Complexity of the Maximum k-Path Vertex Cover Problem

Eiji Miyano[1], Toshiki Saitoh[1], Ryuhei Uehara[2], Tsuyoshi Yagita[1], Tom C. van der Zanden[3]

[1] Kyushu Institute of Technology, Japan
{miyano@, toshikis@, yagita@theory.}ces.kyutech.ac.jp
[2] Japan Advanced Institute of Science and Technology, Japan
uehara@jaist.ac.jp
[3] Utrecht University, The Netherlands
T.C.vanderZanden@uu.nl

Abstract. This paper introduces the maximum version of the k-path vertex cover problem, called the Maximum k-Path Vertex Cover problem (MaxP_kVC for short): A path consisting of k vertices, i.e., a path of length $k-1$ is called a k-path. If a k-path P_k includes a vertex v in a vertex set S, then we say that S or v covers P_k. Given a graph $G=(V,E)$ and an integer s, the goal of MaxP_kVC is to find a vertex subset $S \subseteq V$ of at most s vertices such that the number of k-paths covered by S is maximized. MaxP_kVC is generally NP-hard. In this paper we consider the tractability/intractability of MaxP_kVC on subclasses of graphs: We prove that MaxP_3VC and MaxP_4VC remain NP-hard even for split graphs and for chordal graphs, respectively. Furthermore, if the input graph is restricted to graphs with constant bounded treewidth, then MaxP_3VC can be solved in polynomial time.

1 Introduction

One of the most important and most fundamental computational problems in graph theory, combinatorial optimization, and theoretical computer science is the Minimum Vertex Cover problem (MinVC). Indeed, as one of the seminal results in computational complexity theory, the decision version of MinVC was listed in Karp's original 21 NP-complete problems in [10].

Very recently, Brešar, Kardoš, Katrenič, and Semanišin introduced a generalized variant of MinVC, called the Minimum k-Path Vertex Cover problem (MinP_kVC), motivated by the need to secure the data integrity of wireless sensor networks from attackers [5]: Let $G=(V,E)$ be a simple undirected graph, where V and E denote the set of vertices and the set of edges, respectively. $V(G)$ and $E(G)$ also denote the vertex set and the edge set of G, respectively. A path consisting of k vertices, i.e., a path of length $k-1$ is called a *k-path*. If a k-path P_k contains a vertex v in a vertex set S, then we say that the set S or the vertex v *covers* P_k. Given a graph G, the goal of MinP_kVC is to find a vertex subset $S \subseteq V(G)$ of *minimum* cardinality such that S covers all the k-paths in

G. In the same paper, Brešar et al. proved the NP-hardness of $\mathsf{Min}P_k\mathsf{VC}$ and designed a linear-time algorithm for $\mathsf{Min}P_k\mathsf{VC}$ on trees for $k \geq 3$. Furthermore, the authors proved that $\mathsf{Min}P_k\mathsf{VC}$ can be expressed by Extended Monadic Second Order Logic, which implies that $\mathsf{Min}P_k\mathsf{VC}$ can be solved in linear time on graphs with bounded treewidth by Courcelle's theorem [8]. Subsequently, due to its wide applicability to many practical problems, $\mathsf{Min}P_k\mathsf{VC}$ has been studied intensively. Indeed, for example, a large number of results on approximation [6, 12, 15, 16, 19], fixed-parameter tractability [11, 14] and exact algorithms [17] for $\mathsf{Min}P_3\mathsf{VC}$ and $\mathsf{Min}P_4\mathsf{VC}$ have been reported.

The classical/original MinVC has several variants; one of the most popular variants is the Maximum Vertex Cover problem (MaxVC), which is often called the Partial Vertex Cover problem: Given a graph G and an integer s, the goal of MaxVC is to find a vertex subset $S \subseteq V(G)$ of s vertices such that the number of edges covered by S is *maximized*. It is known that MaxVC also has many applications in real life (see, e.g., [7]). It is known [1, 7] that MaxVC is NP-hard even on bipartite graphs, though the minimization version MinVC is solvable in polynomial time on them.

For the general version $\mathsf{Min}P_k\mathsf{VC}$, therefore, it would be natural to consider the maximization problem; this paper introduces the Maximum k-Path Cover problem ($\mathsf{Max}P_k\mathsf{VC}$): Given a graph G and an integer s, the goal of $\mathsf{Max}P_k\mathsf{VC}$ is to find a vertex subset $S \subseteq V(G)$ of size at most s such that the number of k-paths covered by S is *maximized*. One can see that $\mathsf{Max}P_2\mathsf{VC}$ is generally NP-hard since it is identical to MaxVC. Therefore, we focus on the case where $k \geq 3$. For any fixed integer $k \geq 3$, $\mathsf{Max}P_k\mathsf{VC}$ is NP-hard in the general case since $\mathsf{Min}P_k\mathsf{VC}$ can be considered as a special case of $\mathsf{Max}P_k\mathsf{VC}$. In this paper, we are interested in the tractability and the intractability of $\mathsf{Max}P_k\mathsf{VC}$ on subclasses of graphs.

Our main results are summarized as follows:

(i) $\mathsf{Max}P_3\mathsf{VC}$ remains NP-hard for the class of split graphs.
(ii) $\mathsf{Max}P_4\mathsf{VC}$ remains NP-hard for the class of chordal graphs.
(iii) $\mathsf{Max}P_3\mathsf{VC}$ can be solved in polynomial time if the input graph is restricted to graphs with constant bounded treewidth.

2 Preliminaries

Let $G = (V, E)$ be a simple undirected graph, where V and E denote the set of vertices and the set of edges, respectively. $V(G)$ and $E(G)$ also denote the vertex set and the edge set of G, respectively. We denote an edge with endpoints u and v by $\{u, v\}$. A path of length $k-1$ from a vertex v_1 to a vertex v_k is represented as a sequence of vertices such that $P_k = \langle v_1, v_2, \ldots, v_k \rangle$, which is called a k-path. For a vertex v, the set of vertices adjacent to v, i.e., the *open neighborhood* of v is denoted by $N(v)$. Let $deg(v) = |N(v)|$ be the *degree* of v. Let $G[S]$ denote the subgraph of G induced by a vertex subset $S \subseteq V(G)$.

A graph G is *chordal* if each cycle in G of length at least four has at least one chord, where the chord of a cycle is an edge between two vertices of the

cycle that is not an edge of the cycle. A graph G is *split* if there is a partition of $V(G)$ into a clique set V_1 and an independent set V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V(G)$. A *treewidth* of a graph is defined in Section 5.

The problem MaxP_kVC that we study in this paper is defined as follows for any fixed integer k:

> Maximum k-Path Vertex Cover (MaxP_kVC)
> Given a graph G and an integer s, the goal of MaxP_kVC is to find a vertex subset $S \subseteq V(G)$ of size at most s such that the number of k-paths covered by S is maximized.

As mentioned in Section 1, it is known [5] that the minimum variant MinP_kVC of our problem is NP-hard for any fixed integer $k \geq 2$. It is important here to note that MinP_kVC can be considered as a special case of MaxP_kVC, i.e., the essentially equivalent goal of MinP_kVC is to find a vertex subset S of size at most s such that S covers all the k-paths in the input graph. Therefore, the NP-hardness of MaxP_kVC is straightforward:

Theorem 1. *[5] For any fixed integer k, MaxP_kVC is NP-hard.*

Moreover, it is known that MinP_3VC is a *dual* problem of the Maximum Dissociation Set problem, which was introduced in [18]. Yannakakis [18], and Papadimitriou and Yannakakis [13] proved that the problem is NP-hard even on bipartite graphs, and on planar graphs, respectively. Similarly to the above, we can obtain the following theorem:

Theorem 2. *[13, 18] MaxP_3VC is NP-hard on (i) bipartite graphs and (ii) planar graphs.*

3 NP-hardness of MaxP_3VC on split graphs and MaxP_4VC on chordal graphs

In this section, we prove the NP-hardness of MaxP_3VC on split graphs and MaxP_4VC on chordal graphs. Let us define a decision version of MaxP_3VC, denoted by MaxP_3VC(t): Given a graph G, and two integers s and t, determine if the graph G has a vertex subset $S \subseteq V(G)$ of size at most s such that the total number of 3-paths covered by S is at least t. The first result of this section is:

Theorem 3. *MaxP_3VC(t) is NP-complete, even on split graphs.*

Proof. First, we prove that MaxP_3VC(t) is in NP. Every path of three vertices in the graph G can be enumerated in $O(|V|^3)$ time, thus if we nondeterministically guess a set S of s vertices, we can check whether at least t 3-paths are covered by those s vertices in polynomial time.

Next, we show that there exists a polynomial-time reduction from the Restricted Exact Cover by Three Sets (RX3C) problem to MaxP_3VC(t). The input is a finite set $X = \{x_1, x_2, \ldots, x_{3q}\}$ of $3q$ elements and a collection $\mathcal{C}$

of $3q$ 3-element subsets of X, where each element of X appears in exactly three subsets of $\mathcal{C}$. RX3C asks if $\mathcal{C}$ contains an exact cover for X, that is, a subcollection $\mathcal{C}' \subseteq \mathcal{C}$ such that every element of X occurs in exactly one member of $\mathcal{C}'$. RX3C is shown to be NP-complete by Gonzalez [9]. We give the reduction such that the original instance of RX3C is a yes-instance if and only if the MaxP_3VC(t) instance is also a yes-instance. Let $n = 3q$ for a while. As an input of RX3C, let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n elements. Also, let $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ be a collection of n 3-element sets. Then, we construct a graph $G = (V, E)$ corresponding to an instance $(X, \mathcal{C})$ of RX3C as follows: The constructed graph G consists of the following vertices: (i) n vertices, v_{C_1} through v_{C_n}, called the *set vertices*, corresponding to the n sets, C_1 through C_n, respectively, (ii) n vertices, v_{x_1} through v_{x_n}, called the *element vertices*, corresponding to the n elements, x_1 through x_n, respectively, and (iii) corresponding to each set C_i ($i \in \{1, 2, \ldots, n\}$), n^2 vertices, $v_{C_i,1}$ through v_{C_i,n^2}, i.e., n^3 vertices in total, called *pendant vertices*. Let $C = \{v_{C_1}, v_{C_2}, \ldots, v_{C_n}\}$, $EL = \{v_{x_1}, v_{x_2}, \ldots, v_{x_n}\}$, and $P = \{v_{C_1,1}, \ldots, v_{C_1,n^2}, v_{C_2,1}, \ldots, v_{C_n,n^2}\}$. The edge set $E(G)$ is as follows: (iv) The subgraph induced by the set C of n vertices forms a clique K_n of size n, i.e., we add all possible edges between any pair of vertices in C into $E(G)$. (v) If $x_i \in C_j$ for $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, n\}$, then we add an edge $\{v_{x_i}, v_{C_j}\}$ into $E(G)$. Note that each set vertex v_{C_i} is adjacent to exactly three element vertices and furthermore each element vertex v_{x_j} is adjacent to exactly three set vertices. (vi) For each $i \in \{1, 2, \ldots, n\}$ and each $j \in \{1, 2, \ldots, n^2\}$, the pendant vertex $v_{C_i,j}$ is connected to v_{C_i} by adding an edge $\{v_{C_i,j}, v_{C_i}\}$. Finally, we set $s = q$ and $t = 81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$. This completes the reduction, which clearly can be done in polynomial time. One can verify that the constructed graph G is split since the set vertices form a clique, and the remaining vertices form an independent set.

As an example, if we are given $X = \{1, 2, 3, 4, 5, 6\}$ and a collection $\mathcal{C} = \{C_1, C_2, \ldots, C_6\} = \{\{1, 3, 5\}, \{1, 4, 5\}, \{3, 4, 6\}, \{2, 4, 6\}, \{1, 2, 6\}, \{2, 3, 5\}\}$ as an RX3C instance, the graph constructed above is illustrated in Figure 1. One can see that $\mathcal{C}' = \{C_1, C_4\}$ is a possible solution.

Before showing the correctness of our reduction, we make important observations: (1) Each set vertex v_{C_i} can cover at least $\binom{n^2}{2} = \Omega(n^4)$ 3-paths, i.e., $\langle v_{C_i,j}, v_{C_i}, v_{C_i,k} \rangle$ for $1 \leq j, k \leq n^2$ and $j \neq k$. (2) On the other hand, every element or pendant vertex can cover at most $O(n^2)$ 3-paths. Therefore, in order to cover as many 3-paths as possible, it would be the most effective to select set vertices into a solution of MaxP_3VC(t).

The following lemma shows the correctness of the reduction:

Lemma 1. *RX3C is yes if and only if MaxP_3VC(t) is yes, i.e., there is a vertex subset S of size at most q such that S can cover at least $81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$ 3-paths. (The proof is in Appendix A due to the space limitation.)*

This completes the proof of Theorem 3. □

By using a very similar reduction with small modification, we can obtain the following theorem:

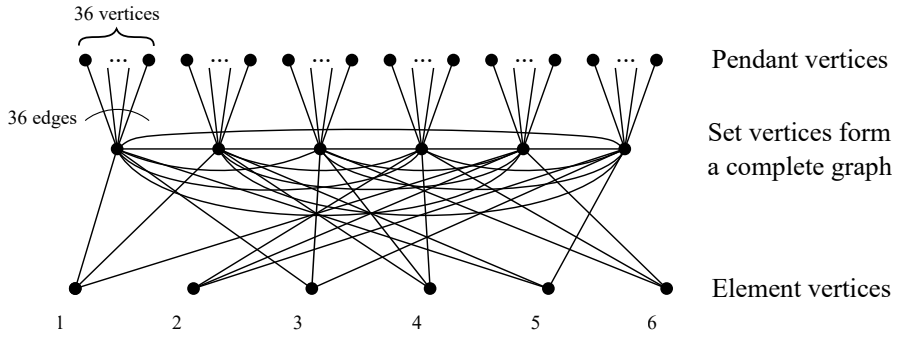

Fig. 1. Constructed graph G

Theorem 4. *MaxP_4VC(t) is NP-complete, even on chordal graphs. (The proof sketch is in Appendix B.)*

4 Algorithm for MaxP_3VC on trees

In the next section we present a polynomial-time algorithm for MaxP_3VC on graphs with bounded treewidth, but, in order to make our basic ideas clear, this section provides a simpler algorithm running in polynomial time only for MaxP_3VC on trees. In the following, let T denote the given tree, and especially, let $T_{v_{root}}$ denote the subtree of T whose root is v_{root}.

Intuitively, our algorithm is based on dynamic-programming, keeping the *minimum number of uncovered* 3-paths from the bottom to the top of the tree. For every vertex, the following two steps are considered in our algorithm: [I] **Introduce Step** and [II] **Join Step**, and in each step, the table in which the minimum number of uncovered 3-paths is stored is updated. After computing the minimum number of uncovered 3-paths of a certain subtree, our algorithm proceeds to the parent vertex u of the root of the subtree. Then, we say that u is in Introduce Step (see Figure 2). Also, there may exist some subtrees whose parent of the root of each subtree is u. In such case, our algorithm merges those subtrees one by one, by joining the same parent u, and computes the minimum number of uncovered 3-paths. In this joining step, we say u is in Join Step (see Figure 3).

Why we can compute the minimum number of uncovered 3-paths in the subtree is as follows: Assume that now we are looking at the vertex u during the bottom-up procedure. The number of uncovered 3-paths of the subtree T_u, the subtree whose root is u, can be easily counted by three information: (i) whether u is selected in the cover or not, (ii) the size of the solution given as input, which corresponds to the number of the vertices we are allowed to pick, (iii) the number of u's children not selected as the covering vertices. Since the given graph is tree,

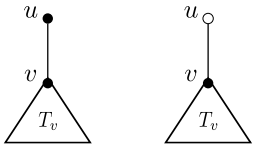

Fig. 2. Vertex u is in Introduce Step; u is in the cover (Left), or not (Right)

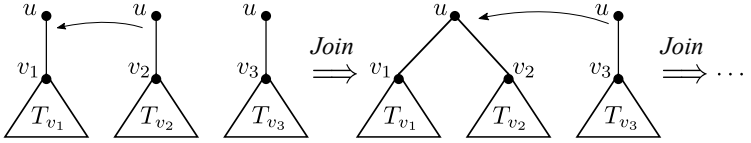

Fig. 3. Vertex u is in Join Step

the number of uncovered 3-paths only increases in the following two cases: [I] u is in **Introduce Step**. In this case, u has not started its **Join Step** yet. Thus u has a child vertex below itself which is a root of certain subtree. If neither u nor the child of u, say, v are selected in the cover, then the number of uncovered 3-paths increases depending on the number of the children of v which are not selected in the cover. [II] u is in **Join Step**. In this case, if u is *not* selected in the cover, then new uncovered 3-paths appear between the left subtree and right subtree of u. The number of those 3-paths increases depending on the number of the vertices not selected in the cover, existing in both two subtrees.

Now, we are going to show the recursive formulas with precise notation. Let $c[v; b, \ell, r]$ denote the number of uncovered 3-paths, where v denotes the vertex we are currently looking at, $b \in \{1, 0\}$ denotes whether the vertex is selected in the cover ($b = 1$) or not ($b = 0$) as a root, $\ell \in \{0, 1, 2, \ldots, s\}$ denotes the size of the solution, and $r \in \{0, \ldots, n-1\}$ denotes the number of unselected children. Note that, when a vertex is in Introduce Step and chosen in the cover, we do not need to consider the fourth argument r. This is because the 3-paths including the vertex in Introduce Step and its children are already covered by the vertex in Introduce Step. We show the tables in Table 1 and Table 2, where each entry denotes the minimum number of uncovered 3-paths under a set of some arguments. When a vertex v is in Introduce Step, first we consider two cases; $b = 1$ or $b = 0$, that is, whether we put v in the cover or not. If $b = 1$, then we only consider the solution size ℓ, ranging it from 1 to s. For example, in Table 1, *in_1* stands for $c[v; 1, 1, *]$, and there is stored the minimum number of uncovered 3-paths under these conditions. Similarly, if $b = 0$, then there are $s+1$ and n options for the solution size and the number of unselected children of the root. Each of the entry, such as *out_00*, *out_01* and so on, stores the minimum number of the 3-paths with each argument. Utilizing this table, the algorithm proceeds from the bottom to the top.

Leaf: If the vertex u is a leaf, then there are no uncovered 3-paths, thus we have $c[u; 0, 0, \text{-}] = c[u; 1, 1, \text{-}] = 0$.

Table 1. Table when the root is in the cover

Solution size	-
0	∞
1	*in_1*
2	*in_2*
$\vdots$	$\vdots$
s	*in_s*

Table 2. Table when the root is NOT in the cover

Solution size	*The number of children of the root NOT chosen in cover*			
	0	1	$\cdots$	$n-1$
0	*out_00*	*out_01*	$\cdots$	*out_0(n-1)*
1	*out_10*	*out_11*	$\cdots$	*out_1(n-1)*
2	*out_20*	*out_21*		*out_2(n-1)*
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s	*out_s0*	*out_s1*		*out_s(n-1)*

Introduce Step: If the vertex u is in Introduce Step, assuming v, the child vertex of u, has d children, we consider two cases: u is in the cover or not.

(i) u **is in the cover:** As mentioned before, we do not need to consider the fourth argument, so we have only to take care of the size of the solution which ranges from 1 to s. If the size of the solution is 1, then we refer to *out_d* of v, $c[v; 0, 0, d]$. This is because the root v is in the cover and the solution size we assume now is 1, v is not in the cover and the size of the solution for v is 0, and also v has d unselected children. If the solution size is 2, then it becomes little complicated. We have to take the minimum of {*in_1*, *out_1d*, *out_1(d-1)*} of v. This is because if the solution size is 2, from the assumption that we put u into the cover set, then we have to consider where one more vertex in the cover is in the subtree T_u. There are following three options in this case: (i) v is also in the cover set, (ii) even the children of v do not have the selected vertex, in other words, all of the d children of v are all unselected vertices, and (iii) one of the d children is in the cover set. Thus we refer three entries, and take the minimum of them. In the same manner, $c[u; 1, *, \text{-}]$ is calculated as follows, and also the table for u when the root is in (see Table 1) is updated with the following values:

$$c[u; 1, i, \text{-}] = \begin{cases} \infty & \text{if } i = 0 \\ c[v; 0, 0, d] & \text{if } i = 1 \\ \min_{0 \le j \le i-1}\{c[v; 1, i-1, *], c[v; 0, i-1, d-j]\} & \text{if } 2 \le i \le s \end{cases}$$

(ii) u **is *not* in the cover:** Since G is tree, we do not have to consider the case where the number of unselected children is $2, \cdots, n-1$. Thus we can set all

the entries of Table 2 whose number of unselected children is $2, \cdots, n-1$ with ∞. In other words, we have only to consider the case where the number of unselected children is 0 or 1. Furthermore, if u has 0 unselected children (which means v is in the cover) and the solution size is $1, \cdots, s$, it is clear that we refer to the root-in table of v, corresponding to the solution size. If u has 1 uncovered child (which similarly means v is *not* in the cover), then we have to take the minimum depending on the solution size. Thus $c[u; 0, *, *]$ is calculated as follows:

$$c[u;0,i,j] = \begin{cases} \infty & \text{if } i = 0 \text{ and } j = 0 \\ c[v;1,i,\text{-}] & \text{if } 1 \leq i \leq s \text{ and } j = 0 \\ c[v;0,0,d] & \text{if } i = 0 \text{ and } j = 1 \\ \min_{0 \leq d' \leq i}\{c[v;0,i,d-d'] + d - d'\} & \text{if } 1 \leq i \leq s \text{ and } j = 1 \end{cases}$$

Join Step: If the vertex u is in Join Step, then we update the table of u. As with **Introduce Step**, we consider two cases: the root is in the cover or not. Let us assume that u is in Join Step, and let v_L and v_R be the left and right child of u, respectively. Also, for clarity, we specially introduce u_L and u_R such that $u = u_L = u_R$, whose child is v_L and v_R respectively.

(i) u **is in the cover:** We do not have to consider the fourth argument, as we mentioned. We update the table ranging the size of the solution from 0 to s. If the size of the solution is 0, we set the entry as ∞. If the size of the solution is 1, then we just add the number of uncovered 3-paths $c[u_L; *, *, *]$ and $c[u_R; *, *, *]$. Note that u is selected in the cover, therefore the solution S has only u in this case. If the size of the solution is 2, then we have to take the minimum from two choices: one more solution is in the left subtree or the right subtree, say T_{v_L} or T_{v_R}. Thus $c[u; 1, *, \text{-}]$ is calculated as follows:

$$c[u;1,i,\text{-}] = \begin{cases} \infty & \text{if } i = 0 \\ \min_{1 \leq j \leq i}\{c[u_L;1,i-j+1,\text{-}] + c[u_R;1,j,\text{-}]\} & \text{if } 1 \leq i \leq s \end{cases}$$

(ii) u **is *not* in the cover:** If the vertex which is not selected in the cover is in Join Step, then uncovered 3-paths whose central vertex is u, in other words, the uncovered 3-paths going through from the left subtree T_{u_L} to the right subtree T_{u_R} may exist. Thus we have to take them into consideration in updating the table of u. There are two tables for subtrees T_{u_L} and T_{u_R}, so we have to take the the minimum among all the possible combinations of the size of the solution and the number of unselected children of those subtrees, considering newly appearing uncovered 3-paths going from T_{u_L} to T_{u_R}. Note that these newly appearing uncovered 3-paths can be calculated by multiplying the two numbers of unselected children, the number of the unselected children in T_{u_L} and T_{u_R}. Let i, i_L, and i_R be the variables which respectively denotes the size of the solution in the subtree T_u, T_{u_L}, and T_{u_R}. Note that $i_R = i - i_L$ holds. Also, let j, j_L, and j_R be the variables which respectively denotes the number of the unselected children in the subtree T_u,

T_{u_L}, and T_{u_R}. Note that $j_R = j - j_L$ also holds. $c[u; 0, *, *]$ is calculated as follows:

$$c[u; 0, i, j] = \min_{0 \le i \le s} \min_{0 \le j \le n-1} \{c[u_L; 0, i_L, j_L] + c[u_R; 0, i_R, j_R] + j_L \cdot j_R\}$$

Note that, since we can assume that for any vertex v, the number of unselected children r is always at least $deg(v) - s - 1$, the number of cases in the dynamic programming table is $O(s^2)$. The running time for the algorithm is dominated for that of **Join Step**, which (using this observation) is $O(s^4)$.

Theorem 5. *MaxP$_3$VC on trees of n vertices can be solved in $O(s^4 \cdot n)$ time, where s is the prescribed size of the 3-path vertex cover.*

5 Algorithm for MaxP$_3$VC on graphs with bounded treewidth

In this section, we show that MaxP$_3$VC admits a polynomial-time algorithm for graphs with bounded treewidth. In particular, we show that there exists an $O((s+1)^{2tw+4} \cdot 4^{tw} \cdot n)$-time algorithm, where tw denotes the *treewidth*, which is defined later. Thus, MaxP$_3$VC is in XP with respect to the parameter treewidth (and FPT with respect to the combined parameter $s + tw$).

Our algorithm uses dynamic programming on a *nice tree decomposition* [2] of the input graph G. Given a graph G, a *tree decomposition* of G is a tree T with for each node $v_T \in V(T)$ a subset $X_{v_T} \subseteq V(G)$ (called *bag*) such that

- for every $(u, v) \in E(G)$, there is a $v_T \in V(T)$ such that $\{u, v\} \subseteq X_{v_T}$, and
- for every $v \in V(G)$, the subset $\{v_T \in V(T) \mid v \in X_{v_T}\}$ induces a connected subtree of T.

The *width* of a tree decomposition is $\max_{v_T \in T} |X_{v_T}| - 1$, and the *treewidth* of a graph G is the minimum width taken over all tree decompositions of G. To avoid confusion, from now on we shall refer to the vertices of T as "nodes", and "vertex" shall refer exclusively to vertices of G.

We designate an arbitrary node of T as *root* of the tree decomposition. Given a node $v_T \in T$, we denote by $G[v_T]$ the subgraph of G induced by X_{v_T} and the vertices in bags of nodes which are descendants of v_T in T. We moreover assume that our (rooted) decomposition is *nice*, that is, each of the nodes $v_T \in T$ is one of the four types:

- **Leaf:** v_T is a leaf of T, and $|X_{v_T}| = 1$.
- **Introduce:** v_T has a single child node u_T, and X_{v_T} differs from X_{u_T} only by the inclusion of one additional vertex w. We say that w is *introduced* in v_T.
- **Forget:** v_T has a single child node u_T. X_{v_T} differs from X_{u_T} only by the removal of one vertex w. We say that vertex w is *forgotten* in v_T.
- **Join:** v_T has two children u_T and u'_T. Moreover, $X_{u_T} = X_{u'_T} = X_{v_T}$.

We note that a tree decomposition can be converted into a nice tree decomposition of the same width. Moreover, we can assume that the size of a tree decomposition (i.e. the number of bags) is linear in $|V(G)|$ [2].

Given a node v_T of a tree decomposition of G, a *partial solution* is a subset $S \subseteq V(G[v_T])$ of size at most s. Since the number of 3-paths in G is equal to

$$\sum_{v \in V} \frac{1}{2} deg(v)(deg(v) - 1),$$

we define the cost of a partial solution (relative to a node v_T) S to be

$$\sum_{v \in V(G[v_T]) \setminus (S \cup X_{v_T})} \frac{1}{2} deg_{v_T,S}(v)(deg_{v_T,S}(v) - 1),$$

where $deg_{v_T,S}(v)$ is taken to be the degree of v in the subgraph of G induced by $V(G[v_T]) \setminus S$. This definition, which does not take into account the degrees of the vertices in X_{v_T}, is convenient because the degrees of the vertices in X_{v_T} are not yet fixed, and may change as new vertices are introduced. However, if we assume the root bag of the tree decomposition is empty (which may be accomplished by introducing a series of forget bags after the root bag), then a partial solution with minimum cost corresponds to an optimal solution to the $\mathsf{Max}P_3\mathsf{VC}$ instance.

As is usual for dynamic programming on tree decompositions, we group partial solution by *characteristics*. Given a partial solution S for node v_T of the nice tree decomposition (with associated bag X_{v_T} and subgraph $G[v_T]$), its characteristic (ℓ, S', f) consists of the size of the solution $\ell = |S|$, its intersection with the bag $S' = X_{v_T} \cap S$, together with a function $f : X_{v_T} \to \{0, 1, \ldots, s\}$ such that $f(v) = |\{u \in S \mid u \in N(v)\}|$, which, for each vertex v in the bag X_{v_T}, tells us how many of its neighbors are in the partial solution.

For each characteristic, we store the minimum cost of a partial solution with that characteristic, which we denote by $c(\ell, S', f)$. Next, we show how to recursively compute for each type of node in a nice tree decomposition the set of characteristics of a partial solutions, and for each such characteristic, the minimum number of 3-paths not covered by a partial solution with that characteristic.

Leaf: If $v_T \in V(T)$ is a leaf node, then $X_{v_T} = \{v\}$ for some $v \in V(G)$. Then there are exactly two partial solutions for $G[v_T]$: the empty partial solution, which has characteristic $(0, \emptyset, f)$ where $f(v) = 0$ and the partial solution that includes v, which has characteristic $(1, \{v\}, f)$, where $f(v) = 0$. In both cases, $c(0, \emptyset, f) = c(1, \{v\}, f) = 0$.

Introduce: Suppose that $v_T \in V(T)$ is an introduce node, and v is the vertex being introduced. Let (ℓ, S', f) be a characteristic for the child node of v_T. In the partial solutions (for the child node) with this characteristic, we may (if $\ell < s$) choose to either add the vertex v or not. In the case where we add v, the corresponding partial solutions have characteristic $(\ell+1, S' \cup \{v\}, f')$, where $f'(v) = |\{u \in S' \mid u \in N(v)\}|$, and, if $u \neq v$ and $u \notin N(v)$, $f'(u) = f(u)$, and, if $u \neq v$ and $u \in N(v)$, $f'(u) = f(u) + 1$. In the case where we do not add v, the corresponding partial solutions will have characteristic (ℓ, S', f'), where $f'(v) =$

$|\{u \in S' \mid u \in N(v)\}|$, and, if $u \neq v$, $f'(u) = f(u)$. Since v is not adjacent to any vertex in $G[v_T] \setminus X_{v_T}$, the cost of these partial solutions remains unchanged. Note, however, that taking two partial solutions with distinct characteristics may end up having the same characteristic after vertex v is introduced. In this case, we should take the cost (for the new characteristic) to be the minimum of the costs for the original partial solutions.

Forget: Suppose that $v_T \in V(T)$ is a forget node, and v is the vertex being forgotten. Let (ℓ, S', f) be a characteristic for the child node of v_T. If S is a partial solution with this characteristic, then, viewed as a partial solution with respect to node v_T, it will have characteristic $(\ell, S' \setminus \{v\}, f')$, where f' is the restriction of f to the domain $S' \setminus \{v\}$. If $v \notin S'$ and $f(v) < deg(v)$, then the cost of this partial solution increases by $\frac{1}{2}(deg(v) - f(v))(deg(v) - f(v) - 1)$, otherwise it remains unchanged. As before, since multiple characteristics for the child node may end up having the same characteristic in v_T, and we should take the new cost of the characteristic to be the minimum of the updated costs.

Join: Suppose that $v_T \in V(T)$ is a join node, and v_T^1 and v_T^2 are its children. Let (ℓ_1, S_1', f_1) (resp., (ℓ_2, S_2', f_2)) be a characteristic for v_T^1 (resp., v_T^2). Assume that $S_1' = S_2'$, which we henceforth denote simply by S', and that $\ell_1 + \ell_2 - |S| \leq s$. If we take the union of partial solutions, S_1 relative to v_T^1 with characteristic (ℓ_1, S', f_1) and S_2 relative to v_T^2 with characteristic (ℓ_2, S', f_2), then we obtain a new partial solution (relative to v_T) with characteristic $(\ell_1 + \ell_2 - |S'|, S', f')$, where $f'(v) = f_1(v) + f_2(v) - |\{u \in S' \mid u \in N(v)\}|$. Since $V(G[v_T^1]) \cap V(G[v_T^2]) = X_{v_T}$, in $G[v_T]$, the degree of a vertex $v \in V(G[v_T]) \setminus (X_{v_T} \cup S)$ is equal to its degree in (the subgraph of G induced by) $G[v_T^1] \setminus S$ (resp., $G[v_T^2] \setminus S$) if $v \in V(G[v_T^1]) \setminus S$ (resp., $v \in V(G[v_T^2]) \setminus S$). Therefore, the cost of this new partial solution is equal to the sum of the costs of the partial solutions S_1 and S_2. Since once again, multiple (combinations of) characteristics for the child nodes may give rise to the same characteristic for v_T, we can find the minimum cost of a partial solution for a given characteristic by taking the minimum over all (combinations of) characteristics for the child nodes.

For any node, there are at most $(s+1)^{tw+2}2^{tw+1}$ characteristics. The running time is dominated by the time taken for a join node, which is $O((s+1)^{2tw+4} \cdot 4^{tw+1})$. Since we can assume that our tree decomposition has at most $O(n)$ nodes, we obtain a $O((s+1)^{2tw+4} \cdot 4^{tw} \cdot n)$-time algorithm. This assumes a tree decomposition is given as part of the input. A tree decomposition can be computed in $2^{O(tw^3)}n$ time [3], or a 5-approximate tree decomposition can be computed in time $O(1)^{tw}n$ [4].

Theorem 6. *MaxP$_3$VC on n-vertex graphs of treewidth tw can be solved in $O((s+1)^{2tw+4} \cdot 4^{tw} \cdot n)$ time, where s is the prescribe size of the 3-path vertex cover.*

References

1. Nicola Apollonio and Bruno Simeone. The maximum vertex coverage problem on bipartite graphs. *Discrete Applied Mathematics*, Vol. 165, pp. 37–48, 2014.

2. Nadja Betzler, Rolf Niedermeier, and Johannes Uhlmann. Tree decompositions of graphs: saving memory in dynamic programming. *Discrete Optimization*, Vol. 3, pp. 220–229, 2006.
3. Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, Vol. 25 (6), pp. 1305–1317, 1996.
4. Hans L Bodlaender, Pål Grønås Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $O(c^k n)$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, Vol. 45 (2), pp. 317–378, 2016.
5. Boštjan Brešar, František Kardoš, Ján Katrenič, and Gabriel Semanišin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, Vol. 159 (12), pp. 1189–1195, 2011.
6. Eglantine Camby. *Connecting hitting sets and hitting paths in graphs*. PhD thesis, Doctoral Thesis, 2015.
7. Bugra Caskurlu, Vahan Mkrtchyan, Ojas Parekh, and K Subramani. On partial vertex cover and budgeted maximum coverage problems in bipartite graphs. In *IFIP Inte. Conf. on Theoretical Computer Science*, pp. 13–26. Springer, 2014.
8. Bruno Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of Theoretical Computer Science*, Vol. B, pp. 193–242, 1990.
9. Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, Vol. 38, pp. 293–306, 1985.
10. Richard Karp. Reducibility among combinatorial problems. *Compleixty of Computer Computations*, pp. 85–103, 1972.
11. Ján Katrenič. A faster FPT algorithm for 3-path vertex cover. *Information Processing Letters*, Vol. 116 (4), pp. 273–278, 2016.
12. Xiaosong Li, Zhao Zhang, and Xiaohui Huang. Approximation algorithms for minimum (weight) connected k-path vertex cover. *Discrete Applied Mathematics*, Vol. 205, pp. 101–108, 2016.
13. Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM*, Vol. 29, No. 2, pp. 285–309, 1982.
14. Jianhua Tu and Zemin Jin. An FPT algorithm for the vertex cover P4 problem. *Discrete Applied Mathematics*, Vol. 200, pp. 186–190, 2016.
15. Jianhua Tu and Wenli Zhou. A factor 2 approximation algorithm for the vertex cover P3 problem. *Information Processing Letters*, Vol. 111 (14), pp. 683–686, 2011.
16. Jianhua Tu and Wenli Zhou. A primal–dual approximation algorithm for the vertex cover P3 problem. *Theoretical Computer Science*, Vol. 412 (50), pp. 7044–7048, 2011.
17. Mingyu Xiao and Shaowei Kou. Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theoretical Computer Science*, Vol. 657, pp. 86–97, 2017.
18. Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal of Computing*, Vol. 10, pp. 310–327, 1981.
19. Zhao Zhang, Xiaoting Li, Yishuo Shi, Hongmei Nie, and Yuqing Zhu. PTAS for minimum k-path vertex cover in ball graph. *Information Processing Letters*, Vol. 119, pp. 9–13, 2017.

Table 3. Type of 3-paths and the number of covered 3-paths

Type of 3-paths	*Number of Covered 3-paths*
Set-Set-Set	$(3q \cdot (3q-1) \cdot (3q-2) - 2q \cdot (2q-1) \cdot (2q-2))/2$
Pendant-Set-Pendant	$q \cdot (9q^2) \cdot (9q^2-1)/2$
Pendant-Set-Set	$9q^2 \cdot 3q \cdot (3q-1) - 9q^2 \cdot 2q \cdot (2q-1)$
Pendant-Set-Element	$3 \cdot q \cdot (9q^2)$
Set-Set-Element	$3q \cdot (3q-1) \cdot 3 - 2q \cdot (2q-1) \cdot 3$
Element-Set-Element	$3q$
Set-Element-Set	$6q$

APPENDIX

This appendix provides the proofs of the results that have been omitted due to space reasons. They may be read to the discretion of the program committee.

A Proof of Lemma 1

Lemma 1. RX3C is yes if and only if MaxP_3VC(t) is yes, i.e., there is a vertex subset S of size at most q such that S can cover at least $81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$ 3-paths.

Proof. Note that if the RX3C instance is a yes-instance, then the number of sets selected in $\mathcal{C}'$ is exactly q $(= n/3)$.
($\Rightarrow$) Suppose that $\mathcal{C}' \subseteq \mathcal{C}$ is an exact cover and let $\mathcal{C}' = \{C_1, C_2, \ldots, C_q\}$ of q sets. We select q vertices v_{C_1} through v_{C_q} into a solution set S (i.e., $S \subseteq C$) corresponding to q sets C_1 through C_q in $\mathcal{C}'$, respectively. To count the number of 3-paths covered by those q vertices, we divide the 3-paths into seven different types according to the types of three vertices contained in them. See Table 3. In the first column, **Set**, **Element**, and **Pendant** stand for the set, element, and pendant vertices, respectively. The second column shows the number of each type of 3-paths covered by S.

Here is the detailed estimation:

– **Set-Set-Set:** Recall that the graph induced by the set vertices is a clique of $3q$ vertices and q vertices are selected into the solution S. Note that the number of 3-paths in the clique is $3q \cdot (3q-1) \cdot (3q-2)/2$. Then, we can

show that the number of "uncovered" 3-paths is $2q \cdot (2q-1) \cdot (2q-2)/2$ as follows: Now we choose q vertices in S, and thus $G[C \setminus S]$ forms a clique of $2q$ vertices, which means there are $2q \cdot (2q-1) \cdot (2q-2)/2$ uncovered 3-paths. Hence, $(3q \cdot (3q-1) \cdot (3q-2) - 2q \cdot (2q-1) \cdot (2q-2))/2$ 3-paths are covered for this type.

- **Pendant-Set-Pendant:** We may pick any of the q covered set vertices, and form a 3-path by picking any two of its $9q^2$ incident pendant vertices. Thus $q \cdot (9q^2) \cdot (9q^2-1)/2$ 3-paths of this type are covered.
- **Pendant-Set-Set:** There are $9q^2 \cdot 3q \cdot (3q-1)$ 3-paths of this type, of which $9q^2 \cdot 2q \cdot (2q-1)$ are *not covered*. Thus $9q^2 \cdot 3q \cdot (3q-1) - 9q^2 \cdot 2q \cdot (2q-1)$ 3-paths of this type are covered.
- **Pendant-Set-Element:** We may pick any of the q covered set vertices, and form a 3-path by picking one of its $9q^2$ adjacent pendant vertices and one of its 3 adjacent set vertices. Thus $3 \cdot q \cdot (9q^2)$ 3-paths of this type are covered.
- **Set-Set-Element:** There are $3q \cdot (3q-1) \cdot 3$ 3-paths of this type, of which $2q \cdot (2q-1) \cdot 3$ are not covered. Thus $3q \cdot (3q-1) \cdot 3 - 2q \cdot (2q-1) \cdot 3$ 3-paths of this type are covered.
- **Element-Set-Element:** We may pick any of the q covered set vertices, and form a 3-path by picking two out of the three adjacent element vertices. There are thus $3q$ 3-paths of this type that are covered.
- **Set-Element-Set:** Consider any of the $3q$ element vertices. It is adjacent to 3 set vertices, and since the set vertices in the vertex cover correspond to an exact cover, exactly one of the three adjacent set vertices is in the vertex cover. Therefore, for a given element vertex, one of the paths of this type is not covered (namely the path that uses the two set vertices not in the cover), while the remaining two are. Thus $6q$ 3-paths of this type are covered.

Summing up the numbers in Table 3, we obtain that the total number of 3-paths covered by S is $81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$.

($\Leftarrow$) Conversely, suppose that there is a vertex subset $S \subseteq V$ of size at most $s = q$ which covers at least $81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$ 3-paths.

Note that if we select a pendant vertex, then the number of 3-paths covered by it is at most $9q^2 - 1$ **Pendant-Set-Pendant** 3-paths, $3q - 1$ **Pendant-Set-Set** 3-paths, and 3 **Pendant-Set-Element** 3-paths. If we select an element vertex, then the number of 3-paths covered by it is at most 6 **Element-Set-Element** 3-paths, $9q$ **Element-Set-Set** 3-paths, and $18q^2$ **Element-Set-Pendant** 3-paths.

On the other hand, if we select a set vertex, then the number of 3-paths covered by it is at least $(9q^2 - q + 1) \cdot (9q^2 - q) \geq 62q^4 \geq 33q^2$ **Pendant-Set-Pendant** 3-paths. Since this is (for $q \geq 1$) strictly more than the number of 3-paths covered by a pendant vertex or an element one, we can assume that the solution consists of exactly q set vertices.

The number of **Set-Set-Set**, **Pendant-Set-Pendant**, **Pendant-Set-Set**, **Pendant-Set-Element**, **Set-Set-Element** and **Element-Set-Element** 3-paths that is covered, depends only on the number of set vertices selected and is (if we

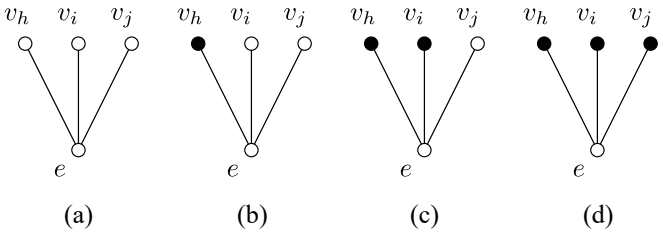

Fig. 4. 4 ways of choosing set vertices v_h, v_i, v_j

select q set vertices) equal to $81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + q$. Thus, at least $6q$ **Set-Element-Set** 3-paths must be covered.

Figure 4 shows a situation from the viewpoint of an element vertex; there are four situations on the selection of the three neighbors of each element vertex: (a) None of the set vertices adjacent to it is chosen, and (b) one, (c) two, and (d) all three set vertices are chosen. **Set-Element-Set** 3-paths involving this vertex are covered, in the second situation (b) two out of the three possible 3-paths are covered, and in the situations (c) and (d) all of the three possible 3-paths are covered.

Since every set vertex is adjacent to exactly three element vertices, there are exactly $3q$ edges from a set vertex in S to an element vertex. Situation (a) covers 0 3-paths at the cost of 0 such edges, situation (b) covers 2 3-paths at the cost of 1 edge, and situations (c) and (d) cover 3 3-paths at the expense of 2 or 3 such edges. Since, on average, one such edge should cover 2 3-paths, and there is no situation in which one edge covers more than 2 3-paths, we know that situations (c) and (d) do not occur (or else we would not be able to cover at least $6q$ **Set-Element-Set** 3-paths, since this would cause the average to drop below 2 3-paths covered per vertex-in-cover-to-element edge). Since the cover includes $3q$ vertex-in-cover-to-element edges, each element vertex must be in situation (b) (or else, we would have less than $3q$ such edges in total). This shows that the sets corresponding to the selected set vertices form an exact cover. □

B Proof Sketch of Theorem 4

Theorem 4. MaxP_4VC(t) is NP-complete, even on chordal graphs.

Proof. (Sketch) We only give the proof sketch of the NP-hardness of MaxP_4VC on chordal graphs. Our basic idea is very similar to that of MaxP_3VC on split graphs; we also reduce from RX3C. We only show how to construct the graphs in the reduction, and explain the correctness of the reduction intuitively. The detailed proof will appear in the full version of this paper.

As with MaxP_3VC, the input of RX3C is a finite set $X = \{x_1, x_2, \ldots, x_{3q}\}$ of $3q$ elements and a collection $\mathcal{C}$ of $3q$ 3-element subsets of X, where each element

of X appears in exactly three subsets of $\mathcal{C}$. See Figure 5. Roughly speaking, we replace every element vertex with one triangle, called a *element triangle*, i.e., we prepare one triangle of 3 vertices for each element x_i, instead of one vertex in the case of MaxP_3VC. Then, we connect the set vertex to the three vertices of the element triangle if the above corresponding set contains the element. Furthermore, we replace $9q^2$ *pendant edges* with $9q^2$ pendant vertices in the case of MaxP_3VC. The constructed graph is clearly a chordal graph.

Let us take a look at an example shown in Figure 6. Similarly to the example in Section 3, assume that $X = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{C} = \{C_1, C_2, \ldots, C_6\} = \{\{1, 3, 5\}, \{1, 4, 5\}, \{3, 4, 6\}, \{2, 4, 6\}, \{1, 2, 6\}, \{2, 3, 5\}\}$ as an RX3C instance. Figure 6 illustrates the constructed graph, but, for simplicity, some edges and triangles for elements are omitted.

Now we make observations to (sketchily) prove the correctness of this reduction in the following: (i) The set vertices can "effectively" cover many 4-paths since there are $\Omega(q^4)$ paths, say, **Pendant-Pendant-Set-Pendant** or **Pendant-Set-Pendant-Pendant** paths. This enforces the optimal solution of MaxP_4VC to consist of all the set vertices. (ii) In MaxP_3VC, there is the difference between the numbers of covered 3-paths in the case whether the selected vertices cover the same element or not. If the selected vertices cover the same element, there is at least one doubly covered 3-path, denoted as **Set-Element-Set** path in Section 3. Similarly in this MaxP_4VC reduction shown in Figure 6, for example, if we select two vertices which correspond to two sets C_4 and C_5 as the solution of MaxP_4VC, then two 4-paths $\langle C_4, e_6^2, e_6^1, C_5\rangle$ and $\langle C_4, e_6^1, e_6^3, C_5\rangle$ are covered twice. Therefore, the number of covered 4-paths does not increase much. However, such paths do not appear if we select two vertices C_1 and C_4 as the solution of MaxP_4VC, which corresponds to the optimal solution of RX3C. As a result, the number of covered 4-paths should be maximized. Details are omitted here. □

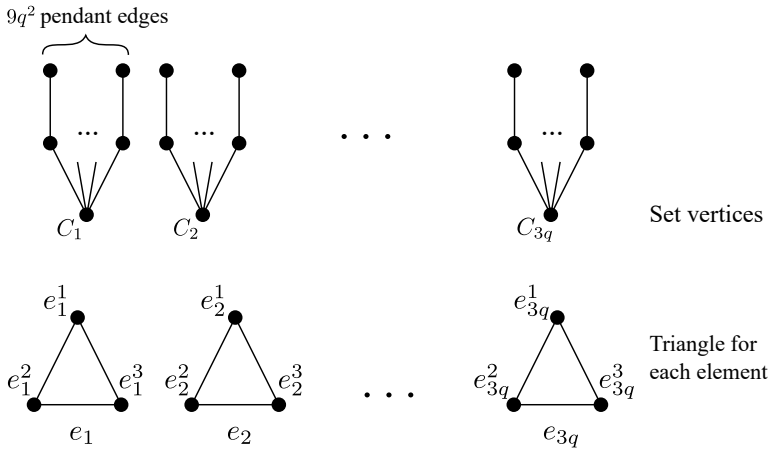

Fig. 5. (Bottom) Each element vertex is replaced with one element triangle of 3 vertices, and (top) each pendant vertex is replaced with one pendant edge

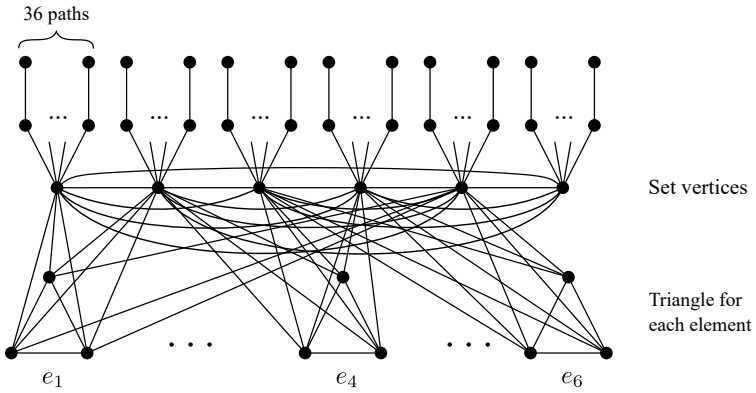

Fig. 6. Example of the constructed graph G