

Chapter 1

Fixed-Parameter Algorithms for Graph-Modeled Data Clustering

Falk Hüffner*, Rolf Niedermeier, and Sebastian Wernicke†

*Institut für Informatik, Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{hueffner, niedermr, wernicke}@minet.uni-jena.de
Web: <http://theinf1.informatik.uni-jena.de/>*

Fixed-parameter algorithms can efficiently find optimal solutions to some NP-hard problems, including several problems that arise in graph-modeled data clustering. This survey provides a primer about practical techniques to develop such algorithms; in particular, we discuss the design of *kernelizations* (data reductions with provable performance guarantees) and *depth-bounded search trees*. Our investigations are circumstantiated by three concrete problems from the realm of graph-modeled data clustering for which fixed-parameter algorithms have been implemented and experimentally evaluated, namely CLIQUE, CLUSTER EDITING, and CLIQUE COVER.

1.1. Introduction

The central idea behind graph-modeled data clustering is to depict the similarity between a set of entities as a graph: Each vertex represents an entity—such as a gene or protein—and two vertices are connected by an edge if the entities that they represent have some (context-specific) similarity; for instance, two genes have a similar expression profile or two proteins have a high sequence similarity. Groups of highly connected vertices in the resulting graph represent clusters of mutually similar entities. Hence, detecting these dense groups can identify clusters in the graph-encoded data.

Graph-modeled data clustering has been shown to have useful applications in many areas of bioinformatics, including the analysis of gene expres-

*Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

†Supported by the Deutsche Telekom Stiftung.

sion,^{1–4} proteins,^{5,6} gene networks,⁷ allergic reactions,⁸ and marine ecosystems.⁹ There is a catch, however: Most problems that are concerned with the detection of cluster structures in a graph are known to be NP-hard, that is, there is probably no algorithm that can solve all instances efficiently.¹⁰ Thus, whenever such a problem is encountered and large instances need to be solved, it is common to employ heuristic algorithms,¹¹ approximation algorithms,^{12,13} or similar techniques. These usually come with some disadvantages: The solutions are not guaranteed to be optimal or there are no useful guarantees concerning the running time of the algorithm. Further, approximation algorithms and—to some extent—heuristic algorithms are not suited to cope with enumerative tasks. There are many scenarios where these disadvantages seem too severe, that is, where we need to solve a combinatorially hard problem both optimally and yet at the same time somewhat efficiently. For some combinatorial problems, this can be achieved by means of *fixed-parameter algorithms*.^{14–16} These are based on the observation that not all instances of an NP-hard problem are equally hard to solve; rather, this hardness depends on the particular *structure* of a given instance. Opposed to “classical” computational complexity theory—which sees problem instances only in terms of their size—fixed-parameter algorithms and the underlying theory of *fixed-parameter tractability (FPT)* reflect such differences in structural hardness by expressing them through a so-called *parameter*, which is usually a nonnegative integer variable denoted k .

Whenever the parameter k turns out to be small, fixed-parameter algorithms may solve an NP-hard problem quite fast (sometimes even in linear time)—with provable bounds on the running time and guaranteeing the optimality of the solution that is obtained. More precisely, a size- n instance of a fixed-parameter tractable problem can be solved in $f(k) \cdot p(n)$ time, where f is a function solely depending on k , and $p(n)$ is a polynomial in n .

The purpose of this survey is twofold: First, we provide a primer about some important and practically relevant techniques for the design of fixed-parameter algorithms in the realm of graph-modeled data clustering (Section 1.2); in particular, Section 1.2.1 exhibits *kernelizations* (data reductions with provable performance guarantees) and Section 1.2.2 discusses *depth-bounded search trees*. Second, we present three concrete case studies from the realm of graph-modeled data clustering where fixed-parameter algorithms have been devised, implemented, and successfully tested:

- CLIQUE (Section 1.3.1). Using techniques that were originally de-

veloped for the fixed-parameter tractable VERTEX COVER problem, it is possible to detect a size- $(n - k)$ clique in an n -vertex and m -edge graph in $O(1.3^k + kn + m)$ time.^{17,18} So-called *k-isolated cliques*, that is, i -vertex cliques that have less than $k \cdot i$ edges to vertices that lie outside of them, can be exhaustively enumerated in $O(4^k \cdot k^2 m)$ time.^{19–21}

- CLUSTER EDITING (Section 1.3.2). In this problem, the assumption is that the input graph has an underlying cluster structure that is a disjoint union of cliques, which has been distorted by adding and removing at most k edges. For an n -vertex input graph, this underlying cluster structure can be found in $O(1.92^k + n + m)$ time.^{22–24}
- CLIQUE COVER (Section 1.3.3). The assumed underlying cluster structure in this problem is an overlapping union of cliques, that is, the task is to cover the edges of a given graph with a minimum number of cliques. Fixed-parameter algorithms allow for optimal problem solutions within a running time that is competitive with common heuristics.^{25,26}

Practical experiments that we discuss in the case studies suggest that the presented fixed-parameter algorithms are capable of solving many real-world instances in reasonable time. In particular, they perform much better on real-world data than the provable worst-case bounds suggest. Thus, for some NP-hard clustering problems, fixed-parameter tractability theory offers algorithms which are both efficient and capable of delivering optimal solutions. It should hence be part of the algorithmic toolkit for coping with graph-based clustering problems.

We conclude our survey with advice on employing fixed-parameter algorithms in practice (Section 1.4.1) and with a list of specific challenges for future research (Section 1.4.2).

1.2. Fixed-Parameter Tractability Basics and Techniques

In this section, we introduce the basics of fixed-parameter tractability, in particular exhibiting two techniques that are of major practical importance^a and have by now facilitated many success stories in bioinformatics, namely

- kernelizations, that is, data reductions with provable performance

^aA broader view on fixed-parameter algorithm design techniques can be found in Ref. 16.

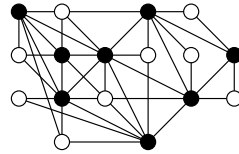


Fig. 1.1. A graph with a size-8 vertex cover (cover vertices are marked black, the solution size is optimal).

- guarantees (Section 1.2.1) and
- depth-bounded search trees (Section 1.2.2).

Both techniques are introduced by means of a single natural and easy to grasp problem, namely the NP-hard VERTEX COVER problem.

VERTEX COVER

INPUT: An undirected graph $G = (V, E)$ and a nonnegative integer k .

TASK: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C .

This problem is illustrated in Figure 1.1 and is—among many other applications—of central importance to practically solving the CLIQUE problem that we discuss in Section 1.3.1.^b

Throughout this work, we assume basic knowledge from algorithmics^{28,29} and graph theory.^{30,31} For a given undirected graph $G = (V, E)$, we always use n to denote the number of its vertices and m to denote the number of its edges. For $v \in V$, we use $N_G(v)$ to denote the neighbor set $\{v \in V \mid \{u, v\} \in E\}$ and $N_G[v]$ to denote the closed neighborhood $N_G(v) \cup \{v\}$, omitting the indices whenever they are clear from the context.

The core approach of fixed-parameter tractability^{14–16} is to consider *parameterized problems*—that is, problems that consist of the instance I and a parameter k —and ask whether there is an algorithm that confines the combinatorial explosion that is involved in solving the problem to the parameter.

^bVERTEX COVER is the *Drosophila* of fixed-parameter research in that many initial discoveries that influenced the whole field originated from studies of this single problem (e.g., see Guo et al.²⁷).

Definition 1.1. An instance of a parameterized problem consists of a problem instance I and a parameter k . A parameterized problem is *fixed-parameter tractable* if it can be solved in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function solely depending on the parameter k , and not on the input size $|I|$.

For NP-hard problems, $f(k)$ will of course not be polynomial, since otherwise we would have an overall polynomial-time algorithm.

As parameterized complexity theory points out, there are problems that are likely not to be fixed-parameter tractable.^{14–16} It is important to note in this respect that a problem can have *various* parameterizations such as the size of the solution that is sought after or some structural parameter that characterizes the input. A problem that is not fixed-parameter tractable with respect to some parameter may still be so with respect to others. Also, the choice of the parameter can greatly affect the efficiency of the algorithm that is obtained.

Besides the classic reference,¹⁴ two new monographs are available on parameterized complexity, one focusing on theoretical foundations¹⁵ and one focusing on techniques and algorithms.¹⁶

1.2.1. Kernelizations

Before firing up a computationally expensive algorithm to solve a combinatorially hard problem, one should always try to perform a *reduction* on the input data, the idea being to quickly presolve those parts of the input data that are relatively easy to cope with and thus to shrink the input to those parts that form the “really hard” core of the problem. Costly algorithms need then only be applied to the reduced instance. In some practical scenarios, data reduction may even reduce a seemingly hard problem to triviality.^{25,32,33}

Clearly, practitioners are likely to already be aware of data reduction rules. The reason why they should also consider fixed-parameter tractability in this context is that fixed-parameter theory provides a way to use data reduction rules not only in a heuristic way, but to *prove their power* by so-called *kernelizations*. These run in polynomial time and give an upper bound on the size of a reduced instance that solely depends on the parameter value, that is, they come with a performance guarantee both concerning their running time as well as their effectiveness. Having a quantitative measure for the performance of a data reduction can moreover help to guide the search for further improved data reductions in a constructive

way.³⁴

1.2.1.1. An Introductory Example

Consider our running example VERTEX COVER. To reduce the input size for a given instance of this problem, it is clearly permissible to remove isolated vertices, that is, vertices with no adjacent edges. This leads to a first simple data reduction rule.

REDUCTION RULE VC1. Remove all isolated vertices.

In order to cover an edge in the graph, one of its two endpoints *must* be in the vertex cover. If one of these is a degree-1 vertex, then the other endpoint has the potential to cover more edges than the degree-1 vertex, leading to a second reduction rule.

REDUCTION RULE VC2. For degree-1 vertices, put their neighboring vertex into the cover.^c

Note that this reduction rule assumes that we are only looking for *one* optimal solution to the VERTEX COVER instance we are trying to solve; there may exist other minimum vertex covers that do include the reduced degree-1 vertex.

After having applied the easy rules VC1 and VC2, we can further do the following in the fixed-parameter setting where we ask for a vertex cover of size at most k .

REDUCTION RULE VC3. If there is a vertex v of degree at least $k + 1$, put v into the cover.

The reason this rule is correct is that if we did not take v into the cover, then we would have to take every single one of its $k + 1$ neighbors into the cover in order to cover all edges adjacent to v . This is not possible because the maximum allowed size of the cover is k .

After exhaustively performing the rules VC1–VC3, no vertex in the remaining graph has a degree higher than k , meaning that choosing a vertex into the cover can cause at most k edges to become covered. Since the solution set may be no larger than k , the remaining graph can have at most k^2 edges if it is to have a solution. By rules VC1 and VC2, every

^c“Put into the cover” means adding the vertex to the solution set and removing it and its incident edges from the instance.

vertex has degree at least two, which implies that the remaining graph can contain at most k^2 vertices.

1.2.1.2. The Kernelization Concept

Abstractly speaking, what have we done in the previous section? After applying a number of rules *in polynomial time* to an instance of VERTEX COVER, we arrived at a reduced instance whose size can *solely be expressed in terms of the parameter k* . Since this can be easily done in $O(n)$ time, we have found a data reduction for VERTEX COVER with guarantees concerning its running time as well as its effectiveness. These properties are formalized in the concepts of a *problem kernel* and the corresponding *kernelization*.¹⁴

Definition 1.2. Let \mathcal{L} be a parameterized problem, that is, \mathcal{L} consists of input pairs (I, k) , where I is the problem instance and k is the parameter. A *reduction to a problem kernel* (or *kernelization*) means to replace an instance (I, k) by a *reduced* instance (I', k') called *problem kernel* in polynomial time such that

- (1) $k' \leq k$,
- (2) I' is smaller than $g(k)$ for some function g only depending on k ,
and
- (3) (I, k) has a solution if and only if (I', k') has one.

While this definition does not formally require that it is possible to reconstruct a solution for the original instance from a solution for the problem kernel, all kernelizations we are aware of easily allow for this.

The methodological approach of kernelization, including various techniques of data reduction, is best learned by the concrete examples that we discuss in Section 1.3; there, we will also discuss kernelizations for VERTEX COVER that even yield a kernel with a *linear* number of vertices in k .

To conclude this section, we state some useful general observations and remarks concerning Definition 1.2 and its connections to fixed-parameter tractability. Most notably, there is a close connection between fixed-parameter tractable problems and those problems that have a problem kernel—they are exactly the same.

Theorem 1.3 (Cai et al.³⁵). *Every fixed-parameter tractable problem is kernelizable and vice-versa.* \square

Unfortunately, the practical use of this theorem is limited: the running times of a fixed-parameter algorithm directly obtained from a kernelization is usually not practical; and, in the other direction, the theorem does not constructively provide us with a data reduction scheme for a fixed-parameter tractable problem. Hence, the main use of Theorem 1.3 is to establish the fixed-parameter tractability or amenability to kernelization of a problem—or show that we need not search any further (e.g., if a problem is known to be fixed-parameter intractable, we do not need to look for a kernelization).

Rule VC3 explicitly needed the value of the parameter k . We call this a *parameter-dependent* rule as opposed to the parameter-independent rules VC1 and VC2, which are oblivious to k . Of course, one typically does not know the actual value of k in advance and then has to get around this by iteratively trying different values of k .^d While, in practice, one would naturally prefer to avoid this extra outer loop, assuming explicit knowledge of the parameter clearly adds some leverage to finding data reduction rules and is hence frequently encountered in kernelizations.

1.2.2. *Depth-Bounded Search Trees*

After preprocessing the given input data of a problem by a kernelization and cutting away its “easy parts,” we are left with the “really hard” problem kernel to be solved. A standard way to explore the huge search space of a computationally hard problem is to perform a systematic exhaustive search. This can be organized in a tree-like fashion, which is the main subject of this section.

Certainly, search trees are no new idea and have been extensively used in the design of exact algorithms (e.g., see Ref. 37–41). The main contribution of fixed-parameter theory to search tree approaches is the consideration of search trees whose depth is bounded by the parameter, usually leading to search trees that are much smaller than those of naïve brute-force searches. Additionally, the speed of search tree exploration can (provably) be improved by exploiting kernelizations.⁴²

An extremely simple search tree approach for solving VERTEX COVER is to just take one vertex and branch into two cases: either this vertex is in the vertex cover or not. This leads to a search tree of size $O(2^n)$. As we outline

^dIn general, the constraint $k < n$ is easily established. As Dehne et al.³⁶ point out in their studies of CLUSTER EDITING, it depends on the concrete problem which search strategy for the “optimum” value of k is most efficient to employ in practice.

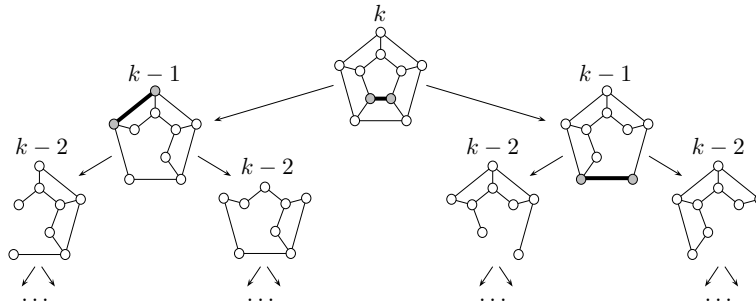


Fig. 1.2. Simple search tree for finding a vertex cover of size at most k in a given graph. The size of the tree is upper-bounded by $O(2^k)$.

in this section, we can do much better than that and obtain a search tree whose depth is upper-bounded by k , giving a size bound of $O(2^k)$. Since usually $k \ll n$, this can draw the problem into the zone of feasibility even for large graphs (as long as k is small).

The basic idea is to find a small subset of the input instance in polynomial time such that at least one element of this subset *must* be part of an optimal solution to the problem. In the case of VERTEX COVER, the most simple such subset is any two vertices that are connected by an edge. By definition of the problem, one of these two vertices *must* be part of a solution. Thus, a simple search-tree algorithm to solve VERTEX COVER on a graph G proceeds by picking an arbitrary edge $e = \{v, w\}$ and recursively searching for a vertex cover of size $k - 1$ both in $G - v$ and $G - w$.^e That is, the algorithm *branches* into two subcases knowing one of them must lead to a solution of size at most k —if one such solution exists.

As shown in Figure 1.2, these recursive calls of the simple VERTEX COVER algorithm can be visualized as a tree structure. Because the depth of the recursion is upper-bounded by the parameter value and we always branch into two subcases, the size of this tree is upper-bounded by $O(2^k)$. This means that the size of the tree is *independent of the size of the initial input instance* and only depends on the value of the parameter k .

The main idea behind fixed-parameter algorithmics is to get the combinatorial explosion as small as possible. For our VERTEX COVER example, one can easily achieve a size- $o(2^k)$ search tree by distinguishing more detailed branching cases rather than just picking single endpoints of edges

^eFor a vertex $v \in V$, we define $G - v$ to be the graph G with v and the edges incident to v removed.

to be in the cover.^f An example for such an “improved” search-tree is given in our case study of CLUSTER EDITING in Section 1.3.2. The currently “best” search trees for VERTEX COVER are of size $O(1.28^k)^{18}$ and mainly achieved by extensive case distinguishing. However, it should be noted for practical applications that it is always concrete implementation and testing that has to decide whether the administrative overhead caused by distinguishing more and more cases pays off. A simpler algorithm with slightly worse bounds on the search tree size often turns out to be preferable in practice. Here, recent progress with the analysis of search tree algorithms using multivariate recurrences⁴⁴ might help: with this method, it was shown that some simple algorithms perform in fact much better than previously proved.³⁹ Also, new algorithms were developed guided by the new analysis methods;³⁹ however, there is no practical experience yet with these approaches.

In combination with data reduction (see Section 1.3.1), the use of depth-bounded search trees has proven itself quite useful in practice, allowing to find vertex covers of more than ten thousand vertices in some dense graphs of biological origin.⁴⁵ Search trees also trivially allow for a parallel implementation: when branching into subcases, each processor in a parallel setting can further explore one of these branches with no additional communication required. Cheetham et al.⁴⁶ expose this in their parallel VERTEX COVER solver to achieve a near-optimum (i.e., linear with the number of processors employed) speedup on multiprocessor systems. Finally, it is generally beneficial to augment search tree algorithms with admissible heuristic evaluation functions in order to further increase their performance and memory efficiency by cutting away search tree parts that cannot lead to good solutions.^{47,48}

1.3. Case Studies From Graph-Modeled Data Clustering

This section surveys fixed-parameter algorithms and experimental results for three important NP-complete problems from the realm of graph-modeled data clustering, namely CLIQUE, CLUSTER EDITING, and CLIQUE COVER. The purpose of these case studies is twofold: First, they serve to

^fNote that analogously to the case of data reduction, most of these branchings assume that only *one* minimum solution is sought after. Since some graphs can have 2^k minimum vertex covers, a size- $o(2^k)$ search tree for enumerating *all* minimum vertex covers requires the use of *compact solution representations* as outlined by Damaschke⁴³ and is beyond the scope of this work.

teach in more detail the methodological approaches of designing kernelizations and depth-bounded search trees. Second, the encouraging experimental results that are known for these problems underpin the general usefulness of fixed-parameter algorithms for optimally solving NP-hard problems in graph-modeled data clustering.

1.3.1. *Clique*

A “classical” combinatorial problem that is closely connected to graph-modeled data clustering is to find a clique in a graph, that is, a subset of vertices that are fully connected.

CLIQUE

INPUT: An undirected graph $G = (V, E)$ and a nonnegative integer k .

TASK: Find a k -vertex clique in G .

It is also a common task to *enumerate* maximal cliques in a graph, that is, all cliques that are not a proper subset of any other clique.

CLIQUE is NP-hard¹⁰ and hard to approximate in polynomial time.⁴⁹ In a similar sense as it is generally assumed that $P \neq NP$, it is strongly believed that CLIQUE is not fixed-parameter tractable when parameterized by the size of the cliques that are sought after.¹⁴ Nevertheless, CLIQUE has a close connection to the fixed-parameter tractable VERTEX COVER problem that we used as our running example to introduce fixed-parameter techniques: If an n -vertex graph contains a size- k clique, then its *complement graph*[§] contains a size- $(n - k)$ vertex cover and vice versa. This can be made use of when seeking after or enumerating cliques.

1.3.1.1. *Finding Maximum Cardinality Cliques*

The catch when solving CLIQUE for a graph G by means of finding a minimum-cardinality vertex cover for the complement graph G' is that if the maximum size k of a clique in G is rather small compared to its total number of vertices n , then G' will have a rather large minimum-size vertex cover. Therefore, one has to rely on effective data reduction rules that preprocess the complement graph G' so that depth-bounded search tree algorithms become practically applicable for the reduced graph that remains.

[§]That is, the graph that contains exactly those edges that are not contained in the original graph.

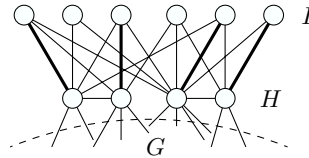


Fig. 1.3. A graph G with a crown $I \cup H$. The thick edges constitute a maximum matching of size $|H|$ in the bipartite graph that is induced by the edges between I and H .

One kernelization for VERTEX COVER that has proven itself to be of particular practical importance in this respect is the so-called *crown reduction*,⁵⁰ which generalizes the VERTEX COVER data reduction rule VC2 (the elimination of degree-1 vertices by taking their neighbors into the cover) and thus leads to a data reduction that requires no explicit knowledge of the parameter k and yields a kernel with a number of vertices *linear* in k .

A *crown* in a graph consists of an independent set I (that is, no two vertices in I are connected by an edge) and a set H containing all vertices adjacent to I . In order for $I \cup H$ to be a crown, there has to be a size- $|H|$ matching in the bipartite graph induced by the edges between I and H (i.e., one in which every vertex of H is matched). An example for a crown structure is given in Figure 1.3. If there is a crown $I \cup H$ in the input graph G , then we need *at least* $|H|$ vertices to cover all edges in the crown. But since all edges in the crown can be covered by taking *at most* $|H|$ vertices into the cover (as I is an independent set), there is a minimum-size vertex cover for G that contains all the vertices in H and none of the vertices in I . We may thus delete any given crown $I \cup H$ from G , reducing k by $|H|$.

It turns out that finding crowns can be achieved in polynomial time by computing maximum matchings.⁵¹ The size of the thus reduced instance is upper-bounded via the following theorem.

Theorem 1.4 (Abu-Khazam et al.⁵⁰). *A graph that is crown-free and has a vertex cover of size at most k can contain at most $3k$ vertices.* \square

There are several kernelizations for VERTEX COVER that achieve a kernel of $O(k)$ vertices; some of these even yield an at-most- $2k$ -vertex kernel, e.g., see Ref. 16,50. However, it has been found that crown reductions often offer a good balance between the polynomial time that is required to compute the kernel and the size that the reduced graphs usually turn out

to have in practice.^{45,50,52}

Some quite successful implementations for solving CLIQUE rely on kernelization techniques (especially crown reductions) for VERTEX COVER that are combined with depth-bounded search trees.^{50,52,53} The exploration of the search trees is usually highly optimized, for instance, by using efficient data structures and ensuring proper load balancing in parallel scenarios; details of these techniques are described, e.g., by Abu-Khzam et al.⁵⁰ and Zhang et al.⁵³ Even attempts to implement parts of the algorithms in hardware have been reported.⁴⁵

With the combination of kernelizations and depth-bounded search trees, it is currently possible to find cliques that consist of over 400 vertices in some dense graphs of biological origin within hours.⁴⁵

1.3.1.2. *Enumerating Maximal Cliques*

Instead of finding a single maximum-size clique in a graph, one would often like to enumerate all cliques of maximal size. In graph-modeled data clustering, this can have mainly two reasons: First, an enumeration obviously identifies *all* clusters that are present in the data. Second, with an enumerative solution one can include expert knowledge as to what cliques that are present in the input graph are (biologically) “meaningful.”

Analogously to the task of finding a maximum-size clique, the task of enumerating maximal cliques in a graph is equivalent to enumerating minimal vertex covers for its complement graph. To enumerate all maximal cliques in a graph, one can therefore rely on kernelizations for VERTEX COVER that are suited for finding *all* vertex covers up to a certain size⁵⁴ and on enumerative depth-bounded search trees such as discussed by Damaschke.⁴³ However, so far there is no empirical evidence of the practical viability of this approach.

An interesting result concerning the enumeration of maximal cliques that makes a more “indirect” use of VERTEX COVER was recently shown by Ito et al.¹⁹ It is based on an *alternative parameterization* other than the clique size. This parameterization is based on the observation that the hardness of finding a large clique in a graph is determined by the *isolation* of that clique, meaning that if we restrict ourselves to finding cliques that have only a few edges to “external” vertices outside of the clique, then this is a much easier task compared to finding cliques that have many edges to external vertices. The intuitive reason for this is that isolated cliques are better distinguishable from the remaining graph. To quantify the isolation

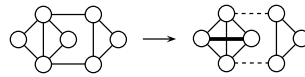


Fig. 1.4. Illustration for the CLUSTER EDITING problem: By removing two edges from and adding one edge to the graph on the left (that is, $k = 3$), we can obtain a graph that consists of two disjoint cliques.

of a clique, Ito et al.¹⁹ introduced the notion of an *isolation factor* k . An i -vertex clique is said to be k -isolated if it has less than $k \cdot i$ edges to external vertices. It turns out that enumerating all k -isolated cliques is fixed-parameter tractable with respect to k . Komusiewicz et al.^{20,21} pointed out an error in the algorithm and provided a corrected version.

Theorem 1.5 (Ito et al.,¹⁹ Komusiewicz et al.^{20,21}). *All k -isolated cliques in an m -edge graph can be enumerated in $O(4^k \cdot k^2 m)$ time.*

The underlying algorithm of this result is based on a search tree for VERTEX COVER; the main achievement lies in showing that the isolation factor k can also serve as a bound for the depth of this search tree, which is achieved by a parameter-dependent data reduction. This nicely demonstrates the benefit of alternative parameterizations for a problem. Ito et al.¹⁹ mentioned that some preliminary experiments suggest that the detection of isolated cliques is quite efficient in practice.

1.3.2. Cluster Editing

The CLUSTER EDITING problem is based on the assumption that the input graph is a disjoint union of cliques—a so-called *cluster graph*—that has been perturbed by adding or removing edges. CLUSTER EDITING appears as an important problem in the analysis of data from synthetic genetic arrays.^{1,36}

CLUSTER EDITING

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Task: Modify G to consist of disjoint cliques by adding or deleting at most k edges.

Figure 1.4 illustrates this problem. CLUSTER EDITING is NP-hard,^{3,55} and its minimization version can be approximated in polynomial time

within a factor of 4.⁵⁶ A randomized expected factor-3 approximation algorithm was given by Ailon et al.⁵⁷ CLUSTER EDITING is also a special case of the CORRELATION CLUSTERING problem occurring in machine learning.⁵⁸

In what follows, we concentrate on a search tree-based fixed-parameter approach towards exactly solving CLUSTER EDITING. Here, the overall strategy is based on an easy-to-see observation, namely that a cluster graph has a very special structure: If two vertices are connected by an edge, then their neighborhoods must be the same. Hence, whenever we encounter two connected vertices u and v in the input graph G that are connected by an edge and where one vertex, say u , has a neighbor w that is not connected to v , we call $\{u, v, w\}$ a *conflict triple* of vertices because it compels us to do one of three things: Either remove the edge $\{u, v\}$, or connect v with w , or remove the edge $\{u, w\}$. Each of these three modifications counts with respect to the parameter k and, therefore, exhaustively branching into these cases for at most k forbidden substructures, we obtain a search tree of size $O(3^k)$ to solve CLUSTER EDITING.

The search tree size can be significantly reduced. More specifically, a more sophisticated branching strategy gives a search tree size of $O(2.27^k)$,²² which—using computer-generated branching rules—has been further improved to a size of $O(1.92^k)$.²³ The computer-generated result, however, is based on a quite complicated branching with lots of case distinctions that might not be of practical value. In the following, we describe the key observations and fundamental ideas behind the improved search trees for CLUSTER EDITING. As a remark, there also exist size- $o(3^k)$ search trees for *enumerating* all solutions to a given instance of CLUSTER EDITING.⁵⁹

The basic approach to obtain the improved search trees for CLUSTER EDITING is to do a case distinction, where we provide for every possible situation additional branching steps. The analysis of successive branching steps, then, yields a better worst-case bound on the search tree size. To this end, we make use of two annotations for unordered vertex pairs:

- “*permanent*”: In this case, $\{u, v\} \in E$ and it is not allowed to delete $\{u, v\}$;
- “*forbidden*”: In this case, $\{u, v\} \notin E$ and it is not allowed to add $\{u, v\}$.

Clearly, if an edge $\{u, v\}$ is deleted, then the vertex pair is made *forbidden*. If an edge $\{u, v\}$ is added, then the vertex pair is made *permanent*.

We distinguish three main situations that may apply when considering the conflict triple $\{u, v, w\}$:

- (C1) Vertices v and w do not share a common neighbor, that is, $\forall x \in$

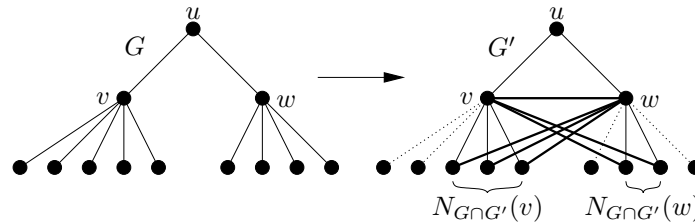


Fig. 1.5. In case (C1), adding the edge $\{v, w\}$ does not need to be considered. Here, G is the given graph and G' is a clustering solution of G that adds the edge $\{v, w\}$. The dashed lines denote edges being deleted to transform G into G' , and the bold lines denote edges being added. Observe that the drawing only shows that parts of the graphs (in particular, edges) which are relevant for our argument.

$$V, x \neq u : \{v, x\} \notin E \text{ or } \{w, x\} \notin E.$$

(C2) Vertices v and w have a common neighbor $x \neq u$ and $\{u, x\} \in E$.

(C3) Vertices v and w have a common neighbor $x \neq u$ and $\{u, x\} \notin E$.

Regarding case (C1), the following lemma shows that a branching into two subcases suffices.

Lemma 1.6. *Given a graph $G = (V, E)$, a nonnegative integer k , and a conflict triple $u, v, w \in V$ of vertices that satisfy case (C1) from above, adding the edge $\{v, w\}$ cannot yield a better solution than deleting one of the edges $\{u, v\}$ or $\{u, w\}$.*

Proof. Consider a clustering solution G' for G where we did add $\{v, w\}$ (see Figure 1.5 for an example). We use $N_{G \cap G'}(v)$ to denote the set of vertices that are neighbors of v in G and in G' . Without loss of generality, assume that $|N_{G \cap G'}(w)| \leq |N_{G \cap G'}(v)|$. We then construct a new graph G'' from G' by deleting all edges adjacent to w . It is clear that G'' is also a clustering solution for G . We compare the cost of the transformation $G \rightarrow G''$ to that of the transformation $G \rightarrow G'$:

- -1 for not adding $\{v, w\}$,
- $+1$ for deleting $\{u, w\}$,
- $-|N_{G \cap G'}(v)|$ for not adding all edges $\{w, x\}, x \in N_{G \cap G'}(v)$,
- $+|N_{G \cap G'}(w)|$ for deleting all edges $\{w, x\}, x \in N_{G \cap G'}(w)$.

Here, we omitted possible vertices which are neighbors of w in G' but not in G : they would only increase the cost of transformation $G \rightarrow G'$.

In summary, the cost of $G \rightarrow G''$ is not higher than the cost of $G \rightarrow G'$, that is, we do not need more edge additions and deletions to obtain G'' from G than to obtain G' from G . \square

As a consequence of Lemma 1.6, the search tree only has to branch into two instead of three subcases in case (C1). Making use of the markers “permanent” and “forbidden,” the standard branching into three subcases can also be avoided in cases (C2) and (C3). Each of these cases, however, requires specific considerations²² which have to be omitted here.

Besides a search tree strategy for CLUSTER EDITING, also data reductions yielding problem kernels are known for this problem. The first result was a problem kernel with $O(k^2)$ vertices²² and this has recently been improved to a problem kernel with only $O(k)$ vertices.^{60,61} For a practical solving algorithm, the search tree has to be combined with the kernelization.^{22,36,42} By splitting up cases (C2) and (C3) further using a computer, we arrive at the following theorem.

Theorem 1.7 (Gramm et al.,²³ Protti et al.²⁴). CLUSTER EDITING can be solved in $O(1.92^k + n + m)$ time.

For a recent implementation of the above strategy, experiments indicated that the fixed-parameter approach outperforms a solution based on linear programming and appears to be of practical use.³⁶ The implementation can solve certain synthetic instances with $n = 100$ and 40 edit operations within an hour.

There are several problems closely related to CLUSTER EDITING that deserve similar studies. Among these are the more general CORRELATION CLUSTERING problem^{56,58} and the BICLUSTER EDITING problem,^{24,62} the latter also known to be fixed-parameter tractable.²⁴

1.3.3. Clique Cover

The model assumption for CLIQUE COVER is that the data forms *overlapping* cliques, as opposed to the disjoint cliques that were the underlying model for CLUSTER EDITING. In general, this clustering model is applicable whenever various items carry an unknown subset of *features*, and two items will be measured as similar whenever they have at least one feature in common. Then, the set of items that carry one particular feature forms a clique. Therefore, finding a set of cliques that covers all edges gives the most parsimonious explanation of the data under this model: each clique corresponds to one feature. Guillaume and Latapy⁶³ argue that this model

is very widely applicable to discover underlying structure in complex real-world networks.

Formally, the CLIQUE COVER problem is defined as follows:

CLIQUE COVER

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Task: Find a set of at most k cliques in G such that each edge in E has both its endpoints in at least one of the selected cliques.

CLIQUE COVER is NP-hard,⁶⁴ and there is strong evidence that it cannot be approximated to a constant factor.⁶⁵ Therefore, the standard approach to solving CLIQUE COVER in practice so far is to employ heuristics.^{66–69} Behrisch and Taraz⁷⁰ give simple greedy algorithms for CLIQUE COVER that provide asymptotically optimal solutions for certain random intersection graphs. However, because of the fundamental inapproximability of the problem, the results of these algorithms can become nearly arbitrarily bad for particular inputs. This makes fixed-parameter algorithms, which provide optimal solutions, attractive. A natural parameter is the size of the feature set k , since it seems reasonable to assume that there are much fewer features than items.

Gramm et al.²⁵ presented a parameterized approach to CLIQUE COVER, which is based on data reduction. In fact, the fixed-parameter tractability of CLIQUE COVER with respect to k is based on a *problem kernel*. The first two rules that lead to this kernel are easy to see:

REDUCTION RULE CC1. Remove isolated vertices and vertices that are only adjacent to covered edges.

REDUCTION RULE CC2. If there is an edge $\{u, v\}$ whose endpoints have exactly the same closed neighborhood, that is, $N[u] = N[v]$, then delete u . To reconstruct a solution for the unreduced instance, add u to every clique containing v .

Rules CC1 and CC2 together suffice to show the problem kernel (the basic underlying observation was already made by Gyarfas⁷¹).

Theorem 1.8 (Gyarfas⁷¹ and Gramm et al.²⁵). *A CLIQUE COVER instance reduced with respect to Rules CC1 and CC2 contains at most 2^k*

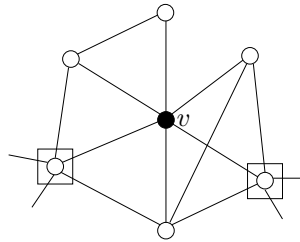


Fig. 1.6. An illustration of the partition of the neighborhood of a vertex v . The two vertices with rectangles are exits, the other white ones are prisoners.

vertices or, otherwise, has no solution.

Proof. Consider a graph $G = (V, E)$ that is reduced with respect to Rules CC1 and CC2 and has a clique cover C_1, \dots, C_k of size k . We assign to each vertex $v \in V$ a binary vector b_v of length k where bit i , $1 \leq i \leq k$, is set iff v is contained in clique C_i . If we assume that G has more than 2^k vertices, then there must be $u \neq v \in V$ with $b_u = b_v$. Since Rule CC1 does not apply, every vertex is contained in at least one clique, and since $b_u = b_v$, u and v are contained in the same cliques. Therefore, u and v are connected. As they also share the same neighborhood, Rule CC2 applies, in contradiction to our assumption that G is reduced with respect to Rule CC2. Consequently, G cannot have more than 2^k vertices. \square

By Theorem 1.3, this implies that CLIQUE COVER is fixed-parameter tractable with respect to parameter k . Unfortunately, the worst-case size of a reduced instance is still exponential, as opposed to the polynomially-sized kernels that are known for VERTEX COVER and CLUSTER EDITING.

As an example of an advanced data reduction rule, we now formulate a generalization of Rule CC2. While one can show that this rule finds a strict superset of the reduction opportunities of Rule CC2, it does not seem possible to use it to improve the worst-case problem kernel size bound in Theorem 1.8. Nevertheless, Rule CC2 improves the running time of solving CLIQUE COVER in practice, as described below.

To discuss the advanced data reduction rule, we need some additional terminology: For a vertex v , we partition the set of vertices that are connected by an edge to v into *prisoners* p with $N(p) \subseteq N(v)$ and *exits* x with $N(x) \setminus N(v) \neq \emptyset$. We say that the prisoners *dominate* the exits if every exit x has an adjacent prisoner. An illustration of the concept of prisoners and exits is given in Figure 1.6.

REDUCTION RULE CC3. Consider a vertex v that has at least one prisoner. If each prisoner is connected to at least one vertex other than v via an edge, and the prisoners dominate the exits, then delete v . To reconstruct a solution for the unreduced instance, add v to every clique containing a prisoner of v .

Lemma 1.9. *Reduction Rule CC3 is correct.*

Proof. By definition, every neighbor of v 's prisoners is also a neighbor of v itself. If a prisoner of v participates in a clique C , then $C \cup \{v\}$ is also a clique in the graph. Therefore, it is correct to add v to every clique containing a prisoner in the reduced graph. Next, we show that all edges adjacent to v are covered by the cliques resulting by adding v to the cliques containing v 's prisoners. We consider separately the edges connecting v to prisoners and edges connecting v to exits. Regarding an edge $\{v, w\}$ to a prisoner w , vertex w has to be part of a clique C of the solution for the instance after application of the rule. Therefore, the edge $\{v, w\}$ is covered by $C \cup \{v\}$ in the solution for the unreduced instance. Regarding an edge $\{v, x\}$ to an exit x , the exit x is dominated by a prisoner w and therefore x has to be part of a clique C with w . Hence, the edge $\{v, x\}$ is covered by $C \cup \{v\}$ in the solution for the unreduced instance. \square

Concerning experimental results, Gramm et al.²⁵ implemented an algorithm to optimally solve CLIQUE COVER that is based on four data reduction rules (CC1–CC3 and one additional rule) and a simple branching strategy. The implementation was able to solve within a few seconds 14 real-world instances from an application in graphical statistics, with up to 124 vertices and more than 2700 edges.²⁶ Further experiments on random graphs showed that in particular sparse instances could be solved quickly. The algorithm relied mainly on the data reduction rules: many instances were reduced to an empty instance before the branching even began. By way of contrast, when the data reduction was not successful, running times increased sharply. Additional experiments for the feature model mentioned at the beginning of this section showed that random instances with 100 items and up to 30 features could be solved within a few minutes.

Two reduction rules of Gramm et al. (Rule CC3 and another rule that we do not discuss here) have so far not been proved to improve the size of the problem kernel and are also quite slow to compute. For some instances, however, both rules were beneficial, meaning that the obtained speedups

for them instances were much larger than observed slowdowns for other instances. This suggests the general application of these rules, possibly with an additional heuristic to disable them based on instance properties.

As a final note, unlike the CLUSTER EDITING model, CLIQUE COVER does not take perturbed data into account. In practice, probably edges within feature clusters are missing, and spurious edges exist. However, this can be handled with a post-processing: as long as there are not too many errors, spurious edges will be covered by size-2 cliques, which can be easily filtered; and a clique missing an edge will be optimally covered by two cliques, and so in a post-processing one can check for all pairs of cliques whether they could be merged by adding a small number of edges.

1.4. Conclusion

We conclude our survey with a list of guidelines for the practical design of fixed-parameter algorithms and some open challenges in the field.

1.4.1. *Practical Guidelines*

The following list sums up the experiences of our and other research groups who have implemented fixed-parameter algorithms:

Fixed-Parameter Tractability in General

- (1) Do not despair of bad upper bounds for fixed-parameter algorithms that are given in theoretically oriented papers—the analysis is worst-case and often much too pessimistic.
- (2) Fixed-parameter algorithms are the better the smaller the parameter value is. Hence, parameterizations with small parameter values should be sought after.
- (3) Most existing fixed-parameter algorithms in the literature are concerned with optimization problems on unweighted graphs. Solving enumerative problems and problems on weighted graphs therefore usually requires some additional work to be done.
- (4) Exponential memory consumption usually turns out to be more harmful in practice than exponential time, especially in enumerative tasks. Using memory-saving techniques generally pays off, even if this means some decrease in time efficiency.

Data Reductions and Kernelizations

- (1) One should always start by designing data reduction rules because these are helpful in combination with basically any algorithmic technique that is subsequently applied.
- (2) The *order* of applying data reduction rules can significantly influence the practical effectiveness and efficiency of the overall data reduction. Experimental work is important to find good orderings.
- (3) Even if a data reduction has no provable performance guarantee, it can still turn out to be very effective in practice.

Depth-Bounded Search Trees

- (1) The branching in a search tree can produce new opportunities for data reduction. Therefore, data reductions should be applied repeatedly during the course of the whole algorithm and not only as a preprocessing step.⁴² To achieve maximum efficiency, the exact frequency of applying data reductions may require some tuning.
- (2) Search tree algorithms can be parallelized rather easily.
- (3) Complicated case distinctions for the branching should be avoided when a simpler search strategy is available that yields almost the same worst-case running time bounds. The simpler strategy usually turns out to be faster.

1.4.2. Challenges

While there has been substantial work on fixed-parameter algorithms for clustering problems and several examples show the potential of this approach, the field is still quite young, and there remain a couple of challenges for future research:

- The CLIQUE model is often too restricted in applications; one would rather prefer a notion of “dense subgraph” (e. g., Ref. 72–74). Except for Ref. 19–21, we are not aware of fixed-parameter approaches for such scenarios.
- For simplicity, many fixed-parameter approaches drop the requirement to be able to handle weighted problems or to handle enumeration. Extensions of known results in this direction are desirable.
- Of our three case studies, CLIQUE COVER seems to be the least explored problem. While kernels with a linear number of vertices are known for VERTEX COVER and CLUSTER EDITING, the only

known kernel for CLIQUE COVER is of exponential size. Also, no search tree with a fixed-parameter bound on its size is known for CLIQUE COVER except for a trivial brute-force exploration of the problem kernel.

- Some works consider the variant of CLUSTER EDITING where there are *don't care*-edges that have zero editing cost.⁵⁸ It is not yet known whether a fixed-parameter algorithm exists for this problem.

A particularly important challenge for future work is to bring progress from fixed-parameter algorithmics to a broader audience by providing easily accessible software tools that are finely tuned by algorithm engineering and additional tools such as heuristics and parallelization.

References

1. A. Ben-Dor, R. Shamir, and Z. Yakhini, Clustering gene expression patterns, *Journal of Computational Biology*. **6**(3–4), 281–297, (1999).
2. E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, D. W. Threadgill, K. F. Manly, and R. W. Williams, Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function, *Nature Genetics*. **37**, 233–242, (2005).
3. R. Shamir, R. Sharan, and D. Tsur, Cluster graph modification problems, *Discrete Applied Mathematics*. **144**(1–2), 173–182, (2004).
4. S. Seno, R. Teramoto, Y. Takenaka, and H. Matsuda, A method for clustering expression data based on graph structure, *Genome Informatics*. **15**(2), 151–160, (2004).
5. H. Kawaji, Y. Takenaka, and H. Matsuda, Graph-based clustering for finding distant relationships in a large set of protein sequences, *Bioinformatics*. **20**(2), 243–252, (2004).
6. H. Kawaji, Y. Yamaguchi, H. Matsuda, and A. Hashimoto, A graph-based clustering method for a large set of sequences using a graph partitioning algorithm, *Genome Informatics*. **12**, 93–102, (2001).
7. B. H. Voy, J. A. Scharff, A. D. Perkins, A. M. Saxton, B. Borate, E. J. Chesler, L. K. Branstetter, and M. A. Langston, Extracting gene networks for low-dose radiation using graph theoretical algorithms, *PLoS Computational Biology*. **2**(7), e89, (2006).
8. M. Benson, M. A. Langston, M. Adner, B. Andersson, Å. Torinsson Naluai, and L. O. Cardell, A network-based analysis of the late-phase reaction of the skin, *The Journal of Allergy and Clinical Immunology*. **118**(1), 220–225, (2006).
9. M. A. Langston, A. D. Perkins, D. J. Beare, R. W. Gauldie, P. J. Kershaw, J. B. Reid, K. Winpenny, and A. J. Kenny. Combinatorial algorithms and high performance implementations for elucidating complex ecosystem rela-

- tionships from North Sea historical data. In *Proc. International Council for the Exploration of the Sea Annual Science Conference*, (2006).
10. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. (W. H. Freeman, 1979).
 11. Z. Michalewicz and B. F. Fogel, *How to Solve it: Modern Heuristics*. (Springer, 2004), 2nd edition.
 12. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. (Springer, 1999).
 13. V. V. Vazirani, *Approximation Algorithms*. (Springer, 2001).
 14. R. G. Downey and M. R. Fellows, *Parameterized Complexity*. (Springer, 1999).
 15. J. Flum and M. Grohe, *Parameterized Complexity Theory*. (Springer, 2006).
 16. R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*. (Oxford University Press, 2006).
 17. L. S. Chandran and F. Grandoni, Refined memorisation for Vertex Cover, *Information Processing Letters*. **93**(3), 125–131, (2005).
 18. J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for Vertex Cover. In *Proc. 31st MFCS*, vol. 4162, *LNCS*, pp. 238–249. Springer, (2006).
 19. H. Ito, K. Iwama, and T. Osumi. Linear-time enumeration of isolated cliques. In *Proc. 13th ESA*, vol. 3669, *LNCS*, pp. 119–130. Springer, (2005).
 20. C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for enumerating dense subgraphs. In *Proc. 13th COCOON*, *LNCS*. Springer, (2007). To appear.
 21. C. Komusiewicz. Various isolation concepts for the enumeration of dense subgraphs. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, (2007).
 22. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier, Graph-modeled data clustering: Exact algorithms for clique generation, *Theory of Computing Systems*. **38**(4), 373–392, (2005).
 23. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier, Automated generation of search tree algorithms for hard graph modification problems, *Algorithmica*. **39**(4), 321–347, (2004).
 24. F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized Bicluster Editing. In *Proc. 2nd IWPEC*, vol. 4169, *LNCS*, pp. 1–12. Springer, (2006).
 25. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction, exact, and heuristic algorithms for clique cover. In *Proc. 8th ALENEX*, pp. 86–94. SIAM, (2006). Long version to appear under the title “Data reduction and exact algorithms for clique cover” in *ACM Journal of Experimental Algorithms*.
 26. J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H.-P. Piepho, and R. Schmid, Algorithms for compact letter displays: Comparison and evaluation, *Computational Statistics & Data Analysis*. (2006). To appear.
 27. J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of gen-

- eralized Vertex Cover problems. In *Proc. 9th WADS*, vol. 3608, LNCS, pp. 36–48. Springer, (2005). Long version to appear under the title “Parameterized complexity of Vertex Cover variants” in *Theory of Computing Systems*.
28. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. (MIT Press, 2001), 2nd edition.
 29. J. Kleinberg and É. Tardos, *Algorithm Design*. (Addison Wesley, 2005).
 30. R. Diestel, *Graph Theory*. (Springer, 2005), 3rd edition.
 31. D. B. West, *Introduction to Graph Theory*. (Prentice Hall, 2001), 2nd edition.
 32. S. Mecke and D. Wagner. Solving geometric covering problems by data reduction. In *Proc. 12th ESA*, vol. 3221, LNCS, pp. 760–771. Springer, (2004).
 33. K. Weihe. Covering trains by stations or the power of data reduction. In *Proc. 1st ALEX*, pp. 1–8, (1998).
 34. J. Guo and R. Niedermeier, Invitation to data reduction and problem kernelization, *ACM SIGACT News*. **38**(1), 31–45, (2007).
 35. L. Cai, J. Chen, R. Downey, and M. R. Fellows, Advice classes of parameterized tractability, *Annals of Pure and Applied Logic*. **84**, 119–138, (1997).
 36. F. K. H. A. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The Cluster Editing problem: Implementations and experiments. In *Proc. 2nd IWPEC*, vol. 4169, LNCS, pp. 13–24. Springer, (2006).
 37. M. Davis, G. Logemann, and D. W. Loveland, A machine program for theorem-proving, *Communications of the ACM*. **5**(7), 394–397, (1962).
 38. L. Drori and D. Peleg, Faster exact solutions for some NP-hard problems, *Theoretical Computer Science*. **287**(2), 473–499, (2002).
 39. F. V. Fomin, F. Grandoni, and D. Kratsch, Some new techniques in design and analysis of exact (exponential) algorithms, *Bulletin of the EATCS*. **87**, 47–77, (2005).
 40. K. Mehlhorn, *Data Structures and Algorithms, Volume 2: NP-Completeness and Graph Algorithms*. EATCS Monographs on Theoretical Computer Science, (Springer, 1984).
 41. G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Proc. 5th IWCO*, vol. 2570, LNCS, pp. 185–208. Springer, (2003).
 42. R. Niedermeier and P. Rossmanith, A general method to speed up fixed-parameter-tractable algorithms, *Information Processing Letters*. **73**, 125–129, (2000).
 43. P. Damaschke, Parameterized enumeration, transversals, and imperfect phylogeny reconstruction, *Theoretical Computer Science*. **351**(3), 337–350, (2006).
 44. D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proc. 15th SODA*, pp. 788–797. ACM/SIAM, (2004).
 45. F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons, Scalable parallel algorithms for FPT problems, *Algorithmica*. **45**(3), 269–284, (2006).
 46. J. Cheetham, F. K. H. A. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon, Solving large FPT problems on coarse-grained parallel machines, *Journal of Computer and System Sciences*. **67**(4), 691–706, (2003).
 47. R. E. Korf, M. Reid, and S. Edelkamp, Time complexity of iterative-deepening-A*, *Artificial Intelligence*. **129**, 199–218, (2001).

48. A. Felner, R. E. Korf, and S. Hanan, Additive pattern database heuristics, *Journal of Artificial Intelligence Research*. **21**, 1–39, (2004).
49. J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Mathematica*. **182**(1), 105–142, (1999).
50. F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the Vertex Cover problem: theory and experiments. In *Proc. 6th ALENEX*, pp. 62–69. SIAM, (2004).
51. B. Chor, M. R. Fellows, and D. W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proc. 30th WG*, vol. 3353, *LNCS*, pp. 257–269. Springer, (2004).
52. F. N. Abu-Khzam, M. A. Langston, and W. H. Suters. Fast, effective vertex cover kernelization: A tale of two algorithms. In *Proc. 3rd AICCSA*. ACS/IEEE, (2005). 16 pages.
53. Y. Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova. Genome-scale computational approaches to memory-intensive applications in systems biology. In *Proc. 18th SC*, p. e12. IEEE Computer Society, (2005).
54. M. Chlebík and J. Chlebíková. Improvement of Nemhauser–Trotter theorem and its applications in parameterized complexity. In *Proc. 9th SWAT*, vol. 3111, *LNCS*, pp. 174–186. Springer, (2004).
55. M. Krivánek and J. Morávek, NP-hard problems in hierarchical-tree clustering, *Acta Informatica*. **23**(3), 311–323, (1986).
56. M. Charikar, V. Guruswami, and A. Wirth, Clustering with qualitative information, *Journal of Computer and System Sciences*. **71**(3), 360–383, (2005).
57. N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proc. 37th STOC*, pp. 684–693. ACM, (2005).
58. N. Bansal, A. Blum, and S. Chawla, Correlation clustering, *Machine Learning*. **56**(1), 89–113, (2004).
59. P. Damaschke. On the fixed-parameter enumerability of Cluster Editing. In *Proc. 31st WG*, vol. 3787, *LNCS*, pp. 283–294. Springer, (2005).
60. M. R. Fellows, M. A. Langston, F. Rosamond, and P. Shaw. Polynomial-time linear kernelization for Cluster Editing. Manuscript, (2006).
61. J. Guo. A more effective linear kernelization for cluster editing. In *Proc. ESCAPE*, *LNCS*. Springer, (2007). To appear.
62. L. Wang, Y. Lin, and X. Liu. Approximation algorithms for bi-clustering problems. In *Proc. 6th WABI*, vol. 4175, *LNBI*, pp. 310–320. Springer, (2006).
63. J.-L. Guillaume and M. Latapy, Bipartite structure of all complex networks, *Information Processing Letters*. **90**(5), 215–221, (2004).
64. J. B. Orlin, Contentment in graph theory: Covering graphs with cliques, *Indagationes Mathematicae (Proceedings)*. **80**(5), 406–424, (1977).
65. C. Lund and M. Yannakakis, On the hardness of approximating minimization problems, *Journal of the ACM*. **41**, 960–981, (1994).
66. E. Kellerman, Determination of keyword conflict, *IBM Technical Disclosure Bulletin*. **16**(2), 544–546, (1973).
67. L. T. Kou, L. J. Stockmeyer, and C.-K. Wong, Covering edges by cliques

- with regard to keyword conflicts and intersection graphs, *Communications of the ACM*. **21**(2), 135–139, (1978).
68. H.-P. Piepho, An algorithm for a letter-based representation of all-pairwise comparisons, *Journal of Computational and Graphical Statistics*. **13**(2), 456–466, (2004).
 69. S. Rajagopalan, M. Vachharajani, and S. Malik. Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints. In *Proc. CASES*, pp. 157–164. ACM, (2000).
 70. M. Behrisch and A. Taraz, Efficiently covering complex networks with cliques of similar vertices, *Theoretical Computer Science*. **355**(1), 37–47, (2006).
 71. A. Gyarfas, A simple lower bound on edge coverings by cliques, *Discrete Mathematics*. **85**(1), 103–104, (1990).
 72. K. Holzapfel, S. Kosub, M. G. Maaß, and H. Taubig, The complexity of detecting fixed-density clusters, *Discrete Applied Mathematics*. **154**(11), 1547–1562, (2006).
 73. H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein, Predicting interactions in protein networks by completing defective cliques, *Bioinformatics*. **22**(7), 823–829, (2006).
 74. S. Butenko and W. E. Wilhelm, Clique-detection models in computational biochemistry and genomics, *European Journal of Operational Research*. **173**(1), 1–17, (2006).