Pruning Rules for Edge Clique Cover

Ben Kallus

Abstract

Covering the edges of a graph with the minimum number of complete subgraphs is an NP-hard problem. We expand upon a previous branch-and-reduce method for this solving problem by Gramm et al. with improvements to previously-known data reduction rules, as well as new pruning rules that significantly reduce the portion of the search tree to be explored.

Introduction

The problem of covering the edges of a graph with complete subgraphs, called the edge clique cover problem, has been of interest to graph theorists since at least the 1960s. More recently, applications of the problem have surfaced in compiler optimization, computational geometry, and applied statistics. The edge clique cover problem is NP-hard, indicating that it is unlikely that a polynomial-time algorithm for the problem will be found. The problem is also APX-hard, so an approximation algorithm accurate to within a constant factor of 2^k does not exist for any $k < 2$ unless P = NP. Thus, polynomial-time heuristic algorithms, such as the one introduced by Conte et al., can offer only no meaningful guarantees about accuracy. Gramm et al. introduced data reductions that provide a significant improvement over brute-force computation when solving the problem exactly in practice for sparse graphs, and are believed to be asymptotically optimal. However, their algorithm can have prohibitive running time on dense graphs, due in part to its exhaustively exploring and reexploring the search tree through iterative deepening. We expand upon the work by Gramm et al. with new pruning rules that, when employed within a branch-and-bound framework, allow for computing minimum edge clique covers with significantly improved running time.

Preliminaries

Definitions

A **graph** G is a collection of vertices $V(G)$ together with a collection of edges $E(G) \subseteq \{uv \mid u, v \in V(G)\}$. We consider only simple, undirected, unweighted graphs. Thus, each edge $e \in E(G)$ is a

2-element unordered subcollection of $V(G)$. Self-loops are not permitted, and any pair of distinct vertices in $V(G)$ is joined by at most one edge. The **neighborhood** of a vertex v, denoted $N(v)$, is the set of vertices $\{u \mid uv \in E(G)\}$. The **closed neighborhood** of a vertex v, denoted $N[v]$, is $N(v) \cup \{v\}$. The **complete graph** of order n, denoted K_n, is the unique (up to isomorphism) graph on n vertices in which $uv \in E(K_n)$ for all distinct $u, v \in V(K_n)$. A **subgraph** of a graph G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The subgraph of G **induced** by a vertex set $C \subseteq V(G)$, denoted $G[C]$, is the subgraph of G with $V(G[C]) = C$ and $E(G[C]) = \{uv \in E(G) \mid u, v \in C\}$. A **$k$-clique** in an undirected graph G is a set $C \subseteq V(G)$ of order k such that $G[C]$ is complete. For example, in the graph shown in Figure 1, $\{a, b, c\}$ is a 3-clique.

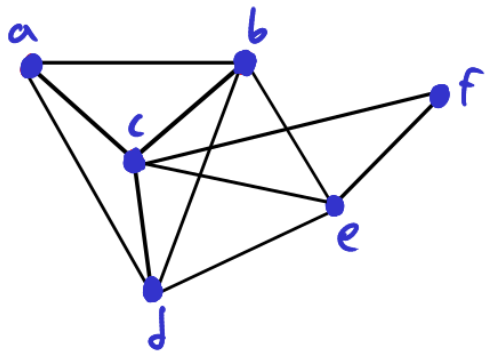

Figure 1:

Note that a clique may be a proper subset of another clique. For example, in Figure 1, the 3-clique $\{a, b, c\}$ is proper subset of the 4-clique $\{a, b, c, d\}$. The 4-clique $\{a, b, c, d\}$ is not a proper subset of any other clique in the graph, so we say $\{a, b, c, d\}$ is a **maximal clique**.

The Edge Clique Cover Problem

An edge clique cover for an undirected graph G is a collection $\mathcal{C}$ of cliques in G such that for all $uv \in E(G)$, $u, v \in C$ for some $C \in \mathcal{C}$.[1] For example, $\mathcal{C}_1 = \{\{a, b, c, d\}, \{c, e, f\}, \{d, e\}, \{b, e\}\}$ constitutes an edge clique cover for the graph from Figure 1.

Observe that $\mathcal{C}_1$ is not of minimum cardinality; the 2-cliques $\{d, e\}$ and $\{b, e\}$ can be substituted for the 3-clique $\{b, d, e\}$ to construct a new edge clique cover $\mathcal{C}_2$ of cardinality 3. The graph from Figure 1 cannot be covered by any fewer than 3 cliques, so $\mathcal{C}_2$ is an edge clique cover of minimum cardinality. Note that a graph may have many minimum edge clique covers; $\mathcal{C}_3 = \{\{a, b, c, d\}, \{c, e, f\}, \{b, c, d, e\}\}$ is another minimum edge clique cover for the graph from Figure 1. The **edge clique cover number** of a graph G, $\mathrm{ecc}(G)$, is the cardinality of a minimum edge clique cover for G.

[1]Note that isolated vertices have no effect on an edge clique cover, because they are incident to no edges. Thus, we consider only graphs without isolated vertices.

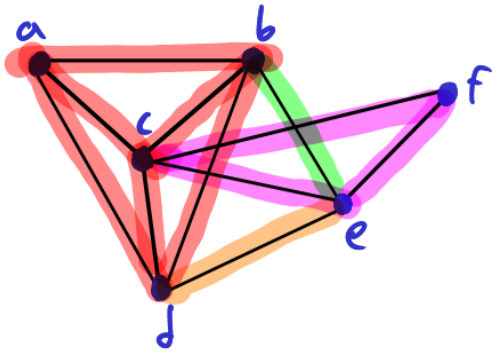

Figure 2: The edge clique cover $\mathcal{C}_1$

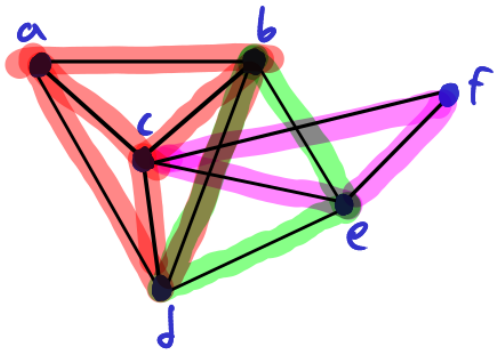

Figure 3: The edge clique cover $\mathcal{C}_2$

Related Work

The edge clique cover problem was first introduced as the intersection number problem in 1966. In 1973, Kellerman produced the first heuristic algorithm for the edge clique cover problem This was improved by Kou et al. in 1978. Additionally, Kou et al. showed through a reduction to and from the related vertex clique cover problem that the edge clique cover problem is NP-complete. The vertex clique cover problem, one of Richard Karp's original 21 NP-complete problems, is the problem of finding a minimum cardinality set of cliques in a graph such that every vertex in the graph is included in at least one clique. An arguably more famous formulation of the vertex clique cover problem is the graph coloring problem, which is equivalent to the vertex clique cover problem on the complement graph.

Existing Reduction Rules

Gramm et al. introduce four polynomial-time reduction rules for the edge clique cover problem. Each of these provides a condition under which either the problem instance can be reduced in size or difficulty, or a portion of the solution can be deduced. They can be applied in any order. We restate them here without proof.

Reduction Rule 1. *Remove isolated vertices and vertices incident only to covered edges.*

Reduction Rule 2. *If an uncovered edge participates in exactly one maximal clique C, cover C.*

Reduction Rule 3. *Suppose that there exists a vertex v such that $N[u] \subseteq N[v]$ for some vertex $u \in N(v)$. Then, we call u a prisoner of v. An exit of v is a vertex in $N(v)$ that is not a prisoner of v. If every prisoner of v is adjacent to at least one vertex other than v via an uncovered edge, and every exit of v is adjacent to at least one prisoner of v, remove v and insert it into any clique containing a prisoner of v covered from now on.*

Reduction Rule 4. *For each vertex v such that the subgraph H induced by $N(v)$ is disconnected, delete v and replace it with one new vertex for each component of H. Connect each new vertex to each vertex in its corresponding component with a new edge, maintaining the covered status of the corresponding edge from v. Once a cover has been computed, replace all instances of the new vertices created by this reduction rule in the clique cover with v.*

Contributions

Correction to Reduction Rule 3

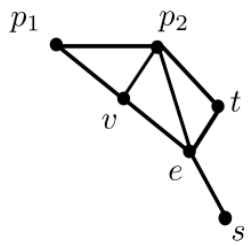

Figure 4: The graph G, with labeled vertices.

As stated above and by Gramm et al., Reduction Rule 3 is incorrect. The following series of reductions will compute an invalid edge clique cover for the graph G in 4 by exploiting an interaction between Reduction Rule 1 and Reduction Rule 3.

1. Apply Rule 2 to the edge et to cover the clique $\{e, t, p_2\}$.

2. Apply Rule 1 to the vertex t to remove it from the graph.

3. Apply Rule 3 to the vertex v to remove it from the graph. Note that its prisoners are p_1, p_2, and its only exit is e.

4. Apply Rule 2 to the edge p_1p_2 to cover the clique $\{p_1, p_2, v\}$. Note that v is added to this clique because of Reduction Rule 3.

5. Apply Rule 2 to the edge es to cover the clique $\{e, s\}$.

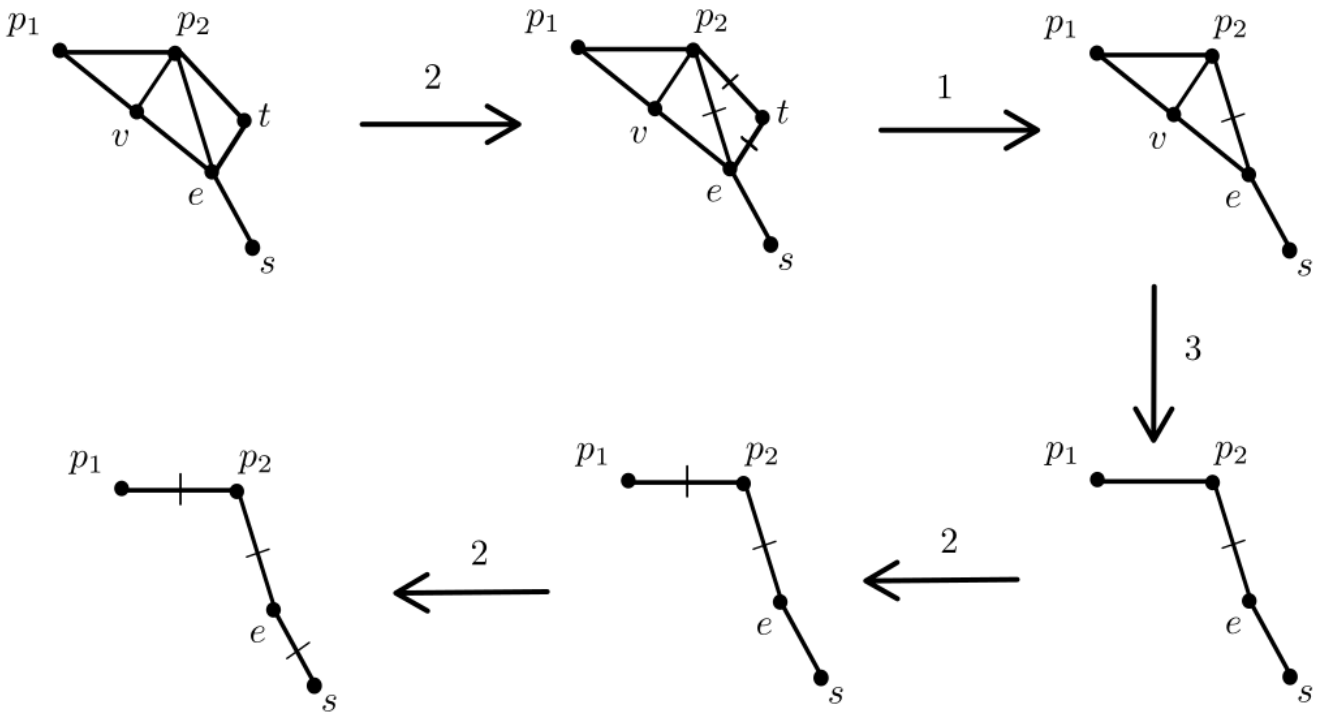

Figure 5: The series of reductions applied to G.

Following this series of reductions, all edges in the reduced graph are covered, so we are to assume that $\{\{e, t, p_2\}, \{p_1, p_2, v\}, \{e, s\}\}$ is a minimum edge clique cover for G. However, this set does not cover the edge ve, and so it is not an edge clique cover at all. To account for this counterexample, we propose a modification to Reduction Rule 3.

Modified Reduction Rule 3. *Suppose that there exists a vertex v such that $N[u] \subseteq N[v]$ for some vertex $u \in N(v)$. Then, we call u a prisoner of v. An exit of v is a vertex in $N(v)$ that is not a prisoner of v. If every prisoner of v is adjacent to at least one vertex other than v via an uncovered edge, and every exit of v is adjacent to at least one prisoner of v* **via an uncovered edge**, *remove v and insert it into any clique containing a prisoner of v covered from now on.*

In the proof of Reduction Rule 3's correctness by Gramm et al., they assert that an edge $\{v, e\}$ from a vertex v to one of its exits e must be in a clique with at least one of the prisoners adjacent to e. However, that assertion does not necessarily hold if pe is covered for every prisoner p adjacent to e. Clearly, Modified Reduction Rule 3 avoids this issue by ensuring that e is adjacent to at least one prisoner via an uncovered edge.

Pruning Rules

In the search tree algorithm by Gramm et al. computes edge clique covers incrementally through recursive branching and reduction. We provide pseudocode for their algorithm below.

Algorithm 1 Branch-and-reduce algorithm from Gramm et al.

```
Input: A graph G.
Output: A minimum edge clique cover for G.
k ← 0
X ← nil
while X = nil do
    X ← BRANCH(G, k, ∅)
    k ← k + 1
return X
function BRANCH(G, k, X)
    if X covers G then
        return X
    REDUCE(G, k, X)
    if k < 0 then
        return nil
    e ← an edge of minimum score in G
    for each maximal clique C in G containing e do
        X′ ← BRANCH(G, k − 1, X ∪ {C})
        if X′ ≠ nil then
            return X′
    return nil
```

Because of this algorithm's iterative deepening approach, the first edge clique cover found is always of minimum cardinality. However, large amounts of computational time are spent computing and recomputing partial covers. It is for this reason that we do away with the iterative deepening in our reimplementation of Gramm et al.'s work. A simple optimization to eliminate wasted search time is to track the size s of the smallest edge clique cover found so far. If at any point the search tree algorithm adds s cliques during the construction of a cover, that branch of the search cannot lead to a minimum edge clique cover, and can therefore be ignored. This simple pruning can be augmented through the use of a lower bound ℓ on the number of cliques remaining to cover. If at

any point the search tree algorithm adds $s-\ell$ cliques during the construction of a cover, that branch of the search can be ignored as well. Thus, in this branch-and-bound model, if a large lower bound can be obtained quickly, large portions of the search tree can be ignored. We provide pseudocode for this algorithm below.

Algorithm 2 Branch-and-bound algorithm

```
Input: A graph G.
Output: A minimum edge clique cover for G.
k ← |V(G)|^2 / 2                          ▷ This is a well-known upper bound on ecc(G)
X ← BRANCH(G, k, ∅)
return X
function BRANCH(G, k, X)
    REDUCE(G, k, X)
    if |X| ≥ k then
        return nil
    else
        if X covers G then
            return X
    e ← an edge of minimum score in G
    ℓ ← COMPUTE-LOWER-BOUND(G, k, X)
    if |X| + ℓ ≥ k then
        return nil
    B ← X
    for each maximal clique C in G containing e do
        X' ← BRANCH(G, k, X ∪ {C})
        if X' ≠ nil then
            k ← |X'|
            B ← X'
    return B
```

We now provide two techniques for computing a lower bound suitable for pruning. It is well known that for a graph G with no isolated vertices, the size of a maximum vertex independent set in G is a lower bound for $\text{ecc}(G)$. We propose a pruning rule based on this bound.

We now prove this lower bound's correctness.

Proof. Because I is an independent set, no two vertices in I can be in the same clique in G. Because each vertex in I is incident to an uncovered edge, each vertex in I must be contained in at least one clique yet to be added to X. Thus, at least $|I|$ cliques must be added to X to produce an edge clique cover. □

We improve upon this lower bound with a second. Another well-known lower bound for $\text{ecc}(G)$ is the size of a maximum independent set of edges in G. We propose a second pruning rule based

Algorithm 3 Vertex set lower bound

Input A graph G, a set X of covered cliques in G.
Output A lower bound for the number of cliques necessary to add to X to produce an edge clique cover.
$I \leftarrow \emptyset$.
for each vertex $v \in V(G)$ **do**
  if v is adjacent to an edge not in a clique in X **and** $I \cup \{v\}$ is an independent set **then**
    $I \leftarrow I \cup \{v\}$
return $|I|$

loosely on this bound.

Algorithm 4 Edge set lower bound

Input A graph G, a set X of covered cliques in G.
Output A lower bound for the number of cliques necessary to add to X to produce an edge clique cover.
$I \leftarrow \emptyset$.
for each edge $uv \in E(G)$ that is not covered by a clique in X **do**
  $valid \leftarrow$ **true**
  for each edge $xy \in I$ **do**
    if uv and xy do not share a vertex **and** $\{u, v\} \cup \{x, y\}$ induces a K_4 in G **then**
      $valid \leftarrow$ **false**
      break
    if uv and xy share a vertex **and** $\{u, v\} \cup \{x, y\}$ induces a K_3 in G **then**
      $valid \leftarrow$ **false**
      break
  if $valid$ **then**
    $I \leftarrow I \cup \{uv\}$
return $|I|$

Figure 6 demonstrates the intuition behind the edge pruning rule. For each pair of black edges in Figure 6, there is at least one edge missing between them, emphasized in red. Thus, no two of uv, xy, ab, ac can belong to a clique together, so at least four cliques remain to be covered. We now prove this lower bound's correctness.

Proof. Let $uv, xy \in I$. Suppose that uv, xy do not share a vertex. Then, the construction of I ensures that $\{u, v\} \cup \{x, y\}$ does not induce a K_4 in G. Thus, at least one of ux, uy, vx, vy is not an edge in G, so uv and xy cannot participate in a clique together in G. Now, suppose without loss of generality that $u = x$. Then, the construction of I ensures that $\{u, v\} \cup \{x, y\} = \{u, v, y\}$ does not induce a K_3 in G. Thus, vy is not an edge in G, so uv and $xy = uy$ cannot participate in a clique

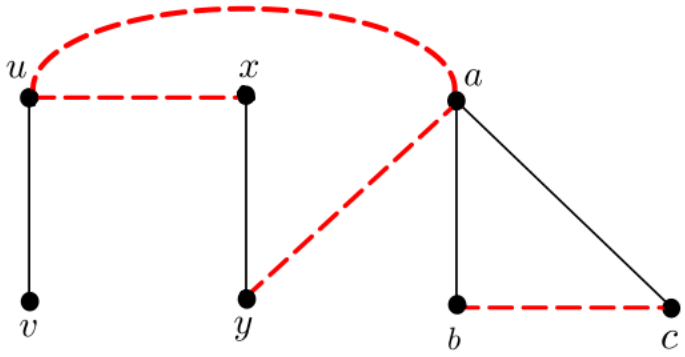

Figure 6:

together in G. Thus, no two edges in I belong to a clique together in G. Thus, because each edge in I is uncovered, at least $|I|$ cliques must be added to X to produce an edge clique cover. □

Implementation

We implemented both the branch-and-reduce algorithm by Gramm et al. and our contributions to it in 762 lines of C++ code, excluding comments and whitespace. Our program and source code are available here. We represent graphs using simple adjacency lists, implmented using `vector` from the g++ STL.

Experiments

Experimental Setup

Experiments were run on an AMD Ryzen 5 3600 with 16 GB of system memory running Arch Linux with a 5.12.4 kernel. The program was written in C++20, and compiled using g++ 11.1.0 with the `-O3` and `-march=native` optimization flags.

Future Directions

Because the search tree algorithm by Gramm et al. requires the enumeration of all maximal cliques containing the branching edge, it breaks dow when the branching edge participates in a number of maximal cliques that cannot be easily enumerated. It remains to be seen whether a better branching heuristic can minimize the total degree of the search tree to a greater extent than selecting the lowest score edge. Reduction Rule 4, as described by Gramm et al., does not improve the running time of the search tree algorithm because the cost of introducing new vertices outweights the benefit

of removing a potentially high-degree vertex. It is possible that selective application of Reduction Rule 4, such as applying it only to cut-vertices, could minimize the negative impact of the rule by guaranteeing that new components are created in the graph.