Parameterized Power Vertex Cover

Eric Angel[1], Evripidis Bampis[2], Bruno Escoffier[2], Michael Lampis[3]

1. IBISC, Université Evry Val d'Essone, angel@ibisc.fr
2. Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, {euripidis.bampis,bruno.escoffier}@upmc.fr
3. CNRS UMR 7243 and Université Paris Dauphine, France michail.lampis@dauphine.fr

Abstract. We study a recently introduced generalization of the Vertex Cover (VC) problem, called Power Vertex Cover (PVC). In this problem, each edge of the input graph is supplied with a positive integer *demand*. A solution is an assignment of (power) values to the vertices, so that for each edge one of its endpoints has value as high as the demand, and the total sum of power values assigned is minimized.
We investigate how this generalization affects the complexity of Vertex Cover from the point of view of *parameterized algorithms*. On the positive side, when parameterized by the value of the optimal P, we give an $O^*(1.274^P)$ branching algorithm, and also an $O^*(1.325^P)$ algorithm for the more general asymmetric case of the problem, where the demand of each edge may differ for its two endpoints. When the parameter is the number of vertices k that receive positive value, we give $O^*(1.619^k)$ and $O^*(k^k)$ algorithms for the symmetric and asymmetric cases respectively, as well as a simple quadratic kernel for the asymmetric case.
We also show that PVC becomes significantly harder than classical VC when parameterized by the graph's treewidth t. More specifically, we prove that unless the ETH is false, there is no $n^{o(t)}$ algorithm for PVC. We give a method to overcome this hardness by designing an FPT *approximation scheme* which obtains a $(1+\epsilon)$-approximation to the optimal solution in time FPT in parameters t and $1/\epsilon$.

1 Introduction

In the classical Vertex Cover (VC) problem, we are given a graph $G = (V, E)$ and we aim to find a minimum cardinality cover of the edges, i.e. a subset of the vertices $C \subseteq V$ such that for every edge $e \in E$, at least one of its endpoints belongs to C. Vertex Cover is one of the most extensively studied NP-hard problems in both approximation and parameterized algorithms [15, 13].

In this paper, we study a natural generalization of the VC problem, which we call Power Vertex Cover (PVC). In this generalization, we are given an edge-weighted graph $G = (V, E)$ and we are asked to assign (power) values to its vertices. We say that an edge e is covered if at least one of its endpoints is assigned a value greater than or equal to the weight of e. The goal is to determine a valuation such that all edges are covered and the sum of all values assigned is minimized. Clearly, if all edge weights are equal to 1, then this problem coincides with VC.

Power Vertex Cover was recently introduced in [1], motivated by practical applications in sensor networks (hence the term "power"). The main question posed in [1] was whether this more general problem is harder to approximate than VC. It was then shown that PVC retains enough of the desirable structure of VC to admit a similar 2-approximation algorithm, even for the more general case where the power needed to cover the edge (u, v) is not the same for u and v (a case referred to as Directed Power Vertex Cover (DPVC)).

The goal of this paper is to pose a similar question in the context of parameterized complexity: is it possible to leverage known FPT results for VC to obtain FPT algorithms for this more general version? We offer a number of both positive and negative results. Specifically:

- When the parameter is the value of the optimal solution P (and all weights are positive integers), we show an $O^*(1.274^P)$ branching algorithm for PVC, and an $O^*(1.325^P)$ algorithm for DPVC. Thus, in this case, the two problems behave similarly to classical VC.

- When the parameter is the *cardinality* k of the optimal solution, that is, the number of vertices to be assigned non-zero values, we show $O^*(1.619^k)$ and $O^*(k^k)$ algorithms for PVC and DPVC respectively, as well as a simple quadratic kernel for DPVC, similar to the classical Buss kernel for VC. This raises the question of whether a kernel of order *linear* in k can be obtained. We give some negative evidence in this direction, by showing that an LP-based approach is very unlikely to succeed. More strongly, we show that, given an optimal *fractional* solution to PVC which assigns value 0 to a vertex, it is NP-hard to decide if an optimal solution exists that does the same.

- When the parameter is the treewidth t of the input graph, we show through an FPT reduction from Clique that there is no $n^{o(t)}$ algorithm for PVC unless the ETH is false. This is essentially tight, since we also supply an $O^*((\Delta + 1)^t)$ algorithm, and is in stark contrast to VC, which admits an $O^*(2^t)$ algorithm. We complement this hardness result with an FPT approximation scheme, that is, an algorithm which, for any $\epsilon > 0$ returns a $(1 + \epsilon)$-approximate solution while running in time FPT in t and $\frac{1}{\epsilon}$. Specifically, our algorithm runs in time $\left(O(\frac{\log n}{\epsilon})\right)^t n^{O(1)}$.

Our results thus indicate that PVC occupies a very interesting spot in terms of its parameterized complexity. On the one hand, PVC carries over many of the desirable algorithmic properties of VC: branching algorithms and simple kernelization algorithms can be directly applied. On the other, this problem seems to be considerably harder in several (sometimes surprising) respects. In particular, neither the standard treewidth-based DP techniques, nor the Nemhauser-Trotter theorem can be applied to obtain results comparable to those for VC. In fact, in the latter case, the existence of edge weights turns a trivial problem (all vertices with fractional optimal value 0 are placed in the independent set) to an NP-hard one. And yet, despite its added hardness, PVC in fact admits an FPT approximation scheme, a property that is at the moment known for only a handful of other W-hard problems. Because of all these, we view the results of this paper as a first step towards a deeper understanding of a natural generalization of VC that merits further investigation.

Due to space constraints, some proofs are in a separate Appendix.

Previous work As mentioned, PVC and DPVC were introduced in [1], where 2-approximation algorithms were presented for general graphs and it was proved that, like VC, the problem can be solved in polynomial time for bipartite graphs.

Vertex Cover is one of the most studied problems in FPT algorithms, and the complexity of the fastest algorithm as a function of k has led to a long "race" of improving results, see [14, 3] and references therein. The current best result is a $O^*(1.274^k)$-time polynomial-space algorithm. Another direction of intense interest has been kernelization algorithms for VC, with the current best being a kernel with (slightly less than) $2k$ vertices [9, 7, 4]. Because of the importance of this problem, numerous variations and generalizations have also been thoroughly investigated. These include (among others): Weighted VC (where each vertex has a cost) [14], Connected VC (where the solution is required to be connected) [5, 12], Partial VC (where the solution size is fixed and we seek to maximize the number of covered edges) [8, 11] and Capacitated VC (where each vertex has a capacity of edges it can dominate) [6, 8]. Of these, all except Partial VC are FPT when parameterized by k, while all except Capacitated VC are FPT when parameterized by the input graph's treewidth t. Partial VC is known to admit an FPT approximation scheme parameterized by k [11], while Capacitated VC admits a *bi-criteria* FPT approximation scheme parameterized by t [10], that is, an algorithm that returns a solution that has optimal size, but may violate some capacity constraints by a factor $(1 + \epsilon)$.

In view of the above, and the results of this paper, we observe that PVC displays a different behavior than most VC variants, with Capacitated VC being the most similar. Note though, that for PVC we are able to obtain a (much simpler) $(1 + \epsilon)$-approximation for the problem, as opposed to the bi-criteria approximation known for Capacitated VC. This is a consequence of a "smoothness" property displayed by one problem and not the other, namely, that any solution that slightly violates the feasibility constraints of PVC can be transformed into a feasible solution with almost the same value. This property separates the two problems, motivating the further study of PVC.

2 Preliminaries

We use standard graph theory terminology. We use n to denote the order of a graph, Δ to denote its maximum degree. We also use standard parameterized complexity terminology, and refer the reader to related textbooks [13] for the definitions of notions such as FPT, kernel, treewidth.

In the DPVC problem we are given a graph $G(V, E)$ and for each edge $(u, v) \in E$ two positive integer values $w_{u,v}$ and $w_{v,u}$. A feasible solution is a function that assigns to each $v \in V$ a value p_v such that for all edges we have either $p_u \geq w_{u,v}$ or $p_v \geq w_{v,u}$. If for all edges we have $w_{u,v} = w_{v,u}$ we say that we have an instance of PVC.

Both of these problems generalize VERTEX COVER, which is the case where $w_{u,v} = 1$ for all edges $(u, v) \in E$. In fact, there are simple cases where the problems are considerable harder.

Theorem 1. PVC *is NP-hard in complete graphs, even if the weights are restricted to* $\{1, 2\}$*. It is even APX-hard in this class of graphs, as hard as approximate as* VC*.*

As a consequence of the above, PVC is hard on any class of graphs that contains cliques, such interval graphs. In the remainder we focus on classes that do not contain all cliques, such as graphs of bounded treewidth.

3 Parameterizing by treewidth

3.1 Hardness for Treewidth

Theorem 2. *If there exists an algorithm which, given an instance* $G(V, E)$ *of* PVC *with treewidth* t*, computes an optimal solution in time* $|V|^{o(t)}$*, then the ETH is false. This result holds even if all weights are polynomially bounded in* $|V|$*.*

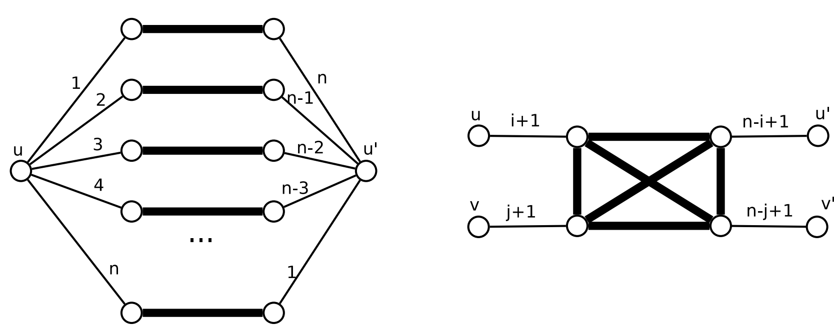

Fig. 1. Main gadgets of Theorem 2. Thick lines represent weight n edges.

Proof. We describe a reduction from k-Multicolored Independent Set. In this problem we are given a graph whose vertex set has been partitioned into k cliques $V_1, \ldots, V_k$ and we are asked if this graph contains an independent set of size k. We assume without loss of generality that $|V_1| = |V_2| = \ldots = |V_k| = n$ and that the vertices of each part are numbered $\{1, \ldots, n\}$. It is known that if an algorithm can solve this problem in $n^{o(k)}$ then the ETH is false.

Our reduction relies on two main gadgets, depicted in Figure 1. We first describe the choice gadget, depicted on the left side of the figure. This gadget contains two vertices u, u' that will be connected to the rest of the graph. In addition, it contains n independent edges, each of which is given weight n. Each edge has one of its endpoints connected to u and the other to u'. The weights assigned are such that no two edges incident on u have the same weight, and for each internal edge the weight of the edges connecting it to u, u' add up to $n+1$.

The first step of our construction is to take k independent copies of the choice gadget, and label the high-degree vertices $u_1, \ldots, u_k$ and $u'_1, \ldots, u'_k$. As we will see, the idea of the reduction is that the power assigned to u_i will encode the choice of vertex for the independent set in V_i in the original graph.

We now consider the second gadget of the figure (the checker), which consists of a K_4, all of whose edges have weight n. We complete the construction as follows: for every edge of the original graph, if its endpoints are the i-th vertex of V_c and the j-th vertex of V_d, we add a copy of the checker gadget, where each of the vertices u_c, u'_c, u_d, u'_d is connected to a distinct vertex of the K_4. The weights are $i+1, n-i+1, j+1, n-j+1$ for the edges incident on u_c, u'_c, u_d, u'_d respectively.

This completes the description of the graph. We now ask if there exists a power vertex cover with total cost at most $k(n^2+n)+3mn$, where m is the number of edges of the original graph. Observe that the treewidth of the constructed graph is $2k+O(1)$, because deleting the vertices $u_i, u'_i, i \in \{1, \ldots, k\}$ turns the graph into a disconnected collection of K_2s and K_4s.

First, suppose that the original graph has an independent set of size k. If the independent set contains vertex i from the set V_c, we assign the value i to u_c and $n-i$ to u'_c. Inside each choice gadget, we consider each edge incident on u_c not yet covered, and we assign value n to its other endpoint. Similarly, we consider each edge incident on u'_c not yet covered and assign value n to its other endpoint. Since all weights are distinct and from $\{1, \ldots, n\}$, we will thus select $n-i$ vertices from the uncovered edges incident on u_i and i vertices from the uncovered edges incident on u'_i, thus the total value spent on each choice gadget is n^2+n. To see that this assignment covers also the weight n edges inside the matching, observe that since the edges connecting each to u, u' have total weight $n+1$, at least one is not covered by u_c or u'_c, thus one of the internal endpoints is taken.

Let us now consider the checker gadgets. Recall that we have one such gadget for every edge. Consider an edge between the i-th vertex of V_c and the j-th vertex of V_d, so that the weights are those depicted in Figure 1. Because we started from an independent set of G we know that for the values we have assigned at least one of the following is true: $p_{u_c} \neq i$ or $p_{u_d} \neq j$, since these values correspond to the indices of the vertices of the independent set. Suppose without loss of generality that $p_{u_c} \neq i$, therefore $p_{u_c} > i$ or $p_{u_c} < i$. In the first case, the edge connecting u_c to the K_4 is already covered, so we simply assign value n to each of the three vertices of the K_4 not connected to u_c. In the second case, we recall that we have assigned $p_{u'_c} = n - p_{u_c}$ therefore the edge incident on u'_c is covered. Thus, in both cases we can cover all edges of the gadget for a total cost of $3n$. Thus, if we started from an independent set of the original graph we can construct a power vertex cover of total cost $k(n^2 + n) + 3mn$.

For the other direction, suppose that a vertex cover of cost at most $k(n^2 + n) + 3mn$ exists. First, observe that since the checker gadget contains a K_4 of weight n edges, any solution must spend at least $3n$ to cover it. There are m such gadgets, thus the solution spends at most $k(n^2 + n)$ on the remaining vertices.

Consider now the solution restricted to a choice gadget. A first observation is that there exists an optimal solution that assigns all degree 2 vertices values either 0 or n. To see this, suppose that one such vertex has value i, and suppose without loss of generality that it is a neighbor of u. We set its value to 0 and the value of u to $\max\{i, p_u\}$. This is still a feasible solution of the same or lower cost.

Suppose that the optimal solution assigns total value at most $n^2 + n$ to the vertices of a choice gadget. It cannot be using fewer than n degree-two vertices, because then one of the internal weight n edges will be uncovered, thus it spends at least n^2 on such vertices. Furthermore, it cannot be using $n + 1$ such vertices, because then it would have to assign 0 value to u_i, u'_i and some edges would be uncovered. Therefore, the optimal solution uses exactly n degree-two vertices, and assigns total value at most n to u_i, u'_i. We now claim that the total value assigned to u_i, u'_i must be exactly n. To see this, suppose that $p_{u_i} + p_{u'_i} < n$. The total number of edges covered by u_i, u'_i is then strictly less than n. There exist therefore $n + 1$ edges incident on u_i, u'_i which must be covered by other vertices. By pigeonhole principle, two of them must be connected to the same edge. But since we only selected one of the two endpoints of this edge, one of the edges must be uncovered.

Because of the above we can now argue that if the optimal solution has total cost at most $k(n^2 + n) + 3mn$ it must assign value exactly $3n$ to each checker gadget and $n^2 + n$ to each choice gadget. Furthermore, this can only be achieved if $p_{u_c} + p_{u'_c} = n$ for all $c \in \{1, \ldots, k\}$. We can now see that selecting the vertex with index p_{u_c} in V_c in the original graph gives an independent set. To see this, suppose that $p_{u_c} = i$ and $p_{u_d} = j$ and suppose that an edge existed between the corresponding vertices in the original graph. It is not hard to see that in the checker gadget for this edge none of the vertices u_c, u'_c, u_d, u'_d covers its incident edge. Therefore, it is impossible to cover everything by spending exactly $3n$ on this gadget. □

3.2 Exact and Approximation Algorithms for Treewidth

In the previous section we showed that PVC is much harder than Vertex Cover, when parameterized by treewidth. This raises the natural question of how one may be able to work around this added complexity. Our first observation is that, using standard techniques, it is possible to obtain FPT algorithms for this problem by adding extra parameters. In particular, if W is the maximum weight of any edge and Δ is the maximum degree of the input graph, we have the following:

Theorem 3. *There exists an algorithm which, given an instance of* DPVC *and a tree decomposition of width* t*, computes an optimal solution in time* $(W+1)^t n^{O(1)}$*. Similarly, there exists an algorithm that performs the same in time* $(\Delta+1)^t n^{O(1)}$*.*

Theorem 3 indicates that the problem's hardness for treewidth is not purely combinatorial; rather, it stems mostly from the existence of large numbers, which force the natural DP table to grow out of proportion. Using this intuition we are able to state the main algorithmic result of this section which shows that, in a sense, the problem's W-hardness with respect to treewidth is "soft": even if we do not add extra parameters, it is always possible to obtain in FPT time a solution that comes arbitrarily close to the optimal.

Theorem 4. *There exists an algorithm which, given an instance of* DPVC*,* $G(V,E)$ *and a graph decomposition of* G *of width* t*, for any* $\epsilon > 0$*, produces a* $(1+\epsilon)$*-approximation of the optimal in time* $\left(O(\frac{\log n}{\epsilon})\right)^t n^{O(1)}$*. Therefore,* DPVC *admits an FPT approximation scheme parameterized by treewidth.*

Proof. (Sketch) The proof relies on two rounding steps. In the first, we deal with the case where the maximum weight is not polynomially bounded in n. In that case, we divide all weights by an appropriate value, so that the maximum weight becomes polynomial in n, while losing a $(1+\epsilon)$ factor in optimality (this is similar to standard techniques, e.g. for Knapsack). We now have an (almost) equivalent instance where $W = n^c$. We now replace all edge weights by setting $w_{u,v} := \lfloor \log_{(1+\epsilon)}(w_{u,v}) \rfloor$. The idea is that this does not significantly affect the feasibility constraints (by more than $(1+\epsilon)$); indeed, a vertex that was receiving value p_v in the old instance may now take value $\lfloor \log_{(1+\epsilon)} p_v \rfloor$ in the new one. We can now modify the algorithm of Theorem 3 to calculate the optimal in the new instance. Because the new maximum value is now $\log_{(1+\epsilon)} W$ we get the promised running time. □

4 Parameterizing by total power

We focus in this section on the standard parameterization: given an edge-weighted graph G and an integer P (the parameter), we want to determine if there exists a solution of total power at most P. We first focus on PVC and show that it is solvable within time $O^*(1.274^P)$, thus reaching the same bound as VC (when parameterized by the solution value). We then tackle DPVC where a more involved analysis is needed, and we reach time $O^*(1.325^P)$.

4.1 PVC

The algorithm for PVC is based on the following simple property.

Property 1. Consider an edge $e = (u,v)$ of maximum weight. Then, in any optimal solution either $p_u = w_e$ or $p_v = w_e$.

This property can be turned into a branching rule: considering an edge $e = (u, v)$ of maximum weight, then either set $p_u = w_e$ (remove u and incident edges), or set $p_v = w_e$ (remove v and incident edges). This already shows that the problem is FPT, leading to an algorithm in $O^*(2^P)$. To improve this and get the claimed bound, we also use the following reduction rule.

(RR1) Suppose that there is (u, v) with $w_{u,v} = M$, and the maximum weight of other edges incident to u and v is $B \leq M - 1$. Then set $w_{u,v} = B$, and do $P \leftarrow P - (M - B)$.

Property 2. (RR1) is correct.

Now, consider the following branching algorithm.

	Algorithm 1
STEP 1:	While (RR1) is applicable, apply it;
STEP 2:	If $P < 0$ return NO;
	If the graph has no edge return YES;
STEP 3:	If the maximum weight of edges is 1:
	Apply an algorithm in $O^*(1.274^k)$ for VC.
STEP 4:	Take two adjacent edges $e = (u, v)$ and $f = (u, w)$ of maximum weight. Branch as follows:
	- either set $p_u = w_e$ (remove u, set $P \leftarrow P - w_e$)
	- or set $p_v = p_w = w_e$ (remove v and w, set $P \leftarrow P - 2w_e$)

Theorem 5. *Algorithm 1 solves* PVC *in time* $O^*(1.274^P)$.

4.2 DPVC

For DPVC, the previous simple approach does not work. Indeed, there might be no pair of vertices (u, v) such that both $w_{u,v}$ is the maximum weight of arcs starting at u, and $w_{v,u}$ is the maximum weight of arcs starting at v. If we branch on a pair (u, v), the only thing that we know is that either $p_u \geq w_{u,v}$, or $p_v \geq w_{v,u}$. Setting a constraint $p_u \geq w$ corresponds to the following operation $Adjust(u, w)$.

Definition 1. $Adjust(u, w)$ *consists of decreasing weights of each arc starting at u by w, and decreasing P by w.*

Of course, if the weight of an arc becomes 0 or negative, then it is removed (as well as the reverse arc). In our algorithm, a typical branching will be to take an edge (u, v), and either to apply $Adjust(u, w_{u,v})$ (and solve recursively), or to apply $Adjust(v, w_{v,u})$ (and solve recursively). Another possibility is to set the power of a vertex u to a certain power w. In this case we must use v to cover (v, u) if $w_{u,v} > w$. Formally:

Definition 2. $Set(u, w)$ *consists of (1) setting $p_u = w$, removing u (and incident edges), (2) decreasing P by w, (3) applying $Adjust(v, w_{v,u})$ for all (v, u) such that $w_{u,v} > w$.*

Using this it is already easy to show that the problem is solvable in FPT-time $O^*(1.619^P)$. To reach the claimed bound of $O^*(1.325^P)$ we need some more ingredients. Let $M(u)$ be the maximum weight of outgoing arcs from u, and $P(u)$ be the sum of weights or arcs (z, u) (incoming in u). We first define two reduction rules and one branching rule.

(RR2) If there exists u such that $P(u) \leq M(u)$, do $Adjust(v, w_{v,u})$ where v is such that $w_{u,v} = M(u)$.

(RR3) If there is (u, v) with $w_{u,v} = w_{v,u} = 2$, and all other arcs outgoing from u and v have weight 1, then set $w_{u,v} = w_{v,u} = 1$, and do $P \leftarrow P - 1$.

(BR1) If there exists u with $P(u) \geq 5$, branch as follows: either $Set(u, 0)$, or $Adjust(u, 1)$.

Property 3. (RR2) and (RR3) are correct. (BR1) has a branching factor (at worst) $(-1, -5)$.

Now, before giving the whole algorithm, we detail the case where the maximum weight is 2, where a careful case analysis is needed.

Lemma 1. *Let us consider an instance where (1) (RR2) and (RR3) have been extensively applied, and (2) the maximum edge-weight is $w_{u,v} = 2$. Then there is a branching algorithm with branching factor (at worst) $(-1, -5)$ or $(-2, -3)$.*

We are now ready to describe the main algorithm. $N(u)$ denotes the neighbors of u, and $N^2(u)$ the set of vertices at distance 2 from u.

	Algorithm 2
STEP 1:	While (RR2) or (RR3) is applicable, apply them;
STEP 2:	If $P < 0$ return NO;
	If the graph has no edge return YES;
STEP 3:	If there is a vertex u with $P(u) \geq 5$: apply (BR1)
STEP 4	If there exists (u, v) with either $w_{u,v} + w_{v,u} \geq 6$, or
	$w_{u,v} = 2$ and $w_{v,u} = 3$: branch by either $Adjust(u, w_{u,v})$,
	or $Adjust(v, w_{v,u})$.
STEP 5:	If there exists (u, v) of weight $w_{u,v} = 3$:
	- if $N^2(u) = \{t\}$ and all arcs $(t, z), z \in N(u)$ have weight 1:
	- either $Adjust(t, 1)$,
	- or $Set(t, 0)$.
	- otherwise:
	- either $Adjust(v, 1)$
	- or $Set(u, 3)$, and $Set(z, 0)$ for all $z \in N(u)$
STEP 6:	If the maximum weight is at most 2, then branch as in Lemma 1

Theorem 6. *Algorithm 2 solves* DPVC *in time* $O^*(1.325^P)$.

Proof. Note that 1.325 corresponds to branching factors $(-1,-5)$ and $(-2,-3)$.

We have already seen that (RR2) and (RR3) are sound, and that (BR1) gives a branching factor $(-1,-5)$.

For Step 4, if $w_{u,v} + w_{v,u} \geq 6$ this gives in the worst case a branching factor $(-1,-5)$. If $w_{u,v} = 2$ and $w_{v,u} = 3$ the branching factor is $(-2,-3)$.

At this point (after Step 4): there cannot remain an (u,v) with weight $w_{u,v} \geq 5$, since Step 4 would have been applied. If there is (u,v) with $w_{u,v} = 4$: either $P(u) \geq 5$ (impossible since (BR1) would have been applied), or $P(u) \leq 4$ (impossible since (RR2) would have been applied). So, the maximum edge weight after step 4 is at most 3.

Thanks to Lemma 1, what remains to do is to focus on Step 5, with $w_{u,v} = 3 = M(u)$.

First, note that then $P(u) = 4$ (otherwise (RR2) or (BR1) would have been applied), and $w_{v,u} = 1$ (otherwise Step 4 would have been applied).

We consider two cases.

- If $N^2(u) = \{t\}$ and all arcs $(t,z), z \in N(u)$ have weight $w_{t,z} = 1$. As explained in the algorithm, we branch on t: either $p_t \geq 1$, and all arcs $(t,z), z \in N(u)$ are covered, so we need to cover edges incident to u (which are now disconnected from the rest of the graph), and we need at least 2 for this. Or $p_t = 0$, and we need to cover optimally edges incident to a vertex in $N(u)$, and we need at least 2 for this. P reduces by at least 3 in one branch, by at least 2 in the other, leading to a branching factor $(-3,-2)$.
- If $|N^2(u)| \geq 2$, or if $N^2(u) = \{t\}$ with at least one arc $(t,z), z \in N(u)$ of weight 2: setting $p_u = 3$ is only interesting if all neighbors of u receive weight 0 - otherwise distributing the power 3 of u on neighbors of u to cover all arcs (v,u) is always at least as good. Then in this case we have to cover arcs between $N(u)$ and $N^2(u)$, so a power at least 2. Then either we have $p_u < 3$ and in this case $p_v \geq 1$, or we set $p_u = 3$, $p_z = 0$ for neighbors z of u, et we fix at least 2 in $N^2(u)$. In the first branch P reduces by at least 1, in the other by at least 5, leading to a branching factor $(-1,-5)$. □

5 Parameterizing by the number k of vertices

We now consider a more general parameter, the number k of vertices that will receive a positive value in the optimal solution. Note that by definition $k \leq P$, therefore, we expect any FPT algorithm with respect to k to have worse performance than the best algorithm for parameter P.

Theorem 7. PVC *is solvable in time* $O^*(1.619^k)$. DPVC *is solvable in time* $O^*(k^k)$.

Following Theorem 7, a natural question is whether DPVC is solvable in single exponential time with respect to k or not. This does not seem obvious. In particular, it is not clear whether DPVC is solvable in single expontential time *with respect to the number of vertices n*, since the simple brute-force algorithm which guesses the value of each vertex needs $n^{O(n)}$ time.

Interestingly, though we are not able to resolve these questions, we can show that they are actually equivalent.

Theorem 8. *If there exists an $O^*(\gamma^n)$ algorithm for* DPVC*, then there exists an $O^*((4\gamma)^k)$ algorithm for* DPVC*.*

6 Kernelization and linear programming

Moving to the subject of kernels, we first notice that the same technique as for VC gives a quadratic kernel for DPVC when the parameter is k (and therefore also when the parameter is P):

Theorem 9. *There exists a kernelization algorithm for* DPVC *that produces a kernel with $O(k^2)$ vertices.*

We observe that the above theorem gives a *bi-kernel* also for PVC. We leave it as an open question whether a pure quadratic kernel exists for PVC.

Let us now consider the question whether the kernel of Theorem 9 could be improved to linear. A way to reach a linear kernel for VC is by means of linear programming. We consider this approach now and show that it seems to fail for the generalization we consider here. Let us consider the following ILP formulation for DPVC, where we have one variable per vertex (x_i is the power of u_i), and one variable $x_{i,j}$ ($i < j$) per edge (u_i, u_j). $x_{i,j} = 1$ (resp. 0) means that u_i (resp u_j) covers the edge.

$$\begin{cases} Min \ \sum_{i=1}^{n} x_i \\ x_i \geq w_{i,j} x_{i,j}, \forall (u_i, u_j) \in E, i < j \\ x_j \geq w_{j,i}(1 - x_{i,j}), \forall (u_i, u_j) \in E, i < j \\ x_{i,j} \in \{0, 1\}, \forall (u_i, u_j) \in E, i < j \\ x_i \geq 0, i = 1, \cdots, n \end{cases}$$

Can we use the relaxation of this ILP to get a linear kernel? Let us focus on PVC, where the relaxation can be written in an equivalent simpler form[1]:

$$\begin{cases} Min \ \sum_{i=1}^{n} x_i \\ x_i + x_j \geq w_{i,j}, \forall (u_i, u_j) \in E, i < j \\ x_i \geq 0, i = 1, \cdots, n \end{cases}$$

Let us call $RPVC$ this LP. We can show that, similarly as for VC, $RPVC$ has the semi-integrality property: in an optimal (extremal) solution x^*, $2x_i^* \in \mathbb{N}$ for all i. However, we *cannot* remove vertices receiving value 0, as in the

[1] A solution of the relaxation of the former is clearly a solution of the latter. Conversely, if $x_i + x_j \geq w_{i,j}$, set $x_{i,j} = x_i / w_{i,j}$ to get a solution of the former.

case of VC. Indeed, there does exist vertices that receive weight 0 in the above relaxation which are in *any* optimal (integer) solution. To see this, consider two edges (u_1, v_1) and (u_2, v_2) both with weight 2, and a vertex v adjacent to all 4 previous vertices with edges of weight 1. Then, there is only one optimal fractional solution, with $p_{u_1} = p_{v_1} = p_{u_2} = p_{v_2} = 1$, and $p_v = 0$. But any (integer) solution has value 5 and gives power 2 to u_1 or v_1, to u_2 or v_2, and weight 1 to v. The difficulty is actually deeper, since we have the following.

Theorem 10. *The following problem is NP-hard: given an instance of* PVC*, an optimal (extremal) solution x^* of RPVC and i such that $x^*_i = 0$, does there exists an optimal (integer) solution of* PVC *not containing v_i?*

References

1. E. Angel, E. Bampis, V. Chau, and A. Kononov. Min-power covering problems. In *Algorithms and Computation: 26th International Symposium, ISAAC 2015*, volume 9472 of *LNCS*, pages 367–377, 2015.
2. H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.
3. J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(4042):3736 – 3756, 2010.
4. M. Chlebík and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.
5. M. Cygan. Deterministic parameterized connected vertex cover. In *SWAT*, volume 7357 of *LNCS*, pages 95–106. Springer, 2012.
6. M. Dom, D. Lokshtanov, S. Saurabh, and Y. Villanger. Capacitated domination and covering: A parameterized perspective. In *IWPEC*, volume 5018 of *LNCS*, pages 78–90. Springer, 2008.
7. M. R. Fellows. Blow-ups, win/win's, and crown rules: Some new directions in FPT. In *WG*, volume 2880 of *LNCS*, pages 1–12. Springer, 2003.
8. J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41(3):501–520, 2007.
9. M. Lampis. A kernel of order 2 k-c log k for vertex cover. *Inf. Process. Lett.*, 111(23-24):1089–1091, 2011.
10. M. Lampis. Parameterized approximation schemes using graph widths. In *ICALP (1)*, volume 8572 of *LNCS*, pages 775–786. Springer, 2014.
11. D. Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.
12. D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory Comput. Syst.*, 43(2):234–253, 2008.
13. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.
14. R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47(2):63 – 77, 2003.
15. D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, New York, NY, USA, 1st edition, 2011.

A Omitted Proofs

A.1 Proof of Theorem 1

Proof. The reduction is from VC. Start with an instance $G = (V, E)$ of vertex cover. Put weight 2 on edges of E, add missing edges with weight 1. Add a new vertex v_0 adjacent to all other vertices with edges of weight 1. Then a vertex cover of size k in the initial graph corresponds to a vertex cover of power $|V|+k$ in the final graph (put power 2 on vertices in the vertex cover, and power 1 to other vertices in V). The reverse is also true: v_0 being useless, a solution of PVC takes all vertices of V with power either 1 or 2; it has power $|V| + k$, where k is the number of vertices with power 2. These vertices must be a vertex cover of G. So the NP-hardness follows.

For the approximation hardness, put weights K instead of 2 of edges of G. Then a solution of size k in the initial graph corresponds to a solution of power $Kk + n$ in the final graph. Take $K = n^2$ for instance to transfer constant ratios. □

A.2 Proof of Theorem 3

Proof. Since these algorithms rely on well-known DP techniques, we only sketch the details. Given a nice rooted tree decomposition [2] of G, we will compute a DP table for every bag, starting from the leaves. For every bag B, and for every assignment of values to the vertices of B our DP table will store the optimal cost of a power vertex cover to the graph induced by vertices which appear in B and bags below it in the tree decomposition. It is not hard to see how to compute such a table for Join and Forget nodes, while for Insert nodes the only complication is that, given a value for the new vertex we need to make sure that it is sufficient to cover edges included in the bag (otherwise we set the cost to ∞).

The claimed algorithms now follow from two easy observations. In the first case, it is easy to see that any optimal solution only assigns values from $\{0, 1, \dots, W\}$ to all vertices, therefore the DP table has size $(W + 1)^t$ for each bag. In the second case, we observe that in an optimal solution, the value of every vertex u, if it is not 0, must coincide with the weight of some edge w_{uv}. Therefore, each vertex has $\Delta + 1$ choices for a possible value, and the size of the DP table becomes $(\Delta + 1)^t$. □

A.3 Proof of Theorem 4

Proof. Our strategy will be to reduce the given instance to an instance where the maximum weight is $W = \frac{\log n}{\epsilon}$, without distorting the optimal solution too much. Then, we will invoke a slightly modified version of the first algorithm of Theorem 3 to obtain the stated running time.

Before we begin, observe that for PVC, W is already a lower bound on the value of the optimal solution. However, this is not the case for DPVC. We can, however, assume that we know the largest value used in the optimal solution (there are only $|E|$ possible choices for this value, so we can guess it). Let W be this value. Now, we know that any edge with $w_{uv} > W$ must be covered by v, so we give vertex v value w_{vu} and adjust the graph appropriately. In the process we produce a graph where the largest weight is W and the optimal is at least as high as W.

Now, note that in general we cannot say anything about the relation between W and n, indeed it could be the case that the two are not even polynomially related. To avoid this, we will first "round down" all weights, so that we have $W = n^{O(1)}$. To begin with, suppose that $W > n^2$ (otherwise, all weights are polynomial in n and we are done). For every edge $(u, v) \in E$ we calculate a new weight $w'_{uv} = \lfloor \frac{w_{uv} n^2}{W} \rfloor$. Observe that in the new instance we constructed the maximum weight is $W' = n^2$, so all we need to argue is that the optimal solution did not change much. Let OPT denote the value of the optimal solution of the original instance and OPT$'$ denote the optimal of the new instance. It is not hard to see that $\mathrm{OPT}' \leq \lfloor \frac{\mathrm{OPT} n^2}{W} \rfloor \leq \frac{\mathrm{OPT} n^2}{W}$ because if we take an optimal solution of G and divide the value of each vertex by the same value we divided the edge weights, we will obtain a feasible solution. Suppose now that we have a solution SOL$'$ of the new instance such that $\mathrm{SOL}' \leq \rho \mathrm{OPT}'$, for some $\rho \geq 1$. We can use it to obtain an almost equally good solution of the original instance as follows: for every vertex of G assign it the value assigned to it by SOL$'$, multiplied by $\frac{W}{n^2}$, and then add to it $\frac{W}{n^2}$. The total cost of this solution is at most $\mathrm{SOL}' \frac{W}{n^2} + \frac{W}{n} \leq \rho \mathrm{OPT}' \frac{W}{n^2} + \frac{W}{n} \leq \rho \mathrm{OPT} + \frac{\mathrm{OPT}}{n} = (\rho + \frac{1}{n})\mathrm{OPT}$, where we have also used the assumption that $W \geq \mathrm{OPT}$. Because of the above argument, we will assume in the remainder that $W \leq n^2$.

Now, given an instance with $W \leq n^2$, we will produce a new instance with $W = O(\log n/\epsilon)$. For every edge we set $w'_{uv} = \lfloor \log_{1+\epsilon}(w_{uv}) \rfloor = \lfloor \frac{\log(w_{uv})}{\log(1+\epsilon)} \rfloor$. Because for sufficiently small ϵ we have $\ln(1 + \epsilon) \approx \epsilon$, we can conclude that in the new instance the maximum weight is $W' = O(\frac{\log n}{\epsilon})$.

We will now use the algorithm of Theorem 3 to solve a slightly modified variant of DPVC on the new instance. In this instance we again seek an assignment of values to the vertices so that for each edge $(u, v) \in E$, either $p_u \geq w_{uv}$ or $p_v \geq w_{vu}$. The difference is that now the objective function we seek to minimize is the function $\sum_{v \in V} (1 + \epsilon)^{p_v}$. It is not hard to see that the DP algorithm of Theorem 3 works with minimal modifications for this problem (we only need to change the way the cost of a partial solution is calculated).

Let OPT$'$ be the value of the optimal solution in the new instance (according to the modified objective function). Then, we have $\mathrm{OPT}' \leq \mathrm{OPT}$. To see this, take an optimal solution of the original instance, for each vertex v to which this solution assigns value p_v, assign the value $\lfloor \log_{1+\epsilon} p_v \rfloor$. This is a feasible solution of the new instance, and its objective value is $\sum_{v \in V} (1 + \epsilon)^{\lfloor \log_{1+\epsilon} p_v \rfloor} \leq \mathrm{OPT}$. Suppose now that, using the modified algorithm of Theorem 3 we have found an optimal solution OPT$'$ of the new instance. For each vertex v to which this solution gave value p'_v we give value $\lfloor (1 + \epsilon)^{p'_v + 1} \rfloor$. The crucial observation is that this is necessarily a feasible solution of the original instance, because for an edge (v,u) that is covered by v in the new instance we have $p'_v \geq \lfloor \log_{1+\epsilon} w_{vu} \rfloor$ and therefore $p'_v + 1 \geq \log_{1+\epsilon} w_{vu}$. This implies $(1+\epsilon)^{p'_v+1} \geq w_{vu}$, and since w_{vu} is an integer, $\lfloor (1+\epsilon)^{p'_v+1} \rfloor \geq w_{vu}$. The total cost of this solution is at most $\sum_{v\in V} \lfloor (1+\epsilon)^{p'_v+1} \rfloor \leq \sum_{v\in V} (1+\epsilon)^{p'_v+1} = (1+\epsilon)\mathrm{OPT}' \leq (1+\epsilon)\mathrm{OPT}$.

Finally, we note that the running time of this theorem implies an FPT running time for the approximation scheme via standard arguments: if $t < \log n / \log\log n$ then the running time of the algorithm is actually $(1/\epsilon)^t n^{O(1)}$. If on the other hand $t > \log n / \log\log n$ then we have a problem kernel for parameter t. □

A.4 Proof of Property 2

Proof. Indeed, take a solution with the initial weights. Then $p_u = M$ or $p_v = M$, and we get a solution on the modified instance by reducing the power of u (or v) by $M - B$. Conversely, if we have a solution with the modified weights, either $p_u = B$ (or $p_v = B$) and we get a solution with the initial weights by adding power $M - B$ to u (or v). □

A.5 Proof of Theorem 5

Proof. When (RR1) is no longer applicable, any edge of maximum weight is adjacent to another edge of the same (maximum) weight. So in Step 4 we can always find such a pair of edges e, f. Then the validity of the algorithm follows from Properties 1 and 2. For the running time, the only step to look at is Step 4. When branching, the maximum weight is $w_e = w_f \geq 2$. In the first branch P is reduced by at least 2, in the second branch by at least 4, and this gives a branching factor $(-2,-4)$ inducing a complexity smaller than the claimed bound. □

A.6 Proof of Property 3

Proof. For (RR2): it is never interesting to put $p_u = M(u)$, since we can cover all edges incident to u by putting power at most $P(u)$ on the neighbors of u. So, we can assume that the power of u is smaller than $M(u)$, meaning that v covers (v,u).

For (RR3): Take a solution with the initial weights. Then $p_u = 2$ or $p_v = 2$, and we get a solution on the modified instance by reducing the power of u (or v) by 1. Conversely, if we have a solution with the modified weights, either $p_u = 1$ or $p_v = 1$ and we get a solution with the initial weights by adding power 1 to u or v.

For (BR1):When setting $p_u = 0$, we need to cover all edges incident to u using the other extremity. Since $P(u) \geq 5$, P reduces by at least 5 in this branch (and at least 1 in the other branch). □

A.7 Proof of Lemma 1

Proof. Let us consider an instance where (1) (RR2) and (RR3) have been extensively applied, and (2) the maximum edge-weight is $w_{u,v} = 2$.

We show by a case analysis that we can reach branching factor at worst $(-1, -5)$, or $(-2, -3)$.

If $P(u) \geq 5$, then we apply $(BR1)$. Otherwise, $P(u) \leq 4$. Since $P(u) > M(u)$ (otherwise (RR2) applies), $P(u) \in \{3, 4\}$. In particular, the degree of u is $d(u) \in \{2, 3, 4\}$.

Note first that if all (u, z) have weight $w_{u,z} = 2$ then we do either $Set(u, 0)$ (and we fix $P(u) \geq 3$ on neighbors of u), or $Set(v, 2)$, meaning that P reduces by 2. This gives a branching factor at worst $(-3, -2)$.

We need to consider several cases, taking a vertex u (incident to an edge of weight 2) of maximum degree:

1. If $d(u) = 4$. Then since $P(u) \leq 4$ all (z, u) have weight $w_{z,u} = 1$.
   (a) If there is only one z with $w_{u,z} = 1$: it is never interesting to put power 1 on u. So we do either $Set(u, 0)$ (and we fix 4), or $Set(u, 2)$ and we fix 2.
   (b) If there are three neighbors z with $w_{u,z} = 1$: it is never interesting to set $p_u = 2$ (instead set $p_u = 1$ and give the extra power 1 to v to cover (v, u)), so (u, v) is covered by v and without branching we do $Adjust(v, 1)$.
   (c) If $w_{u,v} = w_{u,z} = 2$, $w_{u,s} = w_{u,t} = 1$. If both v and z have degree 1, then it is never interesting to take them, so we do $Set(u, 2)$ without branching. Otherwise, suppose that z has degree at least 2. Then we branch on z: we do either $Set(z, 0)$ (and we fix at least 3: 2 for $Adjust(u, 2)$ and 1 for the other neighbor), or $Set(z, 1)$. But in this latter case it is never interesting to set $p_u = 2$ (set $p_u = 1$ and put extra power 1 on v instead), so we also do $Adjust(v, 1)$. The branching factor is then at worst $(-3, -2)$.
2. If $d(u) = 3$, with $N(u) = \{v, z, t\}$
   (a) Suppose first that $w_{u,z} = 2$ and $w_{u,t} = 1$.
   If $w_{t,u} = 1$ it is never interesting to set $p_u = 1$ (set $p_u = 0$ and put extra weight on t instead), so we do either $Set(u, 0)$ (and fix at least $P(u) \geq 3$ in $N(u)$), or $Set(u, 0)$. We get a branching factor $(-3, -2)$.
   Otherwise, $w_{t,u} = 2$. Since $P(u) \leq 4$, $w_{v,u} = w_{z,u} = 1$. Then if $d(v) = d(z) = 1$, we do not need to branch, we do $Set(u, 2)$. Otherwise, for instance $d(z) \geq 2$. We branch on z: we do either $Set(z, 0)$ and we fix at least 3 in $N(z)$, or $Adjust(z, 1)$ but in this case it is never interesting to set $p_u = 2$ (set $p_u = 1$, and put the extra weight 1 on v instead), so we do $Adjust(v, 1)$. We get a branching factor $(-3, -2)$.
   (b) Now suppose that $w_{u,z} = w_{u,t} = 1$.
   If $w_{v,u} = 1$ it is never interesting to set $p_u = 2$, so we do $Adjust(v, 1)$ without branching. The only remaining case is $w_{v,u} = 2$.
   If v has another outgoing arc (v, s) with weight $w_{v,s} = 2$: we branch on v, by doing either $Set(v, 2)$, or by considering $w(v) < 2$ hence $Adjust(u, 2)$ and $Adjust(s, w_{s,v})$. We get a branching factor $(-2, -3)$.

Otherwise, we have (u, v) with $w_{u,v} = w_{v,u} = 2$, and all other arcs outgoing from u, v have weight 1. But this cannot occur thanks to Rule (RR3).

3. If $d(u) = 2$, $N(u) = \{v, z\}$. Then $w_{u,z} = 1$.
   (a) If $w_{z,u} = 1$ it is never interesting to set $p_u = 1$. As previously, we do either $Set(u, 0)$ (and fix $P(u) \geq 3$ in $N(u)$), or $Set(u, 2)$ and reduce P by two. So again we get a branching factor $(-3, -2)$.
   (b) If $w_{v,u} = 1$, it is never interesting to set $p_u = 2$, so we do $Adjust(v, 1)$.
   (c) Otherwise, $w_{v,u} = w_{z,u} = 2$. Let us look at z: it has degree 2 with $w_{z,u} = 2$ and $w_{u,z} = 1$ so this is a previous case with $u = z$. □

A.8 Proof of Theorem 7

Proof. For PVC, Algorithm 1 can be easily adapted to deal with parameterization by k: when branching, k is reduced by 1 in the first branch, and by 2 in the second branch, so we have a branching factor $(-1, -2)$ in the worst case. The case of DPVC is a bit more involved, but we can show that it is FPT as well. More precisely:

Let us concentrate on DPVC. Suppose first that there is a vertex u of (out)degree at least $k + 1$. To get a solution with at most k vertices, necessarily u must receive a positive power. More precisely, it must cover all but at most k edges incident to it. In other words, order the weights of arcs (u, v) by decreasing order $w_1 \geq w_2 \geq \cdots \geq w_d$. Then in any solution with at most k vertices, $p_u \geq w_{k+1}$. So in this case, we can safely apply $Adjust(u, w_{k+1})$. In the remaining instance, u has (out)degree at most k. After this reduction rule, we get an instance where each vertex has degree at most $k + 1$. Then:

- Take a pair (u, v) such that (u, v) is a maximal weight arc incident to u. Let $\{w_1, w_2, \ldots, w_s\}$ the set of weights at least $w_{(v,u)}$ of arcs incident to v.
- Either set $p_u = w_{u,v}$ (remove v and recurse), or set $p_v = w_i$ (remove v and incident edges of weight at most w_i) for $i = 1, \ldots, s$ and recurse.

If u is chosen to cover the pair (u, v) then this is the first branch. Otherwise, we have to use v to cover this pair. We consider all the possible weights that will be assigned to v in any (optimal) solution. Since $s \leq k$, we have a tree of arity at most $k + 1$ and depth at most k.

Actually, we have to pay more attention to *Adjust*. Indeed, when applying it to a vertex, it means that this vertex will be taken in the solution. This is fine if we have branched on it, but not if it becames at some point of degree 0. So we shall mark vertices that have been adjusted, and do $k \leftarrow k - 1$ when removing a marked vertex of degree 0. □

A.9 Proof of Theorem 8

Proof. We describe a branching algorithm, which eventually needs to solve an instance of DPVC on k vertices. We maintain three sets $C_1, C_2, I \subseteq V$. Initially,

$I = V$ and $C_1, C_2 = \emptyset$. The informal meaning is that we want to branch so that I eventually becomes an independent set and $C_1 \cup C_2$ a vertex cover of G. In addition, we want to maintain the property that there are no edges between C_2 and I (which is initially vacuously true).

As long as $|C_1 \cup C_2| \leq k$ and there are still edges incident on I we repeat the following branching Rules:

1. If there is a vertex $u \in C_1$ such that there are no edges from u to I we set $C_2 := C_2 \cup \{u\}$ and $C_1 := C_1 \setminus \{u\}$.
2. If there is an edge (u, v) induced by I we branch on which of the two vertices covers it in the optimal solution. In one branch, we perform $Adjust(u, w_{u,v})$, and set $I := I \setminus \{u\}$ and $C_1 := C_1 \cup \{u\}$. The other branch is symmetric for v.
3. If I is an independent set, we select a vertex $u \in C_1$. Because of Step 1, we know that there are some edges connecting u to I. Let $w_{u,v}$ be the maximum weight among edges connecting u to I. We branch between two choices: either in the optimal $p_u \geq w_{u,v}$ or $p_v \geq w_{v,u}$. In the first case we $Adjust(u, w_{u,v})$ (observe that this removes all edges connecting u to I, therefore we can immediately apply Rule 1). In the second case, we $Adjust(v, w_{v,u})$ and set $|I| := I \setminus \{v\}$ and $C_1 := C_1 \cup \{v\}$.

It is not hard to see that the branching described above cannot produce more than 4^k different outcomes. To see this, observe that any branching performed either increases $|C_1 \cup C_2|$ or $|C_2|$, while never decreasing either quantity. Therefore, the quantity $|C_1 \cup C_2| + |C_2|$ increases in each branching step. However, this quantity has maximum value $2k$.

The branching algorithm above will stop either when I has no incident edges, or when $|C_1 \cup C_2| = k$. In the latter case, we have selected that $C_1 \cup C_2$ is the set of vertices that take positive values in our solution, so we can perform $Adjust(u, w_{u,v})$ for any $u \in C_1$ and $v \in I$, until no edges are incident on I. We can now delete all the (isolated) vertices of I, and we are left with a k-vertex DPVC instance on $C_1 \cup C_2$, on which we use the assumed exponential-time algorithm. □

A.10 Proof of Theorem 9

Proof. Consider the following rules:

- If there exists a vertex u of (out)degree at least $k + 1$, apply the procedure $Adjust(u, w_{k+1})$ as previously (and mark the vertex).
- If there exists a marked vertex of degree 0, remove it and do $k \leftarrow k - 1$. If there is an unmarked vertex of degree 0, remove it.

Let G' be the graph obtained after this procedure. If $n' > k(k + 1)$ return NO. Else, $n' \leq (k + 1)^2$ which is a kernel.

Suppose that the answer is YES. Then there exists a vertex cover of size at most k, and each vertex has at most k neighbors, and there is no isolated vertex, so $n' \leq k(k+1)$. Hence, this gives a kernel of size $O(k^2)$ for DPVC when parameterized by k (so a kernel of size $O(P^2)$ when parameterized by P).

A.11 Proof of Theorem 10

Proof. Take a graph $G = (V, E)$, instance of VC. We build the following instance $G' = (V', E')$ of PVC:

- There is one vertex for each vertex in V, and two vertices v'_e and v''_e for each edge e in E
- v'_e and v''_e are linked with an edge of weight 2;
- If $e = (u, v)$ is an edge of G, then u is adjacent to v'_e and v is adjacent to v''_e in G', both edges receiving weight 1.

As previously, an optimal solution in the relaxation RPVC is to give power 1 to each "edge-vertex" (v'_e, v''_e), and 0 to vertices corresponding to vertices in V.

Then, consider an optimal solution C' of PVC: for each $e \in G$, $p_{v'_e} = 2$ or $p_{v''_e} = 2$ - to cover the edge. Moreover, if say $p_{v'_e} = 2$, then we can fix $p_{v''_e} = 0$: indeed, there is only one more edge incident to v''_e to cover, so we shall put power 1 to the other neighbor (v) of v''_e. So $C' = W \cup V'$, where W is made of $|E|$ "edge-vertices", and V' is made of vertices of V. To be feasible, V' must be a vertex cover of G. Conversely, if V' is a vertex cover of G, then we can easily add a set W of $|E|$ "edge-vertices" to get a feasible solution of PVC.

Then, a vertex in V is in an optimal VC of G iff it is in an optimal solution of PVC. Since it receives weight 0 in the (unique) optimal solution of the relaxation, the result follows. $\square$