


Solving Edge Clique Cover Exactly via Synergistic Data Reduction

Anthony Hevia 

Hamilton College, Clinton, NY, USA

Benjamin Kallus 

Dartmouth College, Hanover, NH, USA

Summer McClintic 

Hamilton College, Clinton, NY, USA

Samantha Reisner

Hamilton College, Clinton, NY, USA

Darren Strash¹  

Hamilton College, Clinton, NY, USA

Johnathan Wilson 

Hamilton College, Clinton, NY, USA

Abstract

The edge clique cover (ECC) problem – where the goal is to find a minimum cardinality set of cliques that cover all the edges of a graph – is a classic NP-hard problem that has received much attention from both the theoretical and experimental algorithms communities. While small sparse graphs can be solved exactly via the branch-and-reduce algorithm of Gramm et al. [JEA 2009], larger instances can currently only be solved inexactly using heuristics with unknown overall solution quality. We revisit computing minimum ECCs exactly in practice by combining data reduction for both the ECC *and* vertex clique cover (VCC) problems. We do so by modifying the polynomial-time reduction of Kou et al. [Commun. ACM 1978] to transform a reduced ECC instance to a VCC instance; alternatively, we show it is possible to “lift” some VCC reductions to the ECC problem. Our experiments show that combining data reduction for both problems (which we call *synergistic data reduction*) enables finding exact minimum ECCs orders of magnitude faster than the technique of Gramm et al., and allows solving large sparse graphs on up to millions of vertices and edges that have never before been solved. With these new exact solutions, we evaluate the quality of recent heuristic algorithms on large instances for the first time. The most recent of these, EO-ECC by Abdullah et al. [ICCS 2022], solves 8 of the 27 instances for which we have exact solutions. It is our hope that our strategy rallies researchers to seek improved algorithms for the ECC problem.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Packing and covering problems

Keywords and phrases Edge clique cover, Vertex clique cover, Data reduction, Degeneracy

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.61

Related Version *Full Version*: <https://arxiv.org/abs/2306.17804>

Supplementary Material *Software (Source Code)*: <https://github.com/darrenstrash/Redu3ECC>

Acknowledgements We thank the anonymous reviewers for their insightful feedback, David Swartz from Hamilton College for technical support, and Adam Chrisman, Caitlin Matwijec-Walda, and Jon Matwijec-Walda for a cozy space to work at The Copper Easel and Superofficial in Rome, NY.

¹ Corresponding author.



1 Introduction

In the *edge clique cover (ECC) problem*, also called the *clique cover* problem, we are given an unweighted, undirected, simple graph $G = (V, E)$ and asked to find a minimum cardinality set of cliques that cover the edges of G . The ECC problem is NP-hard, however its decision variant did not appear in Karp’s original list of NP-complete problems [23], though the *vertex clique cover (VCC) problem* did. Compared to the VCC problem, the ECC problem has received the lion’s share of attention from researchers, in part because it has many applications. For instance, edge clique covers can be used to succinctly represent constraints for integer program solvers [5] and to detect communities in networks [12].

Data reduction rules, which allow one to transform an input instance to a smaller equivalent instance of the same problem, are powerful tools for solving NP-hard problems in practice [4, 26]. Of particular interest in the field of parameterized algorithms is whether the repeated application of data reduction rules produces a *kernel* – which is a problem instance that has size bounded by a function $O(f(k))$ of some parameter k of the input. Gramm et al. [19] show that repeated application of four simple reduction rules produce a kernel of size 2^k , where the parameter k is the number of cliques in the cover. When intermixed with branch-and-bound (a so-called *branch-and-reduce* algorithm), these reduction rules enable solving sparse graphs of up to 10,000 vertices quickly in practice. Since their seminal work, no progress has been made on solving larger instances exactly. Indeed, the prospect of doing so is grim since polynomial kernels are unlikely to exist for the ECC problem, when parameterized on the solution size [13]. Although researchers have found further FPT algorithms (and smaller kernels) with other parameters [6, 32], these algorithms are still only able to solve relatively small instances in practice. The outlook for the VCC problem is even worse in theory: it is unlikely to have any problem kernel when parameterized on the number of cliques k in the cover, as it is already NP-hard for $k = 3$ (since it is equivalent to 3-coloring the complement graph).

However, recent data reductions for the VCC problem have been shown to significantly accelerate computing minimum VCCs exactly in practice. Strash and Thompson [31] introduce a suite of reduction rules and show that data reduction can solve real-world sparse graphs with up to millions of vertices in seconds.

Our Results

We show that combining VCC and ECC data reductions enables the ECC problem to be solved exactly on large instances not previously solvable by Gramm et al. [19]. We do so by modifying the polynomial-time transformation of Kou et al. [25] to transform a reduced ECC instance to a VCC instance, but also show that some VCC data reductions can be “lifted” to ECC data reductions. Their combined reduction power (which we call *synergistic data reduction*) reduces an ECC instance significantly more than Gramm et al.’s reductions alone, enabling us to exactly solve graphs with millions of vertices and edges. With these exact results, we objectively evaluate the quality of heuristic algorithms recently introduced in the literature. On instances not solvable exactly with our method, we give upper and lower bounds for use by future researchers.

2 Related Work

We now briefly review the relevant previous work on the ECC and VCC problems, as well as practical data reduction in related problems.

2.1 Edge Clique Cover

The goal of the edge clique cover (ECC) problem is to cover the edges of the graph G with a minimum number of cliques, denoted $\theta_E(G)$. That is, to find a set of cliques $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ such that each edge is in at least one clique in \mathcal{C} and $k = \theta_E(G)$. Although closely related to the VCC problem (to cover *vertices* with a minimum number of cliques, denoted $\theta(G)$), Brigham and Dutton [8] showed that $\theta(G) \leq \theta_E(G)$, and that these cover numbers can differ significantly: $\theta_E(G)$ can be as large as $\theta(G)(n - \theta(G))$. Gramm et al. [19] introduced four data reductions for the ECC problem, which they show can solve real-world sparse graphs of hundreds of vertices, as well as synthetic instances on up to 10K vertices in practice, when interleaved with branch and bound. Furthermore, they showed that their data reductions produce a kernel of size 2^k , where k is the number of cliques. Cygan et al. [13] showed that it is unlikely that a polynomial-size kernel exists when parameterized by the number of cliques in the cover, as otherwise the polynomial hierarchy collapses to its third level. However, Blanchette et al. [6] gave a linear-time algorithm having running time $O(2^{\binom{k}{2}}n)$ where k is the treewidth of the graph. In practice, their algorithm is effective on graphs with hundreds of vertices and small treewidth. For larger graphs, heuristic methods are used to compute inexact ECCs [12, 2, 1] in practice. No heuristic algorithm performs best on all instances, and their overall quality is unclear.

2.2 Vertex Clique Cover

The vertex clique cover (VCC) problem is NP-hard, and closely related to the maximum independent set and graph coloring problems. The size of a minimum VCC (also called the clique cover number) $\theta(G)$ is lower bounded by the size of a maximum independent set (the independence number $\alpha(G)$) and equivalent to the chromatic number of the complement graph, $\chi(\overline{G})$. There is a rich line of research on the graph coloring problem, which seeks to compute the chromatic number; many of the theoretical results for the VCC problem come via the graph coloring problem. The fastest exact exponential-space algorithm for computing the chromatic number on an n -vertex graph has time $O^*(2^n)$ (where O^* hides polynomial factors) using a generalization of the exclusion-inclusion principle [24], and in polynomial space the problem can be solved in time $O(2.2356^n)$ [18]. Furthermore, there exists no polynomial-time algorithm with approximation ratio better than $n^{1-\epsilon}$ for $\epsilon > 0$ unless $P = NP$ [34].

In terms of data reduction, we note that it is unlikely that a kernel exists when parameterized on the (vertex) clique cover number. Deciding if a cover with even 3 cliques exists is NP-complete (since 3-coloring the complement is NP-hard). A polynomial kernel would have size $O(1)$ and could be computed in polynomial time. Solving the kernel with brute-force computation would solve the VCC problem in polynomial time, implying $P = NP$. However, in practice, the VCC problem can be solved on large, sparse real-world graphs using the data reductions by Strash and Thompson [31].

2.3 Data Reduction in Practice for Related Problems

Other classical NP-hard problems have large suites of data reductions that are effective in practice, including minimum vertex cover [4, 15], maximum cut [16], and cluster editing [7]. Popular data reductions include variations of simplicial vertex removal, degree-2 folding, twin, domination, unconfined, packing, crown, and linear-programming-relaxation-based reductions [4]. Even the simplest reductions can be highly effective when combined with

other techniques [10, 30]. Data reductions are most effective in sparse graphs, which are the graphs that we consider here. Finally, similar to what we propose here, other NP-hard problems are solved by first applying a problem transformation. In particular, algorithms for minimum dominating set problem first transform the problem to an instance of the set cover problem [33].

3 Preliminaries

We consider a simple finite undirected graph $G = (V, E)$ with vertex set V and edge set $E \subseteq \{\{u, v\} \mid u, v \in V\}$. For brevity, we denote by $n = |V|$ and $m = |E|$ the number of vertices and edges in the graph, respectively. When more specificity is needed, we denote the vertex and edge set of a graph G by $V(G)$ and $E(G)$ respectively. We say two vertices $u, v \in V$ are *adjacent* (or *neighbors*) when $\{u, v\} \in E$. The *open neighborhood* of a vertex $v \in V$ is the set of its neighbors $N(v) := \{u \mid \{u, v\} \in E\}$, and the *degree* of v is $|N(v)|$. We further define the *closed neighborhood* of a vertex $v \in V$ to be $N[v] := N(v) \cup \{v\}$. Extending these definitions, the open neighborhood of a set $A \subseteq V$ is $N(A) := \bigcup_{v \in A} N(v) \setminus A$ and the closed neighborhood of A is $N[A] := \bigcup_{v \in A} N[v]$. The subgraph of G induced by a vertex set $V' \subseteq V$, denoted $G[V']$, has vertex set V' and edge set $E' = \{\{u, v\} \in E \mid u, v \in V'\}$. The *degeneracy* d of a graph G is the smallest value such that every nonempty subgraph of G has a vertex of degree at most d [27]. It is possible to order the vertices of a graph G in time $O(n + m)$ so that every vertex has d or fewer neighbors later in the ordering; such an ordering is called a *degeneracy ordering* [14].

A vertex set $C \subseteq V$ is called a *clique* if, for each pair of distinct vertices $u, v \in C$, $\{u, v\} \in E$. A set of cliques \mathcal{C} is called an *edge clique cover (ECC)* (or just a *clique cover*) of G if for every edge $\{u, v\} \in E$ there exists at least one $C \in \mathcal{C}$ such that $\{u, v\} \subseteq C$. That is, there is some clique in \mathcal{C} that *covers* $\{u, v\}$. The set of cliques \mathcal{C} is said to cover the graph G . An ECC of minimum cardinality is called a *minimum ECC*, and its cardinality is denoted by $\theta_E(G)$, called the edge clique cover number.

Similarly, in a *vertex clique cover (VCC)*, every vertex $v \in V$ is covered by some clique. The cardinality of a minimum VCC is the *clique cover number*, denoted by $\theta(G)$.

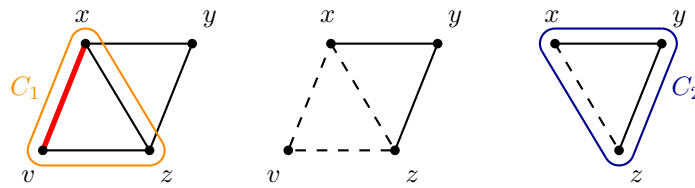
4 Existing Tools Discussion

In this section, we discuss basic tools that we will use to solve the ECC problem, together with insights into their behavior on sparse graphs. We begin by describing the existing ECC data reductions by Gramm et al. [19]. We then discuss how to convert an input ECC instance to an equivalent VCC instance using the technique of Kou et al. [25]. We will extend these tools to develop our full algorithm combining ECC and VCC reductions in the next section.

4.1 ECC Reduction Rules

Gramm et al. [19] introduce four data reduction rules that either cover edges by a clique known to be in a minimum cardinality ECC or add edges to the input graph G . Once all of a vertex v 's incident edges are covered, v can be removed from the graph.

With each edge $\{u, v\}$, Gramm et al. store the common neighbors in G , denoted by $N_{\{u, v\}}$, as well as a count $c_{\{u, v\}} = |E(G[N_{\{u, v\}}])|$ of the edges between common neighbors. These values are updated in ECC Reduction 1 and are used in ECC Reduction 2.



■ **Figure 1** Illustrating Gramm et al. [19]’s data reductions: (left) edge $\{v, x\}$ is in exactly one maximal clique C_1 , triggering ECC Reduction 2 and covering edges $\{v, x\}$, $\{v, z\}$, and $\{x, z\}$ (middle). Vertex v can then be removed with ECC Reduction 1. The remaining triangle (right) is covered by clique C_2 by applying ECC Reduction 2 to either $\{x, y\}$ or $\{y, z\}$.

Throughout the application of data reductions, vertices are removed from G and edges are covered. Figure 1 illustrates an example of the reductions. Set let edge set $E' \subseteq E$ be the set of uncovered edges (by extension, $E \setminus E'$ are the covered edges). The graph G only changes when a vertex is removed.

We note that the data reductions by Gramm et al. [19] are particularly effective for sparse graphs; however, the original data reductions were not written with efficiency in mind. Although these reductions have (very) slow theoretical running times, we offer insights as to why their reductions are faster in practice than indicated by the theoretical running time from Gramm et al. [19].

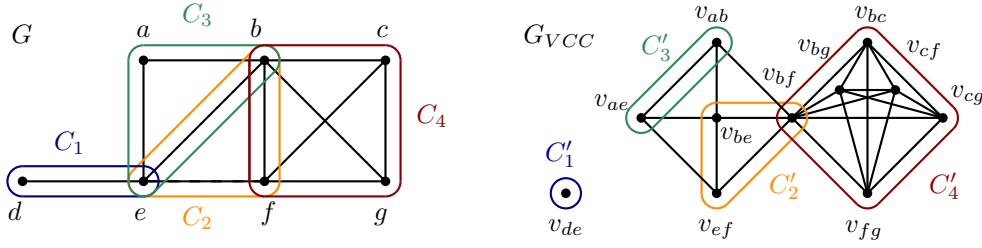
► **ECC Reduction 1** ([19]). *Let $v \in V$ be a vertex whose incident edges are all covered (i.e., in $E \setminus E'$). Then remove v from the graph G , along with its incident edges, and update values $c_{\{w,x\}}$ and $N_{\{w,x\}}$ for all uncovered edges $\{w, x\} \in E'$ whose endpoints are both adjacent to v , i.e., $\{w, x\} \subseteq N(v)$.*

As noted by Gramm et al. [19], this step can be applied to all vertices in running time $O(n^2m)$ by iterating over each vertex v and updating $N_{\{u,w\}}$ for all edges $\{u, w\} \in E'$ whose endpoints are adjacent to v . However, in sparse graphs the maximum degree in G , denoted Δ , is significantly smaller than n . Each edge $\{u, w\}$ has its set $N_{\{u,w\}}$ updated at most Δ times, taking $O(\Delta)$ time to update each time, giving a more reasonable running time of $O(\Delta^2m)$. We note that with adjustments, this can be run faster by enumerating all triangles in G in time $O(dm)$ using the triangle listing by Chiba and Nishizeki [11] and updating $N_{\{u,w\}}$ for edge $\{u, w\}$ in each triangle; however, this is a different implementation than that done by Gramm et al. [19] and not our focus here.

► **ECC Reduction 2** ([19]). *Let edge $\{u, v\} \in E'$ be an uncovered edge such that $c_{\{u,v\}} = \binom{|N_{\{u,v\}}|}{2}$ (i.e., the edge is in exactly one maximal clique in G'). Then $C = N_{\{u,v\}} \cup \{u, v\}$ is a maximal clique of G in some minimum ECC. Add the clique C to the clique cover, and cover any uncovered edges in C in G .*

As noted by Gramm et al. [19], ECC Reduction 2 can be implemented in time $O(n^2m)$ by iterating over each edge $\{u, v\} \in E'$, checking if $c_{\{u,v\}} = \binom{|N_{\{u,v\}}|}{2}$ in $O(1)$ time, and covering the edges of $\{u, v\}$ ’s clique in time $O(n^2)$ time.

However, when run on sparse graphs, which tend to have low degeneracy d [14], this rule is much faster. Graphs with degeneracy d have cliques of at most $d + 1$ vertices, therefore the reduction is only triggered when $|N_{\{u,v\}}| < d$. Hence, in practice, we should observe the much faster running time of $O(d^2m)$.



■ **Figure 2** An example graph G with minimum ECC and its transformed graph G_{VCC} with a corresponding minimum VCC.

Gramm et al. introduce two more ECC reductions (which we'll refer to as ECC Reductions 3 and 4), however, they are more complex and less effective than ECC Reductions 1 and 2 in practice. Experiments by Gramm et al. show that these reductions are very slow, and only improve the search tree size by a constant factor when incorporated into branch and reduce [19]. We therefore omit them from our discussion and implementation.

4.2 Transforming an ECC Instance to a VCC Instance

Kou et al. [25] showed that the ECC problem is NP-hard via a polynomial-time reduction from the VCC problem. Furthermore, they gave a polynomial-time reduction *to* the VCC problem, which we use as the basis of our transformation. We describe their transformation and briefly justify why it works.

Given an input graph $G = (V, E)$ for the ECC problem Kou et al. [25] transform G to a new graph $G_{VCC} = (V_{VCC}, E_{VCC})$ that is an equivalent VCC instance as follows. For each edge $\{x, y\} \in E$, create a new vertex $v_{xy} \in V_{VCC}$, then add an edge $\{v_{xy}, v_{wz}\}$ to E_{VCC} if and only if there exists a clique C in G containing both $\{x, y\}$ and $\{w, z\}$. Now, for any given subset $C \subset V_{VCC}$, C is a clique in G_{VCC} iff its vertices' corresponding edges in E also induce a clique in G . Hence, a minimum cardinality VCC in G_{VCC} corresponds to a minimum cardinality ECC in G . (See Figure 2.)

To determine if two edges are in a clique together in G , Kou et al. [25] make the following observation:

► **Observation 1** ([25]). *Two distinct edges $\{x, y\}, \{w, z\}$ are in a clique together in G iff $\{x, y\}$ and $\{w, z\}$ are incident and $\{x, y\} \cup \{w, z\}$ induce a triangle, or $\{x, y\}$ and $\{w, z\}$ are not incident and $\{w, x, y, z\}$ form a 4-clique.*

However, there is a clear issue when using this transformation: G_{VCC} can be very large. We briefly discuss its size and sparsity.

4.2.1 The Effect of Transformation on Graph Size and Sparsity

In the worst case, the size of G_{VCC} is a quadratic factor larger than G . Indeed, if the graph G is itself the complete graph K_n , on n vertices and $\Theta(n^2)$ edges, then the transformed graph is the complete graph $K_{n(n-1)/2}$ having $\Theta(n^2)$ nodes and $\Theta(n^4)$ edges. However, we show that the size of the graph only increases by a factor of $O(d^2)$, where d is the degeneracy of the graph. Real-world sparse graphs have low degeneracy [14], and thus this is a significant improvement over the worst case.

► **Theorem 2.** *Let the degeneracy of $G = (V, E)$ be d . Then $|V_{VCC}| = m \leq dn$ and $|E_{VCC}| = O(d^2m)$.*

Proof. By construction $|V_{VCC}| = m$; hence, to bound $|V_{VCC}|$, we bound the number of edges in G . In a degeneracy ordering of the graph, each vertex has at most d later neighbors in the ordering. Therefore, $|V_{VCC}| = m \leq dn$. To bound $|E_{VCC}|$, we compute an upper bound on the number of triangles and 4-cliques in G . Following Observation 1, each edge in E_{VCC} corresponds to a pair of edges in E contained in a triangle or a pair of non-incident edges in a 4-clique. Each triangle has 3 edges, and each 4-clique has 3 pairs of non-incident edges. Therefore, an asymptotic upper bound of the number of triangles and 4-cliques in G gives an upper bound for $|E_{VCC}|$.

In any triangle, some vertex must come first in a degeneracy ordering, and can be in a triangle with at most $\binom{d}{2}$ of its at most d later neighbors. Therefore each vertex is in $O(d^2)$ triangles with its later neighbors and, summing up over all vertices, contributes at most $O(d^2n)$ edges to E_{VCC} . Similarly, for each edge $\{u, v\}$ we count the number of 4-cliques it is in with (non-incident) edges that come lexicographically after it in the degeneracy ordering. The number of triangles the second vertex can be in with later neighbors is $\binom{d}{2}$ and hence the edge is in at most $O(d^2)$ 4-cliques with v 's at most d later neighbors, giving at most $O(d^2m)$ 4-cliques total. Thus, we conclude that $|E_{VCC}| = O(d^2m)$. ◀

Thus, G_{VCC} has at size at most $O(d^2m)$, a factor $O(d^2)$ larger than G . As a consequence, the average degree of the graph may increase, but by no more than a factor $O(d)$: whereas G has average degree $2|E|/|V| = O(dn)/n = O(d)$, graph G_{VCC} has average degree $2|E_{VCC}|/|V_{VCC}| = O(d^2m)/m = O(d^2)$. Therefore, for input graphs with small degeneracy, the transformed graph is expected to be sparse as well.

However, even if the degeneracy d is small, the graph G_{VCC} may be very large in practice. Hence, to use this transformation, we require techniques to keep the graph size manageable.

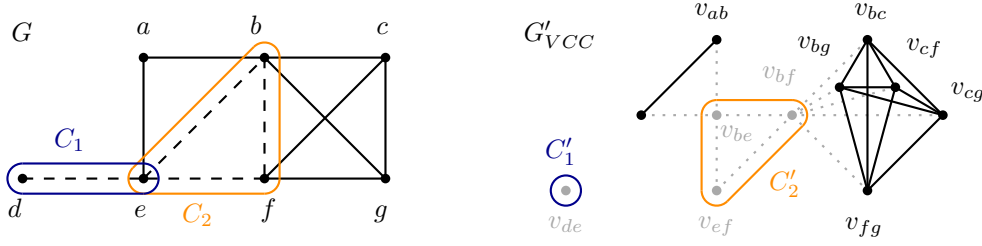
5 Synergistic Reductions: Applying ECC and VCC Reductions

We propose to handle the blow-up by Kou et al. [25] by applying both ECC and VCC reductions to the problem, which we call *synergistic* data reduction. We first show how to adjust the transformation to work on reduced ECC instances, after which we can apply VCC reductions. We also explore the possibility of “lifting” VCC reductions to ECC reductions.

5.1 Transforming a Partially-Covered ECC Problem Kernel

Recall that the data reductions from Gramm et al. [19] result in a graph in which some edges are covered, which is not supported by the transformation of Kou et al. [25]. While it is tempting to modify the transformation to operate on *only* the uncovered edges E' , this does not necessarily result in an equivalent instance, as already-covered edges may still be needed to compute a minimum number of cliques covering E' . For instance, in Figure 1, covering edges $\{x, y\}$ and $\{y, z\}$ with the single clique C_2 uses the already-covered edge $\{x, z\}$.

One way to correct for this is to first perform the transformation on the entire graph $G = (V, E)$, and then take the subgraph induced by the vertices corresponding to uncovered edges in E' . However, this strategy is slow when the edge set E is significantly larger than E' . We show that it is possible to perform the transformation without making vertices for all edges in E . Note that since all that remains is to cover the edges in E' , we now focus on covering all E' using a minimum number of cliques in G . Taken together with already-chosen cliques from ECC reductions, this gives us a covering of all of G . (See Figure 3.)



■ **Figure 3** A partially-covered graph G with cliques C_1, C_2 already added to the cover, and its transformed graph G'_{VCC} . Grayed vertices and (dotted) edges are those in G_{VCC} , but not G'_{VCC} .

We transform G to a graph $G'_{VCC} = (V'_{VCC}, E'_{VCC})$, where $V'_{VCC} = \{v_{xy} \mid \{x, y\} \in E'\}$ and $E'_{VCC} = \{\{v_{xy}, v_{wz}\} \mid \{x, y\}, \{w, z\} \in E' \text{ and } \{x, y\} \cup \{w, z\} \text{ is a clique in } G\}$. This transformation preserves cliques in G that cover edges in E' , which we capture with the following observation.

► **Observation 3.** *If C' is a clique in G'_{VCC} then $C = \cup_{v_{xy} \in C'} \{x, y\}$ is a clique covering $|C'|$ edges of E' in G .*

Furthermore, the transformation gives a correspondence between cliques covering E' in G and VCCs in G'_{VCC} .

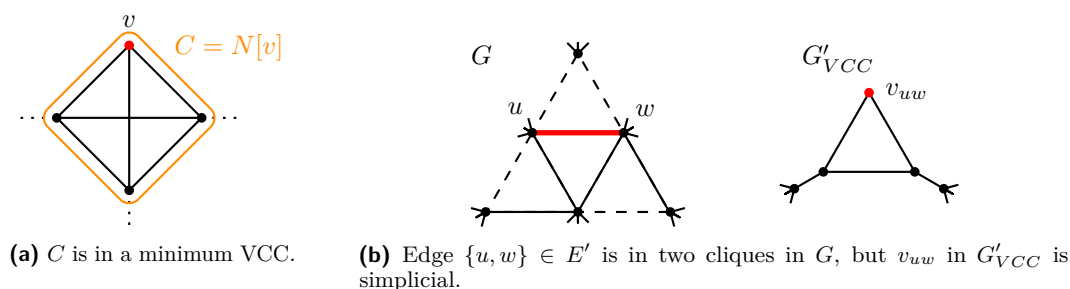
► **Theorem 4.** *If C' is a VCC in G'_{VCC} then $C = \{\cup_{v_{xy} \in C'} \{x, y\} \mid C' \in C'\}$ is a set of cliques covering E' in G .*

Proof. By Observation 3, every clique $C' \in C'$ in G'_{VCC} corresponds to a clique $C = \cup_{v_{xy} \in C'} \{x, y\}$ in G that covers its corresponding edges of E' . Hence, a VCC that covers all V'_{VCC} of G'_{VCC} corresponds to a collection of cliques covering all edges E' in G . ◀

Note that in Theorem 4, $|C| = |C'|$. Hence, a minimum VCC in G'_{VCC} corresponds to a minimum-cardinality set of cliques covering E' in G . This transformation gives us a technique for computing a minimum ECC: First apply the data reductions of Gramm et al., then compute G'_{VCC} and use VCC reductions and any VCC solver to compute a minimum VCC in G'_{VCC} , giving us cliques covering E' in G and, ultimately an entire ECC of G . While applying VCC reductions to G'_{VCC} may produce a smaller instance, these data reductions are not actually producing a smaller *ECC instance*. However, as we now show, we can also “lift” some VCC reductions to the ECC problem, by keeping the equivalence between cliques in the transformation in mind.

5.2 Lifting VCC Reduction Rules to ECC

Unlike the ECC problem, the VCC problem has many data reduction rules [31]. These include reductions based on simplicial vertices, dominance, twins, degree-2 folding, and crowns. We briefly discuss two classes of VCC reductions: clique-removal-based rules and folding-based rules. We place them in the context of the ECC problem, discuss whether it is viable to “lift” them to the ECC problem, and consider if the graph transformation is needed. By combining existing ECC reductions with VCC reductions, we aim to reduce ECC instances even further.



■ **Figure 4** The simplicial vertex VCC reduction can be applied after transforming G to G'_{VCC} .

5.2.1 Clique-Removal-Based VCC Reductions

We call a VCC reduction that removes a set of cliques from the graph a *clique-removal-based* rule. Four VCC reductions (simplicial vertex, dominance, twin removal, and crown) are clique-removal-based rules [31]. Such rules can be easily transformed into an ECC reduction: By the equivalence between cliques in the problem transformation, stated in Observation 3, removing a clique in G'_{VCC} is equivalent to covering its corresponding clique in G . Thus, to apply clique-removal-based VCC reductions directly to the ECC problem, we can compute G'_{VCC} , apply any clique-removal-based rules, and then cover these cliques in G . We capture this with the following theorem.

► **Theorem 5.** *Any clique-removal-based VCC reduction can be lifted to an ECC reduction.*

Of course, we could try to apply these reductions more efficiently to G directly. We discuss two clique-removal-based VCC reductions and discuss whether they are worth implementing for ECC directly, or if we should transform G to G'_{VCC} first.

Simplicial Vertex Reduction

A vertex v is *simplicial* if $N[v]$ forms a clique. In this case, the clique $C = N[v]$ is in some minimum VCC. (See Figure 4a.)

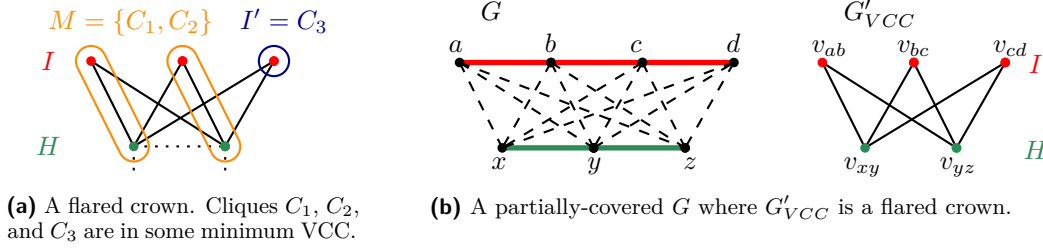
► **VCC Reduction 1** (Simplicial Vertex Reduction [31]). *Let $v \in V$ be a simplicial vertex. Then $C = N[v]$ is a clique in some minimum VCC. Add C to the clique cover and remove C from the graph.*

Applying VCC Reduction 1 on G'_{VCC} is reminiscent of applying ECC Reduction 2 on the untransformed graph G . While it is true that for $\{u, w\} \in E'$, if $N_{\{u, w\}}$ is a clique in G , then v_{uw} is simplicial in G'_{VCC} , the converse is not true in general. Hence, VCC Reduction 1 is more powerful. Consider the counterexample in Figure 4b. Vertex v_{uw} is simplicial in G'_{VCC} , but $\{u, w\} \in E'$ is in two cliques of G .

Thus, we have a new data reduction for the ECC problem, which subsumes ECC Reduction 2:

► **ECC Reduction 5** (Lifted Simplicial Vertex Reduction). *Let edge $\{u, w\} \in E'$ and let set $C = \{x, y \in V \mid \{x, y\} \in E' \text{ and } \{u, w\} \cup \{x, y\} \text{ is a clique in } G\}$ be the set of vertices of edges in some clique with $\{u, w\}$. If C is a clique, then add C to the clique cover, and cover any uncovered edges of C in G .*

To apply our lifted reduction, we could of course first compute G'_{VCC} and then apply VCC Reduction 1. However, we can also apply it directly to G with a slight modification to ECC Reduction 2. For each edge $\{u, w\} \in E'$ compute the common neighborhood $N_{\{u, w\}}$. Instead of checking that the common neighborhood is a clique, collect the uncovered edges



■ **Figure 5** The crown removal VCC reduction can be applied after transforming G to G'_{VCC} .

between vertices in $N_{\{u,w\}}$, and check if they induce a clique. Since $|N_{\{u,w\}}| \leq \Delta$, it takes $O(d\Delta)$ to collect uncovered edges by iterating through the at most d later neighbors of each vertex, which dominates the running time of this step. Exhaustively applying the reduction to all edges takes time $O(d\Delta m)$, which is slightly slower than the $O(d^2 m)$ time for ECC Reduction 2.

Is it worth applying ECC Reduction 5 directly to G , or should we first transform G and run VCC Reduction 1 instead? The transformation can be done in time $O(d^2 m)$ by enumerating all of the triangles and 4-cliques of G [11], hence performing the transformation is faster in theory than applying ECC Reduction 5 to G directly. However, in G'_{VCC} the largest clique may have as many as $\Theta(d^2)$ vertices and $\Theta(d^4)$ edges since a clique of size $d + 1$ in G has $\Theta(d^2)$ edges in G . Therefore, the time to apply VCC Reduction 1 for each of the m vertices of G'_{VCC} is $O(d^4 m)$. Thus, in theory, it is more efficient to apply ECC Reduction 5 directly, rather than first applying a conversion.

However, there are compelling reasons to perform the conversion. For one, most implementations of simplicial vertex reductions limit the degree of the vertex considered – in some cases to as small as two – since large-degree simplicial vertices rarely appear in sparse graphs. Therefore, in practice, it is unlikely that we would observe this large running time. However, a more compelling reason to perform the transformation is that there are two highly effective VCC reductions that we do not currently know how to apply directly to G . The first is the crown removal reduction (a clique-removal-based reduction) and the second is the degree-2 folding-based reduction.

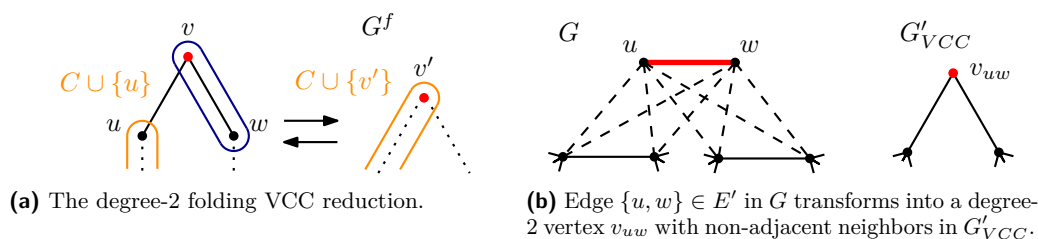
Crown Removal Reduction

The crown removal reduction is arguably one of the most powerful data reductions, successfully reducing sparse instances for the minimum vertex cover and VCC problems [3, 4, 10].

In a pair of vertex sets (H, I) , H is called a *head* and I a *crown* if: I is an independent set, $N(I) = H$, and there exists a matching from H to I of size $|H|$. Figure 5a shows a crown structure. Note that, due to the matching requirement, $|I| \geq |H|$. If $|I| = |H|$, the crown is called *straight*, otherwise it is *flared*. Strash and Thompson [31] give the following data reduction for the VCC problem, adapting a data reduction for the dual coloring problem [17].

► **VCC Reduction 2** (Crown Removal Reduction [31]). *Let (H, I) be a head and crown with matching M and unmatched vertices $I' \subseteq I$. Then add cliques in M and I' to the clique cover and remove $N[I]$ from the graph. (See Figure 5a.)*

Note that it is possible to identify flared crowns by applying a reduction based on an LP relaxation, originally introduced for the minimum vertex cover problem by Nemhauser and Trotter [29]. A variant of this algorithm due to Iwata et al. [21] identifies and removes *all* flared crowns at once by computing a maximum matching on a bipartite graph with $2n$ vertices and $2m$ edges using the Hopcroft-Karp algorithm [20] with running time $O(m\sqrt{n})$.



■ **Figure 6** The degree-2 folding VCC reduction can be applied after transforming G to G'_{VCC} .

As Figure 5b illustrates, after exhaustively applying Gramm et al.'s [19] ECC reductions it is possible to have a crown structure after transforming to G'_{VCC} . Thus, lifting the crown removal reduction can further reduce an ECC instance. However, algorithms for computing a maximum matching for the LP relaxation use an explicit representation of G'_{VCC} and therefore it is unclear how to run this reduction without first transforming G to G'_{VCC} . The transformation and maximum matching can be computed in time $O(d^2m + d^2m\sqrt{m}) = O(d^2m^{3/2})$, since there are $O(m)$ vertices and $O(d^2m)$ edges in G'_{VCC} . We leave the question of whether the LP relaxation reduction can be more efficiently lifted to an ECC reduction as an open problem.

5.2.2 Folding-Based VCC Reductions

In contrast to clique-removal-based reductions, *folding-based* reductions contract a subset $S \subseteq V$ of vertices into a single vertex v' . *Folding* S produces a new graph $G^f = (V^f, E^f)$ with $V^f = (V \setminus S) \cup \{v'\}$ and $E^f = (E \setminus \{\{v, x\} \in E \mid v \in S, x \notin S\}) \cup \{\{v', x\} \mid \exists v \in S, x \notin S, \{v, x\} \in E\}$. We discuss the connections between the ECC problem and the simplest folding-based reduction, folding vertices of degree two.

Degree-2 Folding

The degree-2 folding reduction for VCC contracts a degree-2 vertex v with non-adjacent neighbors u and w that are *crossing independent* [31]. That is, for each edge $\{x, y\} \subseteq N(u) \cup N(w)$ either $\{x, y\} \subseteq N(u)$ or $\{x, y\} \subseteq N(w)$. This condition ensures that no spurious cliques are formed after folding. A vertex v meeting these conditions is *foldable*.

► **VCC Reduction 3** (Degree-2 Folding Reduction [31]). *Let $v \in V$ be a foldable degree-2 vertex with non-adjacent neighbors $N(v) = \{u, w\}$. Let G^f be the graph obtained by folding $\{v, u, w\}$. Let \mathcal{C}^f be a minimum VCC of G^f with clique $C_{v'} \in \mathcal{C}^f$ covering vertex v' and let $C = C_{v'} \setminus \{v'\}$. Then, the clique cover*

$$\mathcal{C} = \begin{cases} (\mathcal{C}^f \setminus \{C_{v'}\}) \cup \{C \cup \{u\}, \{v, w\}\} & \text{if } C \subseteq N(u), \\ (\mathcal{C}^f \setminus \{C_{v'}\}) \cup \{C \cup \{w\}, \{v, u\}\} & \text{otherwise,} \end{cases}$$

is a minimum VCC of G .

See Figure 6a for an example of the degree-2 folding VCC reduction. We note that the transformation from an ECC instance to a VCC instance by Kou et al. [25] does not produce any degree-2 vertices with non-adjacent neighbors, as edges forming a triangle or 4-clique in G form a triangle or 6-clique in G_{VCC} . However, our transformation with covered edges can result in such vertices (see Figure 6b). Thus, the degree-2 folding VCC reduction can be used to further reduce the instance when applied to G'_{VCC} .

We leave as an open problem whether folding-based rules can be lifted to new ECC reductions; we conjecture that it is possible to lift at least degree-2 folding. However, given how effective the degree-2 folding reduction is in practice for the VCC problem, we highly recommend applying it, even though it incurs the overhead of the transformation to G'_{VCC} .

5.3 Wrapping It All Up

With the tools in this section in hand, we have a clear path to solving the ECC problem on sparse graphs: first apply the data reductions due to Gramm et al. [19], then transform the partially-covered graph into a VCC instance, then apply VCC reductions and solve what remains with any VCC solver. We next perform experiments to evaluate this method.

6 Experimental Evaluation

We now compare our technique to the state of the art through extensive experiments on both synthetic instances and real-world graphs.

6.1 Experimental Setup

We implemented the ECC reductions and ECC to VCC transformation in C++ and integrated our methods with the VCC reductions and VCC algorithms by Strash and Thompson² [31], which we then compiled with g++ version 11 using the -O3 optimization flag. Our source code is available under the open source MIT license³. All experiments were conducted on Hamilton College’s High Performance Computing Cluster (HPCC), on a machine running CentOS Linux 7.8.2003, with four Intel Xeon Gold 6248 processors running at 2.50GHz with 20 cores each, and 1.5TB of memory. Each algorithm is run sequentially on its own core.

We run experiments on six different algorithms. Gramm is the original branch-and-reduce code by Gramm et al. [19] written in OCaml, which we compiled with ocamlc version 3.10.2, and provided a sufficiently large stack size due to its heavy use of recursion. We implement three algorithms in C++ that first exhaustively apply ECC Reductions 1 and 2, perform a problem reduction to a VCC instance, apply VCC reductions, and then run a VCC solver: Redu³BnR solves with the VCC branch-and-reduce algorithm by Strash and Thompson [31], Redu³IG solves with the VCC iterated greedy (IG) heuristic algorithm by Chalupa [9], and Redu³ILP solves with an assignment-based ILP formulation [22, 28] for VCC and Gurobi version 9.5.1. Finally, the two heuristic algorithms Conte [12] and EO-ECC [1] are from their respective authors and are compiled with javac version 8 and g++ version 11 with -O3, respectively. Unless stated otherwise, we run each algorithm with a 24-hour time limit. Our stated running times do not include I/O time such as graph reading and writing.

In our tables, “Kernel” denotes the relevant size of the graph after reductions as either uncovered edges (Gramm) or vertices remaining (for VCC-based algorithms). “Time” is the time (in seconds) the solver takes to exactly solve the instance. A “–” indicates that the solver did not finish in the 24-hour time limit. **Bold** values indicate the value is the smallest among all algorithms in the table.

We run our experiments on randomly-generated instances as well as real-world graphs.

² <https://github.com/darrenstrash/ReduVCC>

³ <https://github.com/darrenstrash/Redu3ECC>

Erdős-Rényi Graphs. We generate 70 instances of varying density using the $G(n, p)$ model of generating an n -vertex graph where each edge is selected independently with probability p . We use values of n that are powers of two from 64 to 2048, with two different values of p for each to show the effect of density on the tested algorithms. We generate 5 graphs with each n, p pair using different random seeds to observe the behavior of algorithms on multiple instances of similar size and density.

Real-World Instances. We run our experiments on 52 large, sparse, complex networks from the Stanford Network Data Repository (SNAP)⁴, the Laboratory for Web Algorithmics (LAW)⁵, and the Koblenz Network Collection (KONECT)⁶. These graphs include citation networks, web-crawl graphs, and social networks; the largest graph has 18M vertices, and most graphs follow a scale-free degree distribution: there are many low degree vertices and few high degree vertices. The number of vertices and edges for each instance can be found with experimental results in Tables 2 and 3.

6.2 Results on Synthetic Instances

We begin by comparing the performance of **Gramm** and **Redu³BnR** on synthetic instances generated with the Erdős-Rényi $G(n, p)$ model. In Table 1, we present the average kernel size and running time executing **Gramm** and **Redu³BnR** on the 5 instances of each pair of n and p . We disable ECC Reduction 3 in **Gramm**, since this configuration enables it to solve the largest number of synthetic instances within the time limit.

Focusing on running time, **Gramm** and **Redu³BnR** are equally matched on very sparse graphs, quickly solving many instances in significantly less than one second. Though, as the density increases even slightly, which can be seen when fixing n but increasing p , **Gramm** is no longer able to solve even small instances in a 24-hour time limit. However, on all instances, **Redu³BnR** easily computes exact solutions. The reason why is clear: on instances that **Gramm** is unable to solve, **Gramm**'s kernel is large (for the highest density instance with $n = 64, p = 0.2$, even a kernel of average size 100 is too large for **Gramm** to solve), whereas the VCC kernels for **Redu³BnR** are significantly smaller in all cases. Indeed, for the densest graphs of each value of n , **Gramm** is unable to solve every instance in 24 hours, but **Redu³BnR** solves all graphs in less than a second. This illustrates that the combined reduction power of ECC and VCC reductions is able to handle denser instances than running ECC reductions alone.

6.3 Solving Large Real-World Instances Exactly

We now see which graphs can be solved exactly by one of three algorithms: **Gramm**, **Redu³BnR**, and **Redu³ILP**. We disable ECC Reductions 3 and 4 in **Gramm**, since this configuration enables it to reduce all instances within the time limit. The results are presented in Table 2. **Gramm** was able to solve 12 of the 27 instances exactly; 10 of these graphs were solved because the kernel had 0 uncovered edges and the other two instances (**ca-CondMat** and **ca-GrQc**) had small kernels of less than 100 uncovered edges. However, **Gramm** exceeds the 24-hour time limit on the 15 other instances, even those with as few as 176 uncovered edges.

⁴ <https://snap.stanford.edu/data/>

⁵ <http://law.di.unimi.it/datasets.php>

⁶ <http://konect.cc/>

■ **Table 1** Results on small Erdős-Rényi graphs of varying density. A “*” indicates that not all runs finished in the 24-hour time limit, “–” indicates that no runs finished in the 24-hour time limit.

Graph		Gramm		Redu ³ BnR		
<i>n</i>	<i>p</i>	<i>m</i>	Kernel	Time (s)	Kernel	Time (s)
64	0.15	300	1	< 0.01	0	< 0.01
64	0.2	404	100	1 324.52*	10	< 0.01
128	0.1	805	0	< 0.01	0	< 0.01
128	0.15	1 218	491	–	51	0.03
256	0.075	2 433	42	0.02	0	< 0.01
256	0.1	3 264	1 105	–	12	0.02
512	0.05	6 557	137	0.21	1	0.02
512	0.065	8 513	2 281	–	5	0.04
1 024	0.0365	19 072	1 259	153.21*	4	0.08
1 024	0.0375	19 596	1 704	–	4	0.07
2 048	0.025	52 245	3 147	5.18	4	0.18
2 048	0.0275	57 488	7 235	–	5	0.20

In contrast, Redu³BnR solves 18 of the instances. On all instances, the kernel computed by Redu³BnR was smaller than that of Gramm, the smallest of which is on `zhishi-hudong-int`, which is reduced to 2% of the size of Gramm’s kernel. With the exception of three instances (`email-EuAll`, `web-NotreDame`, and `web-Stanford`), every instance was reduced to at most 10% of Gramm’s kernel size. However, the limitations of branch and reduce for the VCC problem begin to show on these instances. Similar to Gramm, Redu³BnR only finishes within the 24-hour time limit on graphs with kernel size less than 100, and therefore its success is largely due to the reduction of the input instance (a pattern observed in other problems [30]). On the other hand, the Gurobi solver with an ILP formulation is able to solve kernels of much larger size, even up to 536 209 vertices (in the case of `eu-2005`).

6.4 Solving Remaining Instances Heuristically

We now look at the instances that could not be solved in the 24-hour time limit by any exact method. The results are presented in Table 3. Nine instances were reduced to VCC within the time limit of 24 hours, and the remaining instances were too large to finish in the time limit (not in the table). After fully transforming the input ECC instance to a reduced VCC instance, we ran the iterated greedy approach IG due to Chalupa et al. [9], which we call Redu³IG, and compare its best solution with a lower bound from KaMIS, a state-of-the-art evolutionary algorithm for finding near-maximum independent sets on huge networks [26]. Four instances were solved to within 300 cliques of optimum, two of which (`soc-Slashdot0811` and `soc-Slashdot0902`) are within 100 cliques. The remaining instances are solved to within 6 000 cliques of optimum.

6.5 Summarizing the Quality of Existing Heuristic Solvers

Finally, using our exact results, we evaluate the quality of two heuristic solvers designed for large sparse graphs. We compare Conte, an algorithm by Conte et al. [12] designed for large sparse graphs, and EO-ECC by Abdullah et al. [1]. We run Conte and EO-ECC on all instances that were solved exactly (i.e., those from Table 2). The results are presented in Table 4.

■ **Table 2** Comparing exact algorithms Gramm, Redu³BnR, and Redu³ILP on real-world instances solved by at least one of the algorithms in a 24-hour time limit. Times marked with a “*” indicate that the algorithm’s speed was due to programming language differences and not algorithmic improvements.

Graph		Gramm		Redu ³ BnR		Redu ³ ILP	
Name	n	m	Kernel	Time (s)	Kernel	Time (s)	Time (s)
ca-AstroPh	18 772	198 050	2 837	–	0	0.33	0.33
ca-CondMat	23 133	93 439	62	1.74	0	0.10	0.10
ca-GrQc	5 242	14 484	9	0.15	0	0.02	0.02
ca-HepPh	12 008	118 489	491	–	0	0.16	0.16
ca-HepTh	9 877	25 973	176	–	0	0.03	0.03
cnr-2000	325 557	2 738 969	755 617	–	23 880	–	10 727.29
dblp-2010	326 186	807 700	868	–	0	1.98	1.98
dblp-2011	986 324	3 353 618	8 898	–	50	9.13	9.88
email-EuAll	265 214	364 481	20 648	–	5 064	–	6.99
eu-2005	862 664	16 138 468	5 555 826	–	536 209	–	12 966.59
p2p-Gnutella04	10 876	39 994	0	0.34	0	0.05*	0.05*
p2p-Gnutella05	8 846	31 839	0	0.23	0	0.05*	0.05*
p2p-Gnutella06	8 717	31 525	0	0.33	0	0.04*	0.04*
p2p-Gnutella08	6 301	20 777	261	–	17	0.04	0.06
p2p-Gnutella09	8 114	26 013	214	–	5	0.04	0.08
p2p-Gnutella24	26 518	65 369	0	0.91	0	0.10*	0.10*
p2p-Gnutella25	22 687	54 705	0	0.63	0	0.08*	0.08*
p2p-Gnutella30	36 682	88 328	0	1.27	0	0.09*	0.09*
p2p-Gnutella31	62 586	147 892	0	2.14	0	0.23*	0.23*
roadNet-CA	1 965 206	2 766 607	0	115.17	0	5.60*	5.60*
roadNet-PA	1 088 092	1 541 898	0	45.75	0	2.94*	2.94*
roadNet-TX	1 379 917	1 921 660	0	73.21	0	3.64*	3.64*
web-BerkStan	685 230	6 649 470	2 096 936	–	152 581	–	6 753.27
web-Google	875 713	4 322 051	266 455	–	16 440	–	35.58
web-NotreDame	325 729	1 090 108	98 861	–	14 553	–	20.10
web-Stanford	281 903	1 992 636	523 480	–	57 463	–	981.82
zhishi-hudong-int	1 984 484	14 428 382	1 175 068	–	26 536	–	568.26

■ **Table 3** Heuristic solutions for graphs that could not be solved exactly in 24 hours. “lb” is a lower bound on $\theta_E(G)$ from KaMIS, “ub” is the smallest clique cover computed by Redu³IG, and “Time” is the time in seconds for Redu³IG to reach this result.

Graph		KaMIS		Redu ³ IG	
Name	n	m	lb	ub	Time (s)
as-skitter	1 696 415	11 095 298	5 843 072	5 847 591	20 848.17
email-Enron	36 692	183 831	42 141	42 207	2 201.00
soc-Epinions1	75 879	405 740	185 544	186 384	18 064.79
soc-pokec-relationships	1 632 803	22 301 964	12 222 248	12 227 949	21 451.91
soc-Slashdot0811	77 360	469 180	328 018	328 079	3 073.75
soc-Slashdot0902	82 168	504 230	351 012	351 072	3 125.21
wiki-Talk	2 394 385	4 659 565	3 645 692	3 648 312	21 088.53
wiki-Vote	7 115	100 762	34 789	35 004	21 424.48
zhishi-baidu-relatedpages	415 641	2 374 044	1 372 941	1 373 912	9 989.00

61:16 Solving Edge Clique Cover Exactly via Synergistic Data Reduction

From among the 27 graphs, Conte solves five instances exactly. A further nine instances are solved within 50 cliques of optimal, and eight additional graphs are solved within 2000 of optimal. EO-ECC, on the other hand, solves eight instances exactly (a superset of Conte’s five) and solves these faster than Conte. Furthermore, EO-ECC finds 14 smaller solutions faster than Conte (Conte only finds four smaller solutions faster). However, a distinct negative is EO-ECC’s running time and solution quality on `cnr-2000`, `eu-2005`, and `web-BerkStan`, which is much worse than Conte. We conclude that Conte gives consistently fast results with reasonable solutions, and EO-ECC is sometimes very fast and accurate, and other times not.

■ **Table 4** Evaluation of the quality of heuristic solvers Conte and EO-ECC on all graphs with known edge clique cover number $\theta_E(G)$. “ub” is the solution found by the given algorithm, and “Time” is the algorithm’s time in seconds. Values of “ub” marked in **bold** indicates the algorithm found an optimal solution, with its time in **bold** if it did so faster than its competitor. Values of “ub” in *italics* indicate that an algorithm found an ECC smaller than its competitor, with its time in *italics* if it did so faster than its competitor.

Name	Graph G			Conte		EO-ECC	
	n	m	$\theta_E(G)$	ub	Time (s)	ub	Time (s)
ca-AstroPh	18 772	198 050	15 134	15 481	0.92	<i>15 373</i>	<i>0.50</i>
ca-CondMat	23 133	93 439	16 283	16 378	0.54	<i>16 307</i>	<i>0.07</i>
ca-GrQc	5 242	14 484	3 737	3 749	0.15	<i>3 739</i>	<i>0.01</i>
ca-HepPh	12 008	118 489	10 031	10 142	0.69	<i>10 097</i>	<i>0.35</i>
ca-HepTh	9 877	25 973	9 190	9 264	0.19	<i>9 212</i>	<i>0.02</i>
cnr-2000	325 557	2 738 969	752 118	<i>756 905</i>	<i>14.92</i>	763 365	2 820.97
dblp-2010	326 186	807 700	186 834	187 395	2.22	<i>186 968</i>	<i>0.44</i>
dblp-2011	986 324	3 353 618	707 773	713 219	13.56	<i>709 156</i>	<i>3.48</i>
email-EuAll	265 214	364 481	297 092	<i>298 943</i>	2.58	299 257	2.14
eu-2005	862 664	16 138 468	2 832 059	2 883 585	108.67	3 032 337	8 458.21
p2p-Gnutella04	10 876	39 994	38 491	38 491	0.29	38 491	0.04
p2p-Gnutella05	8 846	31 839	30 523	30 527	0.25	<i>30 525</i>	<i>0.04</i>
p2p-Gnutella06	8 717	31 525	30 322	30 327	0.26	<i>30 324</i>	<i>0.04</i>
p2p-Gnutella08	6 301	20 777	19 000	19 042	0.20	<i>19 012</i>	<i>0.03</i>
p2p-Gnutella09	8 114	26 013	24 117	24 150	0.24	<i>24 133</i>	<i>0.03</i>
p2p-Gnutella24	26 518	65 369	63 725	63 726	0.41	63 725	0.06
p2p-Gnutella25	22 687	54 705	53 367	53 367	0.33	53 367	0.05
p2p-Gnutella30	36 682	88 328	85 821	85 823	0.52	85 821	0.10
p2p-Gnutella31	62 586	147 892	144 478	144 478	0.83	144 478	0.15
roadNet-CA	1 965 206	2 766 607	2 537 936	2 537 945	17.90	2 537 936	1.02
roadNet-PA	1 088 092	1 541 898	1 413 370	1 413 370	10.62	1 413 370	0.69
roadNet-TX	1 379 917	1 921 660	1 763 295	1 763 298	13.48	1 763 295	0.89
web-BerkStan	685 230	6 649 470	1 834 074	<i>1 850 605</i>	<i>54.34</i>	1 903 872	2 089.25
web-Google	875 713	4 322 051	1 242 770	1 254 107	24.96	<i>1 251 672</i>	33.10
web-NotreDame	325 729	1 090 108	451 424	453 864	7.09	<i>453 805</i>	7.31
web-Stanford	281 903	1 992 636	562 417	<i>570 958</i>	<i>16.85</i>	591 957	326.92
zhishi-hudong-int	1 984 484	14 428 382	10 557 244	10 698 424	123.45	<i>10 678 121</i>	322.89
Summary (#optimal / #smaller and faster)				(5 / 4)		(8 / 14)	

7 Conclusion and Future Work

We introduced a technique to further reduce ECC problem instances via VCC data reductions, enabling us to solve large sparse real-world graphs that could not be solved before. Critical to this technique is the ability to transform reduced ECC instances to the VCC problem,

through a modification of the polynomial-time reduction of Kou et al. [25]. The combined reduction power of ECC and VCC reductions, which we call *synergistic* data reduction, produces significantly smaller kernels than ECC reductions alone. Of particular interest for future work is integrating data reduction rules with existing heuristic algorithms for the ECC problem, trying to implement a more efficient LP relaxation ECC reduction without a transformation, and to see if folding-based reductions can be lifted to the ECC problem.

References

- 1 Wali M. Abdullah and Shahadat Hossain. A sparse matrix approach for covering large complex networks by cliques. In Derek Groen, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Computational Science - ICCS 2022 - 22nd International Conference, London, UK, June 21-23, 2022, Proceedings, Part III*, volume 13352 of *Lecture Notes in Computer Science*, pages 505–517. Springer, 2022. doi:10.1007/978-3-031-08757-8_43.
- 2 Wali M. Abdullah, Shahadat Hossain, and Muhammad. A. Khan. Covering large complex networks by cliques—A sparse matrix approach. In D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang, editors, *Recent Developments in Mathematical, Statistical and Computational Sciences*, pages 117–127. Springer, 2021. doi:10.1007/978-3-030-63591-6_11.
- 3 Faisal N. Abu-Khizam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theor. Comput. Syst.*, 41(3):411–430, 2007. doi:10.1007/s00224-007-1328-0.
- 4 Tokuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609, Part 1:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.
- 5 Alper Atamtürk, George L. Nemhauser, and Martin W.P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40–55, 2000. doi:10.1016/S0377-2217(99)00015-6.
- 6 Mathieu Blanchette, Ethan Kim, and Adrian Vetta. Clique cover on sparse networks. In *2012 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 93–102. SIAM, 2012. doi:10.1137/1.9781611972924.10.
- 7 Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. A branch-and-bound algorithm for cluster editing. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2022.13.
- 8 Robert C. Brigham and Ronald D. Dutton. On clique covers and independence numbers of graphs. *Discrete Mathematics*, 44(2):139–144, 1983. doi:10.1016/0012-365X(83)90054-7.
- 9 David Chalupa. Construction of near-optimal vertex clique covering for real-world networks. *Computing and Informatics*, 34(6):1397–1417, 2015. URL: <http://www.cai.sk/ojs/index.php/cai/article/view/1276>.
- 10 Lijun Chang, Wei Li, and Wenjie Zhang. Computing a near-maximum independent set in linear time by reducing-peeling. *Proc. 2017 ACM International Conference on Management of Data (SIGMOD '17)*, pages 1181–1196, 2017. doi:10.1145/3035918.3035939.
- 11 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
- 12 Alessio Conte, Roberto Grossi, and Andrea Marino. Large-scale clique cover of real-world networks. *Information and Computation*, 270:104464, 2020. doi:10.1016/j.ic.2019.104464.
- 13 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Trans. Comput. Theory*, 6(2), May 2014. doi:10.1145/2594439.

- 14 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs in near-optimal time. *ACM J. Exp. Algorithmics*, 18, 2013. doi:10.1145/2543629.
- 15 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.
- 16 Damir Ferizovic, Demian Hesper, Sebastian Lamm, Matthias Mnich, Christian Schulz, and Darren Strash. Engineering kernelization for maximum cut. In *Proc. 2020 Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 27–41. SIAM, 2020. doi:10.1137/1.9781611976007.3.
- 17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 18 Serge Gaspers and Edward J. Lee. Faster graph coloring in polynomial space. In Yixin Cao and Jianer Chen, editors, *Proc. 23rd International Computing and Combinatorics Conference (COCOON 2017)*, volume 10392 of *LNCS*, pages 371–383. Springer, 2017. doi:10.1007/978-3-319-62389-4_31.
- 19 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *J. Exp. Algorithmics*, 13, February 2009. doi:10.1145/1412228.1412236.
- 20 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.
- 21 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1749–1761. SIAM, 2014. URL: <https://dl.acm.org/doi/10.5555/2634074.2634201>.
- 22 Adalat Jabrayilov and Petra Mutzel. New integer linear programming models for the vertex coloring problem. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 640–652. Springer, 2018. doi:10.1007/978-3-319-77404-6_47.
- 23 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 24 Mikko Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 583–590, 2006. doi:10.1109/FOCS.2006.11.
- 25 Lawrence T. Kou, Larry J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM*, 21(2):135–139, February 1978. doi:10.1145/359340.359346.
- 26 Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, August 2017. doi:10.1007/s10732-017-9337-x.
- 27 Don R. Lick and Arthur T. White. k -degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970. doi:10.4153/CJM-1970-125-1.
- 28 Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996. doi:10.1287/ijoc.8.4.344.

- 29 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975. doi:10.1007/BF01580444.
- 30 Darren Strash. On the power of simple reductions for the maximum independent set problem. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics (COCOON'16)*, volume 9797 of *LNCS*, pages 345–356. Springer, 2016. doi:10.1007/978-3-319-42634-1_28.
- 31 Darren Strash and Louise Thompson. Effective data reduction for the vertex clique cover problem. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, pages 41–53. SIAM, 2022. doi:10.1137/1.9781611977042.4.
- 32 Ahammed Ullah. Clique cover of graphs with bounded degeneracy. *CoRR*, abs/2108.09851, 2021. arXiv:2108.09851.
- 33 Johan M.M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011. doi:10.1016/j.dam.2011.07.001.
- 34 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007. doi:10.4086/toc.2007.v003a006.