1. Introduction

In this study, we consider a problem, for a given graph $G = (V,E)$, of finding the minimum number of $3$-cliques ($K_3$s) that cover all edges of $G$. Multiple covering or covering one edge by more than one $3$-clique is allowed. Moreover, in this problem, we allow “spilling-out,” i.e., a set of three vertices $\{x,y,z\}$ can be covered by a $3$-clique even if the induced subgraph by them is not a clique. We call this problem $K_3$ edge cover problem in a wide sense.

This problem is a kind of extension of the schoolgirl problem and finite projective planes, and it has applications on experimental designs. Allowing spilling-out is useful for some applications: E.g., when we want to compare $n$ items through some trials of experiments, in which at most three items can be compared simultaneously, and pairs of items that must be compared are given by a graph, finding the minimum number of tries is formalized as this problem. In the known literature, there are many results that considered problems for covering vertices or edges by the minimum number of cliques. However, there is no theoretical result that considers spilling-out. We obtain the following results:

1. The problem is NP-hard even if graphs are restricted to planar, cubic, and $K_4$, $C_5$-free in a sense of subgraphs (i.e., not restricted to induced ones).
2. For the problem with a parameter $k$, which is the number of $3$-cliques in $G$, there is an $O(mn + 2^m)$-time algorithm.
3. If a tree-decomposition of tree-width $t$ is given, there is an $O(2^{2t+3t}n^2)$-time algorithm.

Keywords: graphs, clique, edge cover, spilling-out, NP-complete, FPT, tree-width

Abstract: In this study, we consider a problem, for a given graph $G = (V,E)$, of finding the minimum number of $3$-cliques ($K_3$s) that cover all edges of $G$. Multiple covering or covering one edge by more than one $3$-clique is allowed. Moreover, in this problem, we allow “spilling-out,” i.e., a set of three vertices $\{x,y,z\}$ can be covered by a $3$-clique even if the induced subgraph by them is not a clique. We call this problem $K_3$ edge cover problem in a wide sense. This problem is a kind of extension of the schoolgirl problem and finite projective planes, and it has applications on experimental designs. Allowing spilling-out is useful for some applications: E.g., when we want to compare $n$ items through some trials of experiments, in which at most three items can be compared simultaneously, and pairs of items that must be compared are given by a graph, finding the minimum number of tries is formalized as this problem. In the known literature, there are many results that considered problems for covering vertices or edges by the minimum number of cliques. However, there is no theoretical result that considers spilling-out.

1.1 Definition of $K_3$ Edge Cover Problem in a Wide Sense

From the viewpoint of the experimental design, let us assume that we have to compare $a$ samples. For comparison, we can use $b$ machines. Each machine can compare $c$ samples simultaneously. We must compare every pair $d$ times. Block design is a research on this problem in which $a$, $b$, $c$, and $d$ are variables, and it has been extensively investigated [17]. In many cases of actual experiments, it is likely that there are pairs that do not need to be compared, and in many cases there is no problem if a pair is compared twice or more. For applying such applications we relax the constraints of block design, that is, we consider a problem for minimizing the number of trials under such conditions. In order to formulate the problem, we first define terms in this paper. In this paper, we consider only simple undirected graphs (graphs with no self-loop and no parallel edge). For more details on block design and graphs, see Refs. [2] and [14].

The number of edges of graph $G$ is denoted as $|E|$. We call a subgraph (or its vertex set) that is a complete graph clique. A clique consisting of $k$ vertices is called a $k$-clique and denoted by $K_k$. For a graph $G = (V,E)$, a family $X = \{X_1, \ldots, X_p\}$ of vertex subsets is called $K_3$ edge cover in a wide sense (or $K_3$ edge cover for short) if (1) $|X_i| = k$ for any $i \in \{1, \ldots, p\}$ and (2) for any edge $(u,v) \in E$, there is vertex subset $X_i \subseteq X$ such that $u,v \in X_i$. In this case, we say that edge $(u,v)$ is covered by $X_i$. We call $p$ the size of the $K_3$ edge cover. The minimum size of a $K_3$ edge cover is denoted by $\gamma_3(G)$. In $K_3$ edge cover problem, if one edge $e$ is
The size of the solution is given \[3\], \[9\].

Regarding CBC, polynomial-time algorithms are known for intersection graphs \[10\], and an FPT algorithm with parameter \(k\). For PIC, polynomial-time algorithms are known for circular arc graphs \[7\], for chordal graphs \[6\], for comparability graphs \[8\]. Two problems are known to be NP-complete \[10\], \[16\]. Regarding vertices by cliques. CBC can be regarded as a problem covering edges by cliques with edge repetitions allowed. These algorithms are known to be NP-complete \[5\].

Let \(H = (W, F)\) be a graph. A vertex subset \(V' \subseteq V\) such that there is no edge between any vertices of \(V'\) is called an independent set, and \(|V'|\) is its size. The problem is defined as follows.

**Problem** Maximum Independent Set (IS)

**Instance:** A graph \(H = (W, F)\), a positive integer \(h \geq 1\).

**Question:** Does \(H\) have an independent set with size at least \(h\)?

**Theorem 4.** Ref. \[4\] IS is in NP-complete even if \(H\) is planar and cubic.

We show how to reduce IS to \(K_3\)EC. Let \((H = (W, F), h)\) be an instance of IS, where \(H\) is cubic and planar. We construct an instance \((G = (V, E), k)\) of \(K_3\)EC as follows.

We obtain \(G\) from \(H\) by replacing a vertex \(w \in W\) with a \(K_3\) with vertices \(w_0, w_1, w_2\); next, connecting the three edges \(e_0, e_1, e_2\) incident to \(w\) (note that \(H\) is cubic) to \(w_0, w_1, w_2\) one by one. See an example of this reduction in Fig. 1. Note that if \(H\) is a cubic planar graph, then \(G\) is cubic, planar, and \(C_4, C_5\)-free.

Let \(k = \frac{3}{2}n - h\), where \(n = |W|\).

It is clear that this reduction can be done in polynomial-time. Thus it is sufficient to show that \(H\) has an independent set with the size \(h\) if and only if \(G\) has a \(K_3\) edge cover with the size \(k = \frac{3}{2}n - h\).

Before showing it, we introduce some terms. Each \(K_3\) of \(G\), which corresponds to a vertex in \(W\), is called a triangle. The three edges in a triangle are called triangle-edges. On the other hand, edges connecting distinct triangles are called link-edges. If two distinct triangles are connected by a link-edge, then they are called adjacent.

Let \(X\) be a \(K_3\) edge cover of \(G\) with the size \(p\), where \(p\) is an integer. If \(X \subseteq X\) covers three edges, then \(X\) is called a delta. If \(X \subseteq X\) covers just two edges, then \(X\) is called an \(L\). If \(X \subseteq X\) covers only one edge, then \(X\) is called an \(I\). If a triangle is covered by a delta, then the triangle is called a delta-triangle. If a triangle is covered by three \(Ls\), then the triangle is called a 3L-triangle (see Fig. 2).

If all triangles are covered by delta-triangles or 3L-triangles, the \(K_3\) edge cover is called regular. If there is no pair of adjacent 3L-triangles in a regular \(K_3\) edge cover, the \(K_3\) edge cover is called independent regular.

**Lemma 1.** If \(H\) has an independent set \(U \subseteq W\) with \(|U| = h\), then \(G\) has a \(K_3\) edge cover \(X\) with \(|X| = \frac{3}{2}n - h\).

**Proof:** We construct \(X\) according to \(U\) as follows. For a triangle...
The number of link-edges covered by $3\mathcal{L}$ is cubic, if the corresponding vertex in $H$ is in $U$, then let the triangle be covered by a $3\mathcal{L}$, and otherwise be covered by a delta. The remaining edges in $G$ are covered by $I$s. See Fig. 3 for example.

We calculate the size of $X$: Let $n = |W|$. Since $H = (W, F)$ is cubic, $|F| = \frac{3}{2}n$. From this, it follows that the number of triangles and link-edges in $G$ are $n$ and $\frac{3}{2}n$, respectively. Since the number of $3\mathcal{L}$-triangles is $h$, the number of delta-triangles is $n - h$.

The number of link-edges covered by $3\mathcal{L}$ is $3h$, and the number of link-edges covered by $I$s is $\frac{3}{2}n - 3h$ by summing up the number of deltas, $3\mathcal{L}$s, and $I$s used in $X$, we get the following equations:

$$|X| = 3h + (n - h) + \frac{3}{2}n - 3h = \frac{5}{2}n - h \quad (1)$$

Therefore $X$ is the desired $K_3$ edge cover.

Note that the $K_3$ edge cover $X$ obtained above is independent regular. To show the reverse of this lemma is a little more complicated. We first show the following lemma.

**Lemma 2.** If $G$ has a $K_3$ edge cover $X$, then $G$ has an independent regular $K_3$ edge cover $X'$ with $|X'| \leq |X|$.

**Proof:** First we construct a regular $K_3$ edge cover $X'$ with $|X'| \leq |X|$ from $X$. At the first step, let $X'$ be equal to $X$. If $X'$ is not regular, we modify $X'$ to be regular by using the following operations.

- **Operation I:** If there is a pair $X, X' \in X'$ such that all edges covered by $X'$ are also covered by $X$, then remove $X'$ from $X'$.
- **Operation II:** If there is $X \in X'$ that covers only one triangle-edge and no link-edge, then $X$ is replaced with a delta covering the triangle-edge together with the other two triangle-edges in the triangle.
- **Operation III:** If a triangle-edge $e$ is covered by a delta and an $L$, then the $L$ is replaced with an $I$ covering only $e'$, where $e'$ is the other edge covered by $L$.
- **Operation IV:** If the three edges of a triangle are covered by more than three $L$s, these $L$s are changed to a $3\mathcal{L}$ (At least one $L$ is removed. See Fig. 4).

An algorithm for changing $X'$ to be regular is the following.

**procedure** `Regularize($X'$)

begin
    do while $X'$ is not regular;
        if Operation I can be applied then apply Operation I;
        else if Operation II can be applied then apply Operation II;
        else if Operation III can be applied then apply Operation III;
        else if Operation IV can be applied then apply Operation IV;
    end if
    enddo
end procedure

We show that the above procedure stops in finite steps for any input $X'$. Each operation never increases the size of $X'$. Moreover, both the number of $L$s and the number of $X \in X'$ that cover only one triangle-edge are never increased also. Operations I and IV decrease the size of $X'$, and thus they can be applied in finite times. Operations II and III decrease the number of $X \in X'$ that cover only one triangle-edge and the number of $L$s, respectively, and thus they can be applied in finite times. From these discussions, Procedure Regularize stops in finite steps, totally. It is clear that if `Regularize($X'$)` stops, $X'$ is regular.

Now we obtain a regular $K_3$ edge cover $X'$. We next change it to be independent regular.

- **Operation V:** If there is a pair of adjacent $3\mathcal{L}$-triangles, then replace one of the two $L$s which share the same edge (e.g., the central edge of Fig. 5) with a delta-triangle, and apply Operation III to the triangle (see Fig. 5).

Operation V does not increase the size of the cover. By applying
If $G$ has a $K_3$ edge cover $X$ with $|X| = \frac{3}{2}n - h$, then $H$ has an independent set $U \subseteq W$ with $|U| = h$.

**Proof:** Assume that $G$ has a $K_3$ edge cover $X$. From Lemma 2, $G$ has an independent regular $K_3$ edge cover $X'$ with $|X'| \leq |X|$. Since there are no adjacent 3L-triangles in $G$,

$$U := \{ w \in W \mid \text{the corresponding triangle in } G$$

becomes an independent set. From the discussion made in the proof of Lemma 1, $|X'| = \frac{3}{2}n - |U|$. Thus if $\frac{3}{2}n - |U| = |X'| \leq |X| = \frac{3}{2}n - h$, then $|U| \geq h$. Therefore any $U' \subseteq U$ with $|U'| = h$ is the desired independent set of $H$.

Now we establish the proof of Theorem 1.

**Proof of Theorem 1:** Follows directly from Lemmas 1 and 3.

4. FPT Algorithm with a Parameter of the Number of $K_3$

As shown by Theorem 1, $K_3EC$ is NP-hard even for planar and cubic graphs. We show FPT algorithms for several parameters. In this section, we prove the following theorem.

**Theorem 2 (reshown).** For $K_3EC$, there is an $O(mn + 2^m)$-time algorithm, where $k$ is the number of 3-cliques in $G$.

First, we prepare some lemmas as follows.

**Lemma 4.** If $T = (V, E)$ is a tree, $\gamma_3(T) = [|E|/2]$.

**Proof:** If $G$ is a path, $\gamma_3(P) = [|P|/2]$ is trivial. We consider the case where there is a vertex with degree 3 or more. We call such a vertex a branching vertex (See Fig. 6 (a)).

Let $v$ be a branching vertex. We divide $T$ into a set of subtree $\{T_1, \ldots, T_k\}$ (where $k$ is the degree of $v$) according to the following rule: each $T_i$ is a maximal subtree such that $v$ is one of its leaves (see Fig. 6 (b)). The size of a subtree $T_i$ is defined as $|T_i|$. If the size is odd, the subtree is called an odd-subtree and otherwise it is called an even-subtree. If there are two odd-subtrees, we join them into one even-subtree. By applying this as far as possible, we finally get a set of subtrees $\{T_{i1}, \ldots, T_{ik}\}$ which includes at most one odd-subtree (See Fig. 6 (c)).

By applying this operation to each subtree recursively, we finally get a set of paths which consists of at most one path with odd size. For each path $P$, $\gamma_3(P) = [|P|/2]$. By summing them up, we obtain the following upper bound.

$$\gamma_3(T) \leq [3|T||/2]$$  \hspace{1cm} (3)

Next, we show the lower bound. Since $K_3$ is not included in the tree, each $X$ can cover at most two edges in any $K_3$ edge cover $\{X_1, \ldots, X_k\}$, and thus the following inequality holds.

$$\gamma_3(T) \geq [3|T||/2]$$  \hspace{1cm} (4)

From Eqs. (3) and (4), $\gamma_3(T) = [|E|/2]$ is obtained.

**Lemma 5.** For any connected graph $G = (V, E)$, the minimum size of $K_3$ edge cover in a wide sense without using deltas is $[|E|/2]$.

For proving this lemma, we prepare an operation as follows.

Let $G = (V, E)$ be a graph and $v \in V \in E$ be a vertex with a degree of at least two. Delete $v$ together with the edges adjacent to $v$ from $G$ and add new two vertices $v'$ and $v''$. New edges are added as follows. The vertex set adjacent to $v$ is denoted by $W = \{ w \in V \mid (v, w) \in E \}$. Divide $W$ into two non-empty subsets $X$ and $Y$ such that $X \cup Y = W$, $X \cap Y = \emptyset$, and $|X|, |Y| \neq 0$, and add edge sets $E_X = \{(v', w) \mid w \in X\}$ and $E_Y = \{(v'', w) \mid w \in Y\}$.

The above operation is denoted by a vertex-division on $v$, i.e., it is defined by getting $G' = (V', E')$ such that $V' = (V - \{|v|\} \cup \{v', v''\})$ and $E' = (E - \{(v, w) \in E \mid w \in W\}) \cup E_X \cup E_Y$. Note that since there is arbitrariness in the way of dividing $W$ into $X$ and $Y$, the result of a vertex partition is not defined uniquely if the degree of $v$ is more than two.

**Proof of Lemma 5:** By applying an appropriate vertex-division on an arbitrary vertex $v$ on an arbitrary cycle of $G = (V, E)$, the resulting graph is still connected and the number of edges does not change, and the number of cycles is decreased at least by one. Thus by applying the finite number of vertex-division to $G$, we get a tree that has the same number of edges as $G$. Let $T$ be one of these trees. From Lemma 4, $\gamma_3(T) = [3|T||/2] = [|E|/2]$. Since there is one to one correspondence between the edge set of $T$ and the edge set of $G$, and for any pair of adjacent edges in $T$, the corresponding pair of edges are also adjacent in $G$, there is a $K_3$ edge cover in $G$ with the size of $\gamma_3(T)$. On the other side, the size of $K_3$ edge cover in $G$ without deltas is clearly at least $[|E|/2]$. From the above discussion, this lemma is obtained.

We prove Theorem 2 using this lemma as follows.

**Proof of Theorem 2:** We give an algorithm for solving the problem. The algorithm focuses on an arbitrary triangle of $G$. It divides the case into two cases according to whether or not the triangle is covered by a delta. If it is covered by a delta, the algorithm deletes the three edges in the triangle from $G$. Otherwise, the algorithm gives a label “not covered by a delta” to the triangle. The algorithm recursively does the above operations as far as a non-labeled triangle exists. The graph finally obtained is not allowed to be covered by using any delta, and the solution can be obtained from Lemma 5. From that, the calculation time of each case is $O(m)$ and the number of branches is at most $k$ and the algorithm requires enumerating all triangles and it needs $O(mn)$ time, $O(mn + 2^m)$.
with tree-width as a parameter.

**Theorem 3** (reshown). For $K_t EC$, if a tree-decomposition of tree-width $t$ is given, there is an $O(2^{O(t^2)}n^2)$-time algorithm.

First, we assume that all graphs considered here are connected, otherwise, it is enough to manage each connected component one by one. Under this assumption, we do not need to consider $I$ ($K_t$ covering only one edge) as shown in the following.

**Lemma 6.** If $G = (V, E)$ is connected and $|V| \geq 3$, there is an optimal solution that includes no $I$.

**Proof:** Since $G$ is connected and $|V| \geq 3$, every edge has at least one adjacent edge. Hence if an edge is covered by an $I$, we can replace this $I$ with a delta or an $L$. The number of $K_t$ edge covers used does not change by this change. By applying this operation whenever an $I$ exists finally we obtain a solution that uses no $I$ with the same size. □

Next, we introduce some terms and prepare some other lemmas.

**Definition 1.** Ref. [12] (tree-decomposition) A tree-decomposition of a graph $G = (V, E)$ is a pair $D = (S, T)$ with a family $S = \{B_1, \ldots, B_r\}$ of subsets of $V$ and a tree $T = (I, F)$ such that $\{1, \ldots, r\}$ and the following three conditions are satisfied. For every $i \in I$, $i$ is called a node and $B_i$ is called a bag.

(1) $\bigcup_{i \in I} B_i = V$.

(2) For every edge $(v, w) \in E$, there is at least one bag $B_i$ such that $(v, w) \in B_i$, and

(3) For each vertex $v$ the set of nodes $\{i \in I \mid v \in B_i\}$ forms a subtree of $T$.

**Definition 2.** Ref. [12] (tree-width) The width of a tree-decomposition $D = (S, T)$ is defined as max$_{i \in I}(|B_i| - 1)$. The tree-width of a graph $G$ is the minimum possible width through all tree-decompositions of $G$.

**Definition 3.** Ref. [12] (nice tree-decomposition) A tree-decomposition $D = (S, T)$ is called nice if the following four conditions are satisfied:

(1) $T$ is a rooted tree and each node has at most two children.

(2) If a node $i$ has two children $j$ and $k$, then $B_i = B_j \cup B_k$.

(3) If node $i$ has unique child $j$, then $|B_i| = |B_j| + 1$ and $B_i \subset B_j$ (or $|B_i| = |B_j| - 1$ and $B_i \subset B_j$).

(4) Every bag corresponding to a leaf node of $D$ consists of only one vertex.

**Definition 4.** Ref. [12] (node types of a nice tree-decomposition) In a nice tree-decomposition $D = (B_i \mid i \in I), T = (I, F)$ every node is one of the following four possible types.

Leaf: A node that is a leaf.

Introduction: A node $i$ that has a child $j$ and $|B_i| \geq |B_j|$.

Forget: A node $i$ that has a child $j$ and $|B_i| < |B_j|$.

Join: A node that has two children.

**Lemma 7.** Ref. [12] If a tree-decomposition with tree-width $k$ of graph $G$ is given, a nice tree-decomposition of tree-width $k$ having $O(|V(G)|)$ nodes can be obtained in $O(k^2 \cdot \max(|V(T)|, |V(G)|))$-time.

Our algorithms use a nice tree-decomposition obtained by using Lemma 7. From here every tree-decomposition appearing below is a nice tree-decomposition, unless otherwise stated. In

what follows $T = (I, F)$ is the nice decomposition tree. Let $i$ and $j$ be nodes in $I$ and let $B_i$ and $B_j$ be bags of corresponding $i$ and $j$, respectively. If $j$ is a descendant (an ancestor, resp.) of $i$, then $B_j$ is called a descendant (an ancestor, resp.) of $B_i$. The lower vertex set $B_i$ of a bag $B$ is defined as follows.

$$B_i := \{v \in B' \mid B' \text{ is a descendant of } B\} - \{v \in B\}. \tag{5}$$

$$B_i^1 := B_i \cup B. \tag{6}$$

The upper vertex set $B_i^1$ of the bag $B$ is $B_i = V - B_i^1$. Also, $B_i^2 := B_i \cup B$. For an arbitrary $K_t$ edge cover $X = \{X_1, \ldots, X_k\}$ on a graph $G$ and an arbitrary vertex subset $W \subseteq V$, $X_W := \{X_i \mid X_i \subseteq W\}$, which is a partial solution of $X$ for $W$. The edge set of the subgraph of $G = (V, E)$ induced by $W \subseteq V$ is denoted by $E_W$.

Let $G_{B'} := (B_i^2, E_{B'})$ be the subgraph of $G = (V, E)$ induced by $B_i^2$. Let $E_{B', X}$ be the set of edges not covered by $X_{E_{B'}}$ in $E_{B'}$. $X_{E_{B'}}$ is called an optimal partial solution for $W$ if $X^{*}$ is an optimal solution of $K_t$ edge cover.

Our algorithm traces the rooted tree $T$ in the postorder, and for each bag $B$, constructs a set of partial solutions for $B_i^2$ such that one of them is the partial solution of an optimal solution. For saving the memories, we compress the data of the partial solutions and store them as a table. We will explain the details in the following.

For an intuitive explanation, let us assume a bag $B$ has $k$ vertices. For each bag $B = \{v_1, v_2, \ldots, v_k\}$, we create a table $T[B]$, which is a compressed representation of the partial solutions. Let $G_B := (B, E_B)$ be the subgraph of $G$ induced by $B$. Let $E_{B'} := E_B :\{\{v, w\} \in E_B \mid \{v, w\} \cap B \neq \emptyset\}$, i.e., the set of edges in $E_{B'}$ each of which is incident to at least one vertex in $B$. Let $E_{B'}$ be $E_{B'} := E_{B'} - E_B$, i.e., the set of edges in $E_{B'}$ each of which is incident to just one vertex in $B$. Let $h = |E_B|$ be the number of edges between vertices of $B$, and we give serial numbers to these edges as $e_1, \ldots, e_h$.

Let $X_j$, $j = 1, \ldots, q$ be the partial solutions stored in $T[B]$. $X_j$ is represented as row $j$ of $T[B]$. Each row of $T[B]$ is divided into three parts, the first part consists of $k(= |B|)$ columns, the second part consists of $h$ columns, and the third part is only one column (See, Fig. 8 for an example of $T[B]$). Note that in this figure, the tree-decomposition is not nice so as to make it simple). That is, $T[B]$ consists of $k + h + 1$ columns. The ith cell of the first part of row $j$ stores the number of edges between $v_i$ and vertices in $B_i$ and uncovered by $X_j$ (Note that the set of these edges is
$E_{\Gamma'} \cap E_B = \emptyset$. It will be shown later (in Lemma 10) that this number is enough to be 1 or 0. The $i$th column of the second part of row $j$ is 1 if $e_i$ is covered by $\mathcal{X}$; otherwise 0. The size $|\mathcal{X}'|$ of $\mathcal{X}'$ is stored in the third part of row $j$. For notational simplicity, this value $|\mathcal{X}'|$ is represented by $\omega$ in the following. The compressed expression of a partial solution excluding the third part is called the signature of the partial solution.

In the following, we show some lemmas necessary for supporting our algorithm.

**Lemma 8.** For any bag $B$ of a tree-decomposition of a graph, and two vertices $u \in B_1$ and $v \in B_1$, there is no edge between $u$ and $v$, i.e., $(u, v) \notin E$ from the property of tree-decompositions (Condition 2 of Definition 1).

**Proof:** Since there is no bag containing both $u$ and $v$, $(u, v) \notin E$ from the property of tree-decompositions (Condition 2 of Definition 1).

**Lemma 9.** For any solution $\mathcal{X}$ and a bag $B$, all edges belonging to $E_{\Gamma', \mathcal{X}}$ are incident to vertices in $B$.

**Proof:** Assume that an edge $(u, v) \in E_{\Gamma', \mathcal{X}}$ is not incident to any vertex in $B$, i.e., $(u, v) \notin B$. From $E_{\Gamma', \mathcal{X}} \subseteq E_{\Gamma'}$, it follows that $u, v \in B_1$. Let $X$ be an element of $\mathcal{X}$ that covers $(u, v)$. Since $X \notin X_{\Gamma'}$ and $(u, v) \in E_{\Gamma', \mathcal{X}}$, $X$ covers at least one edge besides edge $(u, v)$.

---

2. $X$ is an element of $\mathcal{X}$ and $X \notin X_{\Gamma'}$, it means that one of the vertices in $X$ is not in $B_1$. $X$ should contain another vertex $w$ other than $u$ and $v$ that is not in $B_1$, because $u, v \in B_1 \subseteq B'$. Therefore, we have that $X$ covers at least one edge besides $(u, v)$, say, $(u, w)$ or $(v, w)$ (if does not exist from Lemma 6).
Table 1 Since $G_{2} = (E = [u], Ex = 0)$ for leaf node $B = [u], X_{B} = 0$ for any solution $X$. Therefore, $T[B]$ looks like what is shown.

| $u$ | $\omega$ |
|-----|---------|
| 0   | 0       |

Lemma 6, contradicting Lemma 8. □

Lemma 13. Let $B$ be a join node and $B'$ and $B''$ be its child nodes. Then, $B' \cap B'' = \phi$.

Proof: From the definition of $B'_i$, any vertex belonging to $B'_i$ does not belong to $B'$. Therefore, from the definition of tree-decomposition (Condition 3 of Definition 1), these vertices do not belong to $B'_i$ either and hence this lemma follows. □

5.1 Algorithm

When creating table $T[B]$, if there are two or more partial solutions whose signatures are the same we can remove these partial solutions except one that has the minimum $\omega$ among them. A strict proof of the correctness of this operation will be shown after describing the algorithm (Lemma 14). We explain the algorithm step by step as follows. This algorithm scans the tree of the nice tree-decomposition in the postorder and creates a table $T[B]$ at each node (bag) $B$ so that there is a partial optimal solution. Finally, the solution of the table at the root node is the optimal solution.

$T[B]$ is created from the tables of the children of $B$. Since the tree decomposition is nice, the number of children is at most two. The basic strategy of creating $T[B]$ is enumerating all possible partial solutions. That is, for each row (partial solution) $X$ in $B'$, which is one of the children of $B$, enumerating all possible cases of covering or uncovering edges in $E_{B',X}$. Since $|E_{B',X}|$ is bounded by a constant number, the enumeration for them can be done in constant-time. However since $E_{B'}$ grows up to $m$ when the trace of the algorithm is close to the root of the decomposition tree, the size of the number of different partial solutions becomes exponential. This problem is resolved by using Lemma 14. That is, from this lemma, keeping only partial solutions whose signatures are different is sufficient.

The operation for constructing $T[B]$ for each node (bag) $B$ consists of two stages: one is Enumerating Stage and the other is Refinement Stage. In Enumerating Stage, all possible partial solutions are enumerated. In Refinement Stage, partial solutions that are unnecessary (i.e., that are not partial solutions of optimal solutions) are removed. In the following, these stages are illustrated. First, Enumerating Stage for each type (leaf, introduce, forget, and join) is shown. Since Refinement Stage is the same for every type, it will be shown after explaining Enumerating Stages of all types.

Enumerating Stage

(1) Leaf nodes

Let $B$ be a leaf node, and $u$ be its (unique) element. Create Table as shown in Table 1.

(2) Introduce nodes

Let $B = [u, v_{1}, v_{2}, \ldots, v_{k}]$ be an introduce node and $B' = \{v_{1}, v_{2}, \ldots, v_{k}\}$ be its child. For each row of $T[B']$, create some rows of $T[B]$ according to the following operations. Let $X_{B'}$ be the partial solution corresponding to the row (of $T[B']$) (See Fig. 9).

Step 1. For each edge $(u, v_{i})$, apply the following operations: Let $w_{i}$ and $w_{j}$ be vertices adjacent to $v_{i}$ and $v_{j}$ in $B'$, respectively, and are not covered by the partial solution that corresponds to the currently focused row of $T[B']$ yet (if exist). Note that $v_{i}$ (resp., $v_{j}$) has at most one such vertex from Lemma 10. List up all possible combinations of the following cases and make a row corresponding to each of the combinations: (1) $\{v_{i}, v_{j}, u\}$ are covered by a $K_{3}$, (2) $\{v_{i}, v_{j}, w_{i}\}$ are covered by a $K_{3}$, and (3) $\{v_{i}, v_{j}, w_{j}\}$ are covered by a $K_{3}$. Create all cases of combinations of the above three cases, i.e, at most $2^{3} = 8$ rows are created for a currently focused row in $T[B']$.

Step 2. For each row created in Step 1, update the values in the cells of the first, the second and the third part.

(3) Forget nodes

Let $B = \{v_{1}, v_{2}, \ldots, v_{k}\}$ be a forget node and $B' = \{u, v_{1}, v_{2}, \ldots, v_{k}\}$ be its child. For each row of $T[B']$, create $T[B]$ according to the following operations. Let $X_{B'}$ be the partial solution corresponding to the row (of $T[B']$) (See Fig. 10).

Step 1. For each edge $(u, v_{i})$, apply the following operations: Let $w_{i}$ and $w_{j}$ be vertices adjacent to $u$ and $v_{i}$ in $B'$, respectively, and are not covered by the partial solution that corresponds to the currently focused row of $T[B']$ yet (if exist). Note that $u$ (resp., $v_{i}$) has at most one such vertex from Lemma 10. Let $v_{i}$ be a vertex adjacent to at least one of $u$ or $v_{i}$. List up all possible combinations of the following cases and make a row corresponding to each of the combinations: (1) $\{u, v_{i}, v_{j}\}$ are covered by a $K_{3}$, (2) $\{u, v_{i}, w_{i}\}$ are covered by a $K_{3}$, and (3) $\{u, v_{i}, w_{j}\}$ are covered by a $K_{3}$. Create all cases of combinations of the above three cases, i.e,
For each row created in Step 1 and Step 2, update $T$ and a row in $B$. For each cell in the first part of the row, store the values in the cells of the second and the third part. Let $B = \{b, c, f\}$ (the child of $B$), on the graph and the tree-decomposition shown in Fig. 7. Each row corresponds to a partial solution of a $K_2$ edge cover, e.g., the first row in table (d) shows a partial solution $\emptyset$ and the second row shows a partial solution $\{b, c, f\}$. (i) Since there is one non-zero cell in the first part of $T[B]$ in (c), the algorithm does nothing in (i). (ii) List up all possible cases to cover edges in $B$. In this case, we consider the combination of whether the two edges $(b, c)$ and $(c, f)$ are covered or not, i.e., the first row of (c) means $(b, c)$ and $(c, f)$ are not covered, the second row means $(b, c)$ and $(c, f)$ are covered, the third row means $(c, f)$ is covered but $(b, c)$ is not. (iii) The value of the column corresponding to vertex $c$ (the second column) is set to 1, because edge $(c, f)$ and $f$ do not exist in $B$. (iv) Because the second, the fourth, the fifth, the sixth and the eighth row of $X$ have 1, and 0 otherwise.

![Fig. 10](image)

An example of making $T[B]$ from $T[B']$, where $B = \{b, c, f\}$ (forget node) and $B' = \{b, c, f\}$ (the child of $B$), on the graph and the tree-decomposition shown in Fig. 7. Each row corresponds to a partial solution of a $K_2$ edge cover, e.g., the first row in table (d) shows a partial solution $\emptyset$ and the second row shows a partial solution $\{b, c, f\}$. (i) Since there is one non-zero cell in the first part of $T[B]$ in (c), the algorithm does nothing in (i). (ii) List up all possible cases to cover edges in $B$. In this case, we consider the combination of whether the two edges $(b, c)$ and $(c, f)$ are covered or not, i.e., the first row of (c) means $(b, c)$ and $(c, f)$ are not covered, the second row means $(b, c)$ and $(c, f)$ are covered, the third row means $(c, f)$ is covered but $(b, c)$ is not. (iii) The value of the column corresponding to vertex $c$ (the second column) is set to 1, because edge $(c, f)$ and $f$ do not exist in $B$. (iv) Because the second, the fourth, the fifth, the sixth and the eighth row of (b) have the same signature and the value of $\omega$ of the second is the smallest, the fourth, the fifth, the sixth and the eighth row are deleted. Consequently, the table of (a) is obtained.

at most $2^3 = 8$ rows are created for a currently focused row in $T[B']$.

**Step 2.** For each row of $T[B]$, if edge $(u, v)$ is not covered (i.e., the value of $(u, v)$ is 0 in the second part of $T[B]$) and $v_1$ has 1 in the cell of the first part of $T[B']$, $(u, v, w_1)$ are covered by a $K_1$ and is set as the value of $v_1$ in the first part of $T[B']$ (Note that $u$ does not exist in $B'$ from Lemma 12).

**Step 3.** For each row created in Step 1 and Step 2, update the values in the cells of the second and the third part.

(4) **Join nodes**

Let $B = \{v_1, v_2, \ldots, v_k\}$ be a join node and $B' = B'' = \{v_1, v_2, \ldots, v_k\} be its children. For each pair of a row in $T[B']$ and a row in $T[B'']$, create a row in $T[B]$ according to the following operations. Let $X_{B_1}'$ and $X_{B_2}'$ be the partial solutions corresponding to the row in $T[B']$ and the row in $T[B'']$, respectively. Create a row in $T[B]$ by applying the following operations to all pairs of rows in $T[B']$ and $T[B'']$.

**Step 1.** For each cell in the first part of the row, store the logical sum of the values of the corresponding cells of $X_{B_1}'$ and $X_{B_2}'$. (The correctness of this operation is supported by the fact $B'_1 \cap B''_2 = \phi$ proved in Lemma 13).

**Step 2.** For each cell in the second part of the row, put 1 if one of the values of the corresponding cells of $X_{B_1}'$ and $X_{B_2}'$ have 1, and 0 otherwise.

**Step 3.** For the cell in the third part of the row, store the sum of the values in the cells of the third part of $X_{B_1}'$ and $X_{B_2}'$.

After Steps 1 to 3, apply the following operations.

**Step 4.** In Step 1, for each $v_i$, if both of the corresponding cells of $T[B]$ and $T[B']$ are 1 (and the corresponding cell in $T[B]$ is 0 as a result of the logical summation), then increase the value of the row in the third part of $T[B]$ by 1.

**Refinement Stage**

After finishing creating columns of $T[B]$ by the above operations, if there are two or more partial solutions (rows) that have the same signature in $T[B]$, then leave only one partial solution that has the minimum $\omega$ among them (i.e., delete the others).

We prove the correctness and the computation time as follows.

**Lemma 14.** If $T[B]$ includes a partial solution of an optimal solution, then at least one of the kept solutions remains after Refinement Stage.

**Proof:** Assume that a partial solution of an optimal solution $X'$ was deleted in Refinement Stage of node $B$. We denote the deleted optimal partial solution by $X'_{B'}$. A partial solution $X'_{B''}$ that has the same signature must remain. From the rule of Refinement Stage,

$$|X'_{B'}| \leq |X'_{B''}|$$

By regarding $X'$ and $X$ as $X'$ and $X$, respectively, of Lemma 11, $X''$ constructed in the proof of Lemma 11 is also a solution. From (6), $|X'''| = |X''| - |X'_{B'}| + |X'_{B''}| \leq |X''|$. Thus $X'''$ is also an optimal solution. $X_{B''}$ can be regarded as a partial solution of $X''$. □

**Lemma 15.** For any bag $B$, the signature set of the partial solution stored in $T[B]$ contains the signature of the optimal partial solution.

**Proof:** This can be easily proved by induction. Hence the detail is omitted here.

Now we establish the proof of Theorem 3 as follows.

**Proof of Theorem 3:** From Lemma 15, since the optimal solution exists in the solution stored in the root node, the algorithm correctly gives the optimal solution. Thus we estimate the computation time. First we calculate the number of columns in a table $T[B]$ of a bag $B$. The number of columns in the first part, which is equal to the number of the vertices in the bag, is at most $t + 1$, the number of columns in the second part, which is equal to the number of edges in the bag, is at most $t + 1 + (t + 2)/2 + 1 = O(t^2)$. Next, we calculate the number of rows. The maximum number of rows in a table after finishing Refinement Stage is equal to the maximum possible variations of signatures. Since there are at most $2^{t+1}$ variations for the first part and $2^{(t+1)/2}$ variations for the second part, the total number of possible variations is at most $2^{t+1} \cdot 2^{(t+1)/2} = 2^{t+1} \cdot 2^{(t+1)/2}$, which is an upper bound of the number of rows of a table after finishing the Refinement Stage. However, in Enumerating Stage, more rows are created in general. In an introduce or forget node, we may create at most $8 \cdot 2^{(t+1)(t+2)/2}$ rows by creating in Enumerating Stage of an introduce or forget node. In a join node, at most $2^{2(t+1)(t+2)/2} = 2^{2(t+1)(t+2)}$ rows are created in Enumerating Stage. By comparing $8 \cdot 2^{(t+1)(t+2)/2}$ and $2^{2(t+1)(t+2)}$, it follows that the number of rows are $O(2^{(t+1)(t+2)})$. 

© 2020 Information Processing Society of Japan

856
In Refinement Stage we compare all pair of rows. Hence we must compare \( O((2^{t+1} \times 2^{t+2}))^2 \) = \( O(2^{2(t+1) + (t+2)}) \) rows in a node. Comparing one pair of rows requires \( O(t^2) \)-time, since the number of columns is \( O(t^2) \). Therefore the computation time required in a node is \( O(2^{2(t+1) + (t+2)} t^2) \).

Finally, from that the number of nodes of the tree is \( O(n) \), it follows that the total computation time is \( O(2^{2(t+1) + (t+2)} t^2 n) \). □

6. Conclusion

In this paper, we dealt with \( K_4 \) edge cover problem in a wide sense. We found that it is NP-complete even for planar and cubic graphs. We also showed FPT algorithms for several parameters, i.e., one is \( O(mn + 2^m) \)-time algorithm and the other is \( O(2^s + \epsilon 2^n) \)-time algorithm, where \( k \) is the number of 3-cliques and \( t \) is the tree-width (under an assumption that a tree-decomposition of tree-width \( t \) is given). Next, we discuss open problems. From the NP-completeness of \( K_4 \) EC, it may be expected that \( K_4 \) edge cover problem in a wide sense for \( k \geq 4 \) is also NP-complete. However, this is not trivially obtained from the proof on \( K_4 \) EC. One of the difficulties is that \( K_4 \) can cover two non-adjacent edges simultaneously. This fact makes \( K_4 \) edge cover problem in a wide sense difficult to show the hardness or give efficient algorithms.

Acknowledgements

We would like to thank the Algorithms on Big Data project (ABD14) of CREST, JST (grant no. JPJICR1402), the ELC project (MEXT KAKENHI Grant Number 24106003), and JSPS KAKENHI Grant Number 15K11985 through which this work was partially supported.

References

[1] Barnier, N. and Brisset, P.: Solving the Kirkman’s schoolgirl problem in a few seconds, International Conference on Principles and Practice of Constraint Programming, pp.477–491, Springer (2002).

[2] Chartrand, G. and Zhang, P.: A first course in graph theory, Courier Corporation (2013).

[3] Cygan, M., Pilipczuk, M. and Pilipczuk, M.: Known algorithms for edge clique cover are probably optimal, SIAM Journal on Computing, Vol.45, No.1, pp.67–83 (2016).

[4] Garey, M.R. and Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete, SIAM Journal on Applied Mathematics, Vol.32, No.4, pp.826–834 (1977).

[5] Garey, M.R. and Johnson, D.S.: Computers and intractability, Vol.29, WH freeman New York (2002).

[6] Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, SIAM Journal on Computing, Vol.1, No.2, pp.180–187 (1972).

[7] Gavril, F.: Algorithms on circular-arc graphs, Networks, Vol.4, No.4, pp.357–369 (1974).

[8] Golumbic, M.C.: The complexity of comparability graph recognition and coloring, Computing, Vol.18, No.3, pp.199–208 (1977).

[9] Gramm, J., Guo, J., Hüffner, F. and Niedermeier, R.: Data reduction and exact algorithms for clique cover, Journal of Experimental Algorithmics (JEA), Vol.13, p.2 (2009).

[10] Karp, R.M.: Reducibility among combinatorial problems, Complexity of Computer Computations, pp.85–103, Springer (1972).

[11] Kaski, P. and Ostergård, P.R.: There are exactly five biplanes with \( k=11 \), Journal of Combinatorial Design, Vol.16, No.2, pp.117–127 (2008).

[12] Kloks, T.: Treewidth: Computations and approximations, Vol.842, Springer Science & Business Media (1994).

[13] Kou, L.T., Stockmeyer, L.J. and Wong, C.-K.: Covering edges by cliques with regard to keyword conflicts and intersection graphs, Comm. ACM, Vol.21, No.2, pp.133–139 (1978).

[14] Ne, J. et al.: Invitation to discrete mathematics, Oxford University Press (2009).

[15] Noshad, M. and Brandt-Pearce, M.: Expurgated PPM using symmetric balanced incomplete block designs, IEEE Communications Letters, Vol.16, No.7, pp.968–971 (2012).

[16] Orlin, J.: Contentment in graph theory: Covering graphs with cliques, Indagationes Mathematicae (Proceedings), Vol.80, No.5, pp.406–424, Elsevier (1977).

[17] Wilson, R.M.: An existence theory for pairwise balanced designs I. Composition theorems and morphisms, Journal of Combinatorial Theory, Series A, Vol.13, No.2, pp.220–245 (1972).

Hiro Ito received B.E., M.E., and Ph.D. degrees in the Department of Applied Mathematics and Physics from the Faculty of Engineering, Kyoto University in 1985, 1987, and 1995, respectively, 1987–1996, 1996–2001, and 2001–2012, he was a member of NTT Laboratories, Toyohashi University of Technology, and Kyoto University, respectively. Since 2012, he has been a full professor in School of Informatics and Engineering at The University of Electro-Communications (UEC). He has been engaged in research in discrete algorithms mainly on graphs and networks, discrete mathematics, recreational mathematics, and algorithms for big data. Dr. Ito is a member of IEICE, the Operations Research Society of Japan, the Information Processing Society of Japan, and the European Association for Theoretical Computer Science. He is also a member of the steering committee of JCDCG³.

Rémy Belmonte received his Ph.D. in computer science from the University of Bergen in 2013. He then held postdoctoral research positions at Kyoto University and Kwansei Gakuin University, Japan. Since 2016, he has been an assistant professor in the Department of Computer and Network Engineering of the University of Electro-Communications in Chofu, Japan. His research interests are theoretical computer science, graph theory and discrete algorithms.

Kyohei Chiba received B.E. from the Department of Communication Engineering and Informatics The University of Electro-Communications in 2017, and received M.E. degrees in the Graduate School of Informatics and Engineering The University of Electro-Communications in 2019. His main research interests are theoretical computer science, graph theory and discrete algorithms.
Michael Lampis received his Ph.D. in computer science from the City University of New York in 2011. He then held post-doctoral research positions at KTH Royal Institute of Technology in Stockholm, Sweden, and at RIMS, Kyoto University, Japan. In 2014 he joined Paris-Dauphine University in Paris as an assistant professor, a position that he holds to this day. His research interests focus on theoretical computer science and more specifically in parameterized complexity and approximation algorithms.

Atsuki Nagao received B.E. from the Faculty of Engineering Kyoto University in 2010, and received M.Info., and Ph.D. degrees in Informatics from the Graduate School of Informatics Kyoto University in 2012 and 2015 respectively. From 2015 to 2017, he was a project researcher in School of Informatics and Engineering The University of Electro-Communications. From 2017 to 2018, he was an assistant professor in Department of Computer and Information Science Seikei University. Since 2018, he has been an assistant professor in Faculty of Core Research Natural Science Division Ochanomizu University. He has majored in computational complexity, log-spaced algorithms and combinatorial games and puzzles. He is a member of IEICE.

Yota Otachi is an associate professor of Nagoya University. Previously, he held positions at Tohoku University, Japan Advanced Institute of Science and Technology, and Kumamoto University. He received B.E., M.E., and Ph.D. degrees from Gunma University in 2005, 2007, and 2010. His research interests include graph algorithms and algorithmic graph theory.