On Efficient Fixed-Parameter Algorithms for Weighted Vertex Cover[1]

Rolf Niedermeier[3]

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
E-mail: niedermr@informatik.uni-tuebingen.de

and

Peter Rossmanith

Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80290 München, Fed. Rep. of Germany
E-mail: rossmani@in.tum.de

Received October 2001; revised December 2002

We investigate the fixed-parameter complexity of 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 vertex subset $C \subseteq V$ of total weight at most k such that every edge of G has at least one endpoint in C. WVC and its natural variants are NP-complete. We observe that, when restricting the range of ω to positive integers, the so-called Integer-WVC can be solved as fast as unweighted Vertex Cover. 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 $O(1.3954^k + k|V|)$. By way of contrast, unless $P = NP$, the problem is not fixed-parameter tractable if arbitrary weights > 0 are allowed. Using dynamic programming, at the expense of exponential memory use, we can improve the running time of Real-WVC to $O(1.3788^k + k|V|)$. The same technique applied to a known algorithm yields the so far fastest algorithm for unweighted Vertex Cover, running in time $O(1.2832^k k + k|V|)$.

[1]A preliminary version of this paper was presented at the 11th Annual International Symposium on Algorithms And Computation (ISAAC'00), Springer-Verlag, LNCS 1969, pages 180–191, held in Taipei, Taiwan, December 2000. This conference version, however, contains a faulty application of the main result to the case of minimum weight vertex covers with a bound on the number of vertices.

[3]Work performed within the PEAL project (parameterized complexity and exact algorithms), supported by the Deutsche Forschungsgemeinschaft (NI-369/1).

1. INTRODUCTION

An interesting and challenging open problem in computational complexity theory is related to the polynomial time approximability of (WEIGHTED) VERTEX COVER [5, 11, 20, 29]: 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 [19]. According to Crescenzi and Kann [12], (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 bounds on their running times.

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 [18]. [5] There are linear time algorithms giving approximation factor 2 for the unweighted case [18] as well as for the weighted case [7]. Both results can be improved to an approximation factor that is asymptotically better: $2 - \log\log|V|/2\log|V|$ [8, 23].

The parameterized complexity [13] of UVC recently has received considerable interest [6, 10, 13, 15, 26, 32]. 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 at most 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 [4, 13, 15, 16]. Until now, the best known result in this direction is an $O(1.2852^k + kn)$ algorithm for UVC [10, 21][6]. 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:

[5] More precisely, the decision version of UVC is NP-complete. In the following, we always present algorithms that do not only solve the decision problem but algorithms that output a solution (i.e., a vertex cover) fulfilling the given demands.

[6] The conference version of [10] was flawed, claiming a running time of $O(1.271^k + kn)$. The error was due to a wrong use of the dynamic programming technique of Robson [30]. Correcting this error, the running time is $O(1.2852^k + kn)$ [10, 21].

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.

In the cases of real number we assume a model where real numbers can be added, subtracted, and compared in unit time.

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 $P = NP$.

In more detail, our observations and results are as follows. INTEGER-WVC can be solved as fast as UVC. Our main result is that REAL-WVC can be solved in time $O(1.3954^k + kn)$. Moreover, using a dynamic programming technique developed by Robson [30], we can further improve the running time for REAL-WVC to $O(1.3788^k + kn)$. This increases, however, the memory requirement of the algorithm from polynomial to modestly exponential (which is significantly less than the exponential term 1.3788^k). In particular, this technique applied to an existing algorithm [26] also yields the fastest algorithm for UVC, slightly improving the time bound $O(1.2852^k + kn)$ of Chen *et al.* [10, 21] (requiring polynomial space) to $O(1.2832^k k + kn)$ (requiring modestly exponential space). Conversely, one easily sees that GENERAL-WVC is fixed-parameter intractable unless $P = NP$. Hence, there is little hope to find an $O(f(k) \cdot n^{O(1)})$ time algorithm for GENERAL-WVC, where f may be a function growing arbitrarily fast in the parameter k.

2. PRELIMINARIES AND BASIC NOTATION

We assume familiarity with the basic notions and concepts of algorithms, computational 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, they have the same number of neighbors. The whole paper only deals 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?

Herein, k is considered as the parameter. Our algorithms are based on two key techniques of parameterized complexity [13]: *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 [6, 22, 25, 26]. If the algorithm solves a problem of size k and calls itself recursively for problems of sizes $k - d_1, \ldots, k - d_r$, then $(d_1, \ldots, d_r)$ 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 $d = \max\{d_1, \ldots, d_r\}$. 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 $(d_1, \ldots, d_r)$. Moreover, if α is a single root, then $t_k = O(\alpha^k)$. 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* [13] (also refer to the surveys [4, 14, 15, 16]). 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 $O(1.2852^k + kn)$ [10, 21], 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 algorithm for small values of k. A problem is called *fixed-parameter tractable* if it can be solved in time $f(k) \cdot n^{O(1)}$ for an arbitrary function f which depends only on k. The corresponding complexity class is called *FPT*.[7]

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 preprocessing step—

[7]The choice of what part of the input is considered as the parameter is completely free. Usually, the parameter is a number. Most oftenly, parameter 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.)

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 (cf. [9]) works based on the following [13]: 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 of size k for G.* 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.* [10] observed that using a well-known theorem of Nemhauser and Trotter [24] (also see, e.g., [8, 29]), one can even obtain a problem kernel with a number of vertices linear in k. Nemhauser and Trotter's theorem can be stated as follows (cf. [10] and see [8] for a constructive proof):

PROPOSITION 3.1. *There is an $O(\sqrt{n}m)$ time algorithm that, given a graph $G = (V, E)$ of n vertices and m edges, constructs two disjoint subsets $C_0 \subseteq V$ and $V_0 \subseteq V$ such that*

1.every minimum vertex cover of the subgraph $G[V_0]$ of G induced by V_0 together with C_0 forms a minimum vertex cover for G, and

2.a minimum vertex cover of $G[V_0]$ contains at least $|V_0|/2$ vertices.

In combination with Buss' reduction to problem kernel, Chen *et al.* then showed the construction of a problem kernel containing at most $2k$ vertices (assuming that the given graph has a size k vertex cover) doable in time $O(kn + k^3)$.

Chen *et al.* 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 $O(1.2852^k k + kn)$. Finally, by applying a simple speed-up technique [27] they obtain running time $O(1.2852^k + kn)$. Note, however, that this kind of improvement only requires a polynomial size problem kernel. Recent work shows that *linear* size problem kernels play a major role in obtaining efficient fixed-parameter algorithms for hard problems on planar graphs [1, 2, 3]. The problem kernel size (as long as polynomial in k) requires no special attention in most parts of 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. By way of contrast, however, the *linear* size problem kernel is of central importance in Section 6, where we apply Robson's dynamic programming technique [30].

4. GENERAL– AND INTEGER–WEIGHTED VERTEX COVER

GENERAL-WVC is *NP*-complete for any fixed $k > 0$, and there is a straightforward reduction from the *NP*-complete, unweighted VERTEX COVER to GENERAL-WVC with $k = 1$. This implies, however, that there cannot be a time $f(k) \cdot n^{O(1)}$ 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. Hence, we have:

PROPOSITION 4.1. GENERAL-WVC *is not fixed-parameter tractable unless* $P = NP$.

In the remaining section, we exhibit that we can easily reduce INTEGER-WVC to UVC via a simple "parameterized many-one reduction" (see [13] 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).

PROPOSITION 4.2. INTEGER–WVC *can be solved as fast as* UVC *up to an additive term polynomial in* k.

Proof. If the graph contains vertices whose weight is at least 6, then we branch on such a vertex. The branching number is good enough to compete with the best UVC algorithm (and we can increase the constant 6 to beat better algorithms that might come up in the future, as well). Repeat this, until all weights are smaller than 6.

We can transform the resulting instances of INTEGER–WVC into instances 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. Herein, 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 any vertex 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 $t(k, wn) =$

$O(t(k, kn))$, where $w = O(1)$ is the maximum vertex weight in the given graph. ∎

Chen *et al.* [10] showed that UVC can be solved in time $O(1.2852^k + kn)$. As a consequence, Proposition 4.2 implies that INTEGER-WVC can be solved in the same time. In Section 6, we further improve on this time bound.

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 $O(1.3954^k + kn)$. Observe, however, that our search tree algorithm, as often, can only guarantee to find at least *one* optimal vertex cover, but *not* necessarily all of them. 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): 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 linear time by dynamic programming (see Subsection 5.1).
2. Execute the lowest numbered, applicable step of the following.

(i) 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.)

(ii) 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.)

(iii) 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.)

(iv) 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.)

We emphasize that the subsequent case descriptions and the corresponding branching numbers are based on the implicit assumption that any vertex has weight at least 1. Finally, let us only mention in passing that the clever trick of so-called "folding degree-2 vertices," as described by Chen *et al.* [10] for UVC, does not apply to Weighted Vertex Cover problems. Folding in the unweighted case works as follows. Assume that there is a degree-2-vertex v with two neighbors u and w which are not connected by an edge. Then a folding step replaces v, u, and w by only one new vertex that obtains as neighbors all neighbors of u and w except for v. It can be shown that the original graph has a vertex cover of size k iff the newly constructed graph has a vertex cover of size $k-1$. The correctness proof, however, heavily relies on the fact that v, u, and w all have exactly the same "weight" which is no longer true in the weighted case. Subsequently, we prove our main theorem, following the outline given above.

Theorem 5.1. Real–WVC *can be solved in time* $O(1.3954^k + kn)$.

Because of step 2.(i) above, in order to prove Theorem 5.1 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 or collections thereof. We can find an optimal vertex cover for them in linear 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 expressed 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. We only mention in passing here that such a kind of "bonus point system" where we obtain "easy graph components" that can be handled without branching of the recursion was already used in designing an efficient fixed-parameter algorithm for the so-called "Constraint Bipartite Vertex Cover" problem from reconfigurable VLSI [17].

Case 2: a has degree 3 and it 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 it 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 in the vertex cover. 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, interchange the roles of y and b. In this way, we get a branching vector of at least $(1, 4)$ (unless the component has only six 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 applies, 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 shall have 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 graph 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 observation was first made by Robson [30].) 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 vertices whose weight is 2 or more. Recall that the values of the branching vectors depend on the weight of the vertices, rather than on their number. 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. Because branching vectors $(2, 3)$ and $(1, 4)$ are better than our worst-case branching vector $(3, 4, 3)$, we can henceforth assume that there is no vertex with weight 2 or more.

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 trivially obtained. 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 (without branching) 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 four 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 *not* 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 3-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 y and z are 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. Observe that $x = y$ is impossible because of the assumption "no triangles." 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 four 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 have to add z 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. Moreover, due to the assumption "no triangles" we know that $b \notin N(y)$.

6. IMPROVED RUNNING TIMES BY DYNAMIC PROGRAMMING

The algorithms presented so far all have exponential running times, but use only a polynomial amount of space. Robson introduced the idea to improve the running time by dynamic programming [30]: Choose a size s and store all induced subgraphs of the input graph G of size s in a database D. Solve all instances in D and store an optimal solution for each of them. Then apply the search tree algorithms of Section 4 or 5 to G. These algorithms find an optimal solution for G by recursively computing optimal solutions for induced subgraphs of G: The only operation to reduce the size of a graph is the deletion of vertices. Normally, the size of the graph is reduced further in the branches of the search tree until the sizes of the graphs in the leaves reaches 0. Having the database D at disposal, the branching can stop earlier: As soon as the size of the graphs in the nodes of the search tree are as small as s, a dictionary lookup replaces the remaining part of the search tree. The size of the search trees is in this way reduced from $O(c^n)$ to $O(c^{n-s})$.

In our case, we can store all induced subgraphs of size up to $s = \alpha k$ in the database. The ratio between s and k is called α and we will choose α rather than s directly. Due to the problem kernel of size $2k$ we can be sure that G has at most $2k$ vertices when the search tree algorithm starts.

The fastest fixed-parameter algorithm for (unweighted) VERTEX COVER that uses only polynomial space takes time $O(1.2852^k + kn)$ [10]. Unfortunately, it does not seem possible to apply dynamic programming to this algorithm as attempted in the conference version of [10] since this algorithm uses a technique, called "folding" by the authors, that contracts edges. This leads to graphs that are not induced subgraphs of the original instance.

In a recent improvement of his algorithm, Robson had to use very complicated case distinctions to be able to avoid having to use foldings—otherwise he would not be able to apply dynamic programming [personal communication] (see [31] for the fastest algorithm for the independent set problem). The second fastest polynomial-size fixed-parameter algorithm for (unweighted) VERTEX COVER [26] (see [25] for a complete version of the paper) does not use foldings and applying dynamic programming to it results in the fastest fixed-parameter algorithm for VERTEX COVER as of today. The resulting running time is $O(1.2832^k k + kn)$. Since Theorem 4.2 states that INTEGER-WVC can be solved equally fast, this will prove the following theorem.

THEOREM 6.1. INTEGER-WVC *can be solved in time* $O(1.2832^k k + kn)$.

Our algorithm for REAL-WVC creates only induced subgraphs in each search tree, too, and is therefore a candidate for dynamic programming. The resulting running time will be $O(1.3788^k k + kn)$.

THEOREM 6.2. REAL-WVC *can be solved in time* $O(1.3788^k k + kn)$.

In the remainder of this section, we explain the dynamic programming algorithm in detail because Robson's technique cannot be directly applied to parameterized problems. Instead of pruning the search tree and look up the optimal vertex cover of the remaining graph in a database when the *size* of the graph drops below some predetermined size, we do the same when the *parameter* reaches or drops below $\alpha k/2$, where k is the initial parameter.

Let $\hat{T}(k)$ be the running time of the search tree algorithm and let $T(k)$ be the time for the new algorithm that combines search trees with dynamic programming. The new algorithm works as follows:

1. Build a dictionary of all induced subgraphs of up to αk vertices.[8] Compute (using the same algorithm recursively) optimal solutions for all of them and store them in a database if the size of the optimal solution is at most $\alpha k/2$. This takes time $O\big(T(\alpha k/2)\binom{2k}{\alpha k}\big)$ because there are at most $2\binom{2k}{\alpha k}$ induced subgraphs of size up to αk as long as $\alpha < 1/2$ and it takes only $O(T(\alpha k/2))$ time to find and store a solution of size up to $\alpha k/2$—if the optimal solution is bigger we can therefore stop after $O(T(\alpha k/2))$ steps.

2. Apply the search tree algorithm to G. If the parameter in a branch reaches $\alpha k/2$, look up an optimal solution in the database. This works because the size of the graph in question can be at most αk (implied by the problem kernel size) and is therefore stored in the database if it has a vertex cover of up to $\alpha k/2$ vertices. The resulting search tree has size $\hat{T}(k - \alpha k/2)$.

To optimize the running time of this algorithm it is crucial to choose αk wisely. Increasing α makes the database bigger and increases the time to build it. Decreasing α increases the time for the search tree algorithm as the size of the search tree grows.

The running time of this algorithm is

$$\begin{aligned} T(k) &\leq \hat{T}(k - \alpha k/2) \cdot O(k) + O\left(T(\alpha k/2) \cdot \binom{2k}{\alpha k}\right) \\ &= \hat{T}(k - \alpha k/2) \cdot O(k) + T(\alpha k/2) \cdot O\left(k^{-1/2}\Big(\frac{4}{\alpha^\alpha (2-\alpha)^{2-\alpha}}\Big)^k\right), \end{aligned}$$

where the last O-term is obtained using Stirling's formula and the extra $O(k)$ factor is the time needed to look up a graph in the database as well as for the work done by the branching algorithm in each node of the search tree. This factor cannot be avoided by the interleaving technique of [27] because the technique is roughly based on the fact that the work done near the leaves in a search tree is asymptotically dominating as nearly all nodes are near the leaves *and* that the size of the graphs processed in those nodes is very small. Now their size is not small, but up to αk vertices big.

We have to choose α such that both terms in the above recurrence, $\hat{T}(k - \alpha k/2)$ and $T(\alpha k/2)\big(4/(\alpha^\alpha(2-\alpha)^{2-\alpha})\big)^k$, have the same size up to a constant factor since this yields the smallest running time possible, and that $\alpha k/2$ is integral. The resulting running time is then $T(k) \leq \hat{T}(k - \alpha k/2) \cdot O(k)$. For unweighted Vertex Cover $\hat{T}(k) = O(1.29175^k + kn)$ [26]

[8] More precisely, this means that the dictionary contains all labeled induced subgraphs, where the labels are taken from the encoding of the input graph.

and we choose $\alpha = 0.05245457 + O(1/k)$, which minimizes the running time. This results in $T(k) = O(1.2831961^k k + kn)$. For Real-WVC $\hat{T}(k) = O(1.3954^k + kn)$ and we choose $\alpha = 0.07188661 + O(1/k)$. This results in $T(k) = O(1.3788^k k + kn)$.

It is finally worthwile to note that using exponential space is a hard price to pay for getting an improved running time. In this case, this bad news is relaxed by the fact that the space used is exponentially smaller than the running time (not as in many cases only polynomially smaller): For unweighted Vertex Cover we use $O(1.275^k k + n^2)$ space and for Real-WVC we use $O(1.363^k k + n^2)$ space.

7. CONCLUSION

In this paper, we contributed to the search for exact fixed-parameter solutions for *NP*-hard problems, a field of increasing importance [4, 15, 16]. More precisely, here we continued and extended the research on the fixed-parameter complexity of Vertex Cover [6, 10, 13, 15, 26, 32]. In this way, we generalized and improved known exact algorithms for one of the most important problems in combinatorial optimization [12]. It would be interesting to perform similar studies of weighted Vertex Cover problems on hypergraphs, where edges may connect more than two vertices. A fixed-parameter algorithm for the case of three vertices per edge, also known as 3-Hitting Set, has recently been obtained for the unweighted case [28]. There is also an easy generalization to d-Hitting Set for $d > 3$. So far, we focussed on the case that the parameter k bounds the weight of the vertex cover we are searching for. Finally, to demonstrate the practical usefulness of the achieved algorithms through an efficient implementation in combination with extensive experimentation on realistic input data is a future task.

Acknowledgement. We are grateful to an anonymous referee of *Journal of Algorithms* for pointing out an error in a preliminary version and for numerous constructive comments that helped improving the presentation of the paper significantly.

REFERENCES

1. J. Alber, M. R. Fellows, and R. Niedermeier. Efficient data reduction for Dominating Set: a linear problem kernel for the planar case. In *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory (SWAT)*, number 2368 in Lecture Notes in Computer Science, pages 150–159. Springer-Verlag, July 2002.

2. J. Alber, H. Fernau, and R. Niedermeier. Graph separators: a parameterized view. In *Proceedings of the 7th Annual International Computing and Combinatorics Conference (COCOON)*, number 2108 in Lecture Notes in Computer Science, pages 318–327. Springer-Verlag, 2001. Long version accepted by *Journal of Computer and System Sciences*.

3. J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming (ICALP)*, number 2076 in Lecture Notes in Computer Science, pages 261–272. Springer-Verlag, 2001.

4. J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics*, 229(1–3):3–27, 2001.

5. 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.

6. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

7. R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the Weighted Vertex Cover problem. *Journal of Algorithms*, 2:198–203, 1981.

8. 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.

9. J. F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993.

10. J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001. A preliminary version appeared at *WG'99*, LNCS 1665, pages 313–324, Springer-Verlag 1999.

11. P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, August 1998.

12. 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.

13. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

14. 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.

15. 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.

16. M. R. Fellows. Parameterized complexity: the main ideas and some research frontiers. In *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC)*, number 2223 in Lecture Notes in Computer Science, pages 291–307. Springer-Verlag, 2001.

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

18. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

19. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.

20. D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. Boston, MA: PWS Publishing Company, 1997.

21. I. A. Kanj. *Vertex Cover: Exact and Approximation Algorithms and Applications*. PhD thesis, Texas A&M University, Department of Computer Science, August 2001.

22. O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

23. B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.

24. G. L. Nemhauser and L. E. Trotter, Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

25. R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover further improved. Technical Report KAM-DIMATIA Series 98-411, Faculty of Mathematics and Physics, Charles University, Prague, November 1998.

26. R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.

27. R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

28. R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 2(1):93–107, 2002.

29. V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, June 1997.

30. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

31. J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, Université Bordeaux I, LaBRI, 2001.

32. U. Stege and M. R. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.