On Efficient Fixed Parameter Algorithms for Weighted Vertex Cover

Rolf Niedermeier[1⋆] and Peter Rossmanith[2]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
niedermr@informatik.uni-tuebingen.de
[2] Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80290 München, Fed. Rep. of Germany
rossmani@in.tum.de

Abstract. We investigate the fixed parameter complexity of one of the most popular problems in combinatorial optimization, Weighted Vertex Cover. Given a graph $G = (V, E)$, a weight function $\omega : V \to \mathbf{R}^+$, and $k \in \mathbf{R}^+$, Weighted Vertex Cover (WVC for short) asks for a subset C of vertices in V of weight at most k such that every edge of G has at least one endpoint in C. WVC and its variants have all been shown to be *NP*-complete. We show that, when restricting the range of ω to positive integers, the so-called Integer-WVC can be solved as fast as unweighted Vertex Cover (time $O(1.271^k + k|V|)$). Our main result is that if the range of ω is restricted to positive reals ≥ 1, then so-called Real-WVC can be solved in time $O(1.3954^k + k|V|)$. If we modify the problem in such a way that k is not the weight of the vertex cover we are looking for, but the number of vertices in a minimum weight vertex cover, then the same running time can be obtained. If the weights are arbitrary (referred to by General-WVC), however, the problem is not fixed parameter tractable unless $P = NP$.

1 Introduction

An interesting and challenging open problem in computational complexity theory is related to the polynomial time approximability of (Weighted) Vertex Cover [3, 9, 19, 25]: A great number of researchers believe that there is no polynomial time approximation algorithm achieving an approximation factor strictly smaller than $2 - \epsilon$, for a positive constant ϵ, unless $P = NP$. Currently, the best known lower bound for this factor is 1.1666 [17]. According to Crescenzi and Kann [10], (Weighted) Vertex Cover is the most popular problem in combinatorial optimization. This motivates the search for *exact* algorithms providing a vertex cover of *optimal* weight. This paper deals with efficient exact "fixed parameter" algorithms for Weighted Vertex Cover problems, which have provable performance bounds.

⋆ Work performed within the "PEAL" project (Parameterized complexity and Exact ALgorithms), supported by the Deutsche Forschungsgemeinschaft (NI-369/1-1).

A set $C \subseteq V$ is called a *vertex cover* of a graph $G = (V, E)$ if every edge in E has at least one endpoint in C. The WEIGHTED VERTEX COVER problem (WVC for short) is: given a graph $G = (V, E)$, a weight function $\omega : V \to \mathbf{R}^+$, and $k \in \mathbf{R}^+$, find a vertex cover C with total weight $\leq k$. In the special case that all vertices have weight 1, one speaks of UNWEIGHTED VERTEX COVER (UVC for short). Even when restricted to planar graphs with maximum vertex degree 3, UVC is *NP*-complete [16]. There are linear time algorithms giving approximation factor 2 for the unweighted case [16] as well as for the weighted case [6]. Both results can be improved to an approximation factor that is asymptotically better: $2 - \log\log|V|/2\log|V|$ [7, 20]. Until now, no further improvements of these bounds have been obtained.

The parameterized complexity [11] of UVC recently has received considerable interest [4, 8, 11, 13, 22, 28]. Here, for a given k, the question is to find a vertex cover of at most k vertices or to report "no" if no vertex cover of size $\leq k$ exists. In many applications, it makes sense to assume that k is small compared to the total number of vertices $n := |V|$. Hence, exact algorithms with running time exponential *only* in the *parameter* k are considered to be valuable [11, 13]. The currently best known result in this direction is an $O(1.271^k + kn)$ algorithm for UVC [8]. In this paper, we study the more general and so far unexplored question concerning the parameterized complexity of WVC. More precisely, for a given k, the problem now is to find a vertex cover of weight at most k. Herein, we consider three natural variants of WVC:

1. INTEGER-WVC, where the weights are arbitrary positive integers.
2. REAL-WVC, where the weights are real numbers ≥ 1.
3. GENERAL-WVC, where the weights are positive real numbers.

Whereas all three versions are clearly *NP*-complete, it turns out that their parameterized complexity differs significantly: While INTEGER-WVC and REAL-WVC are *fixed parameter tractable,* GENERAL-WVC is *not* fixed parameter tractable unless $P = NP$.

Our results are as follows. INTEGER-WVC can be solved as fast as UVC, which currently has running time $O(1.271^k + kn)$ [8]. Our main result is that REAL-WVC can be solved in time $O(1.3954^k + kn)$. As an important corollary, this implies that if we modify the problem in such a way that k is *not the weight* of the vertex cover we are looking for, but *the number of vertices* in a minimum weight vertex cover, then the same running time can be obtained. This enables a direct comparison with the unweighted case, where we also count the number of vertices in the vertex cover. Conversely, one easily sees that GENERAL-WVC is fixed parameter intractable unless $P = NP$. Hence, there is little hope to find an $O(f(k)n^{O(1)})$ time algorithm for GENERAL-WVC, where f may be a function growing arbitrarily fast in the parameter k.

Due to lack of space we omit some details and proofs.

2 Preliminaries and basic notation

We assume familiarity with the basic notions and concepts of algorithms, complexity, and graph theory. If x is a vertex in a graph, then by $N(x)$ we denote the set of its neighbors. A graph is called regular if all vertices in the graph have the same degree, that is, the same number of neighbors. The whole paper only works with *simple* graphs, i.e., there are no double edges between two vertices.

The (parameterized) problem GENERAL-WVC we study is defined as follows:
Given: A graph $G = (V, E)$, a weight function $\omega : V \to \mathbf{R}^+$, and $k \in \mathbf{R}^+$.
Question: Does there exist a vertex cover of weight at most k?

Our algorithms are based on two key techniques of parameterized complexity [11]: *reduction to problem kernel* (see Section 3) and *bounded search tree* (see Section 5). The former deals with reducing the size of the search space and the latter with a clever search through the search space. Both will be explained in detail in later sections. To estimate the size of bounded search trees (and, thus, the complexity of the algorithm), we make use of *recurrence relations.* As a rule, we use linear recurrences with constant coefficients for which there exist several well-known techniques for solving them [4, 22]. If the algorithm solves a problem of size k and calls itself recursively for problems of sizes $k - d_1, \dots, k - d_r$, then $(d_1, \dots, d_r)$ is called the *branching vector* of this recursion. It corresponds to the recurrence

$$t_k = t_{k-d_1} + \cdots + t_{k-d_r}. \tag{1}$$

The characteristic polynomial of this recurrence is

$$z^d = z^{d-d_1} + \cdots + z^{d-d_r}, \tag{2}$$

where $d = \max\{d_1, \dots, d_r\}$. If α is a root of (2) with maximum absolute value, then t_k is bounded by $|\alpha|^k$ up to a polynomial factor. We call $|\alpha|$ the *branching number* that corresponds to the branching vector $(d_1, \dots, d_r)$. Moreover, if α is a single root, then $t_k = O(\alpha^n)$. All branching numbers that will occur in this paper are single roots.

Finally, without going into details, let us briefly say a few words about *parameterized complexity theory* [11] (also refer to the survey articles [2, 12, 14]). Parameterized complexity, as chiefly developed by Downey and Fellows, is one of the latest approaches to attack problems that are *NP*–complete. The basic observation is that for many hard problems the seemingly inherent combinatorial explosion can be restricted to a "small part" of the input, the *parameter.* For instance, the UNWEIGHTED VERTEX COVER problem can be solved by an algorithm with running time $O(kn+1.271^k)$ [8], where the parameter k is a bound on the maximum size of the vertex cover set we are looking for and n is the number of vertices in the given graph. The fundamental assumption is $k \ll n$. As can easily be seen, this yields an efficient, practical algorithm for small values of k. A problem is called *fixed parameter tractable* if it can be solved in time $f(k)n^{O(1)}$

for an arbitrary function f which depends only on k. The corresponding complexity class is called FPT.[1]

3 Reduction to problem kernel

Suppose we are given a graph G and want to find a vertex cover of weight k. By means of *reduction to problem kernel*—a kind of a preprocessing step—we can reduce the original instance to a "smaller one," (G', k'), where G' is a subgraph of G and $k' \leq k$. It holds that G has a vertex cover of weight k iff G' has a vertex cover of weight k'. Assuming positive vertex weights ≥ 1, a simple standard reduction to problem kernel by Buss works based on the following [11]: Each vertex with degree greater than k has to be in the vertex cover set, since, otherwise, not all edges can be covered. From this it is easy to obtain that G can be replaced with G' such that G' consists of at most k^2 edges and at most k^2+k vertices and k' is obtained from k by reducing by the weight of the high-degree vertices added to the cover, *if there is in fact a vertex cover for G of size k*. If G' has more than $k^2 + k$ vertices, it follows that the original problem has no solution and we can stop.

Chen *et. al.* [8] noted that using a well-known theorem of Nemhauser and Trotter [21] (also see, e.g., [7, 25]), one can even obtain a problem kernel with a number of vertices linear in k. They used the linear size of the problem kernel to improve the exponential term in the running time of their algorithm and also to get rid of a factor of k. The resulting algorithm has running time $O(1.271^k k + kn)$. By a new technique, however, the whole factor k^2 can be discarded and the improvement in the exponential term can also be easily achieved without a linear size problem kernel [23]. Therefore, the problem kernel size requires no special attention in this paper, where reduction to problem kernel is assumed as a preprocessing for the bounded search tree algorithms for Integer- and Real-WVC in Sections 4 and 5.

4 Integer– and General–Weighted Vertex Cover

In this section, we show that Integer-WVC can be solved as fast as UVC and that General-WVC is not fixed parameter tractable unless $P = NP$. To see the latter, it suffices to make the following simple observation: General-WVC is NP-complete for any fixed $k > 0$. For example, there is a straightforward reduction from the NP-complete, unweighted Vertex Cover to General-WVC with $k = 1$. However, this implies that there cannot be a time $f(k)n^{O(1)}$ or even $n^{O(k)}$ algorithm for General-WVC unless $P = NP$. This is true because otherwise we would obtain a polynomial time algorithm for an NP-complete problem.

[1] As a rule, k is defined to be a positive integer, but it can also be generalized to the positive reals. (Usually, k is given explicitely as part of the input, but for some problems it is implicit in the encoding of the input).

In the remaining section, we show that we can reduce INTEGER-WVC to UVC via a simple parameterized many-one reduction (see [11] for any details) that does not change the value of the parameter. To prove the following theorem, we may safely assume that the maximum vertex weight is bounded by k (the according preprocessing needs only polynomial time).

Theorem 1. INTEGER–WVC *can be solved as fast as* UVC *up to an additional term polynomial in* k.

Proof. An instance of INTEGER–WVC is transformed into an instance of UVC as follows: Replace each vertex i of weight u with a cluster i' consisting of u vertices. We do not add intra-cluster edges to the graph. Furthermore, if $\{i, j\}$ is an edge in the original graph, then we connect every vertex of cluster i' to every vertex of cluster j'. Now, it is easy to see that both graphs (the instance for INTEGER-WVC and the new instance for UVC) have minimum vertex covers of same weight/size. Here, it is important to observe that the following is true for the constructed instance for UVC: Either all vertices of a cluster are in a minimum vertex cover or none of them is. Assume that one vertex of cluster i is not in the cover but the remaining are. Then all vertices in all neighboring clusters have to be included and, hence, it makes no sense to include the remaining vertices of cluster i in the vertex cover.

Let $t(k, n)$ be the time needed to solve UVC. The running time of the algorithm on the "cluster instance" is clearly bounded by $t(k, wn) \leq t(k, kn)$, where $w \leq k$ is the maximum vertex weight in the given graph. Because of reduction to problem kernel, a bigger graph with same parameter needs only an *additional* polynomial time preprocessing. □

Chen *et al.* [8] state that INTEGER-WVC can be solved in time $O(1.271^k k + kn)$ (which can be improved to $O(1.271^k + kn)$ [23]). As a consequence, Theorem 1 implies that INTEGER-WVC can be solved in the same time, i.e., $O(1.271^k + kn)$.

5 Real–Weighted Vertex Cover

In this section, we prove our main result. We study the case of weights that are real numbers ≥ 1 and we prove that REAL-WVC can be solved in time $O(1.3954^k + kn)$. We proceed as follows. First, we observe that if a graph has maximum vertex degree two, then there is an easy dynamic programming solution. After that, we study in detail three main cases (in the given order): the case when there is a vertex of degree one in the graph, when there is a triangle (i.e., a clique of size 3) in the graph, and when there is no triangle in the graph. Note that the existence of weights makes the reasoning quite different from bounded search tree algorithms for UVC. The overall structure of our algorithm is as follows. The subsequent instructions are executed in a loop until all edges of the graph are covered or $k = 0$, which means that no cover could be found.

1. If there is no vertex with degree > 2, then solve REAL-WVC in polynomial time by dynamic programming (see Subsection 5.1).

2. Execute the lowest numbered, applicable step of the following.
   (a) If there is a vertex x of degree at least 4, then branch into the two cases of either bringing itself or all its neighbors into the vertex cover. (The corresponding branching vector is at least $(1, 4)$, implying branching number 1.3803 or better.)
   (b) If there is a degree-1 vertex, then proceed as described in Subsection 5.2. (The corresponding branching vector is at least $(1, 4)$, implying branching number 1.3803 or better.)
   (c) If there is triangle in the graph, then proceed as described in Subsection 5.3. (The corresponding branching vector is at least $(3, 4, 3)$, implying branching number 1.3954 or better.)
   (d) If there is no triangle in the graph, then proceed as described in Subsection 5.4. (The corresponding branching vector is at least $(3, 4, 3)$, implying branching number 1.3954 or better.)

Finally, let us only mention in passing that the clever trick of so-called "folding degree-2 vertices," as described by Chen *et al.* [8] for UVC, does not apply to weighted Vertex Cover problems. Subsequently, we prove our main theorem, following the outline given above.

Theorem 2. REAL–WVC *can be solved in time* $O(1.3954^k + kn)$.

Because of step 2.(a) above, in order to prove Theorem 2 we only have to deal with graphs with maximum vertex degree 3.

5.1 Graphs with maximum vertex degree 2

Clearly, graphs with maximum vertex degree 2 are either paths or cycles. We can find an optimal vertex cover for them in polynomial time by dynamic programming: Assume that we have a path or cycle of n vertices, numbered consecutively from n to 1. Say we start with vertex n. Then this vertex is in the vertex cover or it is not. If it is not, then its neighbor has to be in the vertex cover. This can easily be reflected by a simple system of recurrences: Let D_n denote the minimum weight cover containing vertex n and let N_n denote the minimum weight cover not containing vertex n, both referring to a path or a cycle of n vertices. One easily verifies that the following recurrences hold:

$$\begin{aligned} N_1 &= 0, \\ D_1 &= w_1, \\ N_n &= D_{n-1}, \\ D_n &= w_n + \min\{D_{n-1}, N_{n-1}\}, \end{aligned}$$

where w_i is the weight of vertex i. This can easily be solved in linear time, using dynamic programming. Moreover, it is easy to extend this in order to explicitly give a vertex cover set of minimum weight, which is $\min\{D_n, N_n\}$.

5.2 Degree one vertices

In this subsection, we assume that there is at least one vertex that has degree 1. Let x be such a vertex and let a be its only neighbor. In addition, let w be the weight of x and let w' be the weight of a. If $w \geq w'$, then it is optimal to include a in the vertex cover. In the following, we handle the more complicated case that $w < w'$.

Case 1: a has degree 2. Then a path starts at x that proceeds over vertices with degree 2 and ends in a vertex y that has degree 1 or 3. If y has degree 1, then we can find an optimal cover for this graph component by dynamic programming as described in Subsection 5.1. Otherwise we branch on y, bringing either y or its three neighbors into the vertex cover. This gives branching vector $(1, 3)$. If we put y into the vertex cover, we create a new graph component that includes x and a and has only vertices with degree at most 2. We can again apply dynamic programming (Subsection 5.1) and we get a branching vector at least $(2, 3)$ for the whole subgraph.

Case 2: a has degree 3 and has at least one neighbor with degree 3. Let y be a's degree-3 neighbor. We branch on y. If y is in the cover, then a will have degree 2 and Case 1 applies. The $(1, 3)$ branching vector thus can be improved to $(1 + 2, 1 + 3, 3) = (3, 4, 3)$.

Case 3: a has degree 3 and has two neighbors with degree 2. Let y and b be a's degree-2 neighbors. We branch on x. If x is in the cover, then a is not and a's other neighbors y and b are in the cover. This gives branching vector $(1, 3)$, which is not yet good enough. Hence, by considering several more subcases, we do a more complicated branching.

Let z be y's other neighbor and assume that y has weight u, z has weight v, and $u \geq v$. Then we can branch on a and get branching vector $(2, 3)$; note that if a is in the cover, then it is optimal to also include z (instead of y).

Assume next that the weight w' of a is at least 2: Then, branching on a, we have branching vector $(2, 3)$. Let w be the weight of x. We can assume in the following that $w' < w + v$ and $u < v$.

Let us return to the branch on x: If x is in the cover, so are y and b. We can now assume that z is *not* in the cover. Otherwise, we could replace x and y with a, which is better and is already covered by the branch that does not include x. Then all neighbors of z are in the cover, too, and among them must be some vertex other than x, y, or b. If not, change the roles of y and b. In this way, we get branching vector of at least $(1, 4)$ (unless the component has only 6 vertices and, thus, can be handled in constant time).

Case 4: Remaining cases. What remains to be considered are the case when a has degree 3 and all its neighbors have degree 1, and when a has degree 3 and two of its neighbors have degree 1 and one has degree 2. The first case is easily handled in constant time, because we then have a graph component of constant size. For to the second subcase, basically the same strategy as in Case 1 can be applied, because the second degree 1 neighbor of a (besides x) only makes necessary a slight, obvious modification to what is done in Case 1.

5.3 Triangles

In this subsection, we assume that the degree of all vertices is between 2 and 3 and that there is at least one triangle, consisting of the vertices a, b, and c, with weights w, u, and v. We distinguish between three cases.

Case 1: Only one of a, b, c has degree 3. We assume that c has degree 3 and a, b have degree 2. Then it is optimal to put a into the vertex cover if $w \leq u$ and b otherwise. No branching of the recursion occurs.

Case 2: Two of a, b, c have degree 3. We assume that b and c have degree 3 and a has degree 2. Then we branch according to b, which directly gives branching vector $(1, 3)$. If we bring b into the cover, then the degree of a becomes 1 and the degree of its neighbor c becomes 2. Hence, we have a $(2, 3)$-subbranch (see Case 1 in Subsection 5.2) and altogether get branching vector $(1+2, 1+3, 3) = (3, 4, 3)$.

Case 3: a, b, c have degree 3. We branch according to a, where a has minimum weight among a, b, c. If a is not in the cover, all its neighbors, and, particularly, b and c are in the cover. Then, however, b's and c's other neighbors are, without loss of generality, *not* in the cover, since it would be equally good to include a instead of b or c. Hence, all neighbors of a and one neighbor of b's and c's neighbors different from a are in the cover (otherwise $N(a) \cup N(b) \cup N(c)$ were a small component, easily solvable), and that makes at least 4. The branching vector, then, is at least $(1, 4)$.

5.4 No triangles

First, note that the only possible case for the graph being regular could be that the graph is 3-regular, that is, each vertex has exactly three neighbors. Branching once on an arbitrary vertex, however, this situation can never occur again, since afterwards, vertices with degree one or two must always exist. Clearly, this "one-time-branch" plays no role for the asymptotic complexity of our algorithm. (This technique was introduced by Robson [26].) Hence, in the following we may assume that the graph has no triangles, it is not regular, and all vertices have degree 2 and 3. Furthermore, there are no triangles and no vertices whose weight is 2 or more. Before we come to the actual case distinction, we verify the correctness of the last assumption: Assume that there is a weight ≥ 2 vertex x. If x has degree 3, then simply branch on x, yielding branching vector $(2, 3)$ or better. If x has degree 2 and at least one degree-3 neighbor y, then branch on y, resulting in a branching vector of at least $(1, 4)$. Finally, if x has two degree-2 neighbors y and z, whose weight is without loss of generality no bigger than the weight of x, then branch on y. In the case of bringing y into the vertex cover, x becomes a degree-1 vertex and its weight is bigger than the weight of z so, without further branching, we know it is optimal to additionally include z into the vertex cover. Hence, we obtain the branching vector $(1+1,3) = (2,3)$ or better.

Case 1: There is a degree-2 vertex that has a degree-2 vertex as its neighbor. Let x and y be degree-3 vertices that are connected by a path consisting of at least two degree-2 vertices. If $x = y$ then branching on x gives easily a branching vector of at least $(2,3)$, since including x splits off a path. If x and y are neighbors, then we branch according to x: If x is in the cover, then the resulting graph corresponds to Case 1 of Subsection 5.2 with branching vector $(2,3)$. Together with x, the branching vector becomes $(3,4)$. If x is not in the cover, its three neighbors are. Altogether, we get the branching vector $(3,4,3)$.

If x and y are not neighbors and $x \neq y$, then we either bring x and y or $x \cup N(y)$ or $N(x)$ into the cover. We claim a branching vector $(3,4,3)$ or better. The vector's third component is trivial. If x and y are in the cover, then the path between x and y becomes an isolated component and can be handled by dynamic programming as described in Subsection 5.1. At least one more vertex comes into the cover, and so the first component of the branching vector becomes 3. Since x and y are not neighbors, x and the neighbors of y are 4 vertices with minimum weight 4, justifying the second component of the branching vector.

Case 2: Every degree-2 vertex has only degree-3 vertices as neighbors and vice versa. Let x, y, and z be degree-3 vertices such that z is connected to x and, resp. to y, by a degree-2 vertex. We make three branches: $N(x)$, $\{x\} \cup N(y)$, and $\{x, y, z\}$. Then, the branching vector is $(3,4,3)$. The branches cover all possibilities because it is optimal to put z into the cover if it already contains x and y, since then, z has two neighbors with degree 1 and the weight of z is less than 2.

Case 3: The first two cases do not apply. This case is slightly more complicated. First, we make a simple observation: The situation is not hard, if a degree-3 vertex x has a neighbor a that is a degree-2 vertex such that the weight of a is no smaller than the weight of x. If y is a's other neighbor, then we can branch as $N(y)$ and $\{x, y\}$, because if y is in the cover then it is optimal to include x, too. This yields a branching vector $(3,2)$. In the following, we can therefore assume that the weight of a degree-3 vertex is bigger than the weights of all those neighbors that are degree-2 vertices.

Now we can find a degree-3 vertex x that has a neighbor a that is a degree-2 vertex, where a has a neighbor z that is a degree-3 vertex. Furthermore, z has a neighbor y that is again a degree-3 vertex, because, otherwise, this situation would already have been handled by Cases 2 and 1: If the first two cases do not apply, then there must be two neighboring degree 3 vertices. Now look at all vertices that are reachable from these two via a path that consists only of degree 3 vertices. This set contains some z that has, of course, itself degree 3 and has one neighbor that has degree 2 (otherwise, there would be a connected regular component, which is not the case, since we assumed that the graph is connected and not 3-regular). Obviously, z also has a neighbor with degree 3 that we call y. (It must exist: Just follow the path one step backwards.) Call z's degree-2 neighbor a. Call a's other neighbor x. Observe that x has degree 3 since otherwise Case 1 would apply.

Now that we have seen that we can find x, a, z, and y, let b be the third neighbor of z. If b has degree 2, then let c be b's other neighbor. We branch into $N(y)$, $N(z)$, and $\{y, z, x, c\}$. This is correct; since if z is in an optimal cover, it cannot be optimal to include a or b, because the weights of a and b each are smaller than the weight of z, and, therefore, we can safely include x and c. The branching vector is $(3, 3, 4)$ unless x and c are identical. If, however, x and c are identical, then we branch on y: Let us first consider an easy special case, namely $y \in N(x)$. Then, however, bringing y into the cover, we get an isolated component of a cycle of 4 vertices. Hence, we get at least two more vertices into the cover without further branching. In total, we get the branching vector $(1+2, 3)$ for this special case. Now, assume that $y \notin N(x)$. Then we branch into $\{y, x\}$, $\{y\} \cup N(x)$, and $N(y)$. In the first branch, however, we get an isolated component consisting of vertices a, b, and z. Hence, we can determine in constant time which of them (at least one) has to be added to the vertex cover. In total, we get branching vector $(3, 4, 3)$.

If b has degree 3, then we branch into $N(b)$, $\{b\} \cup N(y)$, and $\{b, y, a\}$, resulting in a branching vector $(3, 4, 3)$. If b and y are in the cover, then we can also include a because the weight of a is smaller than the weight of z.

5.5 An application to minimum weight vertex covers with a bound on the number of vertices

So far, we always studied the case that the parameter k bounds the weight of the vertex cover we are searching for. It may be even more natural, however, to look for a minimum weight vertex cover with at most k vertices. This is addressed in the following theorem, which we obtain as a corollary to Theorem 2.

Theorem 3. *Given a graph $G = (V, E)$, a weight function $\omega : V \to [1, \infty)$, and $k \in \mathbf{N}$, is there a vertex cover of minimum weight that consists of at most k vertices? This problem can be solved as fast as Real-WVC, i.e., in time $O(1.3954^k + kn)$.*

Proof. (Sketch) We reduce the stated problem to Real-WVC as follows: Given the graph $G = (V, E)$. Let $w := \sum_{v \in V} \omega(v)$. Then consider the following new weight function: If the original weight of vertex v was $\omega(v)$, then it is assigned the new weight $1 + \omega(v)/w$. It is not hard to see that the original graph has a vertex cover of minimum weight that consists of at most k vertices iff the graph with modified weights has a vertex cover of weight at most $k + 1$. This implies the result. □

6 Conclusion and open questions

In this paper, we contributed to the search for exact solutions for *NP*-hard problems, a field of increasing importance [1, 2, 5, 8, 12, 13, 15, 18, 24, 26, 27]. More precisely, here we continued and extended the research on the fixed parameter complexity of unweighted VERTEX COVER [4, 8, 11, 13, 22, 28] to weighted cases. In this way, we generalize and improve known exact algorithms for one of the most important problems in combinatorial optimization [10].

With regard to future work, it is of particular interest to improve the exponential base for REAL-WVC. Moreover, it remains to give efficient implementations of our algorithms and to evaluate and tune them experimentally. Finally, note that it seems possible to apply a dynamic programming technique of Robson [26] in order to improve our exponential terms (search tree size) somewhat. This requires, however, exponential space, whereas our focus in this paper was to develop efficient fixed parameter algorithms using polynomial space.

Acknowledgement. We are greatful to an anonymous referee of *ISAAC 2000* for her/his insightful remarks.

References

1. J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for Planar Dominating Set and related problems. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, number 1851 in Lecture Notes in Computer Science, pages 97–110, Bergen, Norway, July 2000. Springer-Verlag.
2. J. Alber, J. Gramm, and R. Niedermeier. Faster exact solutions for hard problems: a parameterized point of view. Accepted for *Theoretical Computer Science*, August 2000.
3. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation—Combinatorial Optimization Problems and their Approximability Properties*. Springer-Verlag, 1999.
4. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.
5. N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, number 1741 in Lecture Notes in Computer Science, pages 247–258, Chennai, India, Dec. 1999. Springer-Verlag.
6. R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the Weighted Vertex Cover problem. *Journal of Algorithms*, 2:198–203, 1981.
7. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Disc. Math.*, 25:27–46, 1985.
8. J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, number 1665 in Lecture Notes in Computer Science, pages 313–324, Ascona, Switzerland, June 1999. Springer-Verlag.
9. P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, Aug. 1998.
10. P. Crescenzi and V. Kann. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, 29(4):90–97, 1998.

11. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
12. R. G. Downey and M. R. Fellows. Parameterized complexity after (almost) ten years: Review and open questions. In *Combinatorics, Computation & Logic, DMTCS'99 and CATS'99*, Australian Computer Science Communcations, Volume 21 Number 3, pages 1–33. Springer-Verlag Singapore, 1999.
13. R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 49:49–99, 1999.
14. R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *AMS-DIMACS*, pages 49–99. AMS Press, 1999.
15. H. Fernau and R. Niedermeier. An efficient exact algorithm for Constraint Bipartite Vertex Cover. In *Proceedings of the 24th Conference on Mathematical Foundations of Computer Science*, number 1672 in Lecture Notes in Computer Science, pages 387–397, Szklarska Poreba, Poland, Sept. 1999. Springer-Verlag.
16. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.
17. J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.
18. E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.
19. D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. Boston, MA: PWS Publishing Company, 1997.
20. B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.
21. G. L. Nemhauser and L. E. Trotter, Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.
22. R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.
23. R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.
24. R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36:63–88, 2000.
25. V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, June 1997.
26. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.
27. R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.
28. U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.