biology [17]. The parameterized complexity of the cluster editing problem using the number of edits k as a parameter is well-studied. The currently best known algorithm in theory is due to Böcker [16] and has running time $\mathscr{O}(1.62^k+n+m)$, where m is the number of edges.

There has been a wide range of methods applying fixed-parameter techniques in the area. Dehne et al. [53] presented the first practical implementation of a fixed-parameter based method for cluster editing. Their algorithm is exact and implements the kernelization routines of [82] and adds ideas to bound the search space for the parameter k via linear programming. Gramm et al. contributed three reduction rules. For example, if two vertices u and v have more than k common neighbors then the edge $\{u,v\}$ has to be in the solution and is added if it is not present. Moreover, if u and v have more than k non-common neighbors, i.e., vertices that are either neighbors of u but not v or vice versa, then the edge $\{u,v\}$ does not belong to the solution. Lastly, if u and v have more than k common and more than k non-common neighbors, then the given instance has no solution. Overall, their method performs best using a refined branching method with re-kernelization. Interestingly, the experimental analysis of their algorithm shows that binary search may not be the best way to implement a fixed-parameter based approach for cluster editing.

Guo [83] later gave parameter-independent data reductions based on critical cliques, obtaining a linear kernel of $4k$ vertices, which was improved by Chen and Meng [45] to $2k$. Böcker et al. [20] introduced additional parameter-independent data reductions and find that preprocessing is possible if the number of edge modifications is significantly smaller than the number of vertices in the graph. In addition to the parameter-independent rules they combine their technique with the parameter-dependent reductions from above with lower and upper bounds. Böcker et al. find that they can effectively reduce graphs that satisfy $k \leq 25|V|$, whereas the reductions due to Guo [83] are only effective for $k \leq |V|/2$. Their experiments show that computing exact solutions for cluster editing is no longer limited to small or almost transitive graphs. Afterwards, Böcker et al. [18, 19] extended their results to the weighted version of the problem in which the weight of an edge yields the cost of deleting or inserting it, and the goal is to apply a set of edge modifications with minimum total weight. To this end, they include non-trivial extensions of the data reduction rules of the unweighted case. Additionally, they present a technique to merge vertices which drastically improves the running time of their algorithm. Recently, Bastos et al. [135] combine exact methods with local search heuristics. More precisely, the authors propose a GRASP and an ILS metaheuristic with different neighborhoods as well as a new reduction rule for the problem. They show that the used data reduction rules can speed up linear programming for some instances up to 95% decreased runtime after using reduction rules and 41% decreased runtime on average on the instances that the solver could solve to optimality.

Open Problem 20. *Is it possible to compute small kernels in practice if the parameter k is larger than $25|V|$? Are there any specific data reduction rules for that case? If an instance in practice does not reduce well, does that help to obtain bounds on the parameter k?*

Since the parameter k is often large compared to the number of vertices, fixed-parameter algorithms may not always be practical. There has been several attempts to use other parameters such as the number of missing edges per cluster as well as the number of edges between clusters [85], the total number of edge modifications per vertex [3, 110]. Abu-Khzam [3], using local parameters that bound the amount of (either or both) edge addition and deletion per vertex resulted in a number of reduction rules, showed how to solve much larger problem instances and apply the problem effectively in data analysis [11, 12].

2.10 Multiterminal Cut

The *multiterminal cut* problem with k terminals is defined as follows: Its input is an undirected edge-weighted graph $G = (V, E, w)$ with edge weights $w : E \mapsto \mathbb{N}_{>0}$ and its goal is to divide its set of vertices into b blocks such that each block contains exactly one terminal and the weight sum of the edges running between the blocks is minimized. It is a fundamental combinatorial optimization problem that was first formulated by Dahlhaus et al. [50] and Cunningham [48]. It is NP-hard for all $b \geq 3$ [50], even on planar graphs, and reduces to the minimum s-t-cut problem, which is in P, for $b = 2$. The minimum s-t-cut problem aims to find the minimum cut in which the vertices s and t are in different blocks. Most algorithms for the multiterminal cut problem use minimum s-t-cuts as a subroutine. Dahlhaus et al. [50] give a $2(1 - 1/b)$ approximation algorithm with polynomial running time. Their approximation algorithm uses the notion of *isolating cuts*, i.e., a minimum cut separating a terminal from all other terminals. They prove that the union of the $b - 1$ smallest isolating cuts yields a valid multiterminal cut with the desired approximation ratio. The currently best known approximation algorithm by Buchbinder et al. [31] uses linear program relaxation to achieve an approximation ratio of 1.323.

Marx [123] proves that the multiterminal cut problem is fixed-parameter tractable when parameterized by multiterminal cut weight $\mathscr{W}(G)$. Chen et al. [44] give the first fixed-parameter tractable algorithm with running time of $4^{\mathscr{W}(G)} \cdot n^{\mathscr{O}(1)}$, later improved by Xiao [167] to $2^{\mathscr{W}(G)} \cdot n^{\mathscr{O}(1)}$ and by Cao et al. [38] to $1.84^{\mathscr{W}(G)} \cdot n^{\mathscr{O}(1)}$.

Recently, Henzinger et al. [88] engineer an algorithm that combines the branch-and-bound formulation of Xiao [167] with existing and new data reduction rules for the problem and present a shared-memory parallel branch-and-reduce algorithm for the multiterminal cut problem. Experiments indicate that this is orders of magnitude faster than previous ILP formulations for the problem that have been employed by practitioners. Later, reduction rules were combined with local search algorithms for the problem [87 SPP]. The algorithm uses a wide variety of reduction rules with varying computational complexity; using vertex neighborhoods, edge connectivities, articulation points, maximum flows and more criteria to reduce the problem size; Henzinger et al. [87 SPP] report size reductions of up to multiple orders of magnitude in some instances, which make large instances solvable in practice. Additionally, they give an inexact algorithm that aggressively prunes subproblems which likely do not yield an improved solution.

Open Problem 21. *Is there an efficient way to find semi-isolated small clusters that can be contracted (either exact or inexact contraction)?*

Open Problem 22. *The algorithm by Henzinger et al. [88] uses only reductions that guarantee that the optimal solution remains in the graph. Are there reductions that do not guarantee optimality but give good performance in practice?*

3 Recent Advances for Problems in P

3.1 Minimum Cut

Given an undirected graph with non-negative edge weights, the *minimum cut* problem is to partition the vertices into two sets so that the sum of edge weights between the two sets is minimized. The size of a minimum cut is often also referred to as the *edge connectivity* of a graph [91, 130]. Gomory and Hu [81] observed that a (global) minimum cut can be computed with $n-1$ minimum s-t-cut computations. For the following decades, this result by Gomory and Hu was used to find better algorithms for global minimum cut using improved maximum flow algorithms [105]. Hao and Orlin [84] adapt the push-relabel algorithm to pass information to future flow computations. When a push-relabel iteration is finished, they implicitly merge the source and sink to form a new sink and find a new source. Vertex heights are maintained over multiple iterations of push-relabel. With these techniques, they achieve a total running time of $\mathcal{O}(mn\log\frac{n^2}{m})$ for a graph with n vertices and m edges, which is asymptotically equal to a single run of the push-relabel algorithm.

However, for minimum cut algorithms to be viable for applications they must be fast on small data sets and scale to large data sets. Thus, an algorithm should have either linear or near-linear running time, or have an efficient parallelization. *All* existing exact algorithms have non-linear running time [84, 91, 105], the fastest of which is the deterministic algorithm of Henzinger et al. [91] with running time $\mathcal{O}(m\log^2 n\log\log^2 n)$. Although this is arguably near-linear theoretical running time, it is not known how the algorithm performs in practice. Even the randomized algorithm of Karger and Stein [105], which finds a minimum cut only with high probability, has $\mathcal{O}(n^2\log^3 n)$ running time, although this was later improved by Karger [104] to $\mathcal{O}(m\log^3 n)$ and recently improved further by Gawrychowski et al. [76] to $\mathcal{O}(m\log^2 n)$. The algorithm of Karger and Stein can be seen as probabilistic data reduction algorithms as they contract random edges to reduce the problem size, and give the correct answer with a certain probability.

Padberg and Rinaldi [138] give a set of heuristics for edge contraction. Chekuri et al. [43] give an implementation of these heuristics that can be performed in time linear in the graph size. Using these heuristics it is possible to sparsify a graph while preserving at least one minimum cut in the graph. If their algorithm does not find an edge to contract, it performs a maximum flow computation, giving the algorithm worst case running time $\mathcal{O}(n^4)$. However, the heuristics can also be used to improve the expected running time of other algorithms by applying them on interim graphs [43].

Open Problem 23. *Some reductions of Padberg and Rinaldi [138] potentially check each triangle in a graph. Can pruning be used to efficiently identify which subset needs to be checked?*

Nagamochi et al. [130, 131] give a minimum cut algorithm that does not use any flow computations. Instead, their algorithm uses maximum spanning forests to find a non-empty set of contractible edges. The intuition behind the algorithm is as follows: suppose you have an unweighted graph with minimum cut value exactly one. Then any spanning tree must contain at least one edge of each of the minimum cuts. Hence, after computing a spanning tree, every remaining edge can be contracted without losing the minimum cut. Nagamochi, Ono and Ibaraki extend this idea to the case where the graph can have edges with positive weight as well as the case in which the minimum cut is bounded by $\hat{\lambda}$ and show how edges are identified using one modified breadth first search. This contraction algorithm is run until the graph is contracted into a single vertex. The algorithm has a running time of $\mathcal{O}(mn + n^2 \log n)$. Stoer and Wagner [154] give a simpler variant of the algorithm of Nagamochi, Ono and Ibaraki [131], which has a the same asymptotic time complexity. The performance of this algorithm on real-world instances, however, is significantly worse than the performance of the algorithms of Nagamochi, Ono and Ibaraki or Hao and Orlin, as shown in experiments conducted by Jünger et al. [101]. Both the algorithms of Hao and Orlin, and Nagamochi, Ono and Ibaraki achieve close to linear running time on most benchmark instances [43, 101].

Based on the algorithm of Nagamochi, Ono and Ibaraki, Matula [124] gives a $(2+\varepsilon)$-approximation algorithm for the minimum cut problem. The algorithm contracts more edges than the algorithm of Nagamochi, Ono and Ibaraki to guarantee a linear time complexity while still guaranteeing a $(2+\varepsilon)$-approximation factor. Inspired by random contractions, Henzinger et al. [89, 150 SPP] first gave an shared-memory parallel algorithm without guarantees on the cut size. The algorithm is randomized, and has running time $\mathcal{O}(n+m)$ when run sequentially. It repeatedly reduces of the input graph size with both heuristic and exact techniques, and then solve the smallest remaining problem with exact methods. The core idea of the inexact algorithm is that edges in densely connected regions (i.e., inside a cluster of a clustering) are unlikely to be in a minimum cut. The algorithm further uses exact reduction rules from Padberg and Rinaldi [138]. For example, given a bound $\hat{\lambda}$ on the minimum cut, one can obviously contract each edge having weight larger than $\hat{\lambda}$, without losing optimality. Experimental results indicate that the algorithm finds optimal cuts on almost all instances. At the same time, even when run sequentially, the algorithm is significantly faster (up to a factor of 4.85) than other state-of-the-art algorithms.

Later, Henzinger et al. [86, 150 SPP] engineered the fastest known *exact* minimum cut algorithm for the problem. To do so, the authors incorporate the proposed inexact method, use better-suited data structures and other optimizations as well as parallelization of exact methods. More precisely, the exact algorithm uses the *inexact* minimum cut algorithm from above [89, 150 SPP] to obtain a better approximate bound $\hat{\lambda}$ for the problem (recall that the algorithm almost always gave the correct result). As known reduction techniques depend on this bound, the better bound enables us to apply more reductions and to reduce the size of the graph much faster. For example, edges whose incident vertices have a connectivity of at least $\hat{\lambda}$, can be contracted without the contraction affecting the minimum cut. The new exact algorithm outperforms the state-of-the-art by a factor of up to 2.5 already sequentially, and when run in parallel by a factor of up to 12.9. Similar reduction rules were later used by Henzinger et al. [90 SPP, 150 SPP] to find all minimum cuts in graphs.

3.2 Matching

A matching M of a graph $G = (V, E)$ is a subset of edges such that no two elements of M have a common endpoint. Many applications require the computation of matchings with certain properties, like being maximal (no edge can be added to M without violating the matching property), having maximum cardinality, or having maximum total weight $\sum_{e \in M} w(e)$, where w is a positive weight function that assigns weights to edges. Although these problems can be solved optimally in polynomial time, optimal algorithms are not fast enough for many applications involving large graphs where we need near linear time algorithms. For example, the most efficient algorithms for graph partitioning rely on repeatedly contracting maximal matchings, often trying to maximize some edge rating function w. We refer to Holtgrewe et al. [94] for details and examples. For the *maximum cardinality matching* problem, already in the 1980s data reduction rules were proposed by Karp and Sipser [107]. The rules are able to deal with vertices that have degree smaller than two. For example, it is quite easy to see that a vertex having degree zero can be removed from the graph, or if a vertex has degree one, then there is always a maximum matching that has this edge matched.

Möhring and Müller-Hannemann [128] were among the first to use the rules to speed up heuristic algorithms for the general maximum cardinality problem. As exact algorithms for the matching problems typically search for augmenting paths, they can be sped up by using a good initial matching. Hence, later Langguth et al. [117] analyzed the effects of various initializations on the total running time of several exact algorithms for the bipartite maximum cardinality problem and are able to achieve significant speed-ups.

Korenwein et al. [111] implement (near-)linear time data reduction rules for the unweighted case as well as the positive-integer-weight case. Applied reductions include Karp-Sipser rules, as well as rules due to Mertzios et al. [127] who have also shown that the maximum cardinality matching problem admits a kernel with at most $12k$ vertices and $13k$ edges where k is the feedback edge number. Moreover, Koana et al. [111] transfer results from vertex cover to the matching problem, e.g.,crown and LP-based data reductions. Experiments indicate that using data reduction rules can speed up state-of-the-art solvers by a factor of 4.7 for the unweighted case and 12.72 on average in the weighted case.

Open Problem 24. *Can the reduction rules due to Koana et al. [111] be exhaustively applied in linear time? Are there more rules that can be transferred from vertex cover to the matching problem that can be applied in near-linear time?*

Kaya et al. [108] also use Karp-Sipser-based kernels for bipartite graph matching. In particular, the authors describe an efficient implementation as well as modifications to reduce time complexity on worst-case instances. Their implementation is about a factor 2 faster then the general purpose implementation of Koana et al. [111]. Recently, Panagiotas and Uçar [139] engineer fast almost optimal algorithms for bipartite graph matching. To this end, the authors investigate two randomized algorithms by Karp et al. [106] and Goel et al. [80] and convert them to efficient heuristics for bipartite graphs. In particular, the algorithm by Karp [106] incorporates Karp-Sipser rules. Both of their heuristics run in near linear time and obtain matchings whose cardinality is more than 99% of the maximum.

Open Problem 25. *Is it possible to implement the degree-2 vertex Karp-Sipser rule in linear time?*

4 Engineering Techniques

Engineering techniques are necessary to make data reduction algorithms scale in practice. We give a short overview of techniques that are currently used in practice. The techniques we reference here include dependency checking, reduction tracking, plateau/increasing data transformations, limiting to simple and fast reductions, reduce and peeling, limited reductions, on-the-fly reductions and lastly parallelization.

Dependency checking allows pruning of reductions when they will provably not succeed, therefore significantly reducing the number of failed reductions. To compute a kernel, algorithms typically apply their reductions $r_1, \ldots, r_j$ by iterating over all reductions and trying to apply the current reduction r_i to all vertices. If r_i reduces at least one vertex, they restart with reduction r_1. When reduction r_j is executed, but does not reduce any vertex, all reductions have been applied exhaustively, and a kernel is found. Trying to apply every reduction to all vertices can be expensive in later stages of the algorithm where few reductions succeed. The algorithm may repeatedly attempt to apply the same reduction to a vertex even though the graph has not changed sufficiently to allow the reduction to succeed. Checking dependencies between reductions [93], allows to avoid applying certain local reductions when they will provably not succeed, e.g.,if their relevant neighborhood did not change since the reduction was last checked. Therefore dependency checking keeps a set D of *viable* candidate vertices: vertices whose relevant neighborhood has changed and vertices that have never been considered for reductions. Then reductions are only applied to candidates that are in the set D. This avoids a lot of work and can speed up data reduction significantly.

Reduction Tracking. The algorithm by Hespe et al. [93] stops local reductions when they are not effectively reducing the global graph sizes. It is not *always* ideal to apply reductions exhaustively—for example, if only few reductions will succeed and they are costly. During later stages of a data reduction algorithm, local reductions may lead to very few graph changes. Therefore, it may be better to stop local reductions early instead of performing them exhaustively and switch to global, more expensive reductions that may change the graph more significantly. Although the resulting graph is kernel-like, it may be possible to reduce it further. Such a graph is called a *quasi kernel*. Note, however, that this is a trade-off between size of the reduced graph and data reduction speed.

Plateau/Increasing Transformations. The general scheme in data reduction is to apply reductions exhaustively until non of the available reductions can be applied anymore. Gellner et al. [77] engineer new generalized data reduction and transformation rules for the weighted independent set problem. A key feature of this work are some transformation rules that can *increase* the size of the input. Surprisingly, these so-called *increasing transformations* can simplify the problem and also open up the reduction space to yield even smaller irreducible graphs later throughout the algorithm. Overall, for the weighted independent set problem, this yields significant speed ups and enables the authors to solve more instances to optimality than previously possible.

Simple Reductions. Often the smallest kernels (or seemingly equivalently, the most varied reductions) give the best chance at finding solutions. For instance, the reductions used by Akiba and Iwata [5] for the maximum independent set problem are the *only* ones known to compute an exact solution on certain large-scale graphs, and these reductions are further successful in computing exact solutions in an evolutionary approach [114]. However it is not always beneficial to compute the smallest kernel possible. Fast and simple reductions can compute kernels that are "small enough" for local search to quickly find high-quality, and even exact, solutions much faster than the reductions used to find the smallest kernels [42,51]. Fast and simple reductions can even be used to solve many large-scale instances exactly [155] just as quickly as the algorithm by Akiba and Iwata [5].

Reduce and Peel. Lamm et al. [114] showed that including reductions in a branch-and-reduce inspired evolutionary algorithm for the independent set problem enables finding exact solutions much faster than provably exact algorithms. To this end, reductions are applied exhaustively. Once a reduced graph is computed, vertices that are unlikely to be in the solution, e.g.,vertices having a very large degree, are removed from the graph and hence excluded from the solution. The algorithm then proceeds recursively. Chang et al. [42] improved on this result by implementing reduction rules to reduce the lead time for kernelization for local search. They introduce "reducing–peeling" to find a large initial solution for local search. This technique can be viewed as computing one path through the search space of a branch-and-reduce algorithm: they repeatedly exclude high-degree vertices and reduce the graph until it is empty, then they take the solution found as an initial solution for local search.

Limited Reductions. Sometimes reductions can be very expensive, for example if their running time depends on the number of edges in the neighborhood of a certain vertex. However, as mentioned above it is often not necessary to compute the smallest possible kernel in practice. Hence, a common technique in practice is to exclude such reductions, for example, if the degree of a vertex is too large. An application of this technique is due to Ost et al. [137 SPP] for the vertex ordering problem, where the simplicial vertex reduction rule is limited to vertices of degree at most 18.

On-the-Fly Reductions. Data reduction can be used as a preprocessing step to exact algorithms. However, reductions are also used to reduce the size of the search space of local search algorithms without losing solution quality. Dahlum et al. [51] apply a set of simple reductions on the fly for the independent set problem. For this algorithm, they use simple reductions that do not require changing the neighborhoods of vertices. Instead, vertices are marked as removed, e.g.,simplicial vertices. This speeds up local search significantly.

Parallelization. A general technique to speed up algorithms is parallelization. Also in data reduction parallelization is used to speed up preprocessing times. For example, "local" reduction rules have been parallelized by using graph partitioning techniques, i.e., each process works on a subgraph and applied reductions only in his subgraph [93]. At the same time, there are also attempts [93] to parallelize more expensive "global" reductions, e.g., reductions that need to access the whole input instance.

Targeted Branching. Branch-and-reduce algorithms often make use of vertex selection strategies that are carried over from existing branch-and-bound approaches. However, these selection strategies often do not take into account that removing certain vertices from the graph might result in an increase of the reduction space, which in turn might lead to smaller search trees. Gao et al. [72] thus present a dynamic vertex selection strategy that also takes into account one of their reduction rules and uses a degree-based selection as a fallback. Their experiments indicate that this strategy is able to provide better results when compared to a purely degree-based selection rule.

Data-Driven Reductions. Eblen et al. [60] show the benefits of using application-specific reduction rules that exploit prior knowledge of the input space. In particular, they use a reduction rule that is based on the empirical evaluation of large transcriptomic graphs and is able to drastically reduce the running time of their algorithm on similar instances. However, this comes at the drawback of a decrease in performance for random graphs.

5 Open Problems and Future Work

We already discussed problem-specific open problems throughout this article. Here, we list some general open questions that apply to a range of problems touched in this survey. For example, in a branch-and-reduce algorithm can we branch to specifically get graphs that reduce better using the available portfolio of reductions? As a concrete example, as stated above, it may be helpful to end up with a lot of independent connected components and to achieve this one may be able to branch on a small vertex separator first. For most problems, what makes an instance hard to reduce is currently unknown, e.g., when does which data reduction rule work well in practice and why?

From the theory perspective of a practitioner, it would be better to have an analysis of the expected kernel size, rather than the worst case so as to get more realistic results in practice. One does not always need a single optimal solution, but a diverse set of high-quality solutions. Theoretical approaches for this have been proposed [13], however, they remain untested in practice. Probabilistic reductions have not yet been tried in practice. On the other hand, most of the dynamic techniques that maintain a problem kernel have also not yet been implemented. A problem that needs careful investigation is the order in which reduction rules are applied, e.g., when is it good to apply which reduction rule first? Lastly, consider an instance for a problem on which you already applied all data reduction rules at hand exhaustively. Moreover, assume that you already have an optimal solution on the reduced instance. Is it possible to discover new rules by applying machine learning techniques on such instances?

Acknowledgement. Partially supported by DFG grants MN 59/1-1, SCHU 2567/1-2 and SCHU 2567/3-1.

References

1. Abello, J., Pardalos, P.M., Resende, M.: On maximum clique problems in very large graphs. In: External Memory Algorithms, pp. 119–130 (1999). https://doi.org/10.1090/dimacs/050/06
2. Abu-Khzam, F.N.: A kernelization algorithm for d-hitting set. J. Comput. Syst. Sci. **76**(7), 524–531 (2010). https://doi.org/10.1016/j.jcss.2009.09.002
3. Abu-Khzam, F.N.: On the complexity of multi-parameterized cluster editing. J. Discrete Algor. **45**, 26–34 (2017). https://doi.org/10.1016/j.jda.2017.07.003
4. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: theory and experiments. In: Proceedings of ALENEX/ANALCO, pp. 62–69 (2004)
5. Akiba, T., Iwata, Y.: Branch-and-reduce exponential/FPT algorithms in practice: a case study of vertex cover. Theore. Comput. Sci. 609, Part **1**, 211–225 (2016). https://doi.org/10.1016/j.tcs.2015.09.023
6. Alsahafy, M., Chang, L.: Computing maximum independent sets over large sparse graphs. In: Cheng, R., Mamoulis, N., Sun, Y., Huang, X. (eds.) WISE 2020. LNCS, vol. 11881, pp. 711–727. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34223-4_45
7. Andrade, D.V., Resende, M.G., Werneck, R.F.: Fast local search for the maximum independent set problem. J. Heuristics **18**(4), 525–547 (2012). https://doi.org/10.1007/s10732-012-9196-4
8. Arnborg, S., Proskurowski, A.: Characterization and recognition of partial 3-trees. SIAM J. Algeb. Discrete Methods **7**(2), 305–314 (1986). https://doi.org/10.1137/0607033
9. Ashcraft, C.: Compressed graphs and the minimum degree algorithm. SIAM J. Scient. Comput. **16**(6), 1404–1411 (1995). https://doi.org/10.1137/0916081
10. Barahona, F., Grötschel, M., Jünger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Oper. Res. **36**(3), 493–513 (1988). https://doi.org/10.1287/opre.36.3.493
11. Barr, J.R., Shaw, P., Abu-Khzam, F.N., Yu, S., Yin, H., Thatcher, T.: Combinatorial code classification vulnerability rating. In: 2020 Second TransAI, pp. 80–83 (2020). https://doi.org/10.1109/TransAI49837.2020.00017

12. Barr, J.R., Shaw, P., Abu-Khzam, F.N., Chen, J.: Combinatorial text classification: the effect of multi-parameterized correlation clustering. In: Proceedings of GC 2019, pp. 29–36 (2019). https://doi.org/10.1109/GC46384.2019.00013
13. Baste, J., et al.: Diversity of solutions: an exploration through the lens of fixed-parameter tractability theory. In: Proceedings of IJCAI 2020, pp. 1119–1125 (2020). https://doi.org/10.24963/ijcai.2020/156
14. Berlowitz, D., Cohen, S., Kimelfeld, B.: Efficient enumeration of maximal k-plexes. In: Proceedings of SIGMOD 2015, pp. 431–444 (2015). https://doi.org/10.1145/2723372.2746478
15. Bläsius, T., Fischbeck, P., Friedrich, T., Schirneck, M.: Understanding the effectiveness of data reduction in public transportation networks. In: Proceedings of WAW 2019, pp. 87–101 (2019). https://doi.org/10.1007/978-3-030-25070-6_7
16. Böcker, S.: A golden ratio parameterized algorithm for cluster editing. J. Discrete Algor. **16**, 79–89 (2012). https://doi.org/10.1016/j.jda.2012.04.005
17. Böcker, S., Baumbach, J.: Cluster editing. In: Bonizzoni, P., Brattka, V., Löwe, B. (eds.) CiE 2013. LNCS, vol. 7921, pp. 33–44. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39053-1_5
18. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truss, A.: Going weighted: parameterized algorithms for cluster editing. Theor. Comput. Sci. **410**, 5467–5480 (2009). https://doi.org/10.1016/j.tcs.2009.05.006
19. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: A fixed-parameter approach for weighted cluster editing. In: Proceedings of APBC 2008, pp. 211–220 (2008). https://doi.org/10.1142/9781848161092_0023
20. Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: evaluation and experiments. Algorithmica **60**(2), 316–334 (2011). https://doi.org/10.1007/s00453-009-9339-7
21. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996). https://doi.org/10.1137/S0097539793251219
22. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. **75**(8), 423–434 (2009). https://doi.org/10.1016/j.jcss.2009.04.001
23. Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipczuk, M.: A $c^k n$ 5-approximation algorithm for treewidth. SIAM J. Comput. **45**(2), 317–378 (2016). https://doi.org/10.1137/130947374
24. Bodlaender, H.L., Heggernes, P., Villanger, Y.: Faster parameterized algorithms for MINIMUM FILL-IN. Algorithmica **61**(4), 817–838 (2010). https://doi.org/10.1007/s00453-010-9421-1
25. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Preprocessing for treewidth: a combinatorial analysis through kernelization. SIAM J. Discrete Math. **27**(4), 2108–2142 (2013). https://doi.org/10.1137/120903518
26. Bodlaender, H.L., Koster, A.M.: Safe separators for treewidth. Discrete Math. **306**(3), 337–350 (2006). https://doi.org/10.1016/j.disc.2005.12.017
27. Bodlaender, H.L., Koster, A.M., Eijkhof, F.V.d.: Preprocessing rules for triangulation of probabilistic networks. Comput. Intell. **21**(3), 286–305 (2005). https://doi.org/10.1111/j.1467-8640.2005.00274.x
28. Bonnet, É., Sikora, F.: The PACE 2018 parameterized algorithms and computational experiments challenge: the third iteration. In: Proceedings of IPEC 2018, pp. 26:1–26:15 (2018). https://doi.org/10.4230/LIPIcs.IPEC.2018.26

29. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: grouping the minimal separators. SIAM J. Comput. **31**(1), 212–232 (2001). https://doi.org/10.1137/S0097539799359683
30. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Comm. ACM **16**(9), 575–577 (1973). https://doi.org/10.1145/362342.362367
31. Buchbinder, N., Naor, J., Schwartz, R.: Simplex partitioning via exponential clocks and the multiway cut problem. SIAM J. Comput. **47**, 1463–1482 (2018). https://doi.org/10.1137/15M1045521
32. Cai, L., Chen, J., Downey, R.G., Fellows, M.R.: Advice classes of parameterized tractability. Ann. Pure Appl. Logic **84**(1), 119–138 (1997). https://doi.org/10.1016/S0168-0072(95)00020-8
33. Cai, S., Su, K., Luo, C., Sattar, A.: NuMVC: an efficient local search algorithm for minimum vertex cover. J. Artif. Intell. Res. **46**, 687–716 (2013). https://doi.org/10.1613/jair.3907
34. Cai, S.: Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In: Proceedings of IJCAI 2015, pp. 747–753 (2015)
35. Cai, S., Hou, W., Lin, J., Li, Y.: Improving local search for minimum weight vertex cover by dynamic strategies. In: Proceedings of IJCAI 2018, pp. 1412–1418 (2018). https://doi.org/10.24963/ijcai.2018/196
36. Cai, S., Lin, J.: Fast solving maximum weight clique problem in massive graphs. In: Proceedings of IJCAI 2016, pp. 568–574 (2016)
37. Cai, S., Lin, J., Luo, C.: Finding a small vertex cover in massive sparse graphs: construct, local search, and preprocess. J. Artif. Intell. Res. **59**, 463–494 (2017). https://doi.org/10.1613/jair.5443
38. Cao, Y., Chen, J., Fan, J.H.: An $\mathcal{O}(1.84^k)$ parameterized algorithm for the multiterminal cut problem. Inf. Proc. Lett. **114**(4), 167–173 (2014). https://doi.org/10.1016/j.ipl.2013.12.001
39. Cao, Y., Sandeep, R.: Minimum fill-in: inapproximability and almost tight lower bounds. Inf. Comput. **271**, 104514 (2020). https://doi.org/10.1016/j.ic.2020.104514
40. Chang, L.: Efficient maximum clique computation over large sparse graphs. In: Proceedings of KDD 2019, pp. 529–538 (2019). https://doi.org/10.1145/3292500.3330986
41. Chang, L.: Efficient maximum clique computation and enumeration over large sparse graphs. VLDB J. **29**(5), 999–1022 (2020). https://doi.org/10.1007/s00778-020-00602-z
42. Chang, L., Li, W., Zhang, W.: Computing a near-maximum independent set in linear time by reducing-peeling. Proc. SIGMOD **2017**, 1181–1196 (2017). https://doi.org/10.1145/3035918.3035939
43. Chekuri, C., Goldberg, A.V., Karger, D.R., Levine, M.S., Stein, C.: Experimental study of minimum cut algorithms. In: Proceedings of SODA 1997, pp. 324–333 (1997)
44. Chen, J., Liu, Y., Lu, S.: An improved parameterized algorithm for the minimum node multiway cut problem. Algorithmica **55**(1), 1–13 (2009). https://doi.org/10.1007/s00453-007-9130-6
45. Chen, J., Meng, J.: A $2k$ kernel for the cluster editing problem. J. Comput. Syst. Sci. **78**(1), 211–220 (2012). https://doi.org/10.1016/j.jcss.2011.04.001
46. Conte, A., Firmani, D., Mordente, C., Patrignani, M., Torlone, R.: Fast enumeration of large k-plexes. In: Proceedings of KDD 2017, pp. 115–124 (2017). https://doi.org/10.1145/3097983.3098031
47 SPP. Crowston, R., Jones, M., Mnich, M.: Max-cut parameterized above the Edwards-Erdős bound. Algorithmica **72**(3), 734–757 (2014). https://doi.org/10.1007/s00453-014-9870-z

48. Cunningham, W.H.: The optimal multiterminal cut problem. In: Reliability of Computer and Communication Networks, pp. 105–120 (1989). https://doi.org/10.1090/dimacs/005/07
49. Cygan, M., et al.: Parameterized Algorithms. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
50. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM J. Comput. **23**(4), 864–894 (1994). https://doi.org/10.1137/S0097539792225297
51. Dahlum, J., Lamm, S., Sanders, P., Schulz, C., Strash, D., Werneck, R.F.: Accelerating local search for the maximum independent set problem. In: Goldberg, A.V., Kulikov, A.S. (eds.) SEA 2016. LNCS, vol. 9685, pp. 118–133. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38851-9_9
52. Daneshmand, S.V.: Algorithmic approaches to the Steiner problem in networks. Ph.D. thesis, Universität Mannheim, Germany (2004). http://bibserv7.bib.uni-mannheim.de/madoc/volltexte/2004/176/index.html
53. Dehne, F., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006). https://doi.org/10.1007/11847250_2
54. Dell, H., Husfeldt, T., Jansen, B.M.P., Kaski, P., Komusiewicz, C., Rosamond, F.A.: The first parameterized algorithms and computational experiments challenge. In: Proceedings of IPEC 2016, LIPI, vol. 63, pp. 30:1–30:9 (2016). https://doi.org/10.4230/LIPIcs.IPEC.2016.30
55. Dell, H., Komusiewicz, C., Talmon, N., Weller, M.: The PACE 2017 parameterized algorithms and computational experiments challenge: the second iteration. In: Proceedings of IPEC 2017, LIPI, vol. 89, pp. 30:1–30:12 (2017)
56. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999). https://doi.org/10.1007/978-1-4612-0515-9
57. Duff, I.S., Reid, J.K.: Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. ACM Trans. Math. Softw. **22**(2), 227–257 (1996). https://doi.org/10.1145/229473.229480
58. Dzulfikar, M.A., Fichte, J.K., Hecher, M.: The PACE 2019 parameterized algorithms and computational experiments challenge: the fourth iteration (invited paper). In: Proceedings of IPEC 2019, LIPI, vol. 148, pp. 25:1–25:23 (2019). https://doi.org/10.4230/LIPIcs.IPEC.2019.25
59. Ebenegger, C., Hammer, P., De Werra, D.: Pseudo-boolean functions and stability of graphs. In: North-Holland mathematics studies, vol. 95, pp. 83–97 (1984). https://doi.org/10.1016/S0304-0208(08)72955-4
60. Eblen, J.D., Phillips, C.A., Rogers, G.L., Langston, M.A.: The maximum clique enumeration problem: algorithms, applications, and implementations. In: BMC Bioinformatics, p. S5 (2012). https://doi.org/10.1186/1471-2105-13-S10-S5
61. Edwards, C.S.: Some extremal properties of bipartite subgraphs. Can. J. Math. **25**(3), 475–485 (1973). https://doi.org/10.4153/CJM-1973-048-x
62. Edwards, C.: An improved lower bound for the number of edges in a largest bipartite subgraph. In: Recent Advances in Graph Theory, pp. 167–181 (1975)
63. van den Eijkhof, F., Bodlaender, H.L., Koster, A.M.C.A.: Safe reduction rules for weighted treewidth. Algorithmica **47**(2), 139–158 (2007). https://doi.org/10.1007/s00453-006-1226-x
64. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in large sparse real-world graphs in near-optimal time. J. Exp. Algorithmics **18**, 3–1 (2013). https://doi.org/10.1145/2543629

65. Erickson, R.E., Monma, C.L., Jr., A.F.V.: Send-and-split method for minimum-concave-cost network flows. Math. Oper. Res. **12**(4), 634–664 (1987). https://doi.org/10.1287/moor.12.4.634

66 SPP. Etscheid, M., Mnich, M.: Linear kernels and linear-time algorithms for finding large cuts. Algorithmica **80**(9), 2574–2615 (2017). https://doi.org/10.1007/s00453-017-0388-z

67. Fafianie, S., Kratsch, S.: A shortcut to (sun)flowers: kernels in logarithmic space or linear time. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9235, pp. 299–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48054-0_25

68 SPP. Ferizovic, D., Hespe, D., Lamm, S., Mnich, M., Schulz, C., Strash, D.: Engineering kernelization for maximum cut. In: Proceedings of ALENEX 2020, pp. 27–41 (2020). https://doi.org/10.1137/1.9781611976007.3

69. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56**(5), 25:1–25:32 (2009). https://doi.org/10.1145/1552285.1552286

70. Fomin, F.V., Lokshtanov, D., Saurabh, S., Zehavi, M.: Kernelization: Theory of Parameterized Preprocessing. Cambridge University Press, Cambridge (2019). https://doi.org/10.1017/9781107415157

71. Fomin, F.V., Villanger, Y.: Subexponential parameterized algorithm for minimum fill-in. SIAM J. Comput. **42**(6), 2197–2216 (2013). https://doi.org/10.1137/11085390X

72. Gao, J., Chen, J., Yin, M., Chen, R., Wang, Y.: An exact algorithm for maximum k-plexes in massive graphs. In: Proceedings of IJCAI 2018, pp. 1449–1455 (2018). https://doi.org/10.24963/ijcai.2018/201

73. Gao, W., Friedrich, T., Kötzing, T., Neumann, F.: Scaling up local search for minimum vertex cover in large graphs by parallel kernelization. In: Proceedings of ACAI 2017, pp. 131–143 (2017). https://doi.org/10.1007/978-3-319-63004-5_11

74. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete problems. In: Proceedings of STOC 1974, pp. 47–63 (1974). https://doi.org/10.1145/800119.803884

75. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Co., San Francisco, Calif. (1979). A Guide to the Theory of NP-Completeness

76. Gawrychowski, P., Mozes, S., Weimann, O.: Minimum cut in $\mathcal{O}(m \log^2 n)$ time. In: Proceedings of ICALP 2020, LIPI, vol. 168, pp. 57:1–57:15 (2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.57

77. Gellner, A., Lamm, S., Schulz, C., Strash, D., Zaválnij, B.: Boosting data reduction for the maximum weight independent set problem using increasing transformations. In: Proceedings of ALENEX 2021, pp. 128–142. https://doi.org/10.1137/1.9781611976472.10

78. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**(2), 345–363 (1973). https://doi.org/10.1137/0710032

79. George, A., Liu, J.W.: The evolution of the minimum degree ordering algorithm. SIAM Rev. **31**(1), 1–19 (1989). https://doi.org/10.1137/1031001

80. Goel, A., Kapralov, M., Khanna, S.: Perfect matchings in $\mathcal{O}(n \log n)$ time in regular bipartite graphs. SIAM J. Comput. **42**(3), 1392–1404 (2013). https://doi.org/10.1137/100812513

81. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. J. Soc. Ind. Appl. Math. **9**(4), 551–570 (1961). https://doi.org/10.1137/0109047

82. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: fixed-parameter algorithms for clique generation. In: Petreschi, R., Persiano, G., Silvestri, R. (eds.) CIAC 2003. LNCS, vol. 2653, pp. 108–119. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44849-7_17
83. Guo, J.: A more effective linear kernelization for cluster editing. Theor. Comput. Sci. **410**(8), 718–726 (2009). https://doi.org/10.1016/j.tcs.2008.10.021
84. Hao, J., Orlin, J.B.: A faster algorithm for finding the minimum cut in a graph. In: Proceedings of SODA 1992, pp. 165–174 (1992)
85. Heggernes, P., Lokshtanov, D., Nederlof, J., Paul, C., Telle, J.A.: Generalized graph clustering: recognizing (p,q)-cluster graphs. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 171–183. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16926-7_17
86. Henzinger, M., Noe, A., Schulz, C.: Shared-memory exact minimum cuts. In: Proceedings of IPDPS 2019, pp. 13–22 (2019). https://doi.org/10.1109/IPDPS.2019.00013
87 SPP. Henzinger, M., Noe, A., Schulz, C.: Faster parallel multiterminal cuts. Technical report (2020). https://arxiv.org/abs/2004.11666
88. Henzinger, M., Noe, A., Schulz, C.: Shared-memory branch-and-reduce for multiterminal cuts. In: Proceedings of ALENEX 2020, pp. 42–55 (2020). https://doi.org/10.1137/1.9781611976007.4
89. Henzinger, M., Noe, A., Schulz, C., Strash, D.: Practical minimum cut algorithms. ACM J. Exp. Algorithmics **23** (2018). https://doi.org/10.1145/3274662
90 SPP. Henzinger, M., Noe, A., Schulz, C., Strash, D.: Finding all global minimum cuts in practice. In: Proceedings of ESA 2020, pp. 59:1–59:20 (2020). https://doi.org/10.4230/LIPIcs.ESA.2020.59
91. Henzinger, M., Rao, S., Wang, D.: Local flow partitioning for faster edge connectivity. SIAM J. Comput. **49**(1), 1–36 (2020). https://doi.org/10.1137/18M1180335
92. Hespe, D., Lamm, S., Schulz, C., Strash, D.: WeGotYouCovered: the winning solver from the PACE 2019 challenge, vertex cover track. In: Proceedings of CSC 2020, pp. 1–11 (2020). https://doi.org/10.1137/1.9781611976229.1
93. Hespe, D., Schulz, C., Strash, D.: Scalable kernelization for maximum independent sets. J. Exp. Algor. **24**(1), 1–22 (2019). https://doi.org/10.1145/3355502
94. Holtgrewe, M., Sanders, P., Schulz, C.: Engineering a scalable high quality graph partitioner. In: Proceedings of IPDPS 2010, pp. 1–12 (2010). https://doi.org/10.1109/IPDPS.2010.5470485
95. Iwata, Y., Oka, K., Yoshida, Y.: Linear-time FPT algorithms via network flow. In: Proceedings of SODA 2014, pp. 1749–1761 (2014). https://doi.org/10.1137/1.9781611973402.127
96. Iwata, Y., Shigemura, T.: Separator-based pruned dynamic programming for Steiner tree. In: Proceedings of AAAI 2019, pp. 1520–1527 (2019). https://doi.org/10.1609/aaai.v33i01.33011520
97. Jaffke, L., Jansen, B.M.P.: Fine-grained parameterized complexity analysis of graph coloring problems. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) CIAC 2017. LNCS, vol. 10236, pp. 345–356. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_29
98. Jansen, B.M.P.: On sparsification for computing treewidth. Algorithmica **71**(3), 605–635 (2014). https://doi.org/10.1007/s00453-014-9924-2
99. Jansen, B.M.P., Pieterse, A.: Optimal data reduction for graph coloring using low-degree polynomials. Algorithmica **81**(10), 3865–3889 (2019). https://doi.org/10.1007/s00453-019-00578-5
100. Jiang, H., Li, C., Manyà, F.: An exact algorithm for the maximum weight clique problem in large graphs. In: Proceedings of AAAI 2017, pp. 830–838 (2017)

101. Jünger, M., Rinaldi, G., Thienel, S.: Practical performance of efficient minimum cut algorithms. Algorithmica **26**(1), 172–195 (2000). https://doi.org/10.1007/s004539910009
102. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. SIAM J. Comput. **28**(5), 1906–1922 (1999). https://doi.org/10.1137/S0097539796303044
103. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal and interval graphs: minimum fill-in and physical mapping. In: Proceedings of FOCS 1994, pp. 780–791 (1994). https://doi.org/10.1109/SFCS.1994.365715
104. Karger, D.R.: Minimum cuts in near-linear time. J. ACM **47**(1), 46–76 (2000). https://doi.org/10.1145/331605.331608
105. Karger, D.R., Stein, C.: A new approach to the minimum cut problem. J. ACM **43**(4), 601–640 (1996). https://doi.org/10.1145/234533.234534
106. Karp, R.M., Kan, A.H.G.R., Vohra, R.V.: Average case analysis of a heuristic for the assignment problem. Math. Oper. Res. **19**(3), 513–522 (1994). https://doi.org/10.1287/moor.19.3.513
107. Karp, R.M., Sipser, M.: Maximum matchings in sparse random graphs. In: Proceedings of FOCS 1981, pp. 364–375 (1981). https://doi.org/10.1109/SFCS.1981.21
108. Kaya, K., Langguth, J., Panagiotas, I., Uçar, B.: Karp-Sipser based kernels for bipartite graph matching. In: Proceedings of ALENEX 2020, pp. 134–145 (2020). https://doi.org/10.1137/1.9781611976007.11
109. Kobayashi, Y., Tamaki, H.: Treedepth parameterized by vertex cover number. In: Proceedings of IPEC 2016, LIPI, vol. 63, pp. 18:1–18:11 (2016). https://doi.org/10.4230/LIPIcs.IPEC.2016.18
110. Komusiewicz, C., Uhlmann, J.: Cluster editing with locally bounded modifications. Discrete Appl. Math. **160**(15), 2259–2270 (2012). https://doi.org/10.1016/j.dam.2012.05.019
111. Korenwein, V., Nichterlein, A., Niedermeier, R., Zschoche, P.: Data reduction for maximum matching on real-world graphs: theory and experiments. In: Proceedings of ESA 2018, LIPI, vol. 112, pp. 53:1–53:13 (2018). https://doi.org/10.4230/LIPIcs.ESA.2018.53
112. Korhonen, T.: SMS in PACE 2020. Technical report (2020). https://arxiv.org/abs/2006.07302

113 SPP. Lamm, S., Sanders, P., Schulz, C.: Graph partitioning for independent sets. In: Bampis, E. (ed.) SEA 2015. LNCS, vol. 9125, pp. 68–81. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20086-6_6

114. Lamm, S., Sanders, P., Schulz, C., Strash, D., Werneck, R.F.: Finding near-optimal independent sets at scale. J. Heurist. **23**(4), 207–229 (2017). https://doi.org/10.1007/s10732-017-9337-x

115 SPP. Lamm, S., Schulz, C., Strash, D., Williger, R., Zhang, H.: Exactly solving the maximum weight independent set problem on large real-world graphs. In: Proceedings of ALENEX 2019, pp. 144–158 (2019). https://doi.org/10.1137/1.9781611975499.12

116. Lange, J.H., Andres, B., Swoboda, P.: Combinatorial persistency criteria for multicut and max-cut. In: Proceedings of IEEE Conference Computer Vision Pattern Recognition, pp. 6093–6102 (2019). https://doi.org/10.1109/CVPR.2019.00625
117. Langguth, J., Manne, F., Sanders, P.: Heuristic initialization for bipartite matching problems. ACM J. Exp. Algorithmics **15** (2010). https://doi.org/10.1145/1671970.1712656

118. Lavallee, B., Russell, H., Sullivan, B.D., van der Poel, A.: Approximating vertex cover using structural rounding. In: Proceedings of ALENEX 2020, pp. 70–80 (2020). https://doi.org/10.1137/1.9781611976007.6
119. Li, C.M., Jiang, H., Manyà, F.: On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. Comput. Oper. Res. **84**, 1–15 (2017). https://doi.org/10.1016/j.cor.2017.02.017
120. Li, R., Hu, S., Cai, S., Gao, J., Wang, Y., Yin, M.: NuMWVC: a novel local search for minimum weighted vertex cover problem. J. Oper. Res. Soc., 1–12 (2019). https://doi.org/10.1080/01605682.2019.1621218
121. Lin, J., Cai, S., Luo, C., Su, K.: A reduction based method for coloring very large graphs. In: Proceedings of IJCAI 2017, pp. 517–523 (2017). https://doi.org/10.24963/ijcai.2017/73
122. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. ACM Trans. Algor. **14**(2) (2018). https://doi.org/10.1145/3170442
123. Marx, D.: Parameterized graph separation problems. Theor. Comput. Sci. **351**(3), 394–406 (2006). https://doi.org/10.1016/j.tcs.2005.10.007
124. Matula, D.W.: A linear time $2+\varepsilon$ approximation algorithm for edge connectivity. In: Proceedings of SODA 1993, pp. 500–504 (1993)
125. Mellor, D., Prieto-Rodríguez, E., Mathieson, L., Moscato, P.A.: A kernelisation approach for multiple d-hitting set and its application in optimal multi-drug therapeutic combinations. PLoS ONE **5**, 1–13 (2010)
126. Méndez-Díaz, I., Zabala, P.: A branch-and-cut algorithm for graph coloring. Discrete Appl. Math. **154**(5), 826–847 (2006). https://doi.org/10.1016/j.dam.2005.05.022
127. Mertzios, G.B., Nichterlein, A., Niedermeier, R.: The power of linear-time data reduction for maximum matching. Algorithmica **82**(12), 3521–3565 (2020). https://doi.org/10.1007/s00453-020-00736-0
128. Möhring, R., Müller-Hannemann, M.: Cardinality matching: heuristic search for augmenting paths. Technical Report 439, Technische Universität Berlin, Fachbereich 3 (1995)
129. Moser, H.: Finding optimal solutions for covering and matching problems. Ph.D. thesis, Friedrich-Schiller-Universität Jena (2010). http://d-nb.info/999819399
130. Nagamochi, H., Ibaraki, T.: Computing edge-connectivity in multigraphs and capacitated graphs. SIAM J. Discrete Math. **5**(1), 54–66 (1992). https://doi.org/10.1137/0405004
131. Nagamochi, H., Ono, T., Ibaraki, T.: Implementing an efficient minimum capacity cut algorithm. Math. Prog. **67**(1), 325–341 (1994). https://doi.org/10.1007/BF01582226
132. Natanzon, A., Shamir, R., Sharan, R.: A polynomial approximation algorithm for the minimum fill-in problem. SIAM J. Comput. **30**(4), 1067–1079 (2000). https://doi.org/10.1137/S0097539798336073
133. Nemhauser, G., Trotter, L.E., J.: Vertex packings: structural properties and algorithms. Math. Prog. **8**(1), 232–248 (1975). https://doi.org/10.1007/BF01580444
134. Niedermeier, R., Rossmanith, P.: An efficient fixed-parameter algorithm for 3-hitting set. J. Discrete Algor. **1**(1), 89–102 (2003). https://doi.org/10.1016/S1570-8667(03)00009-1
135. Bastos, L., Ochi, L.S., Protti, F., Subramanian, A., Martins, I.C., Pinheiro, R.G.S.: Efficient algorithms for cluster editing. J. Comb. Optim. **31**(1), 347–371 (2014). https://doi.org/10.1007/s10878-014-9756-7
136. Olesen, K.G., Madsen, A.L.: Maximal prime subgraph decomposition of Bayesian networks. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **32**(1), 21–31 (2002). https://doi.org/10.1109/3477.979956

137 SPP. 1 Ost, W., Schulz, C., Strash, D.: Engineering data reduction for nested dissection. In: Proceedings of ALENEX 2021, pp. 113–127 (2021). https://doi.org/10.1137/1.9781611976472.9

138. Padberg, M., Rinaldi, G.: An efficient algorithm for the minimum capacity cut problem. Math. Prog. **47**(1), 19–36 (1990). https://doi.org/10.1007/BF01580850

139. Panagiotas, I., Uçar, B.: Engineering fast almost optimal algorithms for bipartite graph matching: Extended version. Research Report RR-9321, Inria Research Centre Grenoble, Rhône-Alpes (2020). https://hal.inria.fr/hal-02463717

140. Pelofske, E., Hahn, G., Djidjev, H.: Solving large minimum vertex cover problems on a quantum annealer. In: Proceedings of CF 2019, pp. 76–84 (2019). https://doi.org/10.1145/3310273.3321562

141. Polzin, T.: Algorithms for the Steiner problem in networks. Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany (2003). http://scidok.sulb.uni-saarland.de/volltexte/2004/218/index.html

142. Pothen, A.: The complexity of optimal elimination trees. Technical report, Pennsylvania State University, Department of Computer Science (1988). https://www.cs.purdue.edu/homes/apothen/Papers/shortest-etree1988.pdf

143. Rehfeldt, D., Koch, T.: SCIP-Jack - a solver for STP and variants with parallelization extensions: an update. In: Proceedings of OR 2017, pp. 191–196 (2017). https://doi.org/10.1007/978-3-319-89920-6_27

144. Rehfeldt, D., Koch, T., Maher, S.J.: Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem. Networks **73**(2), 206–233 (2019). https://doi.org/10.1002/net.21857

145. Reidl, F., Rossmanith, P., Villaamil, F.S., Sikdar, S.: A faster parameterized algorithm for treedepth. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 931–942. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_77

146. Robertson, N., Seymour, P.: Graph minors. II. Algorithmic aspects of tree-width. J. Algor. **7**(3), 309–322 (1986). https://doi.org/10.1016/0196-6774(86)90023-4

147. Rose, D.J.: Triangulated graphs and the elimination process. J. Math. Anal. Appl. **32**(3), 597–609 (1970). https://doi.org/10.1016/0022-247X(70)90282-9

148. Sanders, P., Schulz, C.: KaHIP v3.00 - Karlsruhe High Quality Partitioning - User Guide. Technical report (2013). https://arxiv.org/abs/1311.1714

149. Schäffer, A.A.: Optimal node ranking of trees in linear time. Inf. Proc. Lett. **33**(2), 91–96 (1989). https://doi.org/10.1016/0020-0190(89)90161-0

150 SPP. Schulz, C.: Scalable Graph Algorithms. Habilitation (2019). http://arxiv.org/abs/1912.00245

151. Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. J. Math. Sociol. **6**(1), 139–154 (1978). https://doi.org/10.1080/0022250X.1978.9989883

152. Shinano, Y., Rehfeldt, D., Koch, T.: Building optimal steiner trees on supercomputers by using up to 43,000 cores. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 529–539. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_35

153. Stallmann, M.F., Ho, Y., Goodrich, T.D.: Graph profiling for vertex cover: targeted reductions in a branch and reduce solver. Technical report (2020). https://arxiv.org/abs/2003.06639

154. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM **44**(4), 585–591 (1997). https://doi.org/10.1145/263867.263872

155. Strash, D.: On the power of simple reductions for the maximum independent set problem. In: Dinh, T.N., Thai, M.T. (eds.) Proccedings of COCOON 2016. LNCS, vol. 9797, pp. 345–356. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42634-1_28

156. Tamaki, H.: Positive-instance driven dynamic programming for treewidth. In: Proceedings of ESA 2017, LIPI, vol. 87, pp. 68:1–68:13 (2017). https://doi.org/10.4230/LIPIcs.ESA.2017.68
157. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. **13**(3), 566–579 (1984). https://doi.org/10.1137/0213035
158. Tarjan, R.E., Yannakakis, M.: Addendum: simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. **14**(1), 254–255 (1985). https://doi.org/10.1137/0214020
159. Tinney, W.F., Walker, J.W.: Direct solutions of sparse network equations by optimally ordered triangular factorization. Proc. IEEE **55**(11), 1801–1809 (1967). https://doi.org/10.1109/PROC.1967.6011
160. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. **37**(1), 95–111 (2007). https://doi.org/10.1007/s10898-006-9039-7
161. Trimble, J.: An algorithm for the exact treedepth problem. In: Proceedings of SEA 2020, LIPI, vol. 160, pp. 19:1–19:14 (2020). https://doi.org/10.4230/LIPIcs.SEA.2020.19
162. Bevern, R.: Towards optimal and expressive kernelization for d-hitting set. Algorithmica **70**(1), 129–147 (2013). https://doi.org/10.1007/s00453-013-9774-3
163. van Bevern, R., Smirnov, P.V.: Optimal-size problem kernels for d-hitting set in linear time and space. Inf. Process. Lett. **163**, 105998 (2020). https://doi.org/10.1016/j.ipl.2020.105998
164. Verma, A., Buchanan, A., Butenko, S.: Solving the maximum clique and vertex coloring problems on very large sparse networks. INFORMS J. Comput. **27**(1), 164–177 (2015). https://doi.org/10.1287/ijoc.2014.0618
165. Wang, L., Li, C.M., Zhou, J., Jin, B., Yin, M.: An exact algorithm for minimum weight vertex cover problem in large graphs. Technical report (2019). https://urldefense.com/v3/__https://www.mdpi.com/2227-7390/7/7/603__;!!NLFGqXoFfo8MMQ!ryv0VjrmlwLawl0j6PQDtgV3XzU7mM4U8uFD6oX3d4bPcT9yMMYD958fi7tNg1IaVc81OzW7E7AEb5NnCFGAplRjt2vxhvOs
166. Weihe, K.: Covering trains by stations or the power of data reduction. In: Proceedings of ALEX 1998, pp. 1–8 (1998)
167. Xiao, M.: Simple and improved parameterized algorithms for multiterminal cuts. Theory Comput. Syst. **46**(4), 723–736 (2010). https://doi.org/10.1007/s00224-009-9215-5
168. Xiao, M., Lin, W., Dai, Y., Zeng, Y.: A fast algorithm to compute maximum k-plexes in social network analysis. In: Proceedings of AAAI 2017, pp. 919–925 (2017)
169. Xiao, M., Nagamochi, H.: Confining sets and avoiding bottleneck cases: a simple maximum independent set algorithm in degree-3 graphs. Theor. Comput. Sci. **469**, 92–104 (2013). https://doi.org/10.1016/j.tcs.2012.09.022
170. Xiao, M., Nagamochi, H.: Exact algorithms for maximum independent set. Inf. Comput. **255**, 126–146 (2017). https://doi.org/10.1016/j.ic.2017.06.001
171. Yannakakis, M.: Computing the minimum fill-in is NP-complete. SIAM J. Algeb. Discrete Meth. **2**(1), 77–79 (1981). https://doi.org/10.1137/0602010

172. Zheng, W., Gu, J., Peng, P., Yu, J.X.: Efficient weighted independent set computation over large graphs. In: Proceedings of ICDE 2020, pp. 1970–1973 (2020). https://doi.org/10.1109/ICDE48307.2020.00216
173. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput. **3**(1), 103–128 (2007). https://doi.org/10.4086/toc.2007.v003a006