Electronic Preprint for Journal of Information Processing Vol.28

Regular Paper

K_3 Edge Cover Problem in a Wide Sense

Kyohei Chiba[1,a)] Rémy Belmonte[1,b)] Hiro Ito[1,c)] Michael Lampis[4,d)] Atsuki Nagao[2,e)]
Yota Otachi[3,f)]

Received: January 6, 2020, Accepted: September 10, 2020

Abstract: In this study, we consider a problem, for a given graph $G = (V, E)$, of finding the minimum number of 3-cliques (K_3s) 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 tries 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 $C4, C5$-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 $O(mn + 2^k m)$-time algorithm. (3) If a tree-decomposition of tree-width t is given, there is an $O(2^{2(t+1)(t+2)}t^2 n)$-time algorithm.

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

1. Introduction

In this study, we consider a problem, for a given graph $G = (V, E)$, for finding the minimum number of 3-cliques (K_3s) 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 tries 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 [6], [7], [8], [10], [13], [16]. However, there is no theoretical result that considers spilling-out.

[1] School of Informatics and Engineering, The University of Electro-Communications (UEC), Chofu, Tokyo 182–8585, Japan
[2] Faculty of Core Research Natural Science Division Ochanomizu University, Bunkyo, Tokyo 112–0012, Japan
[3] Graduate School of Informatics, Nagoya University, Nagoya, Aichi 464–8601, Japan
[4] Université Paris-Dauphine, PSL University, CNRS, LAMSADE; 75016, Paris, France
a) chiba.050505@gmail.com
b) remybelmonte@gmail.com
c) itohiro@uec.ac.jp
d) michail.lampis@lamsade.dauphine.fr
e) a-nagao@is.ocha.ac.jp
f) otachi@nagoya-u.jp

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, 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 $\|G\|$. 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 $\mathcal{X} = \{X_1, \ldots, X_p\}$ of vertex subsets is called *K_k edge cover in a wide sense* (or *K_k 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 $X_i \in \mathcal{X}$ 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_k edge cover. The minimum size of a K_k edge cover is denoted by $\gamma_k(G)$. In K_k edge cover problem, if one edge e is

contained in a clique in a solution set, then we say e is covered. In this paper, we consider only the case of $k = 3$, i.e., K_3 edge cover problem in a wide sense. The problem is defined as follows.

Problem K_3-EDGE-COVER-IN-A-WIDE-SENSE (K_3EC)

Instance: A graph $G = (V, E)$, and a positive integer $h \geq 1$.

Question: $\gamma_3(G) \leq h$?

On this problem we obtain the following results. C_4 and C_5 are the cycles of length 4 and 5, respectively.

Theorem 1. *K_3EC is NP-complete even if graphs are restricted to planar, cubic, and $C4, C5$-free in a sense of subgraphs (i.e., not restricted to induced ones).*

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

Theorem 3. *For K_3EC, if a tree-decomposition of tree-width t is given, there is an $O(2^{2(t+1)(t+2)}t^2 n)$-time algorithm.*

2. Previous Work

School girl problem, block design, and covering by cliques have been extensively studied. The school girl problem is a classical combinatorial mathematical problem, and fast algorithms for solving this problem have been studied [1]. Block design is a well-known problem in the field of discrete mathematics, concrete patterns [11] and applications [15] have been considered.

K_3EC has the following directly related problems. Problem PARTITION-INTO-CLIQUES, PIC for short, is a problem, given a graph $G = (V, E)$ and an integer K, for deciding whether or not V can be partitioned into $k(\leq K)$ cliques. Problem COVERING BY CLIQUES, CBC for short, is a problem, given a graph $G = (V, E)$ and an integer K, for deciding whether or not V can be covered by $k(\leq K)$ cliques, which are subgraphs of G and cover (include) all edges in E. PIC can be regarded as a problem covering vertices by cliques. CBC can be regarded as a problem covering edges by cliques with edge repetitions allowed. These two problems are known to be NP-complete [10], [16]. Regarding PIC, polynomial-time algorithms are known for circular arc graphs [7], for chordal graphs [6], for comparability graphs [8]. Regarding CBC, polynomial-time algorithms are known for intersection graphs [10], and an FPT algorithm with parameter k (the size of the solution) is given [3], [9]. K_3EC differs from them in the following two parts: (1) it allows spilling-out and (2) the size of the clique is limited. There have been no studies on this problem. Clearly, problems for covering by K_1 or K_2 are trivial. For a problem for covering by P_2, which is a path consisting of two edges, a polynomial-time algorithm is easily obtained by modifying our algorithm given in the proof of Lemma 5. If there is no size restriction of cliques, the problem becomes trivial, since covering by a $|V|$-clique is optimal.

3. NP-completeness

In this section, we show a proof of Theorem 1.

Theorem 1 (reshown)**.** *K_3EC is NP-complete even if graphs are restricted to planar, cubic, and $C4, C5$-free in a sense of subgraphs (i.e., not restricted to induced ones).*

K_3EC is clearly in NP. Thus we will show NP-hardness. It will be done by reducing the *Maximum Independent Set* (*IS*), which is a well-known NP-complete problem [5].

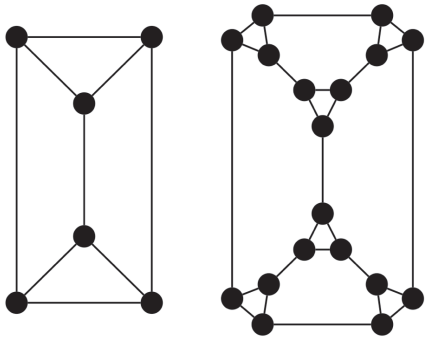


Fig. 1 Reduction to K_3EC from IS.

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_3EC. 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_3EC as follows.

We obtain G from H by replacing a vertex $w \in W$ with a K_3 with vertices w_0, w_1, and w_2; next, connecting the three edges e_0, e_1, and e_2 incident to w (note that H is cubic) to w_0, w_1, and 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 $k = \frac{5}{2}n - h$, 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 $k = \frac{5}{2}n - h$.

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 $\mathcal{X}$ be a K_3 edge cover of G with the size p, where p is an integer. If $X \in \mathcal{X}$ covers three edges, then X is called a *delta*. If $X \in \mathcal{X}$ covers just two edges, then X is called an L. If $X \in \mathcal{X}$ 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 $\mathcal{X}$ with $|\mathcal{X}| = \frac{5}{2}n - h$.*

Proof: We construct $\mathcal{X}$ according to U as follows. For a trian-

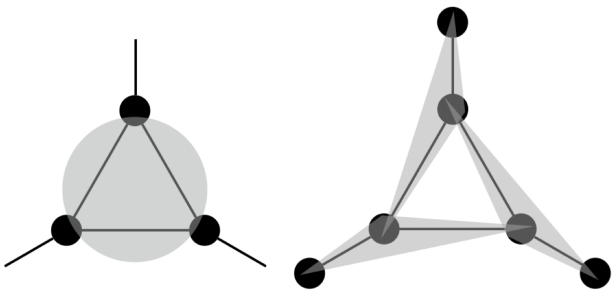


Fig. 2 Delta-triangle (left) and 3L-triangle (right).

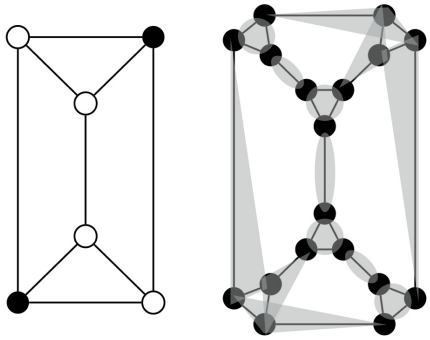


Fig. 3 U (left) and the corresponding $\mathcal{X}$ (right): In the left figure, black vertices are in U, the triangle covered by a 3L in $\mathcal{X}$.

gle in G, if the corresponding vertex in H is in U, then we let the triangle be covered by a 3L, and otherwise be covered by a delta. The remaining edges in G are covered by Is. See **Fig. 3** for example.

We calculate the size of $\mathcal{X}$: 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 3L-triangles is h, the number of delta-triangles is $n-h$. The number of link-edges covered by 3L is 3h, and the number of link-edges covered by Is is $\frac{3}{2}n - 3h$ by summing up the number of deltas, 3Ls, and Is used in $\mathcal{X}$, we get the following equations:

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

Therefore $\mathcal{X}$ is the desired K_3 edge cover. □

Note that the K_3 edge cover $\mathcal{X}$ 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 $\mathcal{X}$, then G has an independent regular K_3 edge cover $\mathcal{X}'$ with $|\mathcal{X}'| \leq |\mathcal{X}|$.*

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

- **Operation I:** If there is a pair $X, X' \in \mathcal{X}'$ such that all edges covered by X' are also covered by X, then remove X' from $\mathcal{X}'$.
- **Operation II:** If there is $X \in \mathcal{X}'$ 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 Ls, these Ls are changed to a 3L (At least one L is removed. See **Fig. 4**).

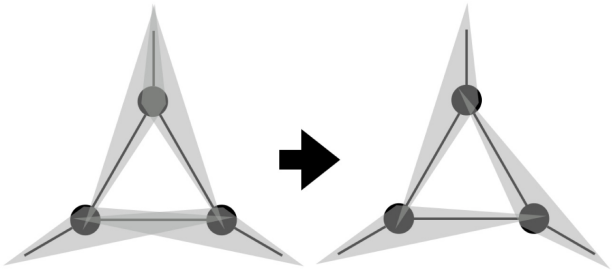


Fig. 4 Operation IV.

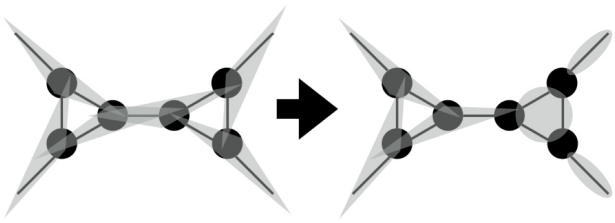


Fig. 5 Operation V.

An algorithm for changing $\mathcal{X}'$ 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
end procedure
```

We show that the above procedure stops in finite steps for any input $\mathcal{X}'$. Each operation never increases the size of $\mathcal{X}'$. Moreover, both the number of Ls and the number of $X \in \mathcal{X}'$ that cover only one triangle-edge are never increased also. Operations I and IV decrease the size of $\mathcal{X}'$, and thus they can be applied in finite times. Operations II and III decrease the number of $X \in \mathcal{X}'$ that cover only one triangle-edge and the number of Ls, 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($\mathcal{X}'$) stops, $\mathcal{X}'$ is regular.

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

- **Operation V:** If there is a pair of adjacent 3L-triangles, then replace one of the two Ls 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 apply-

ing the above operation whenever possible, $\mathcal{X}'$ finally becomes independent regular. □

Now we show the reverse of Lemma 1.

Lemma 3. *If G has a K_3 edge cover $\mathcal{X}$ with $|\mathcal{X}| = \frac{5}{2}n - h$, then H has an independent set $U \subseteq W$ with $|U| = h$.*

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

$$U := \{\ w \in W \mid \text{the corresponding triangle in } G \text{ is a } 3L\text{-triangle in } \mathcal{X}'\} \quad (2)$$

becomes an independent set. From the discussion made in the proof of Lemma 1, $|\mathcal{X}'| = \frac{5}{2}n - |U|$. Thus if $\frac{5}{2}n - |U| = |\mathcal{X}'| \leq |\mathcal{X}| = \frac{5}{2}n - h$, 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 $O(mn + 2^k m)$-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, $\gamma_3(T) = \lceil |E|/2 \rceil$.*

Proof: If G is a path, $\gamma_3(P) = \lceil \|P\|/2 \rceil$ 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 $\{T'_1, \ldots, T_{k'}\}$ 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, $\gamma_3(P) = \lceil \|P\|/2 \rceil$. By summing them up, we obtain the following upper bound.

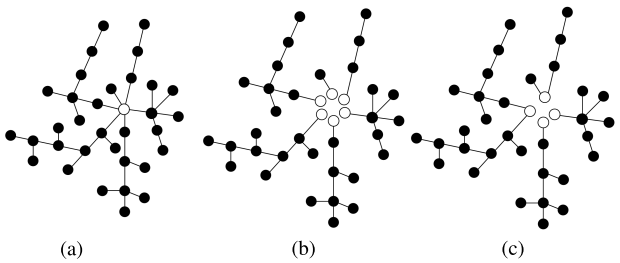


Fig. 6 (a) example of tree (one of branch point is the white circle), (b) branch point partition $\{T_1, \ldots, T_6\}$, (c) $\|T_i\|$ is connected to $\{T_1{}', \ldots, T_4{}'\}$.

$$\gamma_3(T) \leq \lceil \|T\|/2 \rceil \quad (3)$$

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 $\{X_1, \ldots, X_p\}$, and thus the following inequality holds.

$$\gamma_3(T) \geq \lceil \|T\|/2 \rceil \quad (4)$$

From Eqs. (3) and (4), $\gamma_3(T) = \lceil |E|/2 \rceil$ 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 $\lceil |E|/2 \rceil$.*

For proving this lemma, we prepare an operation as follows. Let $G = (V, E)$ be a graph and $v \in V$ 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 $|X|, |Y| \neq \emptyset$, and add edge sets $E_X = \{(v', w) \mid w \in X\}$ and $E_Y = \{(v'', w) \mid w \in Y\}$ to E. 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, $\gamma_3(T) = \lceil \|T\|/2 \rceil = \lceil |E|/2 \rceil$. 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 $\lceil |E|/2 \rceil$. 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, $O(mn + 2^k m)$. □

5. FPT Algorithm with Tree-width as a Parameter

Here we give another FPT algorithm, i.e., an FPT algorithm

with tree-width as a parameter.

Theorem 3 (reshown)**.** *For K_3EC, if a tree-decomposition of tree-width t is given, there is an $O(2^{2(t+1)(t+2)}t^2n)$-time algorithm.*

First, we assume that all graphs considered here are connected, otherwise, it is enough to manage each connected component one by one [*1]. Under this assumption, we do not need to consider I (K_3 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_3 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 $I = \{1, \ldots, r\}$ 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 $B_i = B_j = B_k$.

(3) If node i has unique child j, then "$|B_i| = |B_j| + 1$ and $B_j \subset B_i$ " 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 $(\{B_i \mid i \in I\}, T = (I, F))$ every node is one of the following four possible types.*

Leaf: A node that is a leaf.

Introduce: A node i that has a child j and $|B_i| > |B_j|$.

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* $B_\downarrow$ of a bag B is defined as follows.

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

$B^+_\downarrow := B_\downarrow \cup B$. The upper vertex set $B_\uparrow$ of the bag B is $B_\uparrow = V - B^+_\downarrow$. Also, $B^+_\uparrow := B_\uparrow \cup B$. For an arbitrary K_3 edge cover $\mathcal{X} = \{X_1, \ldots, X_p\}$ on a graph G and an arbitrary vertex subset $W \subseteq V$, $\mathcal{X}_W := \{X_i \in \mathcal{X} \mid X_i \subseteq W\}$, which is a partial solution of $\mathcal{X}$ for W. The edge set of the subgraph of $G = (V, E)$ induced by $W \subseteq V$ is denoted by E_W.

Let $G_{B^+_\downarrow} = (B^+_\downarrow, E_{B^+_\downarrow})$ be the subgraph of $G = (V, E)$ induced by $B^+_\downarrow$. Let $E^-_{B^+_\downarrow, \mathcal{X}}$ be the set of edges not covered by $\mathcal{X}_{B^+_\downarrow}$ in $E_{B^+_\downarrow}$. $\mathcal{X}^*_W$ is called an optimal partial solution for W if $\mathcal{X}^*$ is an optimal solution of K_3 edge cover.

Our algorithm traces the rooted tree T in the postorder, and for each bag B, constructs a set of partial solutions for $B^+_\downarrow$ 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 be $E^+_B := \{(v, w) \in E_{B^+_\downarrow} \mid \{v, w\} \cap B \neq \emptyset\}$, i.e., the set of edges in $E_{B^+_\downarrow}$ each edge 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 edge 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 $\mathcal{X}^j$, $j = \{1, \ldots, q\}$ be the partial solutions stored in $T[B]$. $\mathcal{X}^j$ 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 $B_\downarrow$ and uncovered by $\mathcal{X}^j$ (Note that the set of these edges is

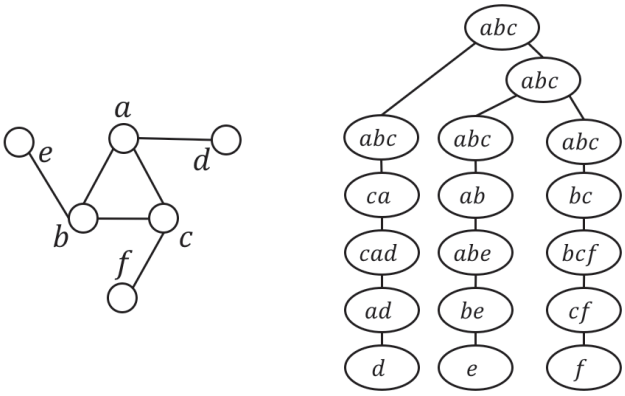
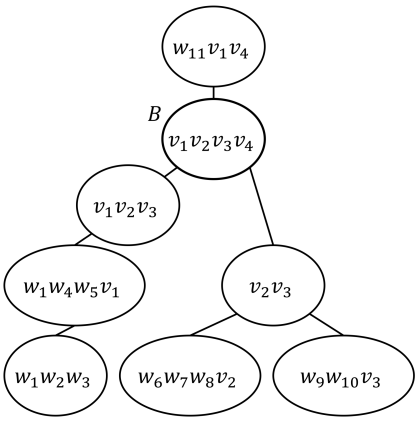
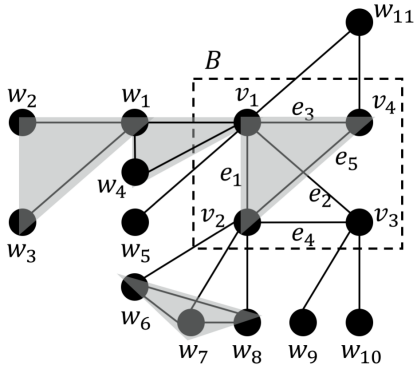


Fig. 7 An example of a graph and a nice tree-decomposition.

[*1] This is valid for K_3 edge cover. If we use K_4, this strategy does not necessarily work.



(a)



(b)

$T[B]$

First part				Second part					Third part
v_1	v_2	v_3	v_4	e_1	e_2	e_3	e_4	e_5	ω
1	1	0	0	1	0	1	0	1	4
⋮				⋮					⋮

(c)

Fig. 8 An example of $T[B]$: (a) a tree-decomposition (note that it is not nice), (b) a covering, (c) Table $T[B]$ and the row corresponding to the covering.

$E^-_{B^+_\downarrow,\mathcal{X}^j} \cap E'_B$). It will be shown later (in Lemma 10) that this number is enough to be 1 or 0. The ith column of the second part of row j is 1 if e_i is covered by $\mathcal{X}^j$; otherwise 0. The size $|\mathcal{X}^j|$ of $\mathcal{X}^j$ is stored in the third part of row j. For notational simplicity, this value $|\mathcal{X}^j|$ is represented by ω 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 $u \in B_\uparrow$ and $v \in B_\downarrow$, there is no edge between u and v, i.e., $(u, v) \notin E$.*

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 $E^-_{B^+_\downarrow,\mathcal{X}}$ are incident to vertices in B.*

Proof: Assume that an edge $(u, v) \in E^-_{B^+_\downarrow,\mathcal{X}}$ is not incident to any vertex in B, i.e., $u, v \notin B$. From $E^-_{B^+_\downarrow,\mathcal{X}} \subseteq E_{B^+_\downarrow}$ it follows that $u, v \in B_\downarrow$. Let X be an element of $\mathcal{X}$ that covers (u, v). Since $X \notin \mathcal{X}_{B^+_\downarrow}$ and $(u, v) \in E^-_{B^+_\downarrow,\mathcal{X}}$, X covers at least one edge besides edge (u, v) [*2]. Let the edge be (v, w) without loss of generality. Here, since there is no edge between $B_\uparrow$ and $B_\downarrow$, $w \in B^+_\downarrow$ follows from Lemma 8, contradicting $X \notin \mathcal{X}_{B^+_\downarrow}$. □

Lemma 10. *If there is a solution $\mathcal{X}$, then there is a solution $\mathcal{X}'$ such that $|\mathcal{X}'| \leq |\mathcal{X}|$ and $|\{(v, w) \in E^-_{B^+_\downarrow,\mathcal{X}'} \mid w \in B_\downarrow\}| \in \{0, 1\}$ for any bag B and any $v \in B$.*

Proof: For any solution $\mathcal{X}$, if there are $X = \{v, u, w\}, X' = \{v, u', w'\} \in \mathcal{X}$ with $v \in B, u, u' \in B_\uparrow (u \neq u')$ and $w, w' \in B_\downarrow$, then X and X' can be replaced with $Y = \{v, u, u'\}$ and $Y' = \{v, w, w'\}$, because there is no edge that is not covered by this replacement from Lemma 8. By applying this procedure as far as possible, the number of uncovered edges between v and $B_\downarrow$ becomes 0 or 1. □

Lemma 10 assures that each cell of the first part of each row is 1 or 0.

Lemma 11. *For two solutions $\mathcal{X}$ and $\mathcal{X}'$ and a bag B, assume that the signatures of the two partial solutions $\mathcal{X}_{B^+_\downarrow}$ and $\mathcal{X}'_{B^+_\downarrow}$ are the same. Then, there is a solution $\mathcal{X}''$ such that $\mathcal{X}''_{B^+_\downarrow} = \mathcal{X}_{B^+_\downarrow}$ and $|\mathcal{X}''| = |\mathcal{X}'| - |\mathcal{X}'_{B^+_\downarrow}| + |\mathcal{X}_{B^+_\downarrow}|$.*

Proof: If $E^-_{B^+_\downarrow,\mathcal{X}} = E^-_{B^+_\downarrow,\mathcal{X}'}$, the statement of this lemma holds by letting $\mathcal{X}'' = \mathcal{X}$ from Lemma 9. Therefore, we assume that $E^-_{B^+_\downarrow,\mathcal{X}} \neq E^-_{B^+_\downarrow,\mathcal{X}'}$ in the following part. Since $\mathcal{X}$ and $\mathcal{X}'$ have the same signature, every edge belonging to $E^-_{B^+_\downarrow,\mathcal{X}} - E^-_{B^+_\downarrow,\mathcal{X}'}$ or $E^-_{B^+_\downarrow,\mathcal{X}'} - E^-_{B^+_\downarrow,\mathcal{X}}$ has one end vertex in B and the other in $B_\downarrow$. Furthermore, for any $v \in B$, the number of edges belonging to $E^-_{B^+_\downarrow,\mathcal{X}} - E^-_{B^+_\downarrow,\mathcal{X}'}$ and incident to v is equal to the number of edges belonging to $E^-_{B^+_\downarrow,\mathcal{X}'} - E^-_{B^+_\downarrow,\mathcal{X}}$ and incident to v. Therefore, it is possible to give a one-to-one correspondence between these elements (if exist). Assume that an element X' of $\mathcal{X}' - \mathcal{X}$ covers an edge (v, w) in $E^-_{B^+_\downarrow,\mathcal{X}'} - E^-_{B^+_\downarrow,\mathcal{X}}$ (where $v \in B$ and $w \in B_\downarrow$). In this case, X' is not a delta: because if $X' = \{v, w, u\}$ is a delta, then from $X' \notin \mathcal{X}'_{B^+_\downarrow}$, $u \in B_\uparrow$ and $(u, v), (w, u) \in E$, but the existence of edge (w, u) contradicts Lemma 8. Moreover X' is not an I: otherwise, $X' = \{v, w\} \in \mathcal{X}'_{B^+_\downarrow}$, contradicts $(v, w) \in E^-_{B^+_\downarrow,\mathcal{X}'}$. Therefore, X' is an L, and in addition to (v, w) the edge to be covered is (u, v) (where $u \in B_\uparrow$) without loss of generality. Let the edge in $E^-_{B^+_\downarrow,\mathcal{X}} - E^-_{B^+_\downarrow,\mathcal{X}'}$ corresponding to (v, w) be (v, w'). If we replace X' with $\{u, v, w'\}$, the signature does not change. We change $\mathcal{X}$ by every element X' of $\mathcal{X}' - \mathcal{X}$, if it covers an edge in $E^-_{B^+_\downarrow,\mathcal{X}'} - E^-_{B^+_\downarrow,\mathcal{X}}$ replacing it as above, otherwise leaving it in the set, and get the resulting set $\mathcal{X}'''$. Here $|\mathcal{X}'''| = |\mathcal{X}' - \mathcal{X}|$ and X''' covers all edges in $E^-_{B^+_\downarrow,\mathcal{X}}$. Therefore, by letting $\mathcal{X}'' = \mathcal{X}''' \cup \mathcal{X}_B$, we obtain the desired solution. □

Lemma 12. *Let B be a forget node and B' be its child node. Let edge (u, w) satisfy $u \in B' - B$ and $w \in B'_\downarrow$. Let $\mathcal{X}$ be an arbitrary solution. Then, there is an $X \in \mathcal{X}$ such that $u, w \in X$ and $X \subseteq B'^+_\downarrow$.*

Proof: Assume that for every $X \in \mathcal{X}$ containing u and w, $X \nsubseteq B'^+_\downarrow$. Then there exists $v \in B'_\uparrow$ and $X \in \mathcal{X}$ such that $X = \{u, w, v\}$. Since B is a forget node, $B \subset B'$ and hence $v \in B_\uparrow$. On the other hand $(u, v) \in E$ or $(w, v) \in E$ must hold, because there is no I from

[*2] X is an element of $\mathcal{X}$ and $X \notin \mathcal{X}_{B^+_\downarrow}$, it means that one of the vertices in X is not in $B^+_\downarrow$. X should contain another vertex w other than u and v that is not in B, because $u, v \in B_\downarrow \subseteq B^+_\downarrow$. Therefore, we have that X covers at least one edge besides (u, v), say, (u, w) or (v, w) (I does not exist from Lemma 6).

Table 1 Since $G_B = (B = \{u\}, E_B = \emptyset)$ for leaf node $B = \{u\}$, $\mathcal{X}_B = \emptyset$ for any solution $\mathcal{X}$. Therefore, $T[B]$ looks like what is shown.

u	ω
0	0

Lemma 6, contradicting Lemma 8. □

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

Proof: From the definition of $B'_{\downarrow}$, any vertex belonging to $B'_{\downarrow}$ does not belong to B'. Therefore, from the definition of tree-decomposition (Condition 3 of Definition 1), these vertices do not belong to $B''_{\downarrow}$ 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 ω 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) $\mathcal{X}$ in B', which is one of the children of B, enumerating all possible cases of covering or uncovering edges in $E^-_{B^+_{\downarrow},\mathcal{X}}$. Since $|E^-_{B^+_{\downarrow},\mathcal{X}}|$ is bounded by a constant number, the enumeration for them can be done in constant-time. However since $E_{B^+_{\downarrow}}$ 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 $\mathcal{X}_{B^+_{\downarrow}}$ be the partial solution corresponding to the row (of $T[B']$) (See **Fig. 9**).

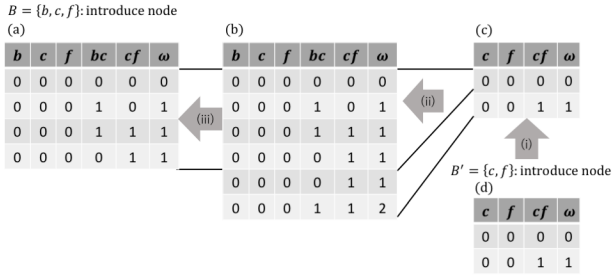


Fig. 9 An example of making $T[B]$ from $T[B']$, where $B = \{b, c, f\}$ (introduce node) and $B' = \{c, f\}$ (the child of B), on the graph and the tree-decomposition shown in Fig. 7. Each row corresponds to a partial solution of a K_3 edge cover, e.g., the first row in Table (d) shows a partial solution $\emptyset$ and the second row shows a partial solution $\{\{c, f\}\}$. Furthermore, in Table (a), the first row shows a partial solution $\emptyset$ and the third row shows a partial solution $\{\{b, c, f\}\}$. (i) Since there is no non-zero cell in the first part of $T[B']$ in (d), 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 (b) means (b, c) and (c, f) are not covered, the second row means (b, c) is covered but (c, f) is not, the third row means both (c, f) and (b, c) are covered by one K_3, and the fourth row means (c, f) is covered but (b, c) is not. (iii) Because the third row and the sixth row of (b) have the same signature and the value of ω of the latter is larger than the former, the latter (the sixth row) is deleted. Furthermore, because the fourth and the fifth rows are completely the same, only one of them is left. Consequently, the table of (a) is obtained.

Step 1. For each edge (v_i, v_j), apply the following operations: Let w_i and w_j be vertices adjacent to v_i and v_j in $B'_{\downarrow}$, 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 $\mathcal{X}_{B^+_{\downarrow}}$ 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_u and w_i be vertices adjacent to u and v_i in $B'_{\downarrow}$, 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_j 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) $\{u, v_i, w_u\}$ are covered by a K_3, and (3) $\{u, v_i, w_i\}$ are covered by a K_3. Create all cases of combinations of the above three cases, i.e,

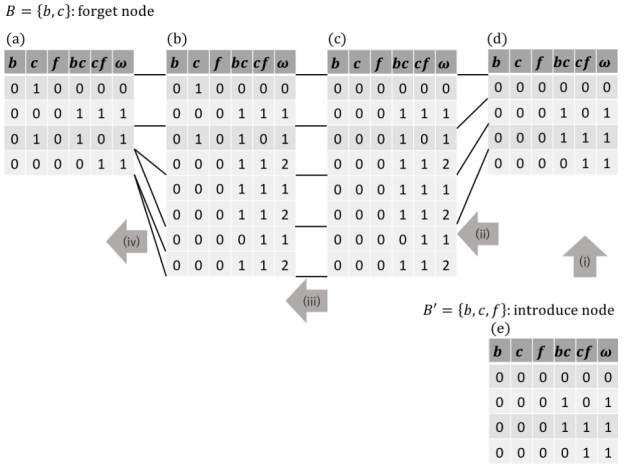


Fig. 10 An example of making $T[B]$ from $T[B']$, where $B = \{b, c\}$ (forget node) and $B' = \{b, c, f\}$ (the child of B), on the graph and the tree-decomposition shown in Fig. 7. Each row corresponds to a partial solution of a K_3 edge cover, e.g., the first row in table (d) shows a partial solution $\emptyset$ and the second row shows a partial solution $\{\{b, c, f\}\}$. (i) Since there is no non-zero cell in the first part of $T[B']$ in (e), 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 ω 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_i) is not covered (i.e., the value of (u, v_i) is 0 in the second part of $T[B]$) and v_i has 1 in the cell of the first part of $T[B']$, $\{u, v_i, w_i\}$ are covered by a K_3 and 0 is set as the value of v_i in the first part of $T[B']$ (Note that u does not exist in $B'_{\uparrow}$ 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 $\mathcal{X}'_{B^+_{\downarrow}}$ and $\mathcal{X}''_{B^+_{\downarrow}}$ be the partial solutions corresponding to the row in $T[B']$ and the row in $T[B'']$, respectively. Create a row in $T[B]$, apply 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 $\mathcal{X}'_{B^+_{\downarrow}}$ and $\mathcal{X}''_{B^+_{\downarrow}}$ (The correctness of this operation is supported by the fact $B'_{\downarrow} \cap B''_{\downarrow} = \phi$ 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 $\mathcal{X}'_{B^+_{\downarrow}}$ and $\mathcal{X}''_{B^+_{\downarrow}}$ 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 $\mathcal{X}'_{B^+_{\downarrow}}$ and $\mathcal{X}''_{B^+_{\downarrow}}$.

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 ω 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 $\mathcal{X}^*$ was deleted in Refinement Stage of node B. We denote the deleted optimal partial solution by $\mathcal{X}^*_{B^+_{\downarrow}}$. A partial solution $\mathcal{X}_{B^+_{\downarrow}}$ that has the same signature must remain. From the rule of Refinement Stage,

$$|\mathcal{X}_{B^+_{\downarrow}}| \leq |\mathcal{X}^*_{B^+_{\downarrow}}| \tag{6}$$

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

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 $\binom{t+1}{2} = t(t + 1)/2$ and there is another column for the third part. Thus the number of columns of a table is at most $t+1+t(t+2)/2+1 = O(t^2)$. 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 $2^{t(t+1)/2}$ variations for the second part, the total number of possible variations is at most $2^{t+1}2^{t(t+1)/2} = 2^{(t+1)(t+2)/2}$, 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 rows for each row of the table of the child node. Thus a total of $8 \cdot 2^{(t+1)(t+2)/2}$ rows may be created in Enumerating Stage of an introduce or forget node. In a join node, at most $(2^{(t+1)(t+2)/2})^2 = 2^{(t+1)(t+2)}$ rows are created in Enumerating Stage. By comparing $8 \cdot 2^{(t+1)(t+2)/2}$ and $2^{(t+1)(t+2)}$, it follows that the number of rows are $O(2^{(t+1)(t+2)})$.

In Refinement Stage we compare all pair of rows. Hence we must compare $O((2^{(t+1)(t+2)})^2) = O(2^{2(t+1)(t+2)})$ 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 $O(2^{2(t+1)(t+2)}t^2n)$. □

6. Conclusion

In this paper, we dealt with K_3 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 $O(mn + 2^k m)$-time algorithm and the other is $O(2^{2(t+1)(t+2)}t^2n)$-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_3EC, it may be expected that K_k 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_3EC. One of the difficulties is that K_4 can cover two non-adjacent edges simultaneously. This fact makes K_k edge cover problem in a wide sense difficult to show the hardness or give efficient algorithms.

Acknowledgments We would like to thank the Algorithms on Big Data project (ABD14) of CREST, JST (grant no. JPMJCR1402), 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 Östergård, P.R.: There are exactly five biplanes with k=11, *Journal of Combinatorial Designs*, 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.135–139 (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).

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.

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. Sine 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 focus on theoretical computer science and more specifically in parameterized complexity and special classes of graphs.

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 on 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 JCDCG[3].

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.