Vertex and Edge Covers with Clustering Properties: Complexity and Algorithms*

Henning Fernau[1,†] and David F. Manlove[2,‡]

[1] *FB IV—Abteilung Informatik, Universität Trier, 54286 Trier, Germany*
[2] *Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK*

Abstract

We consider the concepts of a *t-total vertex cover* and a *t-total edge cover* ($t \geq 1$), which generalise the notions of a vertex cover and an edge cover, respectively. A t-total vertex (respectively edge) cover of a connected graph G is a vertex (edge) cover S of G such that each connected component of the subgraph of G induced by S has at least t vertices (edges). These definitions are motivated by combining the concepts of clustering and covering in graphs. Moreover they yield a spectrum of parameters that essentially range from a vertex cover to a connected vertex cover (in the vertex case) and from an edge cover to a spanning tree (in the edge case). For various values of t, we present $\mathcal{NP}$-completeness and approximability results (both upper and lower bounds) and $\mathcal{FPT}$ algorithms for problems concerned with finding the minimum size of a t-total vertex cover, t-total edge cover and connected vertex cover, in particular improving on a previous $\mathcal{FPT}$ algorithm for the latter problem.

1 Introduction

In graph theory, the notion of covering vertices or edges of graphs by other vertices or edges has been extensively studied (see [35] for a survey). For instance, covering vertices by other vertices leads to parameters concerned with vertex domination [30, 31]. When edges are to be covered by vertices we obtain parameters connected with the classical vertex covering problem [29, p.94]. Covering vertices by edges, i.e., finding edge covers, was first considered by Norman and Rabin [44]. Finally, when edges are to cover other edges, we obtain parameters associated with edge domination (introduced by Mitchell and Hedetniemi [36]). These problems have long been a testbed for the design of parameterised algorithms (or for showing the limitations of that approach) [17]. In particular, whilst variants of vertex cover problems have been considered with respect to parameterised complexity for a number of years (see, e.g., [9, 20]), only more recently has a systematic study of such problems been initiated from the point of view of parameterised complexity [28, 43].

Clustering in graphs is another fundamental concept with a large range of practical applications [24]. Connectedness can be seen as one of the weakest notions of clustering: it is reasonable to assert that a vertex set can be termed a cluster only if it is connected.

*A preliminary version of this paper appeared in the Proceedings of ACiD 2006 [19].

†Part of this work was carried out whilst visiting the University of Glasgow, supported by EPSRC grant EP/D030110/1, and part of the work was carried out whilst affiliated to the University of Hertfordshire, Hatfield, UK. Email `fernau@uni-trier.de`.

‡Supported by EPSRC grant GR/R84597/01 and RSE / Scottish Executive Personal Research Fellowship. Email `davidm@dcs.gla.ac.uk`.

When being used for classification purposes, there is rarely only one cluster, but rather a number of them, each representing some concept, i.e., one is looking for connected components. In order to exclude trivial cases and to define meaningful concepts, it may often be appropriate to bound on the number of elements per cluster.

In this paper we consider a synergy of the notion of clustering with each of the concepts of vertex and edge covering. Throughout we assume that $G = (V, E)$ is a connected graph, where $n = |V|$ and $m = |E| \geq 1$. For $1 \leq t \leq n$, a *t-total vertex cover* (henceforth a *t-tvc*) in G is a vertex cover S in G such that each connected component of $G[S]$, the subgraph of G induced by S, has at least t vertices. Similarly, for $1 \leq t \leq m$, a *t-total edge cover* (henceforth a *t-tec*) in G is an edge cover S of G (i.e., each vertex of G is incident to an edge in S) such that each connected component of $G[S]$, the subgraph of G induced by (the vertices covered by) S, has at least t edges. Hence, if S is a t-tvc or t-tec, then S is a vertex cover or edge cover respectively such that each member of S belongs to a "cluster" containing at least t elements of S.

The concept of a *total dominating set* in a graph, first defined and studied by Cockayne et al. [11], illustrates one case where the notions of clustering and covering (vertices by vertices) have already been brought together. A set of vertices S is a *total dominating set* of G if (i) S is a dominating set (i.e., every vertex in $V \backslash S$ is adjacent to a vertex in S), and (ii) each connected component of $G[S]$ has at least two vertices.

The notion of a 2-tvc was first defined by Jean Blair [4] using the terminology *total vertex cover* (by analogy to the term *total dominating set*). It is straightforward to present relationships between the minimum size of a t-tvc (respectively t-tec) for various values of t and established parameters concerned with vertex covering (respectively edge covering) in G. Throughout this paper, our notation follows and extends that of Harary [29]. Let $\alpha_0(G)$ denote the minimum size of a vertex cover in G. A *connected vertex cover* (henceforth a *cvc*) in G is a vertex cover S in G such that $G[S]$ is connected. Let $\alpha_0^c(G)$ denote the minimum size of a cvc in G. It follows that a 1-tvc is simply a vertex cover (recall that $m \geq 1$), whilst a cvc of size t is a t-tvc. For $t \geq 1$, let $\alpha_{0,t}(G)$ denote the minimum size of a t-tvc in G. Then $\alpha_{0,1}(G) = \alpha_0(G)$. The parameters $\alpha_{0,t}(G)$ for $t \geq 2$ do not appear to have been studied in the literature previously. In Section 2, we present some additional relationships involving the parameters $\alpha_0(G)$, $\alpha_{0,t}(G)$ and $\alpha_0^c(G)$.

Now let $1 \leq t \leq m$ – we turn to the concept of a t-tec. It follows that a 1-tec is simply an edge cover (again recall that $m \geq 1$), whilst a minimum $(n-1)$-tec is a minimum connected edge cover, i.e., a spanning tree. Let $\alpha_{1,t}(G)$ denote the minimum size of a t-tec of G, and let $\alpha_1(G)$ denote the minimum size of an edge cover of G. Then $\alpha_{1,1}(G) = \alpha_1(G)$. The parameters $\alpha_{1,t}(G)$ for $t \geq 2$ do not appear to have been studied in the literature previously. In Section 2, we present some additional relationships between the parameters $\alpha_1(G)$ and $\alpha_{1,t}(G)$.

We remark that, for a t-tvc (respectively t-tec) to exist, G need not be connected; it is sufficient that each connected component of G has at least t vertices (edges), and the results in this paper also hold in such a setting. However for ease of exposition, and due to the correspondence between t-tvcs and t-tecs with connected vertex covers and spanning trees respectively, we choose to assert throughout that G is connected.

Given $t \geq 1$, let VC, t-TVC, t-TEC and CVC denote the problems of computing $\alpha_0(G)$, $\alpha_{0,t}(G)$, $\alpha_{1,t}(G)$ and $\alpha_0^c(G)$ respectively, given a connected graph G where $n = |V|$ and $m = |E| \geq 1$ (additionally $n \geq t$ in the case of t-TVC and $m \geq t$ in the case of t-TEC). Let VC-D, t-TVC-D, t-TEC-D and CVC-D denote the decision versions of VC, t-TVC, t-TEC and CVC, respectively. Hence, the question is, given a graph G and a parameter k, whether there is a cover C (with the additional properties specified by the problem) such that $|C| \leq k$. Note that, in the case of t-TVC-D and t-TEC-D, we lose no generality in assuming

that $t \leq k$, for otherwise the decision problem instance trivially has a NO answer.

For each $t \geq 2$, we show in Section 3 that t-TVC is $\mathcal{NP}$-hard and not approximable within an asymptotic performance ratio of $10\sqrt{5}-21-\delta$ (> 1.3606), for any $\delta > 0$, unless $\mathcal{P}$=$\mathcal{NP}$. However on the other hand we prove that t-TVC is approximable within 2. We also prove that t-TVC-D is $\mathcal{NP}$-complete, even for planar bipartite graphs of maximum degree 3. Moreover we show that there exists a constant $\delta_t > 1$ such that t-TVC in bipartite graphs of maximum degree 3 is not approximable within δ_t unless $\mathcal{P} = \mathcal{NP}$. However, for each $t \geq 2$, we show that t-TVC-D belongs to $\mathcal{FPT}$. In particular, we give a parameterised algorithm for 2-TVC-D with complexity $\mathcal{O}^*(2.3655^k)$. Here the parameter is the size of the 2-tvc.

CVC is $\mathcal{NP}$-hard, even for planar graphs of maximum degree 4 [26], though polynomial-time solvable for graphs of maximum degree 3 [47]. The problem is also solvable in linear time for chordal graphs [15]. For a tree T, finding a minimum cvc is trivial (if $T = K_2$, one vertex will suffice, otherwise the set of non-leaf nodes is a minimum cvc in T). It is known that CVC is approximable within 2 [46, 2] and within $\frac{5}{3}$ for any class of graphs in which VC is solvable in polynomial time [15]. Approximation algorithms for CVC giving rise to performance guarantees based on density parameters of a graph have also been proposed [6]. Furthermore, CVC admits a polynomial-time approximation scheme for planar graphs [15] and unit disk graphs [48]. In Section 4, we show that CVC is not approximable within an asymptotic performance ratio of $10\sqrt{5}-21-\delta$, for any $\delta > 0$, unless $\mathcal{P}$=$\mathcal{NP}$.

Until recently, it appeared that the complexity of CVC in bipartite graphs had not been considered in the literature. We show that CVC-D is $\mathcal{NP}$-complete, even for planar bipartite graphs $G = (V_1, V_2, E)$, where each vertex in V_1 has degree at most 3, and each vertex in V_2 has degree at most 4. This result was proved independently by Escoffier et al. [15], who also showed that CVC is solvable in polynomial time, given a bipartite graph $G = (V_1, V_2, E)$ where each vertex in V_1 has degree at most 2, and there is no upper bound on the degree of a vertex in V_2.

In this paper we also consider the parameterised complexity of CVC. We present a parameterised algorithm for CVC-D with complexity $\mathcal{O}^*(2.9316^k)$, improving on a previous algorithm due to Guo et al. [28], having complexity $\mathcal{O}^*(6^k)$. Here the parameter is the size of the cvc. We remark that Mölle et al. [37] presented a parameterised algorithm for CVC-D with complexity $\mathcal{O}^*(3.2361^k)$. Furthermore, subsequently to the acceptance of the original version of this paper [19], Mölle et al. [38, 40] presented (independently and by using different techniques) an improved algorithm for the same problem, having complexity $\mathcal{O}^*(2.7606^k)$. Finally, Moser [41] describes a parameterised algorithm for CVC-D where the parameter is the treewidth of the input graph.

1-TEC, i.e., the problem of finding a minimum edge cover, is polynomial-time solvable [44]. In Section 5, we give a Gallai identity involving $\alpha_{1,t}(G)$ for each $t \geq 1$. We use this to prove that t-TEC-D is $\mathcal{NP}$-complete for each $t \geq 2$. We also show that t-TEC is approximable within 2 for each $t \geq 2$, though there exists some $\delta > 1$ such that 2-TEC is not approximable within δ unless $\mathcal{P}$=$\mathcal{NP}$. Finally we show that t-TEC-D is in $\mathcal{FPT}$ for each $t \geq 2$ (where the parameter is the size of the t-tec) and the parametric dual of 2-TEC-D is also in $\mathcal{FPT}$. This gives one of the few examples where both a problem and its dual belong to $\mathcal{FPT}$.

2 Preliminary observations involving $\alpha_{0,t}(G)$ and $\alpha_{1,t}(G)$

We begin this section by presenting some relationships involving the parameters $\alpha_0(G)$, $\alpha_{0,t}(G)$ and $\alpha_0^c(G)$.

Proposition 1. *Let $G = (V, E)$ be a connected graph where $n = |V|$, $m = |E| \geq 1$, and let $1 \leq t \leq n$. Then:*

1. *$\alpha_0(G) \leq \alpha_{0,t}(G)$, and for $t < n$, $\alpha_{0,t}(G) \leq \alpha_{0,t+1}(G)$;*
2. *$\alpha_{0,t}(G) \geq t$;*
3. *for $\alpha_0^c(G)/2 < t \leq \alpha_0^c(G)$, $\alpha_{0,t}(G) = \alpha_0^c(G)$;*
4. *for $t \geq \alpha_0^c(G)$, $\alpha_{0,t}(G) = t$;*
5. *the minimum t such that $\alpha_{0,t}(G) = t$ satisfies $t = \alpha_0^c(G)$.*

Proof. 1. If S is a $(t+1)$-tvc then clearly S is a t-tvc. Moreover clearly any t-tvc is a vertex cover.

2. If S is any t-tvc, then as $m \geq 1$, it follows that $G[S]$ has at least one connected component, which contains at least t vertices.

3. Let S be a minimum t-tvc and let C be a minimum cvc. Then C is a t-tvc, so that $|S| \leq |C| = \alpha_0^c(G)$. Now suppose that $G[S]$ contains at least two connected components. Then $|S| \geq 2t > \alpha_0^c(G)$, a contradiction. Hence S is a cvc, so that $|C| \leq |S|$. Hence $\alpha_{0,t}(G) = \alpha_0^c(G)$.

4. Let C be a minimum cvc and let $t' = t - |C|$. As G is connected we may construct a t-tvc S by adding t' vertices to C. Then $|S| = t$, so that $\alpha_{0,t}(G) \leq t$. Hence $\alpha_{0,t}(G) = t$ by Part 2.

5. Let $t = \alpha_0^c(G)$. By Part 4, $\alpha_{0,t}(G) = t$. Now suppose that $t' < t$ and $\alpha_{0,t'}(G) = t'$. Let S be a t'-tvc such that $|S| = t'$. Then $G[S]$ contains one connected component, for otherwise $|S| \geq 2t'$, a contradiction. Hence S is a cvc such that $|S| = t' < \alpha_0^c(G)$, a contradiction. □

We next present some relationships involving the parameters $\alpha_1(G)$ and $\alpha_{1,t}(G)$.

Proposition 2. *Let $G = (V, E)$ be a connected graph where $n = |V|$, $m = |E| \geq 1$, and let $1 \leq t \leq m$. Then:*

1. *$\alpha_1(G) \leq \alpha_{1,t}(G)$, and for $t < m - 1$, $\alpha_{1,t}(G) \leq \alpha_{1,t+1}(G)$;*
2. *$\alpha_{1,t}(G) \geq t$;*
3. *for $\frac{n-1}{2} < t \leq n - 1$, $\alpha_{1,t}(G) = n - 1$;*
4. *for $t \geq n - 1$, $\alpha_{1,t}(G) = t$.*
5. *the minimum t such that $\alpha_{1,t}(G) = t$ satisfies $t = n - 1$.*

Proof. 1. If S is a $(t+1)$-tec then clearly S is a t-tec. Moreover clearly any t-tec is an edge cover.

2. If S is any t-tec, then as $m \geq 1$, it follows that $G[S]$ has at least one connected component, which contains at least t edges.

3. Let S be a minimum t-tec and let T be a spanning tree of G. Then T is a t-tec, so that $|S| \leq |T| = n - 1$. Now suppose that $G[S]$ contains at least two connected components. Then $|S| \geq 2t > n - 1$, a contradiction. Hence $G[S]$ is connected, so that $|S| \geq n - 1$. Thus $\alpha_{1,t}(G) = n - 1$.

4. Let T be a spanning tree of G and let $t' = t - (n-1)$. As G is connected we may construct a t-tec S by adding t' edges to T. Then $|S| = t$, so that $\alpha_{1,t}(G) \leq t$. Hence $\alpha_{1,t}(G) = t$ by Part 2.

5. Let $t = n - 1$. By Part 4, $\alpha_{1,t}(G) = t$. Now suppose that $t' < t$ and $\alpha_{1,t'}(G) = t'$. Let S be a t'-tec such that $|S| = t'$. Then $G[S]$ contains one connected component, for otherwise $|S| \geq 2t'$, a contradiction. Hence S is a spanning tree such that $|S| = t' < n - 1$, a contradiction. □

3 Complexity and approximability of t-TVC

We begin with a lower bound for the approximability of t-TVC in general graphs.

Theorem 3. *For each $t \geq 1$, t-TVC is $\mathcal{NP}$-hard and not approximable within an asymptotic performance ratio of $10\sqrt{5} - 21 - \delta$, for any $\delta > 0$, unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof. For $t = 1$ the result follows by [13]. Now assume that $t \geq 2$. Let $G = (V, E)$ be an instance of VC. We lose no generality in assuming that G is connected and $|V| \geq 2$. Create a new graph $G' = (V', E')$ such that $V' = V \cup W$ and $E' = E \cup E_1 \cup E_2$, where $W = \{w_i : 1 \leq i \leq t\}$ is a set of new vertices, $E_1 = \{\{v, w_1\} : v \in V\}$ and $E_2 = \{\{w_i, w_{i+1}\} : 1 \leq i \leq t-1\}$. Let $W' = W \backslash \{v_t\}$. It is straightforward to verify that if S is a minimum vertex cover in G, then $S \cup W'$ is a t-tvc in G'. Conversely if S' is a minimum t-tvc in G', then $S' \cap W = W'$, and $S' \cap V$ is a vertex cover in G. Hence $\alpha_{0,t}(G') = \alpha_0(G) + t - 1$. The result follows by [13]. □

We now present an upper bound for the approximability of t-TVC.

Theorem 4. *For each $t \geq 1$, t-TVC is approximable within 2.*

Proof. Let $G = (V, E)$ be an instance of t-TVC (then G is a connected graph, where $n = |V| \geq t$ and $m = |E| \geq 1$). Savage [46] presents an approximation algorithm for CVC: the algorithm computes a cvc S in G such that $|S| \leq 2\alpha_0(G)$. Suppose firstly that $t \leq |S|$. Then S is a t-tvc, and $|S| \leq 2\alpha_0(G) \leq 2\alpha_{0,t}(G)$ by Proposition 1, as required. Now suppose that $t > |S|$. Let $t' = t - |S|$. As G is connected, we may construct a t-tvc S' in G by adding t' vertices to S. Then $|S'| = t$, so that S' is in fact a minimum t-tvc by Proposition 1. □

The next two results concern the complexity and approximability of t-TVC in bounded degree bipartite graphs, for each $t \geq 2$.

Theorem 5. *For each $t \geq 2$, t-TVC-D is $\mathcal{NP}$-complete for planar bipartite graphs of maximum degree 3.*

Proof. Clearly t-TVC-D belongs to $\mathcal{NP}$. To show $\mathcal{NP}$-hardness, we give a reduction from the $\mathcal{NP}$-complete restriction of VC-D to planar graphs of maximum degree 3 [27, 26]. Hence let $G = (V, E)$ (a planar graph of maximum degree 3) and k (a positive integer) be an instance of this problem. Let $E = \{e_1, e_2, \ldots, e_m\}$ for some m. We define an instance of t-TVC-D as follows. Construct a graph $G' = (V', E')$ by letting $V' = V \cup W$, where $W = \{w_{i,j} : 1 \leq i \leq m \wedge 1 \leq j \leq t\}$. For each i $(1 \leq i \leq m)$, suppose that $e_i = \{u, v\}$ for some $u, v \in V$. Add the edges $\{u, w_{i,1}\}$, $\{w_{i,j}, w_{i,j+1}\}$ $(1 \leq j \leq t-1)$ and $\{w_{i,1}, v\}$ to E'. Clearly G' can be constructed in polynomial time from G, and G' is planar, bipartite and has maximum degree 3. Let $k' = k + (t-1)m$. We claim that G has a vertex cover of size at most k if and only if G' has a t-tvc of size at most k'.

For, suppose that G has a vertex cover S of size at most k. Let $S' = S \cup W'$, where $W' = W \backslash \{w_{i,t} : 1 \leq i \leq m\}$. Then it may be verified that S' is a t-tvc of G', and $|S'| = |S| + (t-1)m \leq k + (t-1)m = k'$.

Conversely suppose that G' has a t-tvc of size at most k'. Choose S' to be such a set that minimises $|S' \cap W|$. It is straightforward to verify that $W' \subseteq S'$, since $t \geq 2$. Also, $S' \cap W = W'$. For, suppose that $w_{i,t} \in S'$ for some i $(1 \leq i \leq m)$. Let $e_i = \{u, v\}$ for some $u, v \in V$. Define $S'' = (S' \backslash \{w_{i,t}\}) \cup \{u\}$. Then S'' is a t-tvc of G', $|S''| \leq |S'| \leq k'$, and $|S'' \cap W| < |S' \cap W|$, contradicting the choice of S'. Hence the claim is established. Let $S = S' \cap V$. Then it may be verified that S is a vertex cover of G, and $|S| = |S'| - (t-1)m \leq k' - (t-1)m = k$. □

Corollary 6. *For each $t \geq 2$, t-TVC in bipartite graphs of maximum degree 3 is not approximable within $1 + \frac{1}{500t-400}$ unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof. VC in cubic graphs is not approximable within $\frac{100}{99}$ unless $\mathcal{P}$=$\mathcal{NP}$ [10]. By considering this problem as the starting point for the same reduction as in the proof of Theorem 5, it again follows that $\alpha_{0,t}(G') = \alpha_0(G) + (t-1)m$. Now $\alpha_0(G) \geq \beta_1(G) \geq \frac{m}{5}$ [49, Theorem 60], where $\beta_1(G)$ is the size of a maximum matching in G, since G is cubic. It follows that $\alpha_{0,t}(G') \leq (5t-4)\alpha_0(G)$. Hence the reduction of Theorem 5 is an L-reduction (defined in [45]) with parameters $\alpha = 5t - 4$ and $\beta = 1$. The result follows by [49, Theorem 63]. □

We now turn to the parameterised complexity of t-TVC, for $t \geq 2$. Our first result establishes a small kernel for this family of problems, indicating that they belong to the class $\mathcal{FPT}$. To be more precise, we obtain an annotated kernel in the sense of [1].

Theorem 7. *For each $t \geq 2$, t-TVC-D has a kernel of size $\mathcal{O}(k(k+t))$, and hence is in $\mathcal{FPT}$ and can be solved in time $\mathcal{O}^*((k(k+t))^k)$, where k is the size of the t-tvc.*

Proof. Let $G = (V, E)$ (a connected graph) and k (a positive integer where $k \geq t$) be an instance of t-TVC-D. Initially let all vertices in G be unmarked and let $k' = k$. We exhaustively apply the following reduction rules to G, for any remaining vertex v:

1. v is isolated. If v is marked, output NO, otherwise delete v.
2. v is unmarked and has degree $> k$. Mark v and decrement k'.
3. v is marked and $deg(v) \geq t - 1$. Partition $N(v)$ into $N_1(v)$ and $N_2(v)$, where $N_1(v)$ comprises vertices of degree 1, and $N_2(v)$ comprises vertices of degree ≥ 2. Let $N_1^*(v)$ denote those vertices in $N_1(v)$ that are marked. Delete $\min\{|N_1(v)| - |N_1^*(v)|, deg(v) - (t-1)\}$ unmarked vertices from $N_1(v)$.

It is straightforward to verify that, if the above rules output NO, or if k' falls below 0, then $\langle G, k \rangle$ is a NO-instance. Otherwise let $G' = \langle V', E' \rangle$ denote the kernel graph that remains after the reduction rules have been applied exhaustively to G, and let S denote the set of marked vertices at that point. Then $k = k' + |S|$. Moreover it is straightforward to verify that G has a t-tvc of size at most k if and only if G' has a t-tvc of size at most k that contains the vertices in S.

Now suppose that C is such a t-tvc of G'. Let V_1 be the vertices of G' of degree 1, let V_3 be the vertices of G' of degree larger than k, and let V_2 be the remaining vertices of G'. Let $C_i = C \cap V_i$ and let $k_i = |C_i|$ for $i = 1, 2, 3$. Since $V_3 = S$, it follows that $|V_3| = k_3$. Also $k_1 + k_2 \leq k'$ and $k_1 + k_2 + k_3 \leq k$.

Let $v \in V_1$. Either (i) $v \in C_1$, or (ii) $v \notin C_1$. In case (ii), v's unique neighbour w in G' belongs to C. Hence w has degree at least 2 in G', and thus belongs to $V_2 \cup V_3$, since $t \geq 2$. Rule 3 above implies that there are no more than $t-1$ unmarked degree-one neighbours of

w in G'. Since this observation, together with case (i), accounts for all the vertices in V_1, it follows that $|V_1| \leq k_1 + (t-1)(k_2+k_3)$. Hence, $|V_1 \cup V_3| \leq k_1 + (t-1)k_2 + (t-1)k_3 + k_3 \leq tk' + tk_3 \leq tk$.

Let E_2 denote the edges in $G'[V_2]$. Since C_2 is a vertex cover of $G'[V_2]$, it follows that $|E_2| \leq k_2 k$, since each vertex in C_2 can cover at most k edges of E_2. But $|E_2| \geq |V_2|$, since each vertex in V_2 has degree at least 2. It follows that $|V_2| \leq k_2 k \leq k^2$.

The set V' forms our kernel for t-TVC-D, and it follows from the above that $|V'| = \mathcal{O}(k(t+k))$. The second stage of our implied $\mathcal{FPT}$ algorithm enumerates all subsets of $V_1 \cup V_2$ of size k', checking to see whether each is a t-tvc of G (given the marked vertices in V_3 already chosen). It follows that the overall complexity of our approach is $\mathcal{O}^*((k(k+t))^k)$. □

We next present a parameterised algorithm for 2-TVC which improves on the algorithm suggested by Theorem 7 for the case that $t = 2$.

Theorem 8. *2-TVC-D can be solved in time $\mathcal{O}^*(2.3655^k)$, where k is the size of the 2-tvc.*

As a preparatory step, let us first describe an algorithm running in time $\mathcal{O}^*(4^k)$. This will display the main idea of our more refined and intricate algorithm that will yield the claimed running time.

Lemma 9. *2-TVC-D is in $\mathcal{FPT}$ and can be solved in time $\mathcal{O}^*(4^k)$, where k is the size of the 2-tvc.*

Proof. All our algorithms are based on the following simple observation. each 2-tvc of size k contains a minimal (not necessary total) vertex cover of size at most k.

It is known (see [17, 12]) that all minimal vertex covers of size at most k can be enumerated (listed) in time $\mathcal{O}^*(2^k)$. So, the question is how to extend a minimal vertex cover C to a valid 2-tvc C' of minimum size. In other words, how can we find a 2-tvc C' with $C \subseteq C'$, such that there is no other 2-tvc C'' with $C \subseteq C''$ and $|C''| < |C'|$? If this question can be answered in time $\mathcal{O}^*(c^k)$, then 2-TVC-D is in $\mathcal{FPT}$; more precisely, it could be solved in time $\mathcal{O}^*((2c)^k)$.

For each minimal vertex cover C of G described at a leaf of the search tree, we construct a hypergraph H as follows: $V' = V \setminus C$ are the vertices of the hypergraph, and the hyperedges E' are the open neighbourhoods of the vertices in C that do not contain (other) vertices from C. Now, a "minimum extension" of C to a valid 2-tvc corresponds to a minimum hitting set in H. Since $|E'| \leq k$, this can be done in time $\mathcal{O}^*(2^k)$ according to [22]; see also [17, Theorem 8.1]. This shows that 2-TVC-D can be solved in time $\mathcal{O}^*(4^k)$. □

The main idea of how to improve the running time of Lemma 9 is to avoid branching at vertices of small degree in the vertex cover enumeration phase. To implement this idea, we introduce a *marking function* μ (also called a *colouring*) on the vertex set V of the input graph G. So, the algorithm will actually deal with *annotated graph instance*s. We are going to explain the meaning of the marking $\mu : V \to \{-1, 0, 1, 2\}$ in the following:

- Vertices $v \in V$ with $\mu(v) = -1$ are also called *unmarked*. At the very beginning of the algorithm all vertices are unmarked.
- Vertices that are not unmarked will be called *marked*. They are further distinguished by the value of the marking function they received.
- A vertex is *1-marked* if it is known to belong to the (partial) vertex cover (due to branching).

- A *2-marked* vertex also belongs to the vertex cover, but, in addition, also one of its neighbours is known to belong to the vertex cover.
- For a *0-marked* vertex, it is unknown if it belongs to the vertex cover, but one of its neighbours is known to belong to the vertex cover.

Our algorithms will put more and more vertices into a (partial) cover, either by reduction rules or by branching. For the marking function, this means according changes. However, due to the semantics just described, these changes can only increase the value of a marking. More specifically, unmarked vertices should be turned into 1-marked vertices and 0-marked vertices should be turned into 2-marked vertices upon putting them into the cover.

We call an annotated instance *updated* if the following properties are true:

1. There is no unmarked vertex that is known to be in the vertex cover or in the neighbourhood of a vertex from the vertex cover.
2. There is no 1-marked vertex that is neighbour of a vertex known to belong to the vertex cover.
3. There is no vertex that is 2-marked.

As we will see, updated instances allow further simplifications. The updating itself and the mentioned simplifications can be handled by simple rules that we call *colour handling rules*. Details of these rules are described in the actual proof of Theorem 8, since these rule will also modify the parameter budget in a specific way, which in turn affects the run time analysis as given in the proof. Without explaining this parameter budget handling, the rules listed in Fig. 1 can be already read at this point.

To give an idea how the marking could be helpful to get rid of small-degree vertices, consider the case that x is a vertex of degree 0:
Degree-0 rule: If G is an annotated updated graph instance and x is a vertex of degree 0, then do:

- If x is 1-marked, then we have a NO-instance.
- Otherwise, delete x.

Lemma 10. *The degree-0 rule is sound.*

Proof. Consider an isolated vertex x. Since the instance is updated, x is not 2-marked. If x were 1-marked, then—being isolated—x cannot find a neighbour in the cover. Hence, we have a NO-instance in that case. If x is 0-marked or unmarked, we delete it: Because x is isolated, taking it into the cover would neither cover any edges nor help a neighbour (that is in the cover) to form a component of size two. □

Since the degree-1 rule will explicitly deal with the parameter(s) as well, we will defer its discussion to the actual proof of Theorem 8 below.

As a second feature of our improved algorithm, we mention that the hitherto clearly separated vertex cover enumeration and hitting set phases will be interleaved. In particular, this means that the search tree part hidden in the vertex cover enumeration phase will be made explicit. This mixing of the two phases also enables to better deal with the involved parameter(s), which finally yields the better run time estimate.

Figure 1: *Colour handling rules* (parameters that are not changed are not mentioned):

1. If vertex x is unmarked but neighbour of a 1-marked or 2-marked vertex, then 0-mark x.
2. If vertex x is 1-marked and is a neighbour of a 1-marked or 2-marked vertex, then 2-mark x and decrement k_2.
3. Merge two 2-marked vertices.
4. If none of the previously listed rules apply, delete a possibly existing 2-marked vertex.

Proof. (of Theorem 8) The following proof is organised into three subsections: In (A), we discuss and justify the reduction rules. Based on the correctness of the reduction rules, we prove the correctness of our algorithm in part (B). Finally, in (C) we show how the interplay of reduction rules and heuristic priorities positively affect the run time estimate.
(A) Discussion of the reduction rules.

To be coherent with the search tree part, we manage the parameter budget in a way that the 1- and 2-marked vertices are already taken into account; hence we have a NO-instance (corresponding to a particular branch of the search tree) if the parameter budget falls below 0.

We now introduce two budgets (parameters) k_1 and k_2, both bounded by the original parameter size of k at the outset, where k_1 actually bounds the first part (the vertex cover enumeration, although this dictum is not strictly true) and k_2 bounds the second part (the Hitting Set phase) of the search tree. Obviously, $k_1 \leq k_2$ as long as we are in the vertex cover enumeration phase. Moreover, $k_2 - k_1$ equals the number of 1-marked vertices.
Our discussion justifies the following *reject rule*:
If the parameter k_1 drops below 0, we have a NO-instance.

We employ the *colour handling rules* listed in Fig. 1. The correctness of these rules should be clear. Let us call an instance that is reduced with respect to the colour handling rules *CHR-reduced*.

These rules guarantee that the following properties of a CHR-reduced instance are true: Claim 1:

1. A CHR-reduced instance is updated.
2. The set of 1-marked vertices forms an independent set in the graph of a CHR-reduced instance.

Further, notice that there is at most one 2-marked vertex in the graph before applying the last of the colour handling rules due to the third of these rules. ◇

In the following, we always assume that we have already exhaustively executed the colour handling rules.

We have already explained above how to handle vertices of degree zero (see Lemma 10). Vertices of degree one are handled as follows:
Degree-1 rule: If G is a annotated updated graph instance and x is a vertex of degree 1 with unique neighbour y, then do the following:

1. If x is unmarked or 0-marked, we distinguish two subcases:
   (a) If y has degree 1, decrease both parameters k_1 and k_2 by 2 if x and y are both unmarked, and decrease both parameters k_1 and k_2 by 1 if x or y is 0-marked. In both cases, delete both x and y.

(b) Otherwise, y has degree at least 2.

- If y is unmarked, 1-mark y, delete x and decrement k_1.
- If y is 0-marked, 2-mark y, delete x and decrement k_1 and k_2.

2. If x is 1-marked, then 2-mark y and decrement the parameters k_1 and k_2.

Claim 2: The degree-1 rule is sound.
For the proof of this claim, we distinguish three cases: (1) x is 1-marked, (2) x is unmarked or 0-marked and $deg(y) = 1$, and (3) x is unmarked or 0-marked and $deg(y) \geq 2$.
Case (1): If x is 1-marked, then the edge $\{x, y\}$ is covered. Notice that, since we are dealing with CHR-reduced instances, y must be unmarked or 0-marked, because otherwise Property 2 of CHR-reduced instances is violated. This means that y was not known to belong to the vertex cover. The only way to form a valid 2-tvc is to put y into the cover, as well, since otherwise x would find no neighbour in the cover. We model this fact by 2-marking y. Moreover, since y was not known to belong to the vertex cover before, we decrease k_1 and k_2 by one.
Case (2) means that we have a matching edge between x and y.
If x and y are both unmarked, then both x and y must be put into the 2-tvc in order to cover the edge $\{x, y\}$, since this is the only way to guarantee that x or y find a neighbouring vertex in the cover. Since neither x nor y have other neighbours beside y or x, we can safely delete this small component. Moreover, we must update the parameters k_1 and k_2: in particular, since both x and y have a neighbour in the cover (after putting both into the cover), they need not be considered in the Hitting Set phase, which justifies decreasing k_2 by two.
If x is 0-marked, then it is known that there exists a neighbour of x (in the original graph; this vertex might have been deleted by now) that belongs to the vertex cover. So, if we put x into the cover, it will be 2-marked and can hence be finally deleted. Since we are looking for a small 2-tvc, there is no reason to put y into the cover as well. So, we may carry out the operations as described.
Consider now case (3). Let $z \in N(y) \setminus \{x\}$. Consider a minimum 2-tvc C of the original graph that extends the partial cover that has been fixed so far during the run of the algorithm. If $y \notin C$, then $x, z \in C$, since otherwise the edges $\{x, z\}$ and $\{y, z\}$ would not be covered. This means that at the current point of the run of the algorithm, x (being of degree one) must be 0-marked (otherwise, x is in the cover but no neighbour of x is). Moreover, the only reason why x is in the cover is in order to cover $\{x, y\}$. Therefore, $(C \setminus \{x\}) \cup \{y\}$ would be a valid 2-tvc of the same size. Hence, we can assume without loss of generality that y is put into the cover. In that case, if C is a minimum 2-tvc that includes x, then $(C \setminus \{x\}) \cup \{z\}$ is a minimum 2-tvc excluding x. The handling of the markings is now easily understood. ◇

If the vertices that are unmarked or 0-marked form an independent set, a vertex cover has been found; without any 1-marked vertices, we have even found a 2-tvc.

In Alg. 1, the reader can find a summary of how to employ the reduction rules in a way that guarantees their correct usage, so that the soundness claims made above and the derived properties are true.

(B) Correctness of the search tree algorithm.
After having shown the correctness of the reduction rules, we are now going to discuss the correctness of the overall search tree algorithm that can be found in Alg. 2. In the very beginning, i.e., when confronted with an instance (G, k) of 2-TVC-D, the algorithm initialises the marking function μ to constantly deliver (-1), it assigns $k_1 := k_2 := k$, and the initial (partial) cover is empty, as is the branching candidate list L. Throughout the

Algorithm 1 The employment of the reduction rules for 2-TVC-D, called tvc-reduce

Require: an annotated graph $G = (V, E)$ with marking functions $\mu : V \to \{-1, 0, 1, 2\}$, positive integers k_1 and k_2, a set C' of cover vertices

Ensure: return a reduced version of G if G might have a 2-tvc $C \subseteq V$ with $|C| \leq k_1$; NO otherwise

```
    {The rules will implicitly modify G, μ, k1, k2, and C' themselves}
    if k1 < 0 then
      return NO; {reject rule}
    else
 5:   changed := false;
      while possible do
        apply one of the first three colour handling rules; changed := true;
      end while
      if possible then
10:     delete a 2-marked vertex; changed := true;
      end if
      if possible then
        apply a degree-0 or a degree-1 rule; changed := true;
        {An application of such a rule might also modify C'}
15:   end if
      if changed then
        return tvc-reduce(G, μ, k1, k2, C')
      else
        return (G, μ, k1, k2, C')
20:   end if
    end if
```

run of the algorithm, (more specifically, before recursively entering tvc-st) we maintain the following invariants:

1. $|C'| = k - k_1$ (notice that k is not changed by the algorithm)
2. $k_2 - k_1 = |\{v \in V \mid \mu(v) = 1\}|$

We now present a list of give properties. The first four are used to establish the fifth, which is particularly useful for the run time estimate (dealt with under part (C) of the proof below) for the algorithm.

Claim 3: Upon entering lines 18 ff. of Alg. 2, we face an instance with the following properties:
(1) All neighbours of 0-marked vertices are unmarked or 1-marked.
(2) There are unmarked vertices.
(3) All neighbours of unmarked vertices are unmarked or 0-marked.
(4) Except for the very first call of the algorithm, each component of the graph instance contains 0-marked vertices.
(5) Except for the very first call of the algorithm, there are always 0-marked vertices in the graph that have unmarked neighbours.

Namely, 2-marked vertices are deleted after applying the reduction rules. So, (1) would be false if we encounter two adjacent 0-marked vertices. This is impossible due to the branching in lines 7–8 of Alg. 2.
If (2) was false, we would have entered the Hitting Set phase, lines 9ff. of Alg. 2.
If (3) was false, the first colour handling rule would have applied, contradicting the fact that we deal with reduced instances at this point.
Property (4) can be seen by induction on the depth of the search tree. After the very first branching (at y), some 1-marked vertices will appear in each component of the originally connected graph; observe that in the branch where y is not taken into the cover, the

Algorithm 2 An advanced search tree algorithm for 2-TVC-D, called tvc-st

Require: an annotated graph $G = (V, E)$ with marking functions $\mu : V \to \{-1, 0, 1, 2\}$, a positive integers k_1 and k_2, a partial cover C (including those vertices marked 1 or 2 in G and also possible further vertices that have been deleted from the original graph at this point), a preferred branching candidate list L

Ensure: return $C' \supseteq C$ if G has a consistent 2-tvc set $C' \cap V$ with $|C' \cap V| \leq k_1$; NO otherwise

```
   C' := C; {C' collects the cover vertices}
   reduction-result := tvc-reduce(G, μ, k1, k2, C');
   if reduction-result = NO then
      return NO;
 5: else
      (G, μ, k1, k2, C') := reduction-result; {G has no 2-marked vertices}
      if there are two neighboured vertices x, y with μ(x) = μ(y) = 0 then
         branch at y;
      else if G contains no more unmarked vertices (i.e., no x with μ(x) = −1) then
10:      form the corresponding Hitting Set instance with the 0-marked vertices as vertices of the
         hypergraph and the hyperedges N(x) for x being 1-marked;
         compute a minimum hitting set H;
         if |H| > k1 then
            return NO;
         else
15:         return H ∪ C';
         end if
      else
         delete from L all vertices x with μ(x) > 0;
         select a vertex y for branching according to the list of priorities from Fig. 2;
20:      if deg(y) ≤ 6 and L = ∅ then
            L := N(y);
         end if
         branch at y;
      end if
25: end if
```

originally connected graph might decompose into several components. The reduction rules will hence introduce at least one 0-marked vertex in each component. A careful look at all reduction rules and branching scenarios proves that this property is maintained through the course of the algorithm. More specifically, branching itself never poses a problem, since it will 1-mark or 2-mark new vertices and hence 0-mark their neighbours. If 0-marked vertices disappear through the reduction rules, this either means that a component is completely resolved or that (through the degree-1 rule) new vertices are put into the cover, which again means that their neighbours will be 0-marked.

If (5) was not true, all neighbours of 0-marked vertices must be 1-marked due to Property (1). By the colour handling rules, all neighbours of 1-marked vertices must be 0-marked. Therefore, in this scenario, a component that contains 0-marked vertices can contain 0-marked or 1-marked vertices only. By Property (4), this means that the whole graph instance only contains 0-marked and 1-marked instances. Hence, the Hitting Set Phase would have been entered, contradicting our assumption that we entered lines 18 ff. of Alg. 2. ◇

The correctness proof of our algorithm can be easily done by induction, once the following assertions are seen:

Claim 4: The translation into the MINIMUM HITTING SET instance is correct.

Claim 5: The branching is in particular correctly handling the parameters and the markings.

Notice that we can safely defer the discussion of the heuristic priorities to subsection (C) of this proof, since they only affect the order of precedence in which vertices are selected for branching, but they do not interfere with the correctness, since eventually some vertex y will be selected for branching.

The hitting set phase is only entered when there are no more unmarked vertices. Immediately before entering this phase, the reduction rules have been exhaustively applied, as explained in Alg. 1. Therefore, the instance is updated, which implies that there are no more 2-marked vertices. Therefore, all vertices are either 0-marked or 1-marked. In order to be ready to enter the hitting set phase, we must also guarantee that the 1-marked vertices form a vertex cover. This is checked by the branching immediately preceding the hitting set phase: namely, the hitting set phase is only entered when no two neighbouring 0-marked vertices can be found, i.e., only when the 0-marked vertices form an independent set. It is well-known that the complement of an independent set is a vertex cover, which means that in our case, when entering the hitting set phase, the 1-marked vertices form a vertex cover as required. Hence, the translation into the MINIMUM HITTING SET instance is correct; so, by the reasoning given above for the simple version of the search-tree algorithm for 2-TVC-D, we are guaranteed to obtain a minimum valid 2-total vertex cover set that extends our previously determined partial cover. ◇[of Claim 4]

Algorithm 3 The code of "branch at y"

$j := \mu(y) + 1$; {It is known that $j \in \{0, 1\}$.}
Define μ' by setting $\mu'(y) := \mu(y) + 2$ and $\mu'(x) := \mu(x)$ for $x \neq y$;
$C'' :=$tvc-st$(G, \mu', k_1 - 1, k_2 - j, C' \cup \{y\}, L)$;
if $C'' \neq$NO **then**
5: return C'';
else
$j := |\{v \in N(y) \mid \mu(v) = 0\}|$;
$i := |\{v \in N(y) \mid \mu(v) \leq 0\}|$;
modify μ by setting $\mu(v) := \mu(v) + 2$ for all $v \in N(y)$ with $\mu(v) < 1$ at present;
10: delete y from G (and from μ);{keeping the same names G, μ}
return tvc-st$(G, \mu, k_1 - i, k_2 - j, C' \cup N(v), L)$;
end if

To see Claim 5, we have to study Alg. 3. Basically, the branching in Alg. 3 explores two mutually exclusive possibilities: either (a) y is put into the cover, or (b) y is not put into the cover. In case (a), we update C' by adding y. Moreover, the budget k_1 is decremented, which is correct when we assume that we only branch at vertices hitherto not determined to go into the vertex cover (this property can be easily checked by inspection). The parameter budget k_2 is decremented if and only if $\mu(y) = 0$. Case (b) is a little bit more complicated, but it can be analysed completely similarly. The only differences that might need some attention are the following ones: (i) C' might have vertices in common with $N(y)$, the set of vertices added in the recursive call. (ii) We modify G by deleting y from the instance. This is justified by our branching: (I) We will not consider the case that y might go into the cover (also not in the following hitting set phase), since this case is covered under (a). (II) Since all neighbours of y are put into the cover, y should be 0-marked. By (I), it will never be 2-marked, so that it could be deleted from further consideration. ◇

(C) Discussion of the branching in special cases

This leads to an analysis of the run time as claimed. However, let us first understand that even without this more sophisticated case analysis inspired by the heuristic priorities listed in Fig. 2, we obtained a significant gain over the first and simple search-tree algorithm. Namely, since we never branch on vertices of degree zero or one, the vertex cover search-

Figure 2: *Heuristic priorities* used for branching in Alg. 2

1. If possible, select $y \in L$. Modify $L := L \setminus \{y\}$.
2. If possible, select an unmarked y that belongs to a triangle.
3. If possible, select an unmarked y that has a neighbour x with $\mu(x) = 0$.
4. Otherwise, select any unmarked y.
   { A catch-all; possibly needed at the very beginning }

tree phase only takes time $\mathcal{O}^*(1.62^k)$, (ignoring for the moment the branch at neighbouring 0-marked vertices) which totals up to a run time estimate of $\mathcal{O}^*(3.24^k)$.[1]

Our run time analysis contains a feature that could be helpful in analysing other parameterised algorithms, as well, in particular those consisting of two phases: namely, by considering both of the two parameters k_1 and k_2 in the recursions bounding the run times, we could take advantage of reducing either of them.

We only remark on one specific feature of this analysis here which should be understood in order to appreciate the pseudocode of our algorithm, namely the role of the list L. This list is introduced only to enforce a specific branching behaviour of the algorithm that would possibly be violated if not explicitly mentioned. Therefore, vertices on the list are treated with priority, as in Fig. 2. This enforced branching behaviour will be exploited in the run time analysis of the case of branching on vertices of degree at most six. Therefore, only in that case, the list is updated by our algorithm. So, this list handling avoids some special-case branchings that would be necessary otherwise to correctly reflect the run time analysis.

We now present the run-time analysis. In a first step, we analyse the run time of the branching of neighbouring 0-marked vertices (as done before entering the hitting set phase) and the run time for branching in triangles (as done by the second heuristic priority).

Let us look at a first simple example, namely *triangles*, as found as the second item on the priority list of Fig. 2. There, we face a situation where the three vertices x, y, z in the graph that are mutually neighboured and y is selected for branching. At most one vertex from $\{x, z\}$ is 0-marked due to Property (1) from Claim 3. This implies that all of them are either unmarked or 0-marked, since we are dealing with updated instances. In the case when y is put into the cover, both x and z will be 0-marked by the colour handling rules (in the recursive call). Hence, lines 7–8 of Alg. 2 apply, putting either x or z into the cover. Therefore, the branching that is triggered covers three cases that could be summarised as taking two out of the three vertices from $\{x, y, z\}$ into the cover.

Actually, already for the classical vertex cover case, (i.e., 1-TVC-D), we know that two out of these three vertices must go into the cover. So, the point of the preceding paragraph is to show that this scenario will happen according to our algorithm. Notice that in this special case these two vertices will be neighbours, so that this branching scenario is also valid for 2-TVC-D, since clusters are automatically created. Therefore, the vertices put into the cover by this branching need not be taken care of later in the Hitting Set phase. Hence, the budget for that phase can be reduced by 2; in fact, this would be done by the colour handling rule that subsequently deletes two 2-marked vertices. The corresponding recurrence is therefore:

$$T(k_1, k_2) \leq 3T(k_1 - 2, k_2 - 2)$$

Fortunately, we already know the complexity of the second phase, which is $\mathcal{O}^*(2^{k_2})$. As-

[1] A different but similar approach to arrive at this run time is explained in [18].

suming a running time of $\mathcal{O}^*(c^{k_1})$ for the first phase (with c still to be determined), we obtain the condition

$$c^{k_1}2^{k_2} \leq 3c^{k_1-2}2^{k_2-2}.$$

Multiplication with $c^{-k_1+2}2^{-k_2+2}$ yields: $4c^2 \leq 3$, i.e., $c \leq \sqrt{3}/2$. The overall running time of the algorithm can be grossly estimated by assuming $k_1, k_2 \leq k$, so that in that particular case $(2 \cdot c)^k \leq \sqrt{3}^k \leq 1.7321^k$ follows.

Let us now consider another special case with a very nice branching behaviour: that of two neighboured vertices x, y that are 0-marked (see lines 7–8 of Alg. 2). Clearly, either x or y must go into the cover in order to cover the edge $\{x, y\}$. Whichever vertex we put into the cover, notice that it will be immediately 2-marked (and hence it will not be considered in the Hitting Set phase). Therefore, also the parameter k_2 is decreased. We are led to the recurrence:

$$c^{k_1}2^{k_2} \leq 2c^{k_1-1}2^{k_2-1}$$

which is obviously solved by $c \leq 1$, yielding $(2 \cdot c)^k \leq 2^k$.

Due to Property (5) of Claim 3, we can assume in the following that we can always find an unmarked vertex y that is a neighbour of a 0-marked vertex x. We will branch at y or its neighbours according to what we describe in the following. In Alg. 2, this type of branching (on the neighbours) is enforced by using the list L. Notice that we cannot rely on the third heuristic priority here, since we might be tempted to do other branchings at other places of the graph before returning to the neighbourhood (which might be even completely uninteresting for further branching at that point); so, we have to guide our branching with the help of the additional list L. However, notice further that this is the only purpose of this list, so that this can be easily implemented by (after having dealt with all triangles) always first looking within the neighbourhood of the "current" node for further branching candidates.

In the corresponding analysis, we often use the idea that when x is put into the cover, then (since x is 0-marked) also the second parameter is decreased by 1 (actually, it would be possible to decrease the second parameter even by 2 if we knew that x is neighbour of a 1-marked vertex, but the worst case is that all neighbours of x are unmarked).

If $deg(y) \geq 7$, we simply branch at y: either y is put into the cover (decrementing only k_1) or all its at least 7 neighbours are put into the cover. The latter branch decreases k_1 by 7 and decrements k_2, since x is put into the cover. We are led to the recurrence:

$$c^{k_1}2^{k_2} \leq c^{k_1-1}2^{k_2} + c^{k_1-7}2^{k_2-1}$$

which is equivalent to $2c^7 \leq 2c^6+1$; finally yielding $(2 \cdot c)^k \leq 2.3653^k$. This will be (nearly) our worst-case scenario in the end.

If $deg(y) \leq 6$, we use another good branching idea that is indeed rather special to the cluster concept in 2-TVC-D: one of the neighbours of a 1-marked vertex must be put into the cover in order to satisfy the cluster condition. For small-degree 1-marked vertices, this leads to a very satisfactory branching behaviour. We will use this idea in the subcase that takes y into the cover (and hence 1-marked). To actually exploit this idea we had to tweak our algorithm a little bit by introducing a vertex list L that handles the neighbours of y with priority in recursive calls of our algorithm, hence following the pattern of analysis given below.

Consider the case $deg(y) = 6$. Let $N(y) = \{x, z_1, z_2, z_3, z_4, z_5\}$ describe the neighbourhood of y. If we take y into the cover, then one $v \in N(y)$ must go into the cover, since y will then become 1-marked.

- If z_1 is put into the cover, then both k_1 and k_2 are reduced by 2.

- If z_1 is not put into the cover, $N(z_1)$ must be part of the cover. Assume that now z_2 is put into the cover.
- Otherwise, both z_1 and z_2 are not put into the cover but their neighbourhoods are. Assume also that z_3 goes into the cover.
- This argument continues up to the point that none of the z_i are put into the cover but all their neighbours are. In addition, x is put into the cover.

If we do not take y into the cover, then (as before) all neighbours of y are put into the cover, in particular x, so that the second parameter is reduced by 1.

To find a good estimate for the recursion, we have to reason about possible common vertices and minimum degrees. First of all, as we shall see later, we may assume that all vertices z_i have each at least three neighbours. We can also assume that for all i, j: $z_i \notin N(z_j)$ and that $x \notin N(z_j)$, since this would mean triangles in our instance. So the neighbours of z_i, plus z_{i+1} or x, constitutes at least 4 vertices. All these vertices (including y) will be put into the cover in all but the first case of the case distinction above, therefore reducing the first parameter by 4 and the second by 2 (at least). We get as an overall estimate for the recursion:

$$c^{k_1}2^{k_2} \leq c^{k_1-2}2^{k_2-2} + 5c^{k_1-4}2^{k_2-2} + c^{k_1-6}2^{k_2-1}$$

The first term of the right-hand side comes from the case that puts both y and z_1 into the cover, and the last term represents the case that y is not put into the cover but all its neighbours are. A little algebra reveals that $(2c)^k \leq 2.3655^k$ in this case. In fact, this is the overall worst case estimate of our algorithm.

Similarly, in the case $deg(y) = 5$, we can derive

$$c^{k_1}2^{k_2} \leq c^{k_1-2}2^{k_2-2} + 4c^{k_1-4}2^{k_2-2} + c^{k_1-5}2^{k_2-1}$$

leading to $(2c)^k \leq 2.3055^k$. Assuming $deg(y) = 4$, we get:

$$c^{k_1}2^{k_2} \leq c^{k_1-2}2^{k_2-2} + 3c^{k_1-4}2^{k_2-2} + c^{k_1-4}2^{k_2-1}$$

and hence $(2c)^k \leq 2.2361^k$. Similarly, in the case $deg(y) = 3$, we can derive

$$c^{k_1}2^{k_2} \leq c^{k_1-2}2^{k_2-2} + 2c^{k_1-4}2^{k_2-2} + c^{k_1-3}2^{k_2-1}$$

leading to $(2c)^k \leq 2.1454^k$.

We now consider the case that $deg(y) \geq 3$ and that y has a neighbour $z_1 \neq x$ of degree 2 (already excluding the case of degree 1 due to reduction rules). Then, we can branch as follows: either take y into the cover or take all of $N(y)$. The analysis of this branching depends on whether or not z_1 is unmarked. If z_1 is unmarked, then not taking y into the cover not only puts the (three or more) neighbours of y into the cover, but it will (due to the fact that y is deleted from the instance) be the case that z_1 is then of degree 1; hence, a reduction rule will trigger in the recursive call of the procedure and then the unique neighbour of z_1 will be put into the cover as well, in order to satisfy the cluster condition. Therefore, we get the following overall recurrence:

$$c^{k_1}2^{k_2} \leq c^{k_1-1}2^{k_2} + c^{k_1-4}2^{k_2-3}$$

This leads us to $(2c)^k \leq 2.1903^k$. If however z_1 is 0-marked, then in the case that y is not put into the cover at least two 0-marked vertices will be 2-marked (and therefore

they disappear by reduction rules in the next recursive call). This leads to the following recurrence:

$$c^{k_1}2^{k_2} \leq c^{k_1-1}2^{k_2} + c^{k_1-3}2^{k_2-2}$$

which means that $(2c)^k \leq 2.3594^k$.

Finally, we consider the remaining case that $deg(y) = 2$, i.e., $N(y) = \{x, z\}$. If we choose y, then we need to choose either x or z to satisfy the cluster property, since y is unmarked. If we do not choose y, then we need to choose both x and z. When choosing x, then x will become 2-marked. This leads to the following recurrence:

$$c^{k_1}2^{k_2} \leq 2c^{k_1-2}2^{k_2-2} + c^{k_1-2}2^{k_2-1}.$$

Thus $(2c)^k \leq 2^k$. This concludes our case discussion. □

4 Complexity and approximability of CVC

We begin with two results concerning the complexity and approximability of CVC in general graphs and planar bipartite graphs of bounded degree.

Theorem 11. CVC *is not approximable within an asymptotic performance ratio of* $10\sqrt{5} - 21 - \delta$*, for any* $\delta > 0$*, unless* $\mathcal{P}$=$\mathcal{NP}$*.*

Proof. The result follows using the construction in the proof of Theorem 3 for the case that $t = 2$. □

Theorem 12. CVC-D *is* $\mathcal{NP}$*-complete for planar bipartite graphs* $G = (V_1, V_2, E)$*, where each vertex in* V_1 *has degree at most 3, and each vertex in* V_2 *has degree at most 4.*

Proof. Clearly CVC-D belongs to $\mathcal{NP}$. To show $\mathcal{NP}$-hardness, we use the same reduction as in Theorem 5 with $t = 2$, however in this case we reduce from the $\mathcal{NP}$-complete restriction of CVC-D to planar graphs of maximum degree 4 [26]. The graph G' so constructed is then also a planar bipartite graph $G' = (V_1, V_2, E')$, where $V_1 = \{w_{i,1} : 1 \leq i \leq m\}$ and $V_2 = V \cup \{w_{i,2} : 1 \leq i \leq m\}$. Clearly each vertex in V_1 has degree at most 3, whilst each vertex in V_2 has degree at most 4. If S is a connected vertex cover of size at most k in G, then $S \cup W'$ is a connected vertex cover of size at most k' in G'. Conversely if S' is a minimum connected vertex cover of size at most k' in G', then $S \cap W = W'$. It follows that $S' \cap V$ is a connected vertex cover in G of size at most k. □

We remark that independently, Escoffier et al. [15] established APX-completeness for CVC in bipartite graphs with the same degree restrictions as described in the statement of Theorem 12.

We now show how to use the colouring techniques from the proof of Theorem 8 in order to give a parameterised algorithm for CVC-D that improves on the previous $\mathcal{O}^*(6^k)$ algorithm described in [28]. Independently of our results, Mölle et al. [38, 40] obtained a slightly better $\mathcal{O}^*(2.7606^k)$ algorithm for CVC-D by improving on the enumeration phase.

Theorem 13. CVC-D *is in* $\mathcal{FPT}$ *and can be solved in time* $\mathcal{O}^*(2.9316^k)$*, where* k *is the size of the cvc.*

Proof. The algorithm proceeds along the lines of the one suggested in [28], using two stages: firstly, we enumerate all minimal vertex covers of size at most k, and secondly we apply an algorithm for solving the STEINER TREE PROBLEM on (at most) k terminals. Using the $\mathcal{O}^*(2^k)$ enumeration phase for all minimal vertex covers, combined with the well-known $\mathcal{O}^*(3^k)$ Dreyfus-Wagner algorithm [14], the running time of [28] follows.

Recently, the running time of the Dreyfus-Wagner algorithm has been improved to $\mathcal{O}^*((2+\varepsilon)^k)$ for any $\varepsilon > 0$ (the smaller the ε, the bigger the polynomial, but this is hidden in the $\mathcal{O}^*$ notation) [23, 39]. Combining this with the $\mathcal{O}^*(2^k)$ enumeration phase for all minimal vertex covers, we obtain an $\mathcal{O}^*((4+\varepsilon)^k)$ algorithm for CVC-D. The aforementioned hidden constants (covered by the ε) can be avoided by using the fact that the terminal nodes of the Steiner tree instance form a vertex cover; this is detailed in [37] and immediately gives an $\mathcal{O}^*(4^k)$ algorithm for CVC-D.

This can be further improved by using a colour scheme similar to the one given for 2-TVC-D in combination with *catalytic branching*, a technique introduced in [16]. This means that we consider all n cases as to whether a given vertex belongs to the vertex cover set to be constructed. In our case, we will mark a vertex selected in the branching. This can be seen in Alg. 4. Notice that the procedure cvc-annotated takes, besides the parameter(s), a non-empty set of marked vertices as arguments.

Algorithm 4 An advanced search tree algorithm for CVC-D, called cvc-st

Require: a graph $G = (V, E)$ and a non-negative integer k
Ensure: return YES if G has a connected vertex cover of size at most k: NO otherwise

```
    if E = ∅ then
      return YES;
    else
      for all v ∈ V do
5:      if cvc-annotated(G, k − 1, k, {v}) = YES then
          return YES;
        end if
      end for
      return NO;
10: end if
```

The procedure that deals with annotated instances is described in Alg. 5. Similar to our search-tree algorithm for 2-TVC-D, the main ingredients of this algorithm are: (1) reduction rules, (2) a search-tree backbone, and (3) a list of heuristic priorities for branching. Due to the simple branching that is employed, the correctness of the algorithm follows by proving the correctness of (A) the reduction rules and (B) the search-tree backbone. The heuristic priorities (together with the reduction rules) will determine the running time (part (C)).
(A) The reduction we use is described in Alg. 6.
Claim 1: The reduction rules are sound.

Notice that all rules but the last work in the immediate neighbourhood of the catalyst vertex v.

1. Two marked neighboured vertices together are obviously covering all edges outgoing from either of them; this can be equivalently expressed by merging the two of them into a new marked vertex.

2. An unmarked neighbour x of degree 1 of a marked vertex v has no need to go into the cover: the only edge it might cover is already covered by v and it cannot connect to other marked vertices. Hence, we can safely delete x.

3. An unmarked neighbour x of degree at least 2 of a marked vertex v that has an unmarked neighbour y (clearly, $y \neq v$ by the assumed marking) that is of degree at most 2 might be responsible for covering the edge $e = \{x, y\}$. In principle, e could be also covered by y. However, if y was in the cover, then one of its at most two neighbours must go into the cover to satisfy the connectivity requirement. If the degree of y was 1, this means that x must be in the cover, which is just the case

Algorithm 5 A search tree algorithm for annotated CVC-D, called cvc-annotated

Require: a graph $G = (V, E)$, two integers k_1, k_2 (k_1 is bounding the first vertex-cover-like search tree part, while k_2 is bounding the number of terminal points in the subsequent Steiner tree algorithm; hence $k_1 \leq k_2$); a non-empty set $C \subseteq V$ of marked vertices

Ensure: return YES if G has a connected vertex cover of size at most k that contains all vertices from C; NO otherwise

```
produce a reduced instance;
{for simplicity, we use the same namings as before: the reduction is described in Alg. 6}
if k_1 < 0 then
  return NO;
else if C covers all edges of G then
  return (Steiner-Tree (G, C) ≤ k_1);
  {Steiner-Tree computes a Steiner tree for G with terminal vertices C (where |C| ≤ k_2) and
  returns the number of Steiner points}
else
  pick an unmarked vertex u ∈ V to branch at; {how to choose is described in Alg. 7}
  if cvc-annotated(G, k_1 − 1, k_2, G, C ∪ {u}) = YES then
    return YES;
  else
    Let i := |{v ∈ N(u) | v is unmarked}|;
    return cvc-annotated(G − u, k_1 − i, k_2, G, C ∪ N(u));
  end if
end if
```

as claimed by the reduction rule. If y is of degree 2, then y could be in the cover together with another neighbour $z \neq x$ of y. Moreover, there must be a path from z to v within the cover that does not contain y. However, instead of putting v, y, z into the connected cover, we could also take v, x, z into an alternative connected cover of the same size.

4. Consider the triangle given by v, x, y. Since we are aiming at finding a vertex cover, x or y must be in the cover. By the assumption that $N[y] \subseteq N[x]$, any solution that has y but not x within the cover could be converted into a feasible solution (no bigger than the previous solution) having x in the cover, instead, by exchanging y for x. This reasoning is also valid having in mind the connectivity requirement. $\diamond$

Claim 2: After exhaustively applying the reduction rules, the marked vertices together from an independent set.

Namely, neighboured marked vertices are merged. $\diamond$

(B) We basically run a simple vertex cover search-tree algorithm. When the graph has no more edges, then we take the vertex cover set as a set of terminal nodes and look for a minimum Steiner tree (in the modified graph G). The procedure Steiner Tree returns the minimum number of Steiner points. It is called only if there are no more edges in the graph. The result it returns must be bounded by the remaining budget k_1. By induction, the following invariants can be shown:

Claim 3: (invariants) (0) $|\phi(C)| + k_1 = k$; (1) $k_1 + |C| = k_2$; (2) $|C| \leq k_2$.

Here, $\phi(C)$ denotes the set of vertices obtained from C by "undoing" all merge operations undertaken on the specific search tree path.

Namely, each time we put a vertex into the cover C (and hence in $\phi(C)$), k_1 is decreased, proving (0). Also k_2 is decremented if and only if two neighbouring vertices in C are merged, hence $|\phi(C)| - |C| = k - k_2$. This, together with (0), implies (1). The assertion (2) is an easy consequence of (1). $\diamond$

These observations together show the correctness of the search-tree backbone. $\diamond$

Algorithm 6 Reductions for annotated CVC-D, called cvc-reduce

Require: a graph $G = (V, E)$, integers k_1, k_2; the set of marked vertices C
Ensure: return an irreducible instance
repeat
if $\exists x, v \in C : x \in N(v)$ **then**
merge v and x into a new member of C; {implicitly modifying V, E}
$k_2 := k_2 - 1$; {the number of vertices to be considered in the Steiner tree phase of the overall algorithm is reduced}
else if $v \in C$ has an (unmarked) neighbour x of degree 1 **then**
delete x;
else if $v \in C$ has an unmarked neighbour x of degree at least 2 that has an unmarked neighbour y that is of degree at most 2 **then**
mark x;
$k_1 := k_1 - 1$;
else if $v \in C$ has two unmarked neighbours x, y that form a triangle, and $N[y] \subseteq N[x]$ **then**
mark x;
$k_1 := k_1 - 1$;
end if
until no more changes occur to the instance

We now discuss how to pick a vertex $u \notin C$ to branch at, i.e., either take u into the cover or all its neighbours. A pseudocode can be found in Alg. 7. As before, let C be the (assumed) cover so far found (i.e., the set of marked vertices). If none of the listed conditions applies, we have already found a vertex cover of G and can now run some Steiner tree algorithm to ensure connectivity of the cover set (see [28]).

Algorithm 7 Heuristic priorities for annotated CVC-D, called cvc-prio

if possible **then**
Choose a vertex $v \in C$ such that there is a neighbour $u \in N(v)$ that has at least two unmarked neighbours
else if possible **then**
Choose a vertex $v \in C$ such that there is a vertex u in $N(N(v)) \setminus (N[v] \cup C)$ with at least one marked neighbour
else if possible **then**
Choose a vertex $v \in C$ such that there is a vertex u of degree at least 3 in $N(N(v)) \setminus (N[v] \cup C)$
end if

Why can we continue with the Steiner tree phase when none of the possibilities applies?
Claim 4: If none of the possibilities listed in Alg. 7 applies, there are no uncovered edges.

To show this claim, consider the vicinity of a vertex $v \in C$. By Claim 2, all neighbours of v are unmarked. If v has a neighbour with at least two unmarked neighbours, we would branch according to the first heuristic priority. Hence, thereafter any neighbour x of v has at most one unmarked neighbour y. Conversely, by the reduction rules, x must have one neighbour apart from v. If all neighbours of x are marked, there is no need for further branching in the first phase, since all edges incident with x are covered. So, if this observation applies to all neighbours of all $v \in C$, then all edges are covered.

Hence, we can assume in the following discussion that x has exactly one unmarked neighbour y.
(1) If y has a marked neighbour z, then the penultimate reduction rule applies unless both x and y are of degree at least three (notice that z could take on the role of v in the reduction rule). Further, recall that by our previous arguments, all neighbours of x apart from y are marked, and by symmetry of the situation, all neighbours of y apart from x are

marked, too. Hence, the second branching scenario applies, even in the case that $z = v$. Namely, in that case, some marked vertex from $X := N(x) \setminus N[y]$ could take on the role of v. Notice that our last reduction rule ensures that $X \neq \emptyset$.
(2) So, assume that y has no marked neighbours. If y has at most one neighbour besides x, the reduction rules would have triggered. Hence, y has at least two unmarked neighbours besides x, and the third branching scenario considers this final case. ◇

(C) As for the running time analysis, notice that there are now the following possible worst cases:

1. $v \in C$ has a neighbour u of degree ≥ 3 we branch at (with ≥ 2 unmarked neighbours).
2. $v \in C$ has a neighbour x of degree ≥ 2 that has an unmarked neighbour u with at least one marked neighbour z.
3. $v \in C$ has a neighbour of degree ≥ 2 that has an unmarked neighbour u with ≥ 3 unmarked vertices; we branch at u in that case.

In Case 1, we can estimate the running time of the overall search tree (including the Steiner tree computation phase) as follows:

$$T(k_1, k_2) \leq T(k_1 - 1, k_2 - 1) + T(k_1 - 2, k_2).$$

The first term describes the running time of the branch that includes u in the cover and the second term describes the running time of the branch that excludes u from the cover but takes all neighbours into the cover. Notice that if u is taken into the cover, it will be marked and hence merged with its marked neighbour after the recursive call due to the reduction rules. Hence, the second parameter k_2 (bounding the Steiner tree part) is decremented as claimed. A little algebra shows that $T(k_1, k_2) \leq 1.2808^{k_1}(2 + \varepsilon)^{k_2}$.

In Case 2, when u is not put into the cover, x will go into the cover. In either case, the resulting marked vertex will be neighbour of a marked vertex, i.e., in the recursion the reduction rules trigger and reduce the second parameter. Hence, we can estimate

$$T(k_1, k_2) \leq 2T(k_1 - 1, k_2 - 1).$$

Due to the very nice reduction of the second parameter, we can estimate $T(k_1, k_2) \leq (2 + \varepsilon)^{k_2}$.

In Case 3, notice that in the case that u is not put into the cover but all its neighbours, a neighbour of v will be put into the cover which reduces the second parameter k_2. We then obtain the estimate

$$T(k_1, k_2) \leq T(k_1 - 1, k_2) + T(k_1 - 3, k_2 - 1) \leq 1.4655^{k_1}(2 + \varepsilon)^{k_2}.$$

This gives the claimed worst case running time. □

Remark 14. Due to Claim 2 in the previous proof, a reduced graph on which we start the Steiner tree phase is bipartite. However the bipartite property alone does not in general lead to a polynomial-time algorithm for this phase. To see this, it is straightforward to observe that Karp's reduction [33] shows that the Steiner tree problem is NP-hard even if we have a set T of terminal vertices such that G has no edges between two vertices in T and between two vertices in $V \setminus T$. This fact justifies the use of the $\mathcal{O}^*((2+\varepsilon)^k)$ algorithm for the Steiner tree problem [23, 39] here.

Remark 15. Very recently, Björklund et al. [3] obtained (under certain restrictions) an $\mathcal{O}^*(2^{|T|})$ algorithm for MINIMUM STEINER TREE, where T is the set of terminal nodes. This algorithm has no large hidden constants, as in the case of the $\mathcal{O}^*((2 + \varepsilon)^{|T|})$-algorithm.

5 Complexity and approximability of t-TEC

Let $G = (V, E)$ be a connected graph, where $n = |V|$ and $m = |E| \geq 1$. We begin this section by remarking that a t-total edge cover in G does not necessarily correspond to a t-total vertex cover in $L(G)$ (the line graph of G). For example, given any $t \geq 1$, let $H = K_{1,t+1}$. Then $L(H) = K_{t+1}$. Also $\alpha_{0,t}(L(H)) = t$ whilst $\alpha_{1,t}(H) = t + 1$.

Let $1 \leq t \leq n - 1$. We now present a Gallai identity involving the concepts of a t-tec and a *t-tree packing*. A *t-tree packing* of G is a collection $\mathcal{P} = \{G_1, \ldots, G_k\}$ of vertex-disjoint (non-induced) subgraphs of G, each of which is a tree containing exactly t edges. The value k is defined to be the *size* of $\mathcal{P}$. Let $\beta_{1,t}(G)$ denote the maximum size of a t-tree packing of G. Then $\beta_{1,1}(G) = \beta_1(G)$, the size of a maximum matching in G. The following result gives a Gallai identity involving $\alpha_{1,t}(G)$ and $\beta_{1,t}(G)$.

Theorem 16. *Let $G = (V, E)$ be a connected graph, where $n = |V|$, $m = |E| \geq 1$, and let $1 \leq t \leq n - 1$. Then $\alpha_{1,t}(G) + \beta_{1,t}(G) = n$.*

Proof. Let $\mathcal{P} = \{G_1, \ldots, G_k\}$ be a t-tree packing of G such that $k = \beta_{1,t}(G)$. Let S initially contain the edges belonging to the subgraphs in $\mathcal{P}$. Then $|S| = kt$ and S covers $k(t + 1)$ vertices of G, so that $n - k(t+1)$ vertices are as yet uncovered. Pick any uncovered vertex v. Then v is at distance at most t from a covered vertex w, for otherwise we contradict the maximality of $\mathcal{P}$. Let $v_0 = v$, and let $v_0, v_1, \ldots, v_s$ be the vertices (in order) on a path in G from v_0 to v_s, where v_s is covered, v_i is uncovered ($1 \leq i \leq s - 1$), and $1 \leq s \leq t$. Add $\{v_i, v_{i+1}\}$ to S ($0 \leq i \leq s - 1$). Continue in this way until all vertices are covered. Then S is a t-tec of G. Moreover we add one edge for every additional vertex that we cover, so that $|S| = kt + (n - k(t + 1)) = n - k$, i.e., $\alpha_{1,t}(G) \leq n - \beta_{1,t}(G)$.

Conversely let $\mathcal{S} = \{S \subseteq E : S \text{ is a } t\text{-tec in } G \text{ and } |S| = \alpha_{1,t}(G)\}$. Choose $S \in \mathcal{S}$ such that $G[S]$ contains the fewest number of cycles. Let $G_i = (V_i, S_i)$ ($1 \leq i \leq k$) be the connected components of $G[S]$, for some $k \geq 1$. Let i ($1 \leq i \leq k$) be given. Then by definition of S, it follows that G_i contains at least t edges. Now suppose that G_i contains a cycle, and let e be any edge on this cycle. If $k = 1$ then $S' = S \backslash \{e\}$ is a connected subgraph of G that spans V, and hence $|S'| \geq n - 1$, so that S' is a t-tec, contradicting the minimality of S. Hence $k \geq 2$. Since S is an edge cover, there exists an edge $e' = \{u, v\} \notin S$ in G such that u is covered by G_i and v is covered by some G_j ($1 \leq j \neq i \leq k$). Let $S' = (S \backslash \{e\}) \cup \{e'\}$. Then S' is a t-tec, $|S'| = |S|$ and S' has one fewer cycle than S, contradicting the choice of S. Hence G_i is acyclic. It follows that $t \leq |S_i| = |V_i| - 1$, so that

$$|S| = \sum_{i=1}^{k} |S_i| = \sum_{i=1}^{k} (|V_i| - 1) = n - k.$$

Let $\mathcal{P} = \{H_1, \ldots, H_k\}$ be formed by "pruning" each G_i in order to form a tree H_i containing exactly t edges (this may be carried out by repeatedly deleting edges incident to vertices of degree 1 in G_i, until exactly t edges remain). Then $\mathcal{P}$ is a t-tree packing of G, and $|\mathcal{P}| = k = n - \alpha_{1,t}(G)$, so that $\beta_{1,t}(G) \geq n - \alpha_{1,t}(G)$. □

We remark that, in the case $t = 1$, Theorem 16 gives the familiar Gallai identity $\alpha_1(G) + \beta_1(G) = n$ [25]. Also, in the case $t = 2$, a similar (but not quite identical) Gallai identity to Theorem 16 was also observed by De Bontridder et al. [5, Theorem 4.2]. Finally, we remark that Moser and Sikdar [42] prove that $\beta_1^*(L(G)) = \beta_{1,2}(G)$ for a graph G, where $\beta_1^*(G)$ denotes the maximum size of an *induced matching* in G (a matching M in G is *induced* if no two edges in M are adjacent to a common edge). Hence, for a connected graph G with at least two edges, it follows by Theorem 16 that $\alpha_{1,2}(G) = n - \beta_1^*(L(G))$.

For each $t \geq 1$, let t-TREE PACKING denote the problem of computing $\beta_{1,t}(G)$, given a connected graph $G = (V, E)$, where $n = |V| \geq t + 1$. Let t-TREE PACKING-D denote the decision version of t-TREE PACKING. Kirkpatrick and Hell [34] proved the following result concerning t-TREE PACKING-D.

Theorem 17 ([34]). *For each $t \geq 2$, t-TREE PACKING-D is $\mathcal{NP}$-complete.*

The following is an immediate consequence of Theorems 17 and 16.

Corollary 18. *For each $t \geq 2$, t-TEC-D is $\mathcal{NP}$-complete.*

The next two results concern the approximability of t-TEC for $t \geq 2$.

Theorem 19. *For each $t \geq 2$, t-TEC is approximable within 2.*

Proof. Let $G = (V, E)$ be an instance of t-TEC (a connected graph where $n = |V|$ and $m = |E| \geq t$). Any edge cover S of G satisfies $|S| \geq \frac{n}{2}$, since each edge of S covers 2 vertices of G. Now let T be a spanning tree of G. Suppose firstly that $t \leq n - 1$. Then T is a t-tec of G and $|T| = n - 1 \leq 2\alpha_1(G) \leq 2\alpha_{1,t}(G)$ by Proposition 2, as required. Now suppose that $t > n - 1$. Let $t' = t - (n - 1)$. As G is connected, we may construct a t-tec S by adding t' edges to T. Then $|S| = t$, so that S is in fact a minimum t-tec by Proposition 2. □

Theorem 20. 2-TEC *in bounded degree graphs is not approximable within some $\delta > 1$ unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof. 2-TREE PACKING in graphs of maximum degree B is not approximable within some $\varepsilon > 1$ unless $\mathcal{P}$=$\mathcal{NP}$ [32]. We may consider this problem as the starting point for a reduction to 2-TEC that essentially follows the same lines as the proof of Theorem 16 in the case that $t = 2$ and $G = (V, E)$ is a connected graph of maximum degree B, where $n = |V|$ and $m = |E|$. Now $\alpha_{1,2}(G) \geq \alpha_1(G) \geq \frac{n}{2}$ by Proposition 2 and the fact that a given edge can cover at most 2 vertices of G. By Theorem 16, $\alpha_{1,2}(G) + \beta_{1,2}(G) = n$. Hence the reduction described here is an L-reduction (see [45]) with parameters $\alpha = \beta = 1$. The result follows by [49, Theorem 63]. □

We now consider the parameterised complexity of t-TEC ($t \geq 2$).

Theorem 21. *For each $t \geq 2$, t-TEC-D is in $\mathcal{FPT}$.*

Proof. Let $\langle G, k \rangle$ be an instance of t-TEC-D. Then k is a parameter and $G = (V, E)$ is a connected graph where $n = |V|$ and $m = |E| \geq t$. As observed in the proof of Theorem 19, $k \geq \frac{n}{2}$ or else $\langle G, k \rangle$ is a NO-instance. Hence $n \leq 2k$, so $m \leq (2k)^2$. Generating every subset S of E with at most k edges and verifying whether S is a t-tec is a process that takes $\mathcal{O}^*((2k)^{2k})$ overall time. □

We now consider the concept of *parametric duality* (see [8, 17] for a recent exposition), which is in a sense quite related to the family of Gallai identities proved above. Define DUAL-t-TEC-D to be the problem of deciding, given a connected graph $G = (V, E)$ where $n = |V|$ and $m = |E| \geq t$, and a (dual) parameter k_d, whether there a t-tec of size at most $n - k_d$. Using the fact that 2-TREE-PACKING-D is in $\mathcal{FPT}$ and solvable in time $\mathcal{O}^*(2.482^{3k})$, where k is the size of the 2-tree-packing [21], Theorem 16 implies the following result.

Theorem 22. DUAL-2-TEC-D *is in $\mathcal{FPT}$ and can be solved in time $\mathcal{O}^*(2.482^{3k_d})$.*

Theorems 21 and 22 therefore imply that both 2-TEC-D and DUAL-2-TEC-D are in $\mathcal{FPT}$, a result rarely observed in the context of parameterised complexity. However, in the case of t-TVC and CVC, we can show:

Theorem 23. DUAL-*2*-TVC-D *is* $\mathcal{W}$*[1]-complete. Also* DUAL-*t*-TVC-D *(*$t \geq 3$*) and* DUAL-CVC-D *are* $\mathcal{W}$*[1]-hard.*

Proof. To show membership in $\mathcal{W}[1]$ of DUAL-2-TVC-D, we employ the "Turing way" [7, 17]. That is, we exhibit a Turing machine whose $f(k_d)$-step halting problem is solvable if and only if the given instance of DUAL-2-TVC-D is a YES-instance.

A 1-tape nondeterministic Turing machine M_G for graph $G = (V, E)$ would work as follows. The tape alphabet is $V \times \{0, 1\}$ (plus the end markers).

1. Guess k_d letters from $V \times \{0\}$ and write them on the tape.

2. Sweep back and forth on the tape and verify that the vertices are independent. (If two vertices u, v have been guessed with $u \in N(v)$, then the Turing machine would enter an infinite loop.)

   The second part of the tape alphabet can be used to protocol which two vertices are tested.

   If all pairs have been tested, then the tape contains an independent set I.

3. Now use the second part of the tape alphabet to cycle through all subsets of I. For each subset $\emptyset \neq X \subseteq I$, we have to test whether $X = N(v)$ for some $v \notin I$. If this is the case, then we have detected a vertex from the vertex cover $V \setminus I$ that has no neighbour from $V \setminus I$.

   To this end, an n-bit internal memory is used. Initially, this is an all-zero vector. Upon reading X off the tape, at most k_d bits are set to 1. Then, by the internal memory bit vector $X = N(v)$ can be checked in one further step. If the infinite loop is not entered (i.e., $X \neq N(v)$ for all $v \in V \setminus I$), then the k_d bits are set to 0 again, and then the "next set" is selected by the bit vector counter on the tape.

   Finally, the bit vector counter on the tape contains only ones, and then the machine will stop.

Hence, there is a function $f(k_d)$ such that G has a total vertex cover of size $n - k_d$ if and only if M_G stops in at most $f(k_d)$ steps.

We now show that DUAL-*t*-TVC-D is $\mathcal{W}[1]$-hard, for each $t \geq 2$. We use the same reduction as in Theorem 3, where $G = (V, E)$ is a connected graph with $n = |V| \geq 2$ and k_d is a parameter, given as an instance of INDEPENDENT SET-D. Then G has an independent set of size k_d if and only if the $(n + t)$-vertex graph G' has a t-tvc of size $n - k_d + (t - 1) = (n + t) - (k_d + 1)$.

In the case of DUAL-CVC-D, the proof is similar; the same reduction may be used with $t = 2$. □

6 Concluding remarks

In this paper we have defined the concepts of a t-tvc and a t-tec for $t \geq 1$, which are motivated by the notions of covering and clustering in graphs. We have presented $\mathcal{NP}$-completeness, approximability and parameterised complexity results for associated optimisation and decision problems.

Until now, enumeration-based solutions to parameterised decision problems seemed to be doomed to give rise to a complexity function $\mathcal{O}^*(C^k)$ where C is quite large. Our $\mathcal{FPT}$ algorithms in this paper demonstrate how this can be overcome by introducing appropriate "colours" and corresponding reduction rules within the search tree algorithm. A further example is the EDGE DOMINATING SET algorithm described in [18]. Moreover, a novel

way of analysing search trees that can be decomposed into two phases is exhibited; this has proved to be highly effective in the case of 2-TVC-D and CVC-D, and should also be applicable in improving the analysis of other fixed-parameter algorithms.

In Section 4, we presented an $\mathcal{O}^*(2.9316^k)$ algorithm for CVC-D. As mentioned in Section 1, an improved $\mathcal{O}^*(2.7606^k)$ algorithm for CVC-D has been reported in [38, 40]. It is likely that a further improvement could be obtained by combining the approach of Mölle et al. with the reduction rules that we employ for our CVC-D algorithm: while we obtain savings from better estimates of the Steiner tree phase, they obtain savings from improved vertex cover enumeration.

The results in this paper leave open the following problems, among others, that are worthy of further consideration: (1) Formulate polynomial-time algorithms for t-TVC and t-TEC in restricted classes of graphs. (2) Formulate $\mathcal{FPT}$ algorithms for t-TVC-D ($t > 2$) that improve on the general approach suggested by Theorem 7. Are the corresponding parametric dual problems in $\mathcal{W}[1]$? (3) Theorem 21 shows that t-TEC-D is solvable in time $\mathcal{O}^*(2^{O(k \log k)})$ for each $t \geq 2$. Are these problems solvable in time $\mathcal{O}^*(2^{O(k)})$? (4) Consider "clustering" variants of vertex domination and edge domination.

Acknowledgements

The first author is grateful to Mike Fellows for raising the question of whether t-TVC-D has a small kernel; this led to solving an open question from [19] (see Theorem 7). The second author would like to thank Michele Zito for helpful discussions regarding 2-total vertex covers, and Pavol Hell for drawing our attention to reference [34] in connection with t-tree packings. Both authors would like to thank the referees for their comments, which have helped to improve the presentation of this paper.

References

[1] F. Abu-Khzam and H. Fernau. Kernels: annotated, proper and induced. In *International Workshop on Parameterized and Exact Computation IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 264–275. Springer, 2006.

[2] E.M. Arkin, M.M. Halldórsson, and R. Hassin. Approximating the tree and tour covers of a graph. *Information Processing Letters*, 47:275–282, 1993.

[3] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of STOC '07: the 39th Annual ACM Symposium on Theory of Computing*, pages 67–74. ACM, 2007.

[4] J.R.S. Blair. Personal communication, 2001.

[5] K.M.J. De Bontridder, B.V. Halldórsson, M.M. Halldórsson, C.A.J. Hurkens, J.K. Lenstra, R. Ravi, and L. Stougie. Approximation algorithms for the test cover problem. *Mathematical Programming, Series B*, 98:477–491, 2003.

[6] J. Cardinal and E. Levy. Connected vertex covers in dense graphs. In *Proceedings of APPROX + RANDOM 2008: the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, and the 12th International Workshop on Randomized Techniques in Computation,* to appear, Lecture Notes in Computer Science. Springer, 2008.

[7] M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67:654–685, 2003.

[8] J. Chen, H. Fernau, I.A. Kanj, and G. Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. In *Proceedings of STACS 2005: the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2005.

[9] J. Chen and I.A. Kanj. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithmics. *Journal of Computer and System Sciences*, 67:833–847, 2003.

[10] M. Chlebík and J. Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theoretical Computer Science*, 354(3):320–338, 2006.

[11] E.J. Cockayne, R.M. Dawes, and S.T. Hedetniemi. Total domination in graphs. *Networks*, 10:211–219, 1980.

[12] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351:337–350, 2006.

[13] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[14] S.E. Dreyfus and R.A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[15] B. Escoffier, L. Gourvès, and J. Monnot. Complexity and approximation results for the connected vertex cover problem. In *Proceedings of WG '07: the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 4769 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2007.

[16] M.R. Fellows, C. McCartin, F.A. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: an improved FPT algorithm for Max Leaf Spanning Tree and other problems. In *Proceedings of FST TCS 2000: the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2000.

[17] H. Fernau. Parameterized algorithmics: A graph-theoretic approach. Habilitationsschrift, University of Tübingen, 2005.

[18] H. Fernau. Edge dominating set: efficient enumeration-based exact algorithms. In *Proceedings of IWPEC 2006: the Third International Workshop on Parameterized and Exact Computation*, volume 4169 of Lecture Notes in Computer Science, pages 142–153. Springer, 2006.

[19] H. Fernau and D.F. Manlove. Vertex and edge covers with clustering properties: Complexity and algorithms. In *Proceedings of ACiD 2006: the 2nd Algorithms and Complexity in Durham workshop*, volume 7 of *Texts in Algorithmics*, pages 69–84. College publications, 2006.

[20] H. Fernau and R. Niedermeier. An efficient exact algorithm for constraint bipartite vertex cover. *Journal of Algorithms*, 38(2):374–410, 2001.

[21] H. Fernau and D. Raible. A parameterized perspective on packing paths of length two. In *Proceedings of COCOA 2008: the 2nd Annual International Conference on Combinatorial Optimization and Applications,* to appear, Lecture Notes in Computer Science. Springer, 2008.

[22] F. Fomin, D. Kratsch, and G. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *Proceedings of WG '04: the 30th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 3353 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 2004.

[23] B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. Technical Report zaik2005-492, University of Cologne, 2005.

[24] M. Gaertler. Clustering. In U. Brandes and T. Erlebach, editors, *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, chapter 8, pages 178–215. Springer, 2005.

[25] T. Gallai. Über extreme Punkt-und Kantenmengen. *Ann. Univ. Sci. Budapest, Eötvös Sect. Math.*, 2:133–138, 1959.

[26] M.R. Garey and D.S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[27] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[28] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized vertex cover problems. In *Proceedings of WADS 2005: the 9th International Workshop on Algorithms and Data Structures*, volume 3608 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2005.

[29] F. Harary. *Graph Theory.* Addison-Wesley, 1969.

[30] T. Haynes, S.T. Hedetniemi, and P.J. Slater, editors. *Domination in Graphs: Advanced Topics.* Marcel Dekker, 1998.

[31] T. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of Domination in Graphs.* Marcel Dekker, 1998.

[32] V. Kann. Maximum bounded H-matching is MAX SNP-complete. *Information Processing Letters*, 49:309–318, 1994.

[33] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[34] D.G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *Proceedings of STOC '78: the 10th Annual ACM Symposium on Theory of Computing*, pages 240–245. ACM, 1978.

[35] D.F. Manlove. On the algorithmic complexity of twelve covering and independence parameters of graphs. *Discrete Applied Mathematics*, 91:155–175, 1999.

[36] S. Mitchell and S.T. Hedetniemi. Edge domination in trees. In *Proceedings of the 8th South-Eastern Conference on Combinatorics, Graph Theory and Computing*, pages 489–509. Utilitas Mathematica, 1977.

[37] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. In *Proceedings of CSR '06: International Computer Science Symposium in Russia*, volume 3967 of *Lecture Notes in Computer Science*, pages 280–290. Springer, 2006.

[38] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: New runtime bounds for vertex cover variants. In *Proceedings of COCOON 2006: the 12th Annual International Computing and Combinatorics Conference*, volume 4112 of *Lecture Notes in Computer Science*, pages 265–273. Springer, 2006.

[39] D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the Steiner Tree problem. In *Proceedings of STACS 2006: the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 561–570. Springer, 2006.

[40] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory of Computing Systems*, 43:234–253, 2008.

[41] H. Moser. Exact algorithms for generalizations of vertex cover. Master's thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2005.

[42] H. Moser and S. Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics,* to appear, 2008.

[43] N. Nishimura, P. Ragde, and D.M. Thilikos. Fast fixed-parameter tractable algorithms for nontrival generalizations of vertex cover. *Discrete Applied Mathematics*, 152:229–245, 2005.

[44] R.Z. Norman and M.O. Rabin. An algorithm for a minimum cover of a graph. *Proceedings of the American Mathematical Society*, 10:315–319, 1959.

[45] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[46] C. Savage. Depth-first search and the vertex cover problem. *Information Processing Letters*, 14:233–235, 1982.

[47] S. Ueno, Y. Kajitani, and S. Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72:355–360, 1988.

[48] Z. Zhang, X. Gao, and W. Wu. Polynomial time approximation scheme for connected vertex cover in unit disk graph. In *Proceedings of COCOA 2008: the 2nd Annual International Conference on Combinatorial Optimization and Applications,* to appear, Lecture Notes in Computer Science. Springer, 2008.

[49] M. Zito. *Randomised Techniques in Combinatorial Algorithms.* PhD thesis, University of Warwick, Department of Computer Science, 1999.