Vertex Cover Gets Faster and Harder on Low Degree Graphs

Akanksha Agrawal[1], Sathish Govindarajan[1], Neeldhara Misra[1]

Indian Institute of Science, Bangalore
{*akanksha.agrawal|gsat|neeldhara*}*@csa.iisc.ernet.in*

Abstract. The problem of finding an optimal vertex cover in a graph is a classic NP-complete problem, and is a special case of the hitting set question. On the other hand, the hitting set problem, when asked in the context of induced geometric objects, often turns out to be exactly the vertex cover problem on restricted classes of graphs. In this work we explore a particular instance of such a phenomenon. We consider the problem of hitting all axis-parallel slabs induced by a point set P, and show that it is equivalent to the problem of finding a vertex cover on a graph whose edge set is the union of two Hamiltonian Paths. We show the latter problem to be NP-complete, and we also give an algorithm to find a vertex cover of size at most k, on graphs of maximum degree four, whose running time is $1.2637^k n^{O(1)}$.

1 Introduction

Let P be a set of n points in $\mathbb{R}^2$ and let $\mathcal{R}$ be the family of all distinct objects of a particular kind (disks, rectangles, triangles, ...), such that each object in $\mathcal{R}$ has a distinct tuple of points from P on its boundary. For example, $\mathcal{R}$ could be the family of $\binom{n}{2}$ axis parallel rectangles such that each rectangle has a distinct pair of points of P as its diagonal corners. $\mathcal{R}$ is called the set of all objects induced (spanned) by P.

Various questions related to geometric objects induced by a point set have been studied in the last few decades. A classical result in discrete geometry is the *First Selection Lemma* [1] which shows that there exists a point that is present in a constant fraction of triangles induced by P. Another interesting question is to compute the minimum set of points in P that "hits" all the induced objects in $\mathcal{R}$. This is a special case of the classical Hitting Set problem, which we will refer to as *Hitting Set for Induced Objects*.

For most geometric objects, it is not known if the *Hitting Set for induced objects* problem is polynomially solvable. It is known to be polynomial solvable for skyline rectangles and halfspaces. Recently, Rajgopal et al [9] showed that this problem is NP-complete for lines.

The problem of finding an optimal vertex cover in a graph is a classic NP-complete problem, and is a special case of the Hitting Set problem. On the other hand, the hitting set for induced objects problem often turns out to be exactly the vertex cover problem, even on restricted classes of graphs. For example, the problem of hitting set for induced axis-parallel rectangles is equivalent to the vertex cover on the Delaunay graph of the point set with respect to axis-parallel rectangles.

We study a particular phenomenon of this type, where the hitting set question in the geometric setting boils down to a vertex cover problem on a structured graph class. We consider the problem of hitting set for induced axis-parallel slabs (rectangles whose horizontal or vertical sides are unbounded). Note that this is even more structured than general axis-parallel rectangles, and indeed, it turns out that the corresponding Delaunay graph has a very special property — its edge set is the union of two Hamiltonian paths. Since any hitting set for the class of axis-parallel slabs induced by a point set P is exactly the vertex cover of the Delaunay graph with respect to axis-parallel slabs for P, our problem reduces to solving vertex cover on the class of graphs whose edge set is simply the union of two Hamiltonian Paths.

Despite the appealing structure, we show that – surprisingly – deciding k-vertex cover on this class of graphs is NP-complete. This involves a rather intricate reduction from the problem of finding a vertex cover on cubic graphs. We also appeal to the fact that the edge set of four-regular graphs can be partitioned into two two-factors, and the main challenge in the reduction involves stitching the components of the two-factors into two Hamiltonian paths while preserving the size of the vertex cover in an appropriate manner.

Having established the NP-hardness of the problem, we pursue the question of improved fixed-parameter algorithms on this special case. Vertex Cover is one of the most well-studied problems in the context of fixed-parameter algorithm design, it enjoys a long list of improvements even on special graph classes. We note that for Vertex Cover, the goal is to find a vertex cover of size at most k in time $\mathcal{O}(c^k)$, and the "race" involves exploring algorithms that reduce the value of the best known constant c.

In particular, even for sub-cubic graphs (where the maximum degree is at most three, and the problem remains NP-complete), Xiao [11] proposed an algorithm with running time $\mathcal{O}^\star(1.1616^k)$, improving on the previous best record [5] of $\mathcal{O}^\star(1.1940^k)$ by Chen, Kanj and Xia, and prior to this, Razgon [10] had a $\mathcal{O}^\star(1.1864^k)$. The best-known algorithm for Vertex Cover [4] on general graphs has a running time of $\mathcal{O}(1.2738^k + kn)$ and uses polynomial-space.

Typically, these algorithms involve extensive case analysis on a cleverly designed search tree. In the second part of this work, we propose a branching algorithm with running time $O^\star(1.2637^k)$ for graphs with maximum degree bounded by at most four. This improves the best known algorithm for this class, which surprisingly has been no better than the algorithm for general graphs. We note

that this implies faster algorithms for the case of graphs that can be decomposed into the union of two Hamiltonian Paths (since they have maximum degree at most four), however, whether they admit additional structure that can be exploited for even better algorithms remains an open direction.

2 Preliminaries

In this section, we state some basic definitions and introduce terminology from graph theory and algorithms. We also establish some of the notation that will be used throughout.

We denote the set of natural numbers by $\mathbb{N}$ and set of real numbers by $\mathbb{R}$. For a natural number n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a finite set A we denote by $\mathfrak{S}_A$ the set of all permutations of the elements of set A. To describe the running times of our algorithms, we will use the O^* notation. Given $f : \mathbb{N} \to \mathbb{N}$, we define $O^*(f(n))$ to be $O(f(n) \cdot p(n))$, where $p(\cdot)$ is some polynomial function. That is, the O^* notation suppresses polynomial factors in the running-time expression.

Graphs. In the following, let $G = (V, E)$ be a graph. For any non-empty subset $W \subseteq V$, the subgraph of G induced by W is denoted by $G[W]$; its vertex set is W and its edge set consists of all those edges of E with both endpoints in W. For $W \subseteq V$, by $G \setminus W$ we denote the graph obtained by deleting the vertices in W and all edges which are incident to at least one vertex in W.

A *vertex cover* is a subset of vertices S such that $G \setminus S$ has no edges. We denote a vertex cover of size at most k of a graph G by $k-VC(G)$. For $v \in V$ we denote the open-neighborhood of v by $N(v) = \{u \in V | (u, v) \in E\}$, closed-neighborhood of v by $N[v] = N(v) \cup \{v\}$, second-open neighborhood by $N_2(v) = \{u \in V | \exists u' \in N(v)$ s.t. $(u, u') \in E\}$ second-closed neighborhood by $N_2[v] = N_2(v) \cup N[v]$.

When we are discussing a pair of vertices u, v, then the common neighborhood of u and v is the set of vertices that are adjacent to both u and v. In this context, a vertex w is called a *private neighbor* of u if (w, u) is an edge and (w, v) is not an edge. We denote the degree of a vertex $v \in V$ by $d(v)$.

A *path* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_k$ such that (v_i, v_{i+1}) is an edge for all $0 \leq i \leq (k-1)$. A *Hamiltonian path* of a graph G is a path featuring every vertex of G. The following class of graphs will be of special interest to us.

Definition 1 (Braid graphs). *A graph G on the vertex set $[n]$ is a braid graph if the edges of the graph can be covered by two Hamiltonian paths. In other words, there exist permutations σ, τ of the vertex set for which $E(G) = \{(\sigma(i), \sigma(i+1)) \mid 1 \leq i \leq n-1\} \cup \{(\tau(i), \tau(i+1)) \mid 1 \leq i \leq n-1\}$.*

Induced axis-parallel slabs: Axis-parallel slabs are a special class of axis-parallel rectangles where two horizontal or two vertical sides are unbounded. Each pair of points $p(x_1, y_1)$ and $q(x_2, y_2)$ induces two axis-parallel slabs of the form $[x_1, x_2] \times (-\infty, +\infty)$ and $(-\infty, +\infty) \times [y_1, y_2]$. Let $\mathcal{R}$ represent the family of $2\binom{n}{2}$ axis-parallel slabs induced by P.

We refer the reader to [6] for details on standard graph theoretic notation and terminology we use in the paper.

Parameterized Complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where x is a classical problem instance, and k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x, k), decidability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size.

3 Hitting Set for Induced Axis-Parallel Slabs

We show here that the problem of finding a hitting set of size at most k for the family of all axis-parallel slabs induced by a point set is equivalent to the problem of finding a vertex cover of a graph whose edges can be partitioned into two Hamiltonian Paths. In subsequent sections, we establish the NP-hardness of the latter problem, and also provide better FPT algorithms. Due the equivalence of these problems, we note that both the hardness and the algorithmic results apply to the problem of finding a hitting set for induced axis parallel slabs.

Lemma 1. *An instance of k-vertex cover in a braid graph $G = (V, E)$ with permutations $\sigma, \tau \in \mathfrak{S}_V$ can be reduced to the problem of finding a hitting set for the collection of all axis-parallel slabs induced by a point set. d*

Proof (Sketch). Given an instance of Vertex Cover on a braid graph G with permutations σ and τ, we create n points in $\mathbb{R}^2$ in an $(n \times n)$-grid as follows. For every $1 \leq i \leq n$, we let $p_i = (\sigma(i), \tau(i))$. Since we only need to hit empty vertical and horizontal slabs, in the induced setting this amounts to hitting all consecutive slabs in the horizontal and vertical directions. It is easy to check that a hitting set for such slabs would exactly correspond to a vertex cover of G. □

Lemma 2. *The problem of finding a hitting set for all induced axis-parallel slabs by a point set P can be reduced to the problem of finding a Vertex Cover in a braid graph.*

Proof. From the given point set P, we sort the points in P according to their x-coordinates to obtain a permutation of the point set σ. Similarly, we sort with

respect to y-coordinate to get a permutation τ. Note that there exists a empty axis-parallel slab between two points if and only if they are adjacent with respect to at least one of the x- or y-coordinates, These are, on the other hand, precisely the edges in the braid graph with σ and τ as the permutations, which shows the equivalence. □

4 NP-completenes of Vertex Cover on Braids

In this section, we show that the problem of determining a vertex cover on the class of braids is hard even when the permutations of the braid are given as input.

The intuition for the hardness is the following. Consider a four-regular graph. By a theorem of Peterson, we know that the edges of such a graph can be partitioned into two sets, each of which would be a two-factor in the graph G. In other words, every four-regular graph can be thought of as a union of two collections of disjoint cycles, defined on same vertex set. It is conceivable that these cycles can be patched together into paths, leading us to a braid graph. As it turns out, for such a patching, we need to have some control over the cycles in the decomposition to begin with. So we start with an instance of Vertex Cover on a cubic 2-connected planar graphs, morph such an instance to a four-regular graph while keeping track of a special cycle decomposition, which we later exploit for the "stitching" of cycles into Hamiltonian paths.

Formally, therefore, the proof is by a reduction from Vertex Cover on a cubic 2-connected planar graph to an instance of k-vertex cover on a braid graph, noting that [7] shows the NP-hardness of Vertex Cover for cubic planar 2-connected graphs. We describe the construction in two stages, first showing the transformation to a four-regular graph and then proceeding to illustrate the transformation to a braid graph.

Due to space constraints, we only provide the highlights of the reduction. One of the main tasks is to merge the cycles in each decomposition. Let us first illustrate a gadget that combines two cycles into a longer one.[1] Note that the gadget itself must be a braid, and of course, we need to ensure equivalence.

For the purpose of this brief discussion, our starting point is a four-regular graph G. Recall that the edge set of G can be decomposed into two collections of cycles. Note that every vertex v participates in two cycles, say C_v and C'_v — these would be cycles from different collections. Now let the neighbors of v in C_v be v_1, v_2, and let the neighbors in C'_v be v_3 and v_4.

We are now ready to describe the gadget W_v. This gadget has four entry points, namely v', v'', and a, b. The gadget is shown in Figure 1. It is easy to check that

[1] At this point, we are not concerned that this is leading us to, eventually, a Hamiltonian cycle rather than a path, because it is quite easy to convert the former to the latter.

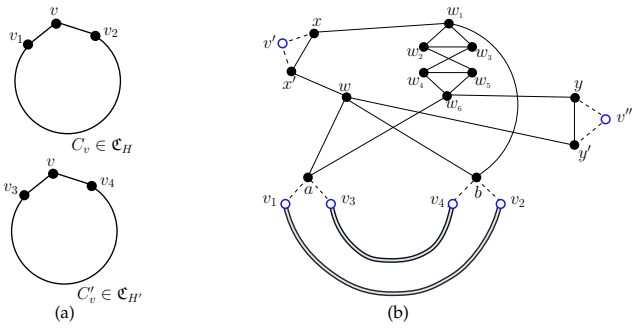

Fig. 1. Stitching together one pair of cycles with a common vertex v

.

the gadget induces a braid. Now, in G, to insert this gadget, we remove v from G, and make v_1, v_2 adjacent to a and v_3, v_4 adjacent to b. Let us denote this graph by G'. Note that there is a path from v_1 to v_2 along the cycle C_v and there is a path from v_3 to v_4 along the cycle C'_v.

For equivalence, we need to be sure that if even one of v_1, v_2, v_3, v_4 is not picked in a vertex cover of G', then we have enough room for the vertex v in the reverse direction. To this end, we show the following crucial property of the gadget W_v.

Lemma 3. *Let S' be any vertex cover of G. If one of a or b belongs to S', then $|S' \cap V(W)| = 10$. On the other hand, there exists a vertex cover S' of G', that contains neither a nor b, for which $|S' \cap V(W)| = 9$.*

Proof. The vertices $\{v', x, x'\}$, $\{v'', y, y'\}$, $\{w_1, w_2, w_3\}$, $\{w_4, w_5, w_6\}$ form triangles, and (w, a) is an edge disjoint from these triangles. Therefore, we clearly require minimum of 9 vertices to cover edges of W alone. If we have $S' = \{x, x', y, y', w_1, w_2, w_4, w_6, w\}$ then we can cover all edges of W with 9 vertices. This proves the second part of the claim.

Now, let S' be a vertex cover such that $a \in S'$. Then, apart from the K_3's present we have an edge (w, b), so including a we need at least 10 vertices. An analogous argument holds when b is present (edge (w, a) left). If we have $V' = \{x, x', y, y', w_1, w_2, w_4, w_6, a, b\}$ then we can cover all edges of W with 10 vertices. □

Corollary 1. *We have a vertex cover of size k in G if and only if G' admits a vertex cover of size $k + 9$.*

Proof. Let S be a vertex cover of G. If $v \notin S$, then $\{v_1, v_2, v_3, v_4\} \subseteq S$. Therefore, we may cover the edges of W using the vertices $\{x, x', y, y', w_1, w_2, w_4, w_6, w\}$ since there is no external obligation to pick either a or b, and this would be an extension of S with at most nine additional vertices. If $v \in S$, then let

$S^\star := S \setminus \{v\}$. We extend $S^\star$ by the set $\{x, x', y, y', w_1, w_2, w_4, w_6, a, b\}$, which also adds up to $k+9$. In the reverse direction, given a vertex cover of size $k+9$, we know that at least nine vertices of S' are from W. Let $S^\dagger$ denote the remaining vertices of S'. If all of $\{v_1, v_2, v_3, v_4\} \in S'$, then note that $S^\star$ is a vertex cover of size k for G. If one of $\{v_1, v_2, v_3, v_4\} \notin S'$, then $S^\dagger \cup \{v\}$ is a vertex cover of size at most k in G. The size bound comes from Lemma 3 and the fact that either a or b belongs to S' due to the case we are considering.

We should be able to use this gadget repeatedly to stitch all cycles into two single long cycles. However, the iterative process involves several challenges. For instance, if some gadgets are already inserted, the paths along the cycles that we had before may not be so readily available. Also, while the process of breaking the cycle at a vertex v is clear, it is not obvious as to how one would mimic this construction for a neighbor of v after v has been suitably replaced. The concern here is that a straightforward application of the gadget will cause vertices from two different gadgets to become adjacent, which we would like to avoid if we are to maintain the braid structure of the gadget itself.

To address the former problem we create a slightly different gadget that creates artificial paths that can be used if the original cycle is broken by some previously inserted gadget. For the second problem, we start our reduction from cubic graphs and proceed in a manner so as to ensure that the cycle decompositions of the reduced graph are somewhat special. This allows us to choose mutually non-adjacent breakpoints v.

Finally, these gadgets must be tied together into a path, which we organize with the help of connection gadgets. The reader is referred to the full version of this work for the complete details.

Theorem 1. *The problem of finding a vertex cover of size at most k in a braid graph is* NP-complete.

5 An Improved Branching Algorithm

In this section we describe an improved FPT algorithm for the vertex cover problem on graphs with maximum degree at most four. The algorithm is essentially a search tree, and the analysis is based on the branch-and-bound technique. We use standard notation with regards to branching vectors as described in [8].The input to the algorithm is denoted by a pair (G, k), where G is a graph, and the question is whether G admits a vertex cover of size at most k.

We work with k, the size of the vertex cover sought, as the measure — sometimes referred to as the *budget*. When we say that we *branch on a vertex v*, we mean that we recursively generate two instances, one where v belongs to the vertex cover, the other where v does not belong to the vertex cover. This is a standard method of exhaustive branching, where the measure drops, respectively, by one

and $d(v)$ in the two branches (since the neighbors of v are forced to be in the vertex cover when v does not belong to the vertex cover).

Preprocessing. We begin by eliminating simplicial vertices, that is, vertices whose neighborhoods form a clique. If the graph induced by $N[v]$ is a clique, then it is easy to see that there is a minimum vertex cover containing $N(v)$ and not containing v (by a standard shifting argument). We therefore preprocess the graph in such a situation by deleting $N[v]$ and reducing the budget to $k-|N(v)|$.

Our algorithm makes extensive use of the *folding* technique, as described in past work [2, 3]. This allows us to preprocess vertices of degree two in polynomial time, while also reducing the size of the vertex cover sought by one. We briefly describe how we might handle degree-2 vertices in polynomial time. Suppose v is a degree-2 vertex in the graph G with two neighbors u and w such that u and w are not adjacent to each other. We construct a new graph G' as follows: remove the vertices v, u, and w and introduce a new vertex $v^\star$ that is adjacent to all neighbors of the vertices u and w in G (other than v). We say that the graph G' is obtained from the graph G by "folding" the vertex v, and we say that $v^\star$ is the vertex generated by folding v, or simply that $v^\star$ is the *folded vertex* (when the context is clear). It turns out that the folding operation preserves equivalence, as shown below.

Proposition 1. *[2, Lemma 2.3] Let G be a graph obtained by folding a degree-2 vertex v in a graph G, where the two neighbors of v are not adjacent to each other. Then the graph G has a vertex cover of size bounded by k if and only if the graph G' has a vertex cover of size bounded by $(k-1)$.*

Note that the new vertex generated by the folding operation can have more than four neighbors, especially if the vertices adjacent to the degree two vertex have, for example, degree four to begin with. The branching algorithm that we will propose assumes that we will always find a vertex whose degree is bounded by 3 to branch on, therefore it is important to avoid the situation where the graph obtained after folding all available degree two vertices is completely devoid of vertices of degree bounded by three (which is conceivable if all degree three vertices are adjacent to degree two vertices that in turn get affected by the folding operation). Therefore, we apply the folding operation somewhat tactfully– we apply it only when we are sure that the folded vertex has degree at most four. We call such a vertex a *foldable* vertex. Further, a vertex is said to be *easily foldable* if, after folding, it has degree at most 3. We avert the danger of leading ourselves to a four-regular graph recursively by explicitly ensuring that vertices of degree at most three are created whenever a folded vertex has degree four. Note that in the preprocessing step we will be folding only easily foldable vertices.

Typically, we ensure a reasonable drop on all branches by creating the following win-win situation: if a vertex is foldable, then we fold it, if it is not, then this is the case since there are sufficiently many neighbors in the second neighborhood

of the vertex, and in many situations, this would lead to a good branching vector. Also, during the course of the branching, we appeal to a couple of simple facts about the structure of a vertex cover, which we state below.

Lemma 4. *[2, First part of Lemma 3.2] Let v be a vertex of degree 3 in a graph G. Then there is a minimum vertex cover of G that contains either all three neighbors of v or at most one neighbor of v.*

This follows from the fact that a vertex cover that contains v (where $d(v) = 3$) and two of its neighbors can be easily transformed into one, of the same size, that omits v and contains all of its neighbors.

Proposition 2. *If x, a, y, b form a cycle of length four in G (in that order), and the degree of a and b in G is two, then there exists an optimal vertex cover that does not pick a or b and contains both x and y.*

Overall Algorithm. To begin with, the branching algorithm tries to branch mainly on a vertex of degree three or two. If the input graph is four-regular, then we simply branch on an arbitrary vertex to create two instances both of which have at least one vertex of degree at most three. We note that this is an off-branching step, in the future, the algorithm maintains the invariant that at each step, the smaller graph produced has at least one vertex whose degree is at most three.

After this, we remove all the simplicial vertices and then fold all easily-foldable vertices. If a degree two vertex v with neighbors u and w is not easily-foldable, then note that there exists an optimal vertex cover that either contains v or does not contain v and includes both its neighbors. Indeed, if an optimal vertex cover S contains, say v and u, then note that $(S \setminus \{v\}) \cup \{w\}$ is a vertex cover of the same size. So we branch on the vertex v:

- when v does not belong to the vertex cover, we pick u, w in the vertex cover, leading to a drop of two in the measure,
- when v does belong to the vertex cover, we have that $N(u) \cup N(w)$ must belong to the vertex cover, and we know that $|N(u) \cup N(w) \setminus \{v\}| \geq 4$ (otherwise, v would be easily-foldable), and this leads to a drop of five in the measure.

So we either preprocess degree two vertices in polynomial time, or branch on them with a branching vector of $(2, 5)$. At the leaves of this branching tree, if we have a sub-cubic graph, then we employ the algorithm of [11]. Otherwise, we have at least one degree three vertex which is adjacent to at least one degree four vertex. We branch on these vertices next. The case analysis is based on the neighborhood of the vertex — broadly, we distinguish between when the neighborhood has at least one edge, and when it has no edges. The latter case is the most demanding in terms of a case analysis. For the rest of this section, we describe all the scenarios that arise in this context.

Degree three vertices with edges in their neighborhood. For this part of the algorithm, we can always assume that we are given a degree three vertex with a degree four neighbor. Let v be a degree three vertex, and let $N(v) := \{u, w, x\}$, where we let u denote a degree four vertex. Note that u, w, x does not form a triangle, otherwise v would be a simplicial vertex and we would have handled it earlier. So, we deal with the case when $N(v)$ is not a triangle, but has at least one edge. If (w, x) is an edge, then we branch on u:

- when u does not belong to the vertex cover, we pick four of its neighbors in the vertex cover, leading to a drop of four in the measure,
- when u does belong to the vertex cover, we delete u from the graph, and we are left with v, w, x being a triangle where v is a degree two vertex, and therefore we may pick w, x in the vertex cover — together, this leads to a drop of three in the measure.

On the other hand, if w, x is not an edge, then there is an edge incident to u. Suppose the edge is u, w (the case when the edge is u, x is symmetric). In this case, we branch on x exactly as above. The measure may drop by three when x does not belong to the vertex cover, if x happens to be a degree three vertex. Therefore, our worst-case branching vector in the situation when $N(v)$ is not a triangle, but has at least one edge is $(3, 3)$.

Degree three vertices whose neighborhoods are independent. Here we consider several cases. Broadly, we have two situations based on whether u, w, x have any common neighbors or not.

Before embarking on the case analysis, we describe a branching strategy for some specific situations — these mostly involve two non-adjacent vertices that have more than two neighbhors in common, with at least one of them of degree 4. This will be useful in scenarios that arise later.

We consider the case when a degree four vertex p non-adjacent to a vertex q has at least three neighbhors in commmon, say a, b, c and let x be the other neighbhor of p that may or may not be adjacent to q. Notice that there always exists an optimal vertex cover that either contains both p and q or omits both p and q. To see this, consider an optimal vertex cover S that contains p and omits q. Then, S clearly contains a, b, c. Notice now that $T := (S \setminus \{p\}) \cup \{x\}$ is also a vertex cover, and T contains neither p or q, and has the same size as S. This suggests the following branching strategy:

1. If p and q both belong to the vertex cover, then the measure clearly drops by two. We proceed by deleting p and q from G. Now note that the degree of the vertices $\{a, b, c\}$ reduces by two, and they become vertices of degree one or two (note that they cannot be isolated because we always begin by eliminating vertices of degree two by preprocessing or branching). If any one of these vertices is simplicial or foldable then we process it or fold it respectively. Otherwise, we branch on a:

   (a) when a does not belong to the vertex cover, we pick its neighbhors in the vertex cover, leading to a drop of two in the measure.
   (b) when a does belong to the vertex cover, we have that its second neighborhood must belong to the vertex cover, and this leads to a drop of six in the measure.
2. If p and q are both omitted from the vertex cover, then we pick a, b, c, x in the vertex cover and the measure drops by four.

Note that if a is foldable in $G \setminus \{p, q\}$, then we have the branch vector $(3, 4)$, otherwise, we have the branch vector $(4, 8, 4)$. We refer to the branching strategies outlined above as the **CommonNeighborBranch** strategy.

A broad overview of all the other cases is as follows.

1. **Scenario A.** There exists a vertex t that is adjacent to at least two vertices in $N(v)$. Further, t is adjacent to u and one other vertex.
   - The vertex t has degree four.
   - The vertex t has degree three, u, w, x have degree four, and $(t, x) \notin E$. We let u' and u'' denote the neighbors of u other than t and v.
     - The degree of both u' and u'' is four.
     - At least one of u' and u'' has degree three.
2. **Scenario B.** There exists a vertex t that is adjacent to at least two vertices in $N(v)$. The vertex t is not adjacent to u and is therefore adjacent to w and x.
3. **Scenario C.** The vertices u, v, w have no common neighbors other than v. We have the following cases based on degree of w, x.
   - The degree of both w and x is three.
   - The degree of both w and x is four.
   - The degree of w is four and x is three.

Theorem 2. *There is an algorithm that determines if a graph with maximum degree at most four has a vertex cover of size at most k in $\mathcal{O}^*(1.2637^k)$ worst-case running time.*

6 Conclusions

In this work we showed that the problem of hitting all axis-parallel slabs induced by a point set P is equivalent to the problem of finding a vertex cover on a graph whose edge set is the union of two Hamiltonian Paths. We established that this problem is NP-complete. Finally, we also gave an algorithm for Vertex Cover on graphs of maximum degree four whose running time is $O^\star(1.2637^k)$. It would be interesting to know if there are better algorithms for braid graphs in particular.

Scenario	Cases	Branch Vector	c
Scenario A	Case 1	$(2,5)$	1.2365
		$(7,4,5)$	1.2365
		$(7,9,5,5)$	1.2498
		$(2,10,6)$	1.2530
		$(7,4,10,6)$	1.2475
		$(7,9,5,10,6)$	1.2575
	Case 2 (I)	$(4,7,5)$	1.2365
		$(9,5,7,5)$	1.2498
		$(4,7,10,6)$	1.2475
		$(9,5,7,10,6)$	1.2575
	Case 2 (II)	$(4,5,6)$	1.2498
		$(4,10,6,6)$	1.2590

Scenario	Cases	Branch Vector	c
CNB		$(2,5)$	1.2365
		$(3,4)$	1.2207
		$(4,8,4)$	1.2465
Degree Two Edge in $N(v)$		$(2,6)$	1.2365
		$(3,3)$	1.2599
Scenario B		$(2,5)$	1.2365
		$(2,6,10)$	1.2530
Scenario C	Case 1	$(2,10,6)$	1.2530
	Case 2	$(8,3,8,7)$	1.2631
	Case 3	$(7,3,5)$	1.2637
		$(5,7,7,6)$	1.2519
		$(10,6,7,7,6)$	1.2592

Fig. 2. The branch vectors and the corresponding running times across various scenarios and cases. (This table is a truncated version due to lack of space.)

References

[1] Endre Boros and Zoltan Füredi. "The number of triangles covering the center of an n-set". In: *Geometriae Dedicata* 17 (1984), pp. 69–77.

[2] Jianer Chen, Iyad A. Kanj, and Weijia Jia. "Vertex Cover: Further Observations and Further Improvements". English. In: *Graph-Theoretic Concepts in Computer Science*. Vol. 1665. 1999, pp. 313–324.

[3] Jianer Chen, Iyad A. Kanj, and Ge Xia. "Improved Parameterized Upper Bounds for Vertex Cover". In: *Mathematical Foundations of Computer Science 2006*. Vol. 4162. 2006, pp. 238–249.

[4] Jianer Chen, Iyad A. Kanj, and Ge Xia. "Improved upper bounds for vertex cover". In: *Theor. Comput. Sci* 411.40-42 (2010), pp. 3736–3756.

[5] Jianer Chen, Iyad A. Kanj, and Ge Xia. "Labeled Search Trees and Amortized Analysis: Improved Upper Bounds for NP-Hard Problems". In: *Algorithmica* 43.4 (2005), pp. 245–273.

[6] Reinhard Diestel. *Graph Theory*. Third. Springer-Verlag, Heidelberg, 2005.

[7] Bojan Mohar. "Face Covers and the Genus Problem for Apex Graphs". In: *Journal of Combinatorial Theory, Series B* 82.1 (2001), pp. 102 –117.

[8] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, 2006.

[9] Ninad Rajgopal et al. "Hitting and Piercing Rectangles Induced by a Point Set". In: *COCOON*. 2013, pp. 221–232.

[10] Igor Razgon. "Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3". In: *J. Discrete Algorithms* 7.2 (2009), pp. 191–212.

[11] Mingyu Xiao. "A Note on Vertex Cover in Graphs with Maximum Degree 3". In: *Computing and Combinatorics*. Vol. 6196. 2010, pp. 150–159.