Data Reduction and Exact Algorithms for Clique Cover

JENS GRAMM
Eberhard-Karls-Universität Tübingen
and
JIONG GUO, FALK HÜFFNER, and ROLF NIEDERMEIER
Friedrich-Schiller-Universität Jena

To cover the edges of a graph with a minimum number of cliques is an NP-hard problem with many applications. We develop for this problem efficient and effective polynomial-time data reduction rules that, combined with a search tree algorithm, allow for exact problem solutions in competitive time. This is confirmed by experiments with real-world and synthetic data. Moreover, we prove the fixed-parameter tractability of covering edges by cliques.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial Algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms: Algorithms; Experimentation; Theory

Additional Key Words and Phrases: Clique cover; Data reduction; Fixed-parameter tractability

1. INTRODUCTION

Data reduction techniques for exactly solving NP-hard combinatorial optimization problems have proven useful in many studies [Guo and Niedermeier 2007]. The point is that by polynomial-time executable reduction rules many input instances of hard combinatorial problems can be significantly shrunk or simplified, without sacrificing the possibility of finding an optimal solution to the given problem. For

A preliminary version appeared under the title "Data reduction, exact, and heuristic algorithms for clique cover" in *Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX '06)*, pages 86–94, SIAM 2006. Note that the conference version contains additional material concerning a polynomial-time heuristic which we omit here to improve the focus.
Jens Gramm was supported by DFG project OPAL, NI 369/2. Jiong Guo and Falk Hüffner were supported by DFG Emmy Noether research group PIAF, NI 369/4.
Authors' addresses: Jens Gramm, Wilhelm-Schickard-Institut für Informatik, Eberhard-Karls-Universität Tübingen, Sand 13, D-72076 Tübingen, Germany; email: gramm@informatik.uni-tuebingen.de. Jiong Guo, Falk Hüffner, and Rolf Niedermeier, Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany; email: {guo,hueffner,niedermr}@minet.uni-jena.de.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 20YY ACM 0000-0000/20YY/0000-0001 $5.00

such reduced instances then often exhaustive search algorithms can be applied to efficiently find optimal solutions. Hence, data reduction techniques are considered as a "must" when trying to cope with computational intractability. Studying the NP-hard problem to cover the edges of a graph with a minimum number of cliques ((Edge) Clique Cover)[1], we add a new example to the success story of data reduction, presenting both empirical as well as theoretical findings.

Our study problem Clique Cover, also known as Keyword Conflict problem [Kellerman 1973] or Covering by Cliques (GT17) or Intersection Graph Basis (GT59) [Garey and Johnson 1979], has applications in diverse fields such as compiler optimization [Rajagopalan et al. 2000], computational geometry [Agarwal et al. 1994], and applied statistics [Piepho 2004; Gramm et al. 2007]. Thus, it is not surprising that there has been substantial work on (polynomial-time) heuristic algorithms for Clique Cover [Kellerman 1973; Kou et al. 1978; Rajagopalan et al. 2000; Piepho 2004; Gramm et al. 2007; Behrisch and Taraz 2006]. In this paper, we extend and complement this work, in particular introducing new data reduction techniques.

A *clique* in an undirected graph $G = (V, E)$ is a set C of vertices such that for any two vertices in C, there is an edge connecting the two. We will also use "clique" to refer to the complete subgraph of G induced by C and assume that the exact meaning will be made clear from context. Formally, as a (parameterized) decision problem, Clique Cover is defined as follows:

> Clique Cover
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Is there 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?

As first observed by Erdős et al. [1966], Clique Cover is equivalent to a problem from intersection graph theory (see McKee and McMorris [1999] for a monograph on intersection graphs). Let $\mathcal{F} = \{S_1, \ldots, S_n\}$ be a family of sets. The *intersection graph* of $\mathcal{F}$, denoted $\Omega(\mathcal{F})$, is the graph having $\mathcal{F}$ as vertex set with S_i adjacent to S_j iff $i \neq j$ and $S_i \cap S_j \neq \emptyset$. It is easy to see that for every *feature* $x \in \mathcal{U}(\mathcal{F}) := \bigcup_{S \in \mathcal{F}} S$, the set $C_x := \{S_i \in \mathcal{F} \mid x \in S_i\}$ forms a clique in $\Omega(\mathcal{F})$, and $\{C_x \mid x \in \mathcal{U}(\mathcal{F})\}$ is a clique cover for $\Omega(\mathcal{F})$. Therefore, finding a minimum cardinality clique cover for a graph G is equivalent to finding a set intersection representation $\mathcal{F}$ for G that minimizes $|\mathcal{U}(\mathcal{F})|$ (called Intersection Graph Basis by Garey and Johnson [1979]). Guillaume and Latapy [2004] argue that this model is very widely applicable to discover underlying structure in complex real-world networks. Behrisch and Taraz [2006] give simple greedy algorithms for Clique Cover that provide asymptotically optimal solutions for certain random intersection graphs.

Clique Cover is NP-hard [Orlin 1977], even when restricted to planar graphs [Chang and Müller 2001] or graphs with maximum degree 6 [Hoover 1992]. It is polynomial-time solvable for chordal graphs [Ma et al. 1989], graphs with maximum

[1]We remark that covering *vertices* by cliques (Vertex Clique Cover or Clique Partition) is of less interest to be studied on its own because it is equivalent to the well-investigated Graph Coloring problem: A graph has a vertex clique cover of size k iff its complement graph can be colored with k colors such that adjacent vertices have different colors.

degree 5 [Hoover 1992], line graphs [Orlin 1977], and circular arc graphs [Hsu and Tsai 1991]. By way of contrast, Clique Cover is not approximable within a factor of $|V|^{\epsilon}$ for some $\epsilon > 0$ unless P = NP [Lund and Yannakakis 1994], and nothing better than a polynomial approximation factor of $O\big(|V|^{2\frac{(\log\log|V|)^2}{(\log|V|)^3}}\big)$ is known [Ausiello et al. 1999].

We examine Clique Cover in the context of parameterized complexity (see Flum and Grohe [2006] and Niedermeier [2006] for new monographs on parameterized complexity analysis). An instance of a parameterized problem consists of a problem instance I and a parameter k being a nonnegative integer. 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, not on the input size $|I|$.

First, as our main algorithmic contribution, we introduce and analyze data reduction techniques for Clique Cover. As a side effect, we provide a so-called problem kernel for Clique Cover, for the first time showing—somewhat surprisingly—that the problem is fixed-parameter tractable with respect to the parameter k. We continue with describing an exact algorithm based on a search tree. For our experimental investigations, we combined our data reduction rules with the search tree, clearly outperforming heuristic algorithms [Kellerman 1973; Kou et al. 1978] in several ways. For instance, we can solve real-world instances from a statistical application [Piepho 2004]—so far solved heuristically [Piepho 2004]—optimally without time loss. This indicates that for a significant fraction of real-world instances our exact approach is clearly to be preferred to a heuristic approach which is without guaranteed solution quality. We also experimented with random graphs of different densities, showing that our exact approach works extremely well for sparse graphs. In addition, our empirical results reveal that for dense graphs a data reduction rule that was designed for showing the problem kernel is often useful. In particular, this gives strong empirical support for further theoretical studies in the direction of improved fixed-parameter tractability results for Clique Cover, nicely demonstrating a fruitful interchange between applied and theoretical algorithmic research.

2. DATA REDUCTION

A (*data*) *reduction rule* replaces, in polynomial time, a given Clique Cover instance (G,k) consisting of a graph G and a nonnegative integer k by a "simpler" instance (G',k') such that (G,k) has a solution iff (G',k') has a solution. An instance to which none of a given set of reduction rules applies is called *reduced* with respect to these rules. A parameterized problem such as Clique Cover (the parameter is k) is said to have a *problem kernel* if, after the application of the reduction rules, the reduced instance has size $f(k)$ for a function f depending only on k. It is a well-known result from parameterized complexity theory that a parameterized problem is fixed-parameter tractable iff it admits a problem kernel [Cai et al. 1997].

Given an n-vertex and m-edge graph G, we use $N(v)$ to denote the neighborhood of vertex v in G, namely, $N(v) := \{u \mid \{u,v\} \in E\}$. The *closed* neighborhood of vertex v, denoted by $N[v]$, is equal to $N(v)\cup\{v\}$. We formulate reduction rules for a generalized version of Clique Cover in which already some edges may be marked

as "covered." Then, the question is to find a clique cover of size k that covers all uncovered edges. Clearly, Clique Cover is the special case of this annotated version where no edge is marked as covered.

We start by describing an initialization routine that sets up auxiliary data structures *once* at the beginning of the algorithm such that the *many* applications of Rule 2 (defined below) become cheaper in terms of runtime. Moreover, the data structures initialized here are also used by the exact algorithm proposed in Section 3. From the reduction rules below, only Rule 1 updates these auxiliary data structures.

Initialization. We inspect every edge $\{u, v\}$ of the original graph. We use two auxiliary variables: We compute a set $N_{\{u,v\}}$ of u and v's common neighbors and we determine whether the vertices in $N_{\{u,v\}}$ induce a clique. More precisely, we compute a positive integer $c_{\{u,v\}}$ which denotes the number of edges interconnecting the vertices of $N_{\{u,v\}}$.

Lemma 1. *The proposed initialization can be done in $O(m^2)$ time.*

Proof. For an edge $\{u, v\}$, we compute $N_{\{u,v\}}$ in $O(n)$ time. Computing $c_{\{u,v\}}$ can be done in $O(m)$ time. Doing this for all edges requires $O(m^2)$ time in total. □

We start the presentation of data reduction rules with a trivial rule removing isolated elements.

Rule 1. *Remove isolated vertices and vertices that are only adjacent to covered edges.*

It is obvious that Rule 1 is correct.

Lemma 2. *Every application of Rule 1 including the update of auxiliary variables can be executed in $O(nm)$ time.*

Proof. The applicability of Rule 1 can be checked in $O(m + n)$ time. Notably, after removing a vertex, Rule 1 requires an update of the data structures provided by the initialization. For one removed vertex v, we have to adjust the sets $N_{\{u,w\}}$ and counters $c_{\{u,w\}}$ for all connected neighbors u and w of v. For an edge $\{u, w\}$ with $v \in N_{\{u,w\}}$, the affected sets and counters can be updated in $O(n)$ time by removing vertex v from $N_{\{u,w\}}$ and decreasing $c_{\{u,w\}}$ by $|N(v) \cap N_{\{u,w\}}|$. For m edges, the asymptotic runtime of Rule 1 amounts to $O(nm)$. □

The next reduction rule is concerned with maximal cliques. Note that we can safely assume that an optimal solution consists of maximal cliques only, since a non-maximal clique in a solution can always be replaced by a maximal clique it is contained in. The following rule identifies maximal cliques which have to be part of *every* optimal solution.

Rule 2. *If an uncovered edge $\{u, v\}$ is contained in exactly one maximal clique C, that is, if the common neighbors of u and v induce a clique, then add C to the solution, mark its edges as covered, and decrease k by one.*

Lemma 3. *Rule 2 is correct. Every application of Rule 2 can be executed in $O(m)$ time.*

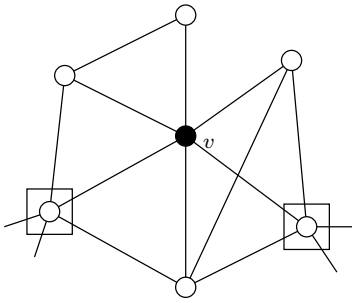

Fig. 1. An illustration of the partition of the neighborhood of a vertex v. The two vertices with rectangles are exits, the other white ones are prisoners.

PROOF. The rule is correct: Edge $\{u, v\}$ *has* to be covered and, as mentioned above, we can assume, without loss of generality, that it is covered by a maximal clique. If there is exactly one maximal clique C covering $\{u, v\}$, then C has to be part of every optimal solution.

Using the proposed initialization, we can apply Rule 2 in $O(m)$ time: For one edge, by looking up $N_{\{u,v\}}$ and $c_{\{u,v\}}$, we can determine in constant time whether the rule is applicable. Scanning through all edges and invoking the rule as soon as we find an edge for which the rule is applicable can be done in $O(m)$ time. □

Rules 1 and 2 remove all degree-1 and degree-2 vertices from the instance. Further, they imply that an isolated clique is deleted: Its edges belong to exactly one maximal clique; the clique, if it contains more than one vertex, is added to the solution by Rule 2 and its vertices are "cleaned up" by Rule 1.

In the following we present two interrelated reduction rules Rules 3' and 3. Rule 3' is subsumed by Rule 3. Nevertheless we choose to present both rules separately since Rule 3' is easier to understand. Moreover, as will be shown in Theorem 1, already Rule 3' is sufficient to show a problem kernel for CLIQUE COVER.

RULE 3'. *If there is an edge $\{u, v\}$ whose endpoints have exactly the same closed neighborhood, that is, $N[u] = N[v]$, then mark all edges incident to u as covered. To reconstruct a solution for the unreduced instance, add u to every clique containing v.*

Comparing $N[u]$ and $N[v]$ for each edge $\{u, v\}$, we can in $O(nm)$ time search an edge for which Rule 3' is applicable and invoke the rule.

For formulating a generalization of Rule 3', we introduce additional terminology. For a vertex v, we partition the set of vertices that are connected by an uncovered edge to v into *prisoners* p with $N(p) \subseteq N(v)$ and *exits* x with $N(x) \setminus N(v) \neq \emptyset$.[2] 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.

[2]We remark that the concept of prisoners and exits (and, in addition, "gates") was introduced for data reduction rules designed for the DOMINATING SET problem [Alber et al. 2004]. The strength of these rules has been proven theoretically [Alber et al. 2004] as well as empirically [Alber et al. 2006].

RULE 3. *Consider a vertex v which has at least one prisoner. If each prisoner is connected to at least one vertex other than v via an uncovered edge (note that this is automatically given when the instance is reduced with respect to Rules 1 and 2), 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.*

Observe that a vertex v is always a prisoner of a vertex u with $u \neq v$ and $N[u] = N[v]$ (and vice versa). In particular, this prisoner dominates all exits. Thus, Rule 3' is subsumed by Rule 3.

LEMMA 4. *Rule 3 is correct. Every application of Rule 3 can be executed in $O(n^3)$ time.*

PROOF. For the correctness note that, 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 incident 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 $\{v, p\}$ to a prisoner p, vertex p has to be part of a clique C of the solution for the instance after application of the rule. Therefore, the edge $\{v, p\}$ 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 p and therefore x has to be part of a clique C with p. Hence, the edge $\{v, x\}$ is covered by $C \cup \{v\}$ in the solution for the unreduced instance.

For executing the rule, we inspect every vertex v to test whether the rule is applicable. To this end, we inspect every neighbor u of v. In $O(n)$ time, we determine whether u is an exit or a prisoner. Having identified all prisoners, we can for every exit u determine in $O(n)$ time whether u is dominated by a prisoner. □

One easily observes that there are cases where Rule 3 applies but Rule 3' does not; however, we could not make use of this fact to improve the theoretical analysis which follows in Theorem 1.

LEMMA 5. *Using Rules 1 to 3, in $O(n^4)$ time one can generate a reduced instance where none of these rules applies any further.*

PROOF. First, we apply Rule 1 to remove the isolated vertices. Then, we repeat the following operation until none of Rules 2 and 3 is applicable: Apply one of Rules 2 and 3, if possible, and, after each application, use Rule 1 to remove the vertices only adjacent to covered edges. Since each application of Rule 2 or Rule 3 results in at least one vertex only incident to covered edges and each application of Rule 1 removes at least one vertex from the graph, the above operation is repeated at most n times. From Lemmas 2, 3, and 4 we can conclude that the total runtime for the application of the three rules amounts to $O(n \cdot (nm+m+n^3))$. Together with the runtime of the initialization $O(m^2)$ shown in Lemma 1, the claimed runtime follows. □

From a theoretical viewpoint, the main result of this section is a problem kernel

with respect to parameter k for Clique Cover:[3]

Theorem 1. *A* Clique Cover *instance reduced with respect to Rules 1 and 3' contains at most* 2^k *vertices or, otherwise, has no solution.*

Proof. Consider any graph $G = (V, E)$ with more than 2^k vertices that has a clique cover $C_1, \ldots, C_k$ 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 to 1 iff v is contained in clique C_i. Since there are only 2^k possible vectors, there must be $u \neq v \in V$ with $b_u = b_v$. If b_u and b_v are all-zero vectors, Rule 1 applies; otherwise, u and v are contained in the same cliques. This means that u and v are connected and share the same neighborhood, and thus Rule 3' applies. □

Corollary 2. Clique Cover *is fixed-parameter tractable with respect to parameter* k.

Proof. By Theorem 1 a Clique Cover instance reduced with respect to Rules 1 and 3' has at most 2^k vertices. It takes $O(n^4)$ time to generate a reduced instance (Lemma 5). This reduced instance can then be solved by exhaustive search. Let $f(|I|)$ denote the runtime of exhaustive search on an instance I. Putting things together, Clique Cover can then be solved in $O(f(2^k) + n^4)$ time, which implies the fixed-parameter tractability with respect to parameter k. □

The result of Corollary 2 might be surprising when noting that many graph problems that involve cliques turn out to be hard in the parameterized sense. For example, the NP-hard Clique problem is known to be W[1]-hard with respect to the clique size [Downey and Fellows 1999], that is, we have a clear indication that this problem is not fixed-parameter tractable with respect to the parameter "clique size." Another example even more closely related to Clique Cover is given by the NP-hard Clique Partition problem, which is also hard in the parameterized sense. Herein, we ask, given an undirected graph and $k > 0$, for a set of k cliques covering all *vertices* of the input graph (in contrast to covering all *edges* as in Clique Cover). Clique Partition is NP-hard already for $k = 3$ [Garey and Johnson 1979]. This can be seen by observing that Clique Partition is equivalent to Coloring on the complement graph; the number of colors required for Coloring corresponds to the number of cliques required for Clique Partition. It is well-known that Coloring is already NP-hard for three colors [Garey and Johnson 1979]. It follows that there is no hope for obtaining fixed-parameter tractability for Clique Partition with respect to parameter k, unless P = NP. In contrast, Clique Cover is shown fixed-parameter tractable in Corollary 2; in both cases the number of required cliques is the considered parameter.

We conclude this section by mentioning an additional rule which does not seem to give any improvement on the problem kernel but may nevertheless be interesting to consider for practical purposes. It differs from the above rules in that it does not make the instance smaller (in fact it makes it larger), but intuitively seems to facilitate application of other data reduction rules since it decomposes the input into separate components.

[3]The central underlying observation was already made by Gyárfás [1990].

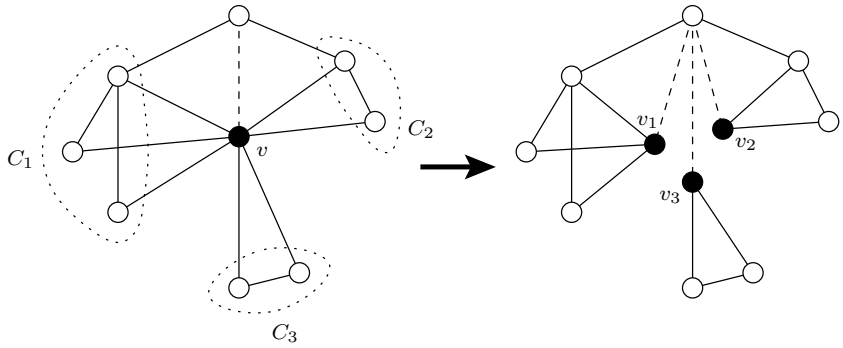

Fig. 2. Example for Rule 4. *Dashed* edges are covered.

RULE 4. *Consider the set N of vertices that are connected to a vertex v via uncovered edges. Let $C_1, \ldots, C_l$ be the connected components induced by N. If there is more than one connected component (that is, $l > 1$), then replace v by l new vertices $v_1, \ldots, v_l$, and connect for each $1 \leq i \leq l$ the vertex v_i to all vertices in C_i. In addition, all of $v_1, \ldots, v_l$ are connected by covered edges to all vertices that are connected to v with a covered edge.*

Figure 2 gives an example. It is straightforward to observe that Rule 4 is correct.

3. A SEARCH TREE ALGORITHM

Search trees are a popular means of exactly solving hard problems. The basic method is to identify for a given instance a small set of simplified instances such that the given instance has a solution if at least one of the simplified instances has one. The algorithm then branches recursively into each of the subcases until a stop criterion is met.

The search tree algorithm presented here for CLIQUE COVER works as follows. We choose an uncovered edge, enumerate all maximal cliques this edge is part of, and then branch according to the clique we add to the clique cover. The recursion stops as soon as a solution is found or k cliques have been chosen without finding a solution. To simplify notation, we use $V = \{1, \ldots, n\}$ as vertex set. The algorithm is presented in pseudo-code in Figure 3.

At first glance, this branching seems impractical, since the number of maximal cliques in a graph can be exponentially large, resulting in a double-exponentially large search tree. However, in practice it turns out that there are usually only a few branching cases. We try to give some intuition for this: Sensible inputs for clustering problems are expected to exhibit transitivity in the sense that if $\{a, b\}$ and $\{b, c\}$ are edges, then probably also $\{a, c\}$ is an edge (that is, its *clustering coefficient* is high). Graphs with many maximal cliques, however, are closely related to the presence of certain complete multipartite graphs [Prisner 1995]; these multipartite graphs are very nontransitive.

Regarding the choice of the edge to branch on, we would, ideally, like to branch on the edge that is contained in the least number of maximal cliques. However, this calculation would be costly. Therefore, we make use of the infrastructure set up for an efficient incremental application of Rule 2. The initialization described in

```
Input:    Graph G = ({1, 2, ..., n}, E).
Output:   A minimum cardinality clique cover for G.
1   k ← 0; X ← nil
2   while X = nil:
3       X ← branch(G, k, ∅)
4       k ← k + 1
5   return X
6   function branch(G, k, X):
7       if X covers G: return X
8       reduce (G, k)
9       if k < 0: return nil
10      choose {i, j} such that (|N_{i,j}| choose 2) − c_{i,j} is minimal
11      for each maximal clique C in N[i] ∩ N[j]:
12          X' ← branch(G, k − 1, X ∪ {C})
13          if X' ≠ nil: return X'
14      return nil
```

Fig. 3. Exact algorithm for Clique Cover.

Section 2 provides a set $N_{\{i,j\}}$ containing the common neighborhood of edge $\{i,j\}$ and a counter $c_{\{i,j\}}$ containing the number of edges in the common neighborhood of its endpoints. Therefore, $\binom{|N_{\{i,j\}}|}{2} - c_{\{i,j\}}$ is the number of edges missing in the common neighborhood of edge $\{i,j\}$ as compared to a clique (called *score*). For branching, we choose the edge with the lowest score. If the score is 0, then the edge is contained in only one maximal clique (and thus will be marked as covered by Rule 2). If the score is 1, the edge is contained in exactly two maximal cliques. Generalizing this, it is plausible to assume that an edge is contained in few maximal cliques if its score is low.

Having chosen the edge to branch on, we determine the set of maximal cliques the edge is contained in using a variant of the classical Bron–Kerbosch algorithm [Bron and Kerbosch 1973] described by Koch [2001]. While the original Bron–Kerbosch algorithm does not exhibit the desired output sensitivity (it runs in exponential time even for a single maximal clique), the variant by Koch turns out to be fast enough for our purposes.

We use the branching routine within an iterative deepening framework, that is, we impose a maximum search depth k and increase this limit by one when no solution is found.

Combining the data reduction rules described in Section 2—which yield a problem kernel for Clique Cover—with the search tree algorithm described here, we obtain a competitive fixed-parameter algorithm for Clique Cover that can solve problem instances of considerable size (a few hundred vertices) efficiently (see Section 4).

4. EXPERIMENTAL RESULTS

We implemented the search tree algorithm from Section 3 and the data reduction rules from Section 2. The program is written in the Objective Caml programming

			Clique cover size	
	n	m	Heuristic	Optimal
A	13	55	4	4
B	17	86	6	5
C	124	4847	50	49
D	121	4706	48	48
E	97	3559	34	31

Table I. Clique cover sizes for five real-world Clique Cover instances, where "Heuristic" is the heuristic by Kellerman [1973] with the postprocessing by Kou et al. [1978].

language [Leroy et al. 1996] and consists of about 1400 lines of code. The source code is free software and available from `http://theinf1.informatik.uni-jena.de/ecc/`. Graphs are implemented using a purely functional representation based on Patricia trees [Okasaki and Gill 1998]. This allows to (conceptually) modify the graph in the course of the algorithm without having to worry about how to restore it when returning from the recursion. Moreover, it allows for quick intersection operations on neighbor sets, as required for some reduction rules. The cache data structure N described in Section 2 is implemented using a priority search queue.

We tested our implementation on various inputs on an AMD Athlon 64 3700+ with 2.2 GHz, 1 MB cache, and 1 GB main memory, running under the Debian GNU/Linux 3.1 operating system.

Real-World Data. We first tested the implementation on five "real-world" instances from an application in graphical statistics [Piepho 2004] (see Table I). Currently, heuristics like that of Kou et al. [1978] are used to solve the problem in practice [Piepho 2004; Gramm et al. 2007]. With our implementation of the heuristic by Kellerman [1973]—which gives no guarantee for the quality of the solution—, the runtime is negligible for these instances (< 0.1 s). Our implementation based on the search tree with data reduction could solve all instances to optimality within less than one second. In all cases, no branching was required: Rules 1 and 2 already completely reduced the instances. We observe that the heuristic produces reasonably good results for these cases; previously nothing was known about its solution quality. In summary, the application of our algorithm in this area seems quite attractive, since we can provide provably optimal results within acceptable runtime bounds.

Random Graphs. Next, we tested the implementation of the exact algorithm on random graphs, that is, graphs where every possible edge is present with a fixed probability ($G_{n,p}$ model). It is known that with high probability a random graph has a large clique cover of size $\Theta(n^2/\log^2 n)$ [Frieze and Reed 1995]. Therefore, relying on branching and a not too large search tree is unlikely to succeed, and data reduction rules are crucial. The results are presented in Figure 4. In the following, the "size" of an instance means the number of vertices. We examine three trials: Sparse graphs with $m \approx n \ln n$, graphs with edge probability 0.1, and graphs with edge probability 0.15. For the denser graphs outliers occur: for example for graphs of size 79 and edge probability 0.15, all of 20 instances could be solved within 10 seconds but one, which took 16 minutes. In contrast, sparse graphs can

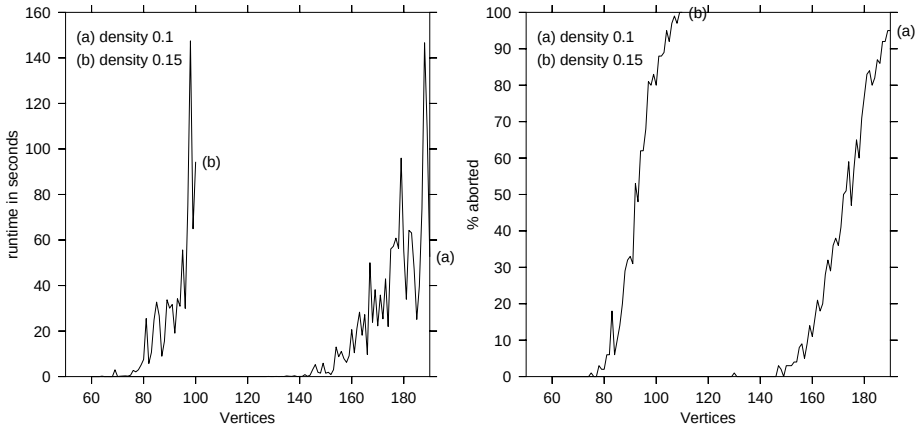

Fig. 4. Runtime for random graphs. Runs were aborted after 10 minutes. Left: average runtime of successful runs; right: percentage of aborted runs.

	n	m	$\lvert C\rvert$	runtime	search tree	Rule 1	Rule 2	Rule 3	Rule 4
sparse	1000	6 954	6 180	1.00	1	1 000	6180	0	0
	1000	6 816	6 022	0.96	1	1 000	6022	0	0
	1000	6 861	6 107	0.96	1	1 000	6107	0	0
$p = 0.1$	156	1 230	653	20.58	254584	845 180	429 193	0	39 042
	156	1 194	644	0.02	27	194	664	0	1
	156	1 226	646	3285.43	21 889 796	112 527 915	63 709 259	0	5 313 473
$p = 0.15$	85	524	273	0.01	1	85	273	0	0
	85	545	272	15.88	132056	705743	382767	0	25032
	85	560	265	1505.94	8725027	47947699	27087827	0	3295196

Table II. Statistics for selected random Clique Cover instances. Here, p is the edge probability, runtime is measured in seconds, $|C|$ is the size of the clique cover, "search tree" is the number of nodes in the search tree, and "Rule r" is the number of applications of Rule r.

be solved uniformly very quickly: Instances of size 5 000 can still be solved within 80 seconds and instances of size 10 000 within 7 minutes, with a standard deviation for the runtime of $< 2\,\%$. Very little branching is required for sparse instances, with most being solved by data reduction alone, and the largest search tree observed in the experiments having 528 nodes. Thus, our approach is very promising for sparse instances up to moderate size, while for denser instances probably a fallback to heuristic algorithms is required to compensate for the outliers.

The presence of extreme outliers for some combinations of parameters makes it difficult to get a clear picture based only on combining statistics such as averages. Therefore, we show measurements for several concrete instances in Table II. For edge probability 0.1 and 0.15, respectively, we select an instance that takes very long, and additionally present two arbitrary instances with similar parameters. For sparse graphs, no such outliers occur, so we show three arbitrary instances of similar size.

The reason for the outliers are the usually but not always effective data reduction

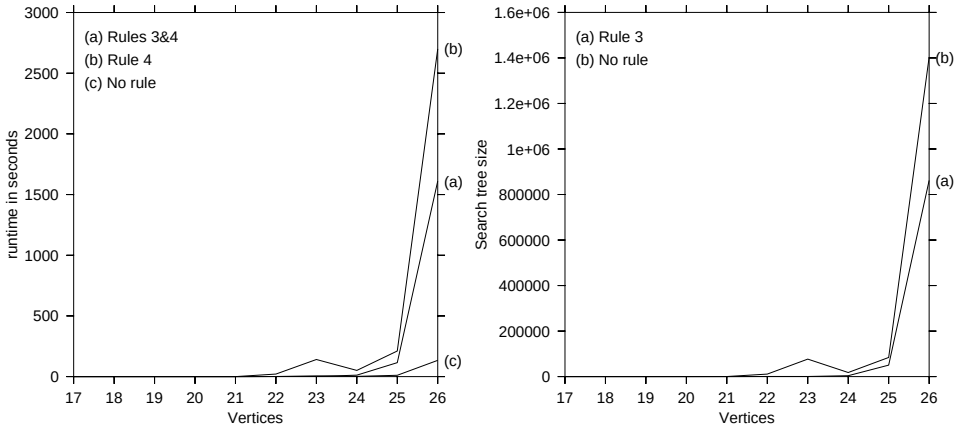

Fig. 5. Runtime and search tree size for random intersection graphs of density 0.3 (average over 20 instances each)

rules. In all instances, we observe an initial reduction phase with many applications of reduction rules. Most of the random graphs with 75 vertices and edge density 0.15 are almost entirely processed by the reduction rules: Out of 50 examined instances we observe search trees with depth more than 3 for 8 instances and 24 instances are completely reduced without branching. However, in rare cases we do encounter instances with an "unreducible core" to which no reduction rule is applicable. Moreover, with an unreducible core it is only rarely the case that reduction rules become applicable after the next branching. Consequently, an unreducible core does cause a significant number of branchings in the search tree.

Synthetic Data. Real instances are not completely random; in particular, in most sensible applications the clique cover is expected to be much smaller than that of a random graph. The fixed-parameter result also promises a better runtime for instances where the clique cover is small. To examine this, we generated random intersection graphs using the $G_{n,m,p}$ model (see Behrisch and Taraz [2006] and references therein), where each of n vertices draws each of m features with probability p (note here m does not denote the number of edges as elsewhere). We can control the size of a clique cover by choosing m: the size of the clique cover must be m or slightly lower in case $C_x \subseteq C_y$ for two features x and y (see Section 1 for the notation). By choosing p, we can generate instances with a desired edge density.

We generated instances with 100 vertices and about 1500 edges (making for a density of 0.3), and varying number of planted cliques m. Figure 5 shows the resulting runtimes. In fact, these quite dense instances can be solved very quickly when the size of the clique cover is small. This makes our exact algorithm also attractive for the numerous applications where we can expect a small clique cover as solution—an observation which is in agreement with our fixed-parameter tractability result.

Effectiveness of Rules 3 and 4. The prisoner-exit-rule (Rule 3) and the neighborhood-splitting-rule (Rule 4) are comparably complicated; Rule 3 has been developed

in context with searching for a problem kernel. Do they really gain any benefit in practice? To examine this question, we repeated the previous experiment with either Rule 3 or Rules 3 and 4 disabled (see Figure 5). Rule 4 does in fact increase the runtime, simply because it is rarely ever applicable in these dense instances. Although Rule 3 increases also the runtime as shown in the left-hand diagram of Figure 5, we recommend to apply it. This is justified by the right-hand side of Figure 5: The reduction of the search tree size indicates that—together with a more efficient implementation of Rule 3, the overall complexity can be (asymptotically) improved. In particular, we believe that, using incremental calculations, the runtime of Rule 3 could be significantly reduced.

5. OUTLOOK

As seen in Table II, there are some outliers with exceedingly high runtimes when compared to "similar" instances. It is an intriguing open question whether there are further data reduction rules that can cope with the remaining outliers. In parallel, this might also lead to a better upper bound on the problem kernel size and improved fixed-parameter tractability for Clique Cover.

In this paper, the proposed data reduction rules were not only used as preprocessing, but they were continuously interleaved with the branching of the search tree algorithm. It remains open how far one may get by combining the data reduction rules with integer linear programming, here using them only as preprocessing.

Acknowledgements. We thank Hans-Peter Piepho (Universität Hohenheim) for providing the test data.

REFERENCES

Agarwal, P. K., Alon, N., Aronov, B., and Suri, S. 1994. Can visibility graphs be represented compactly? *Discrete and Computational Geometry 12*, 347–365.

Alber, J., Betzler, N., and Niedermeier, R. 2006. Experiments on data reduction for optimal domination in networks. *Annals of Operations Research 146,* 1, 105–117.

Alber, J., Fellows, M. R., and Niedermeier, R. 2004. Polynomial-time data reduction for Dominating Set. *Journal of the ACM 51*, 363–384.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. 1999. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer.

Behrisch, M. and Taraz, A. 2006. Efficiently covering complex networks with cliques of similar vertices. *Theoretical Computer Science 355,* 1, 37–47.

Bron, C. and Kerbosch, J. 1973. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM 16,* 9, 575–577.

Cai, L., Chen, J., Downey, R. G., and Fellows, M. R. 1997. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic 84*, 119–138.

Chang, M.-S. and Müller, H. 2001. On the tree-degree of graphs. In *Proc. 27th WG*. LNCS, vol. 2204. Springer, 44–54.

Downey, R. G. and Fellows, M. R. 1999. *Parameterized Complexity*. Springer.

Erdős, P., Goodman, A. W., and Pósa, L. 1966. The representation of a graph by set intersections. *Canadian Journal of Mathematics 18*, 106–112.

Flum, J. and Grohe, M. 2006. *Parameterized Complexity Theory*. Springer-Verlag.

Frieze, A. M. and Reed, B. A. 1995. Covering the edges of a random graph by cliques. *Combinatorica 15,* 4, 489–497.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gramm, J., Guo, J., Hüffner, F., Niedermeier, R., Piepho, H.-P., and Schmid, R. 2007. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*. To appear.

Guillaume, J.-L. and Latapy, M. 2004. Bipartite structure of all complex networks. *Information Processing Letters 90,* 5, 215–221.

Guo, J. and Niedermeier, R. 2007. Invitation to data reduction and problem kernelization. *ACM SIGACT News 38,* 1, 31–45.

Gyárfás, A. 1990. A simple lower bound on edge coverings by cliques. *Discrete Mathematics 85,* 1, 103–104.

Hoover, D. N. 1992. Complexity of graph covering problems for graphs of low degree. *Journal of Combinatorial Mathematics and Combinatorial Computing 11*, 187–208.

Hsu, W.-L. and Tsai, K.-H. 1991. Linear time algorithms on circular-arc graphs. *Information Processing Letters 40,* 3, 123–129.

Kellerman, E. 1973. Determination of keyword conflict. *IBM Technical Disclosure Bulletin 16,* 2, 544–546.

Koch, I. 2001. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science 250,* 1–2, 1–30.

Kou, L. T., Stockmeyer, L. J., and Wong, C.-K. 1978. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM 21,* 2, 135–139.

Leroy, X., Vouillon, J., Doligez, D., et al. 1996. The Objective Caml system. Available on the web. `http://caml.inria.fr/ocaml/`.

Lund, C. and Yannakakis, M. 1994. On the hardness of approximating minimization problems. *Journal of the ACM 41*, 960–981.

Ma, S., Wallis, W. D., and Wu, J. 1989. Clique covering of chordal graphs. *Utilitas Mathematica 36*, 151–152.

McKee, T. A. and McMorris, F. R. 1999. *Topics in Intersection Graph Theory*. SIAM Monographs on Discrete Mathematics and Applications. SIAM.

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

Okasaki, C. and Gill, A. 1998. Fast mergeable integer maps. In *Proc. ACM SIGPLAN Workshop on ML*. 77–86.

Orlin, J. B. 1977. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings) 80,* 5, 406–424.

Piepho, H.-P. 2004. An algorithm for a letter-based representation of all-pairwise comparisons. *Journal of Computational and Graphical Statistics 13,* 2, 456–466.

Prisner, E. 1995. Graphs with few cliques. In *Proc. 7th Conference on the Theory and Applications of Graphs*. Wiley, 945–956.

Rajagopalan, S., Vachharajani, M., and Malik, S. 2000. Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints. In *Proc. CASES*. ACM Press, 157–164.