Upper Bounds for Vertex Cover Further Improved*

Rolf Niedermeier
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
`niedermr@informatik.uni-tuebingen.de`†

Peter Rossmanith
Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80290 München, Fed. Rep. of Germany
`rossmani@in.tum.de`

*An extended abstract of this work appears in the proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99), Springer, *LNCS*, held in Trier, Fed. Rep. of Germany, March 4–6, 1999.

†Supported by a Feodor Lynen fellowship of the Alexander von Humboldt-Stiftung, Bonn, and the Center for Discrete Mathematics, Theoretical Computer Science and Applications (DIMATIA), Prague.

Abstract

The problem instance of Vertex Cover consists of an undirected graph $G = (V, E)$ and a positive integer k, the question is whether there exists a subset $C \subseteq V$ of vertices such that each edge in E has at least one of its endpoints in C with $|C| \leq k$. We improve two recent worst case upper bounds for Vertex Cover. First, Balasubramanian *et al.* showed that Vertex Cover can be solved in time $O(kn + 1.32472^k k^2)$, where n is the number of vertices in G. Afterwards, Downey *et al.* improved this to $O(kn + 1.31951^k k^2)$. Bringing the exponential base significantly below 1.3, we present the new upper bound $O(kn + 1.29175^k k^2)$. We also show how to modify the algorithm to get $O(kn + 1.29175^k)$ as an upper bound, a technique that is also applicable to the two preceeding algorithms mentioned above.

1 Introduction

In recent time, there has been increased interest and progress in lowering the exponential running times of algorithms for fundamental *NP*-complete problems as, for example, Satisfiability [18, 19, 28, 29], Maximum Satisfiability [20, 25], and Vertex Cover [3, 12]. Even small improvements in the bases of the exponential terms are of great interest (cf., e.g., Hirsch [18] for Satisfiability and Downey *et al.* [12] for Vertex Cover). For instance, for the satisfiability of a propositional formula consisting of K clauses, Hirsch [18] improved the upper bound of order (omitting polynomial terms) 1.25993^K of Monien and Speckenmeyer [22] to 1.23883^K. With respect to the length L of the input formula, he improved Kullmann and Luckhardt's [19] upper bound from 1.08006^L to 1.07578^L. Efficient exact algorithms with proven performance bounds are of interest for theoretical and practical reasons. They often build the heart of heuristic algorithms with good average case performance [12, 24].

Vertex Cover is a problem of central importance in computer science:

- It was among the first *NP*-complete problems [15].
- There have been numerous efforts to design efficient approximation algorithms [6], but it is also known to be hard to approximate [2].
- It is of central importance in parameterized complexity theory and has one of the most efficient *fixed parameter*

algorithms [11], which is also subject of [3, 12] and this paper.

- It has important applications, e.g., in computational biochemistry, where it is used to resolve conflicts between sequences by excluding some of them from a sample and, for this reason, the algorithm of Balasubramanian *et al.* [3] has been implemented as part of the DARWIN project at ETH Zürich [12, 16]. In particular, exact algorithms are important here.

An instance of Vertex Cover is an undirected graph $G = (V, E)$ and a positive integer k. The question is whether there exists a *vertex cover set* $C \subseteq V$ with $|C| \leq k$ such that for all edges (u, v) in E, it holds that $u \in C$ or $v \in C$. Vertex Cover, sometimes called Node Cover, is NP-complete. A straightforward greedy algorithm shows that Vertex Cover is approximable to a ratio 1 (cf. [26]), that is, the greedy algorithm always finds a vertex cover of size at most twice as large as as the optimal one. The simple idea behind the greedy algorithm is to pick any edge from the graph, put both endpoints in the vertex cover, and delete these endpoints together with their incident edges from the graph. Surprisingly, this is very close to the best known bound, which gives a ratio $1 - \log\log|V|/(2\log|V|)$ approximation [4, 6, 23]. However, unless $P = NP$, Vertex Cover has no polynomial time approximation scheme [2] and it is known to be *not* approximable to a ratio 0.1666 [17].

Although Vertex Cover is hard to approximate, it has turned out that it is "easy to parameterize": Vertex Cover has seen quite some history of progress with respect to fixed parameter algorithms ("fixed parameter" refers to k, see [11, 24] for details). One of the first results (of mainly theoretical interest) showing its fixed parameter tractability was based on Robertson and Seymour's theory of graph minors leading to an $O(n^3)$ algorithm for constant k and $n = |V|$ [14]. Even a linear time algorithm followed, because graphs with "bounded vertex cover" have bounded treewidth [5]. However, more efficient algorithms based on techniques as *bounded search tree* and *reduction to problem kernel* have been obtained, yielding running time $O(kn + 2^k k^2)$ [10]. Using maximum matching as a subroutine, Papadimitriou and Yannakakis [27] showed that Vertex Cover admits a polynomial time solution whenever the cover size is $O(\log n)$. Surprisingly, in essence, all this already follows from the elementary search tree method described in Mehlhorn's text book on graph algorithms [21, page 216], published before all of the above-mentioned papers. Recently, Balasubramanian *et al.* [3] came up with a greatly improved fixed parameter algorithm for Vertex Cover, running in time $O(kn + 1.324718^k k^2)$. They employ an intricate, improved search tree algorithm. Very recently, this result was slightly improved to $O(kn+1.31951^k k^2)$ by Downey *et al.* [12]. Note that according to the authors this "tiny difference amounts to a 21% improvement in the running time for $k = 60$."

In the following we prove a better upper bound of $O(kn + 1.29175^k k^2)$, thus breaking the 1.3 barrier in the base of the exponential term. Adopting the above example for $k = 60$, our new result means an improvement of 78% to the result of Balasubramanian *et al.*

There is a strong relation between Vertex Cover and Independent Set. A graph has a vertex cover of size k iff it has an independent set of size $n-k$. The fastest known algorithm to compute maximum independent sets [31] takes $O(1.211^n)$ steps and can be used to solve our problem in $O(kn + k^2 1.211^{n-k}$ steps. This is faster than our algorithm if k is very big.

2 Preliminaries and basic notation

Let $G = (V, E)$ be an undirected graph. A set $C \subseteq V$ is a *vertex cover* of G if for every edge $(i, j) \in E$ either $i \in C$ or $j \in C$ or both $i, j \in C$. With other words, the vertices in C cover all edges of G. A vertex cover is *minimal* or *optimal* if it has minimum size, i.e., if there is no vertex cover that has less vertices. Since $\emptyset \subseteq C \subseteq V$ holds for every vertex cover, a minimal cover exists though it needs not to be unique.

By $N(x)$ we denote the set of neighbors, i.e., adjacent vertices, of a vertex x. For the ease of notation, we often write $\{x, N(y)\}$ instead of $\{x\} \cup N(y)$ or $N(\{x, y, \})$ instead of $N(x) \cup N(y)$ to denote sets of vertices. A graph is called *r-regular* if every vertex has degree r; it is called *regular* if it is r-regular for some r. A graph is *connected* if there is a path between each pair of vertices. A *component* of a graph is a maximal connected subgraph. Three vertices a, b, c are a *bridge* of a vertex x if x, a, b, c form a cycle (a closed path). We say b is a *bridge vertex* of x. A cycle of length 3 is a *triangle*.

Our algorithm works recursively. The number of recursions is the number of nodes in the according tree. This number is governed by homogeneous, linear recurrences with constant coefficients. It is well known how to solve them and the asymptotic solution is determined by the roots of the characteristic polynomial. We use the same notation as Kullmann and Luckhardt [19]. If the algorithm solves a problem of size n and calls itself recursively for problems of sizes $n - d_1, \ldots, n - d_k$, then $(d_1, \ldots, d_k)$ is called the *branching vector* of this recursion. It corresponds to the recurrence

$$t_n = t_{n-d_1} + \cdots + t_{n-d_k}. \tag{1}$$

The characteristic polynomial of this recurrence is

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

where $d = \max\{d_1, \ldots, d_k\}$. If α is a root of (2) with maximum absolute value, then t_n is α^n up to a polynomial factor. We call $|\alpha|$ the *branching number* that corresponds to the branching vector $(d_1, \ldots, d_2)$. (In our case α is always real, since $1/\alpha$ is the dominant singularity of the series $\sum t_n z^n$. The dominant singularity of a series with non-negative coefficients is always a positive, real number.) Moreover, if α is a single root, then even $t_n = O(\alpha^n)$ and all branching numbers that will occur in this paper are single roots.

In this paper, the size of the search tree is therefore $O(\alpha^k)$, where k is the parameter and α is the biggest branching number that will occur; it is about 1.291742754 and belongs to the branching vector $(3, 5, 8, 8)$ occuring in Section 5 (Case 5.2, see Figure 1 for all occuring branching vectors and their branching numbers).

branching vector	*branching number*	*cases*
$(3,5,7)$	1.263739	5.5.2
$(3,3)$	1.269921	4.3, 4.4, 5.1, 5.2, 5.3
$(2,4)$	1.272020	4.2
$(3,7,7,7)$	1.282433	5.5.1
$(3,4,7)$	1.288453	5.4
$(4,4,5)$	1.290649	6.1, 6.3
$(4,5,8,8,9)$	1.290649	6.2
$(3,5,8,8)$	1.291743	5.5.2

Figure 1: Branching vectors and their corresponding branching numbers. The third column contains all places, where the branching vectors occur. The first digit is the section, followed by the subcase.

Finally, without going into details, let us briefly say a few words about parameterized complexity theory [11]. Parameterized complexity, as mainly developed by Downey and Fellows [1, 7, 8, 9, 10, 11], 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 restrained to a "small part" of the input, the parameter. So, for instance, the Vertex Cover problem can be solved by an algorithm with running time $O(kn + 1.32472^k k^2)$ [3], 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 that depends only on k. Unfortunately, this $f(k)$ is usually not so small as in the case of Vertex Cover, but grows much faster (e.g., $f(k) = 11^k$ for Planar Dominating Set [10] is still harmless), making the algorithm impractical already for small values of k. This might be one of the main deficiencies of parameterized complexity theory. As Downey *et al.* [12] put it, "the extent to which FPT is really useful is unclear." So far, there are only few examples of problems that are fixed parameter tractable and possess algorithms of comparable efficiency as that for Vertex Cover. To conclude, we mention *klam* values, introduced by Downey and Fellows [11] to evaluate the efficiency of fixed parameter algorithms: The *klam* value of an algorithm A solving a problem L is defined to be the largest k such that

1. L can be solved by A in time $f(k) + n^{O(1)}$ and
2. $f(k) \leq U$, where U is some reasonable absolute bound on the maximum number of steps of any computation, e.g., $U = 10^{20}$.

For example, using $U = 10^{20}$, the klam value for Vertex Cover derived from [3] is about 129, improved to 131 by Downey *et al.* [12]. With the help of our new algorithm, the klam value reaches 141.

3 General outline of the algorithm

Our algorithm works, in essence, as all previous algorithms for Vertex Cover. The main part is to build a *bounded search tree*: To cover an edge, we have to put at least one of its two endpoints into the (optimal) vertex cover set. Thus, starting with an arbitrary edge, we can make a binary decision between its two endpoints. In each subcase, we delete the corresponding vertex chosen and its incident edges and repeat this until we have built a search tree of size 2^k. Altogether, it is easy to see that this leads to an algorithm running in time $O(2^k n)$ [10, 11, 24, 30], where n denotes the number of vertices in the graph. All results (including ours) to get more efficient algorithms are based on efforts to make the search tree smaller. So, Balasubramanian *et al.* [3] presented an algorithm with search tree size 1.32472^k and this was improved to 1.31951^k by Downey *et al.* [12]. We further improve this size to 1.29175^k.

Before giving an overview of our approach, we still have to explain briefly a technique called *reduction to problem kernel*, which is a kind of preprocessing. The main idea is that vertices of degree $> k$ *must* be part of a vertex cover, if its size is at most k. Deleting all those edges leaves a graph, which can still be very big. If, however, it is connected and bigger than $2k^2$ then there cannot exist a vertex cover of size k since there are more than k^2 edges. Hence, after reduction to problem kernel we can assume that the size of the graph is at most $2k^2$.

In parameterized complexity theory the resulting algorithm is known as *Buss' algorithm* [11], but basically the same approach can be traced back to older ideas from VLSI–theory, e.g., Evans [13]. It is not difficult to see, using appropriate subalgorithms, that Buss' algorithm has running time $O(kn+(2k^2)^k k^2)$. Combining reduction to problem kernel with the search tree algorithm described before, we get easily an $O(kn + 2^k k^2)$ algorithm for Vertex Cover. All subsequent improvements concentrated on replacing the exponential term 2^k by a smaller one.

The algorithm we describe is closer in spirit to the one of Balasubramanian *et al.* [3] than to that of Downey *et al* [12]. The main difference between both approaches is that Downey *et al.* employ a different reduction to problem kernel, which not only works as preprocessing, but is also applied during the search tree construction. They handle vertices of degree 1, 2, 3 and of degree greater than k by reduction to problem kernel, so that the search tree takes care only of vertices with degree between 4 and k (the difficult cases being vertices of degree 4 and 5). We refer to Downey *et al.* [12] for details. By way of contrast, Balasubramanian *et al.* [3] and we use the "more classical approach" where the search tree deals also with vertices of degree 2 and 3 and reduction to problem kernel is only applied once as a preprocessing phase. In the rest of the paper, we concentrate on the search tree size.

3.1 Overall structure of the new search tree algorithm

The algorithms finds recursively an optimal vertex cover as follows. Given a graph G, we choose several subgraphs $G_1,\dots,G_k$ and compute optimal vertex covers for all of them. From them we can construct an optimal vertex cover for G. For example, let x be some vertex of G and let G_1 be the subgraph that results from G by deleting x and all incident edges. A vertex cover of G_1, together with x, is then a vertex cover of G. Moreover, if there is an optimal vertex cover for G that contains x, then we can construct an optimal vertex cover from an optimal vertex cover of G_1. Otherwise, if no optimal vertex cover of G contains x, they must contain all neighbors of x. Hence, let G_2 be the graph that results from G by deleting all neighbors of x. Again, we can construct an vertex cover of G by taking a vertex cover of G_2 and adding all neighbors of x. If we start from optimal vertex covers for G_1 and G_2, then one of the resulting covers for G must be optimal, since either x or its neighbors must be part of any vertex cover. We say we *branch according to x and $N(x)$*, where $N(x)$ denotes the neighbors of x. In the first branch, x will be part of the vertex cover and in the second branch it will be $N(x)$. The vertex cover constructed grows in size with each step. Since its size cannot exceed k, the goal, the algorithm terminates.

In principle that is the way our algorithm works, but we choose the subgraphs $G_1, \ldots, G_k$ in a more complicated way and branch according to much more complicated sets. The rules how to choose those branching sets are as follows, provided the graph is connected:

1. If there is a vertex x with degree 1, then branch according to $N(x)$ (and nothing else). There is no other branch, since there is always an optimal vertex cover that contains $N(x)$ and does not contain x.

2. If there is a vertex x with degree 6 or more, then branch according to x and $N(x)$.

3. If there are no vertices with degree 1 or at least 6, but there is a vertex with degree 2, then proceed as shown in Section 4.

4. If 1.–3. do not apply and if the graph is regular, then choose some vertex x with maximum degree and branch according to x and $N(x)$. (This can happen at most three times in each path of the search tree and increases its size at most by a small constant factor.)

5. If 1.–4. do not apply and if there is a vertex with degree 3 then proceed as shown in Section 5.

6. Otherwise, there must be a vertex with degree 4 and all other vertices have degrees between 4 and 5. Proceed as shown in Section 6.

If the graph is not connected, then the algorithm chooses some component G' and tests recursively if G' has a vertex cover of size k or less and, if it has, finds out the optimal size k' of a vertex cover for G'. Then it proceeds to test if $G-G'$, the other components, have a vertex cover of size $k-k'$. In this way the algorithm finds out whether the whole graph has a vertex cover of size k.

3.2 A further improvement

Since the size of the search tree is $O(1.291743^k$, each recursive call can be processed in linear time, and the size of the graphs is $O(k^2)$ (see Section 3), the total running time of the search tree part of our algorithm is $O(1.291743^k k^2)$. Since $1.291743^k k^2 = O(1.29175^k)$ we could claim that the running time is indeed $O(1.29175^k)$, which is technically correct, but constitutes a misuse of big–O–notation: It is based on $k^2 = O(1.000005419^k)$ and there is a huge hidden constant.

There is, however, another way to get rid of the factor k^2: By reduction to problem kernel we can always get a graph whose size is quadratic in the parameter. The parameter is k initially, but gets smaller when we approach the leaves in the search tree. Near the leaves the parameter is small and, hence, the graph to be processed is small, too. Nearly *all* nodes are near the leaves and we can therefore expect that the running time is the size of the search tree times a *small constant*. To make this argument rigorous we analyze the running time directly instead of the size of the search tree. Then we no longer have a homogeneous recurrence, but we get an inhomogenity of the form $c \cdot k^2$, where c

is some constant. We have already seen that all solutions of the homogeneous recurrence are bounded by $O(\alpha^k)$. All solutions of the inhomogeneous recurrence are the sums of all general solutions of the homogeneous recurrence and one special solution of the inhomogeneous recurrence. Since the inhomogenity is a quadratic polynomial, there is usually a special solution that is also a quadratic polynomial (for the branching vector $(2, 4)$, e.g., this solution would be $-c(k^2 + 12k + 52)$). The running time is therefore $O(k^2 + \alpha^k)$ for the search tree part of the algorithm if we apply reduction to problem kernel before each recursive call. The constants are *not huge*, as we expected from the informal argument above. Note that we have to modify the algorithm such that reduction to problem kernel is performed after each recursive call rather than once in the beginning.

4 Degree-2-vertices

If the graph is 2-regular (and connected), all vertices constitute a cycle and it is very easy to construct an optimal cover in linear time. Otherwise let x be a vertex with degree 2 and a, b its neighbors, where a has degree ≥ 3. The algorithm chooses the first of the following four cases that applies.

Case 1.

There is an edge between a and b or x has a bridge whose bridge vector has degree 2. Then include $\{a, b\}$ into the vertex cover, which is optimal. No branching is necessary.

Case 2.

Assume that $|N(a) \cup N(b)| \geq 4$. Then branch according to $\{a, b\}$ and $N(a) \cup N(b)$, whose branching vector is at least $(2, 4)$.

Case 3.

Assume that x has exactly one bridge. Then a's degree must be 3 and b's degree must be 2. Otherwise $|N(a) \cup N(b)| \geq 4$. Then there is an optimal cover that does not contain both a and y, the bridge vertex: If a and y are part of an optimal vertex cover then we can assume that x is also in the cover (but b is not). Replacing a by $N(a)$ produces another vertex cover that is not bigger. Hence, we branch according to $N(y)$ and $N(a)$. The branching vector is at least $(3, 3)$.

Case 4.

Finally, let x have two bridges. Then the degrees of both a and b must be 3 since otherwise $|N(a) \cup N(b)| \geq 4$. Let y and z be the bridge vertices. We can branch according to y and $N(y)$. If y is in an optimal cover, including y and z, but not a or b is optimal, since two further vertices are necessary anyways to cover all incident edges of a and b. Hence, we can branch according to $N(y)$ and $\{x, y, z\}$ with a branching vector at least $(3, 3)$.

All possibilities are covered by these four cases. In particular, if there are more than 2 bridges, Case 2 applies.

5 Degree-3-vertices

In this section, the graph can contain vertices with degrees between 3 and 5. Particularly there must be at least one vertex with degree 3.

For Cases 1, 2, 3, and 4 let x be such a vertex and let a, b, and c be its neighbors. The first four cases distinguish on the structure of the subgraph around x, in particular on the degree of its neighbors and whether x has triangles or bridges. Case 5 is different, it rather assumes that no vertices exist in the whole graph, for which one of the first four cases applies.

Case 1.

Assume that x is part of a triangle, e.g., let $\{x, a, b\}$ be the triangle (but there can be more triangles). Then we can branch according to $N(x)$ and $N(c)$. If x is not part of the cover, $N(x)$ is. If x is part of the cover, then is a or b. If c is also in the cover, then two neighbors of x are and we can replace x by $N(x)$. The branching vector is at least $(3, 3)$.

Case 2.

Assume that x has at least two bridges (separate ones or a double bridge). Let y and z be the middle vertices on the bridges. We can branch according to $N(x)$ and $\{x, y, z\}$. If x is in the cover then we can assume that y and z are in the cover, too: Assume y is not in the cover. Then all neighbors of y must be in the cover and among them are two neighbors of x. Hence, we could replace x by $N(x)$, which is already considered in the first branch. The branching vector is at least $(3, 3)$.

Case 3.

Next, assume that x has exactly one bridge, let us say between a and b. Call the center vertex on the bridge again y. Let us further assume that a or b has degree 3, without loss of generality a. Then we branch according to $N(x)$ and $N(a)$. These two possibilities are sufficient: If x *is* in the cover and y is not, then $N(y)$ is. Hence two neighbors of x are in the cover and we could replace x by $N(x)$. So, we can assume that if x in an optimal cover, y is in it, too. Then, however, we can assume that a is not in the cover. The branching vector is at least $(3, 3)$.

Case 4.

Now assume again that x has exactly one bridge as in the case above, but both a and b have degrees of at least 4. Then we can branch according to $N(x)$, $N(a)$, and $\{a, x, N(b), N(c)\}$. The last branch contains besides x and a also all neighbors of b and c. This is because we can assume that neither b nor c are in the cover: If b or c is in the cover x will have two neighbors that are in the cover and can be replaced by $N(x)$. Since we can assume that x is not part of a triangle and there is exactly one bridge, we get the branching vector $(3, 4, 7)$.

Case 5.

Finally, we can assume that there is no vertex with degree 3 that has a bridge or a triangle.

Case 5.1. Assume that there is a vertex x with degree 3 and neighbors a, b, c, two of which have degree at least 4, say, a and b. We pick either $N(x)$ or $\{x, N(a), N(b)\}$ or $\{x, a, N(b), N(c)\}$ or $\{x, b, N(a), N(c)\}$, using that, in order to get an optimal vertex cover, at most one of the neighbors can be chosen together with x. This yields the branching vector $(3, 7, 7, 7)$.

Case 5.2. Otherwise, we can assume that each degree 3 vertex has at most one neighbor with degree ≥ 4. We assumed further in this section that the graph is not regular and has at least one vertex with degree 3. Since the graph is connected there must be some vertex with degree 3 that has exactly one neighbor with degree 4 or 5.

Case 5.2.1 Let us assume that there is no cycle of length 5 with the following two properties: (1) each vertex on the cycle has degree 3 and (2) there is a vertex on the cycle that has a neighbor with degree at least 4. We choose some vertex with degree 3 that has a neighbor with degree 4 or 5. Call this vertex a_3 and the neighbor b_3. The other two neighbor of a_3 must have degree 3. From each vertex with degree 3 we can inductively follow some path that consists solely of degree–3 vertices: Just choose a neighbor with degree 3, but not that one you came from. Start such a path from a_3 and call the vertices a_2, a_1, a_0.

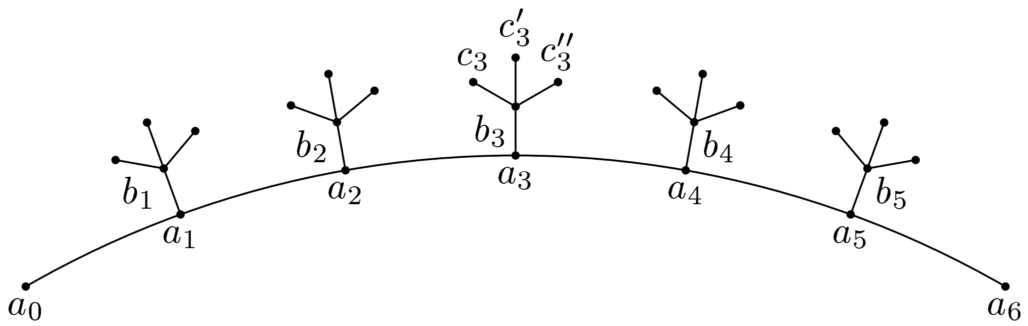

Figure 2: No degree–3 cycle of length 5 (Case 5.2.1)

Start another path and call the vertices a_4, a_5, a_6. Each of the a_i has at least two neighbors with degree 3, i.e., a_{i-1} and a_{i+1}. The third neighbor is called b_i and might have degree 3, 4, or 5.

Figure 2 shows the resulting part of the graph. This picture does not necessarily denote a subgraph of G: Firstly, all vertices b_i are shown with degree 4, but some of them might also have degree 3 or 5. On the other hand, we know that all a_i have *exactly* degree 3. Secondly, not all vertices shown in picture must be necessarily distinct. For example, we could have $b_1 = a_4$ (then b_1 would have degree 3), since it does not violate any assumptions we made for this subcase. The picture in Figure 2 is therefore merely a sceletal structure leaving open many details. The freedom of these details lays in variation of degree of the b_i's and pairs of vertices being identical.

Our algorithm's behavior must depend on these details, but mostly we branch according to

1. $\{a_2, b_3, a_4\}$,

2. $\{a_1, b_2, a_3, b_4, a_5\}$,
3. $\{a_1, b_2, N(b_3), N(b_4), b_5, a_6\}$, and
4. $\{a_0, b_1, N(b_2), N(b_3), b_4, a_5\}$,

which can be found marked in Figure 3. The correctness is seen as follows: The first branch handles the case that a_3 is not in the cover, the remaining branches that it is. The second branch assumes that a_2 and a_4 are not in the cover. The third branch assumes that a_2 is not, but a_4 is in the cover. We can then further assume that b_2 is not in the cover, otherwise there is another optimal cover that contains $N(a_3)$ instead of a_3. Moreover, we can assume that also b_4 and a_5 are not part of the cover, since otherwise we can replace a_4 by $N(a_4)$, which is then handled by the second branch. Hence, the neighbors of b_3, b_4, and a_5 are in the cover and we get altogether $\{a_1, b_2, N(b_3), N(b_4), b_5, a_6\}$. The third and fourth branches are symmetric.

The resulting branching vector is $(3, 5, n_1, n_2)$. Clearly a_2, b_3, and a_4 are pairwise distinct. Furthermore $b_2 \neq b_4$, $a_1 \neq b_4$, $b_2 \neq a_5$ (otherwise a_3 has a bridge) making also a_1, b_2, a_3, b_4, a_5 pairwise distinct and yielding the first two components of the branching vector. ($a_1, \ldots, a_5$ are pairwise distinct, since they do not constitute a cycle.) We concentrate now on the third branch, since the fourth one is quite similar to it and the same reasoning applies.

In $\{a_1, b_2, N(b_3), N(b_4), b_5, a_6\}$ we count 11 or 12 vertices, but some of them might be identical. If we could prove that the size of this set is always at least 8, we got the branching vector $(3, 5, 8, 8)$, which is good enough. Unfortunately, this is

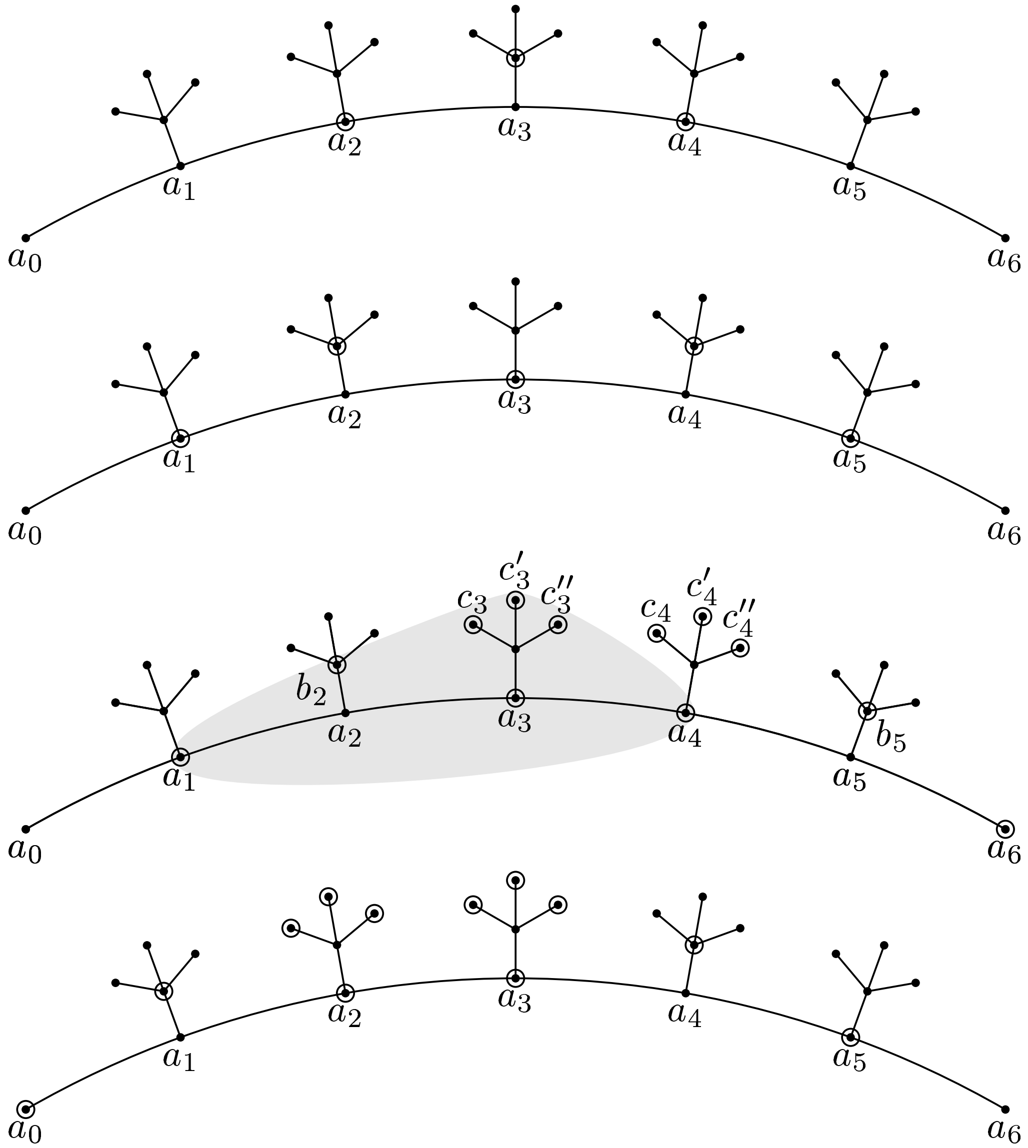

Figure 3: Branching in Case 5.2.1

not possible. We proceed as follows: First we find out under what circumstances the size of the set can be smaller than 8. It will turn out that there is only one pathological possibility. Then we provide a different type of branching suited exactly for this exception. If neither the third nor the symmetrical fourth branch are pathological we get a branching vector of $(3, 5, 8, 8)$ using the above branching scheme.

We distinguish two subcases according to the degree of b_4. But we start with an observation that is used in both subcases. We claim that all vertices in $\{a_1, b_2, a_3, c_3, c_3', c_3'', a_4\}$, highlighted in Figure 3, are pairwise distinct: The a_i's are pairwise distinct as seen above, $b_2 \neq a_4$ (triangle on a_2) and $\{c_3, c_3', c_3''\} \cap \{a_1, a_3, a_4, b_2\} = \emptyset$ (triangle or bridge). Likewise, all marked vertices outside the highlighted area are also pairwise distinct. Hence, the number of marked vertices is at least seven in total and the number is seven only if all marked vertices outside the highlighted area coincide with some vertex inside the area.

Moreover, if the degree of b_3 is bigger than 4, the shaded area already contains 8 vertices. In the following, we can therefore assume that b_3 has at most degree 4.

A The degree of b_4 is at least 4.

In the following we write c_i^* for any vertex in $\{c_i, c_i', c_i''\}$. At least two of c_4^* have to coincide with c_3^*, otherwise not all c_4^*'s can be identical to vertices inside the shaded area ($c_4^* = a_1$ and $c_4^* = b_2$ are not possible simultaneously, since a_2 would have a bridge). If all c_4^*'s coincide with all c_3^*'s, then there is no possibility to match a_6 to a vertex inside the shaded area: Both $a_6 = a_1$ and $a_6 = b_2$ are impossible (cycle of length 5). Hence, there is no possibility to match all vertices out- and inside the area.

If, however, only two of c_4^* coincide with c_3^*, then there is exactly one possibility for a_6 to be matched: $a_6 = c_3^*$. Then there remains only the possibility $b_5 = b_2$ for b_5, since $b_5 = a_1$ leads again to a length–5 cycle. In summary, $c_3 = c_4$, $c_3' = c_4'$, $c_3'' = a_6$, $a_1 = c_4$, $b_2 = b_5$ is the only possibility to match marked vertices in- and outside the shaded area. If this case applies, there are indeed only 7 marked vertices altogether, but it is the only possibility. In this case, however, we might branch according to only $\{a_2, b_3, a_4\}$, $\{a_1, b_2, a_3, b_4, a_5\}$, and $\{a_1, b_2, N(b_3), N(b_4), b_5, a_6\}$, i.e., leaving the last choice $\{a_0, b_1, N(b_2), N(b_3), b_4, a_5\}$ away. To prove that this is legal, Figure 4 contains the subgraph with all identities shown and the fourth branching set marked.

If the marked vertices, i.e., $\{a_0, b_1, N(b_2), N(b_3), b_4, a_5\}$, are indeed part of a minimum vertex cover, we can construct another vertex cover by replacing vertices as indicated by the arrows in Figure 4. The remainder of the graph and the cover is not changed at all and the size of the cover remains also the same. Hence, it is another minimum vertex cover. Particularly, it contains $\{a_1, b_2, N(b_3), N(b_4), b_5, a_6\}$. So we can branch according to only the first three branches. This implies a branching vector of $(3, 5, 7)$ in this case.

B The degree of b_4 is 3.
There are three nodes c_3, c_3', and c_3'', but only two c_4 and c_4'. In contrast to above it is no longer possible that both $c_4 = c_3$

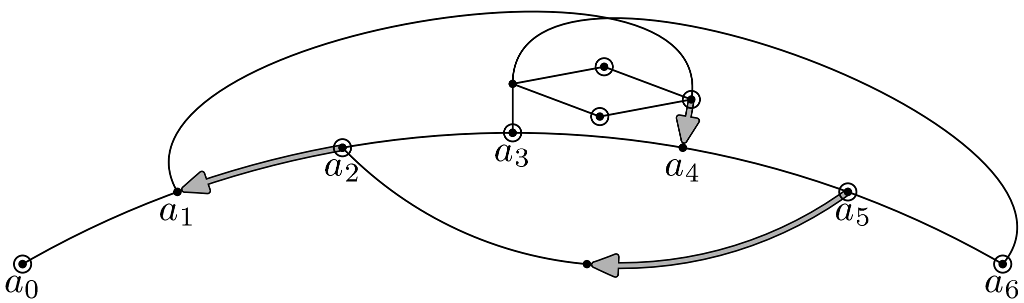

Figure 4: This figure shows the fourth branch, that is, $\{a_0, b_1, N(b_2), N(b_3), b_4, a_5\}$, when $c_3 = c_4$, $c_3' = c_4'$, $c_3'' = a_6$, $a_1 = c_4$, and $b_2 = b_5$. Vertices that are put into the vertex cover in the fourth branch are encircled. By modifying the cover as indicated by the arrows, we get another cover of the same size. This new cover is handled by the third branch $\{a_1, b_2, N(b_3), N(b_4), b_5, a_6\}$. We can therefore skip the fourth branch.

and $c'_4 = c'_3$, since now b_4 has degree 3 and therefore cannot have a bridge. One of c^*_4 can be identical to one of c^*_3, but the other has to be identical to one of $\{a_1, a_2, a_3, a_4, b_2\}$. The only possibility is $c^*_4 = b_2$, since all other matchings would imply a triangle, bridge on a_4, or a forbidden cycle of length 5 (and, of course, $a_4 \neq c^*_4$ by definition). It remains to match b_5 and a_6. It is easy to see that $b_5 = c^*_3$ is the only possibility for b_5. Finally, there is no possibility to match a_6: If $a_6 = c^*_3$, there would be a bridge on a_6 and b_2 is already occupied.

Case 5.2.2 Finally, we assume that there is a cycle of length 5 that consists of vertices with degree 3 and at least one of them has a neighbor with degree at least 4. This cycle shall consist of $a_0, \dots, a_4$ with neighbors $b_0, \dots, b_4$ outside the cycle, where b_2 is the neighbor with degree at least 4. Figure 5 shows the four branches

1. $\{a_1, b_2, a_3\}$,
2. $\{a_0, b_1, a_2, b_3, a_4\}$,
3. $\{a_0, b_1, N(b_2), N(b_3), b_4\}$, and
4. $\{b_0, N(b_1), N(b_2), b_3, a_4\}$.

The first branch covers that a_2 is not in the cover, all other branches that a_2 is in it. The remaining branches distinguish whether a_1 or a_3 are in the cover, or none of them. (We can omit the possibility that both of them are, since then there is an optimal cover that is handled by the first branch.) We can

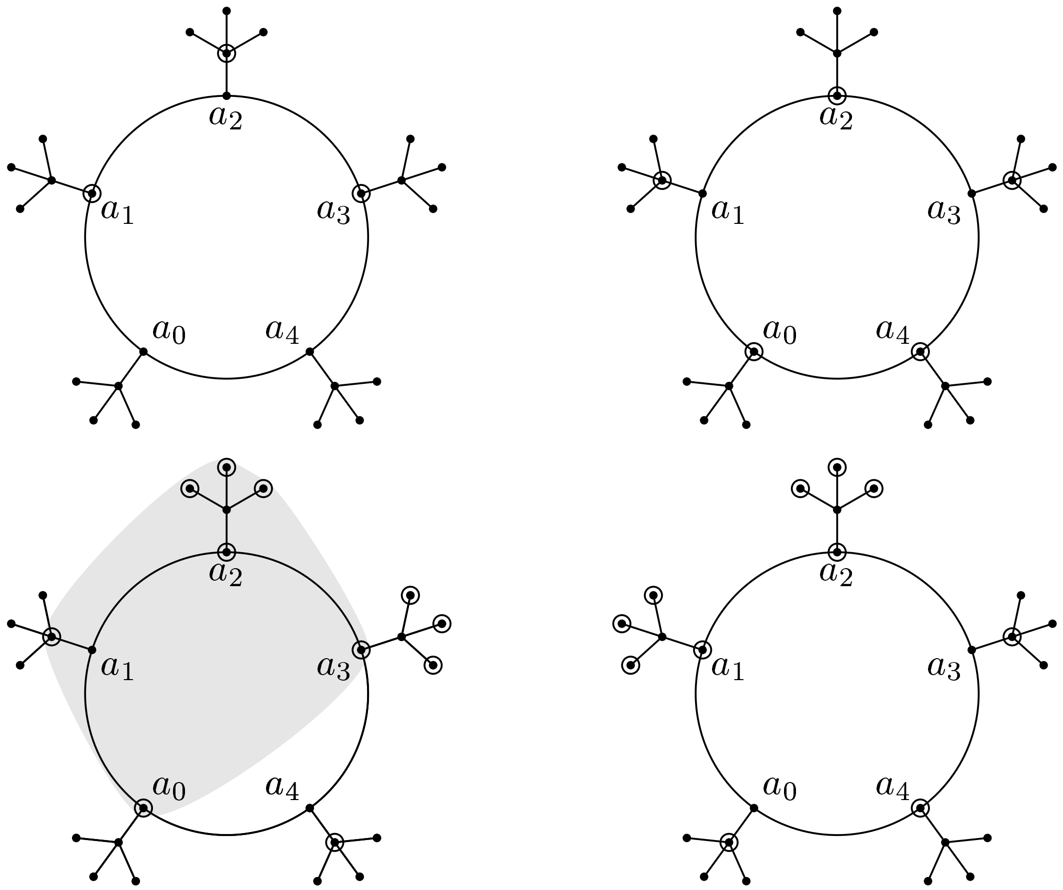

Figure 5: Branching in Case 5.2.2

add $N(a_4)$, i.e., b_4, to the third branch because two of three neighbors of a_4, i.e., a_0 and a_2 have already been picked. Furthermore, we can also add $N(b_2)$, because otherwise this branch could be replaced by the first one. Finally, we can also add $N(b_3)$, because otherwise this branch could be replaced by the second one. The fourth branch is based on picking a_1 and $N(a_3)$ and is extended in a way similar to the third branch.

The shaded area contains at least 7 marked vertices that are pairwise distinct and the marked vertices outside are pairwise distinct, too. Again, if the degree of b_2 is more than 4, then the shaded area contains at least 8 distinct marked vertices and the corresponding component of the branching vector is at least 8. Hence, let the degree of b_2 be 4 in the following.

The total number of marked, pairwise distinct vertices can be 7 only if we can match all marked vertices outside the shaded area to vertices inside the area. For b_4 there is only one possibility: $b_4 = c_2^*$. Before matching the remaining c_3^*, we distinguish again two subcases.

A The degree of b_3 is at least 4.
Then there are at least 3 vertices c_3^*. Only two of them can be matched to c_2^*, since already $c_2^* = b_4$. The remaining c_3^* has to be matched to b_1. So there is essentially only one possibility to get 7 marked vertices: $b_4 = c_2$, $c_3 = b_1$, $c_3' = c_2'$, $c_3'' = c_2''$. The according drawing can be found in Figure 6 with the marking of the fourth branch $\{b_0, N(b_1), N(b_2), b_3, a_4\}$.

If the marked vertices in Figure 6 form a minimum vertex cover we can modify it by replacing vertices as indicated by the

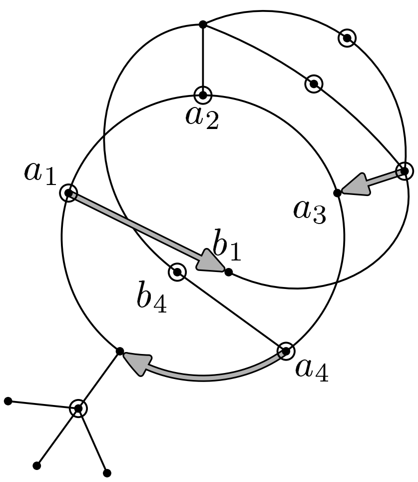

Figure 6: Changing the cover by replacing included vertices as indicated by the shaded arcs also yields a minimum vertex cover.

arrows. The vertex cover remains minimum and all vertices of the third branch are marked. Hence, if there is a minimum cover in the fourth branch, there will be also a minimum cover in the third branch and we can leave the fourth branch away resulting in a branching vector of $(3, 5, 7)$. (This rearrangement does not necessarily work if the degree of b_3 is more than 4, but as we have shown above, we can exclude this possibility.)

B The degree of b_3 is 3.
As we have already seen above, $c_2^* = b_4$ is the only possibility to match b_4. It is now not possible to match both c_3^* to c_2^*, since this would result in a bridge on b_3. So the only possibility is $c_3 = b_1$, $c_3' = c_2'$, $b_4 = c_2$. We can now proceed exactly as in

subcase A resulting in a branching vector of at least $(3, 5, 7)$.

6 Degree-4-vertices

In this section, we can assume that all vertices have either degree 4 or 5 and that there is at least one vertex with degree 4 and at least one vertex with degree 5.

Case 1.

Assume that there is a vertex x of degree 4 that is part of a triangle and has a neighbor y with degree 5. Let a and b be the neighbors of x such that $\{a, b, x\}$ is a triangle (a or b can but need not coincide with y).

First, we assume that $a, b \neq y$. Let $c \notin \{a, b, y\}$ be another neighbor of x. We can branch according to $N(x)$, $N(y)$, and $\{x, y, N(c)\}$: Either x's neighbors or y's neighbors are in a cover or $\{x, y\}$. In the latter case we can assume that c is not in the cover: Assume the contrary. Then y and c are in the cover, two neighbors of x. One of a or b must be in the cover, too. So, three neighbors of x are in the cover and we can replace x by the remaining fourth neighbor. This yields another optimal cover, which is taken into account by the first branch. The resulting branching vector is at least $(4, 5, 4)$.

Now let us assume that $a = y$. Then we can further assume that the remaining two neighbors c and d of x, i.e., not a or b, are *not* connected by an edge. (Otherwise we can choose *them* to play the role of a and b above.) Branch according to $N(x)$,

$N(c)$, and $\{x, c, N(d)\}$, which is sufficient for the same reason as above. The branching vector is again at least $(4, 4, 5)$, because $c \notin N(d)$.

Case 2.

We assume in this case that there is no vertex x with degree 4 that has the following properties simultaneously: (1) x has a neighbor y with degree 5, (2) x has at least two bridges of whom y is not part of (this might be independent bridges or a double bridge). We can further assume that there are no triangles that contain a vertex with degree 4 that has a neighbor with degree 5, i.e., that Case 1 does not apply. Choose some vertex x with degree 4 that has a neighbor a with degree 5 (such vertex exists, otherwise the graph would be regular or not connected). Let b, c, d be the other neighbors of x (which can have degrees between 4 and 5). We can branch according to $N(a)$, $N(x)$, and $\{a, x\}$, but we split the last branch according to whether b and/or d are part of the optimal vertex cover. Altogether, we branch according to $N(a)$, $N(x)$, $\{x, a, N(b), N(d)\}$, $\{x, a, d, N(b), N(c)\}$, and $\{x, a, b, N(c), N(d)\}$ (see Figure 7). We get the branching vector $(5, 4, 8, 9, 9)$ if x has no bridge that a is no part of. This is easy to see: All vertices of the last three branches must be distinct, otherwise there would be such a bridge or a triangle.

There might be *one* bridge that a is no part of. Without loss of generality we can assume that this bridge goes over c and d (b, c, d are symmetric and can be mutually exchanged in the above branching scheme). Then all vertices in the third and fourth branch must be still mutually distinct, but c and

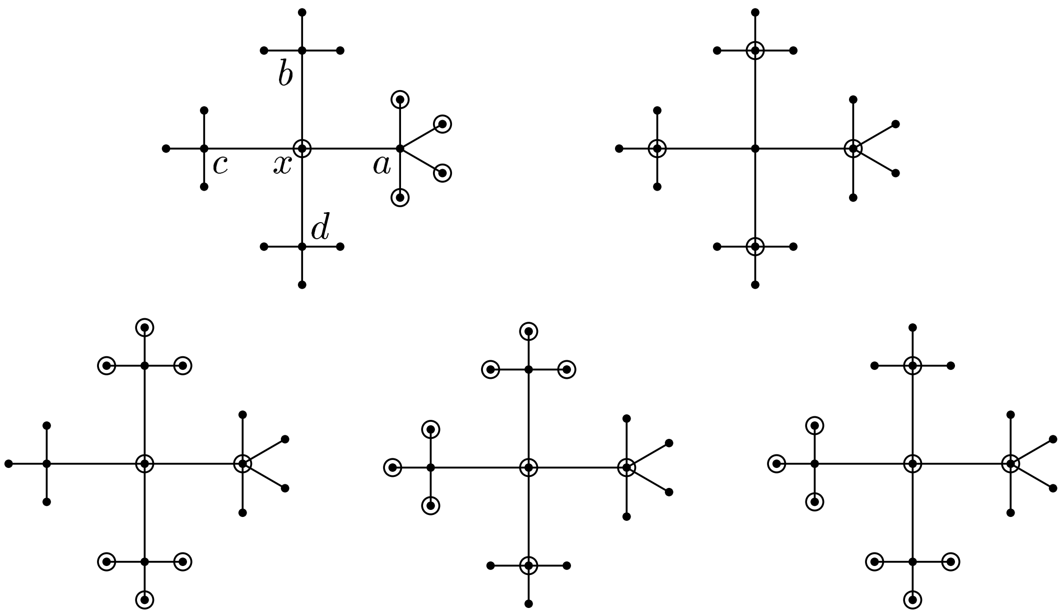

Figure 7: All marked vertices in each of the five branches are distinct if there are no bridges that do not involve a. If there is such a bridge, we can assume it matches two marked vertices in the last branch because of symmetry.

d now share another neighbor besides x and therefore two of the marked vertices in the last branch coincide. This leaves a branching vector of $(5, 4, 8, 9, 8)$.

Case 3.

Now we assume that there is a vertex x that has exactly the properties that were forbidden in Case 2: It has degree 4 and two bridges. There is a neighbor y with degree 5 that is not part of either of the two bridges. There are two possibilities which are depicted in Figure 8: Either x has two separate bridges or a double bridge. The algorithm can branch according to $N(y)$, $N(x)$, and $\{x, y\}$. In the last branch, if $\{x, y\}$ is part of the cover, then we can assume that the vertices on the bridge are members of the cover, too. Otherwise, their neighbors would be part of the cover and that implies that at least three of x's neighbors are in the vertex cover. Then, however, we can take $N(x)$ instead of x. That is, this cover is already handled by the first branch. Altogether, this implies a branching vector $(5, 4, 4)$.

7 Conclusion

Improving previous work [3, 12], we presented the so far best known algorithm for the parametrized Vertex Cover problem, running in time $O(kn + 1.29175^k k^2)$. It is conceivable that refined complexity analysis could still slightly improve the exponential base. Besides the theoretical interest, this result may also be relevant for practical applications. So, for example, the Vertex Cover algorithm of Balasubramanian *et al.* [3] has been implemented for applications in computational biochemistry [12, 16], our algorithm now being a natural candidate to replace or complement it. Note that our as well as previous worst case algorithms have the clear potential to perform much better on average. Thus, they may also qualify as efficient heuristic algorithms, maybe by adding some heuristic features.

As to future work, it remains open to provide a competi-

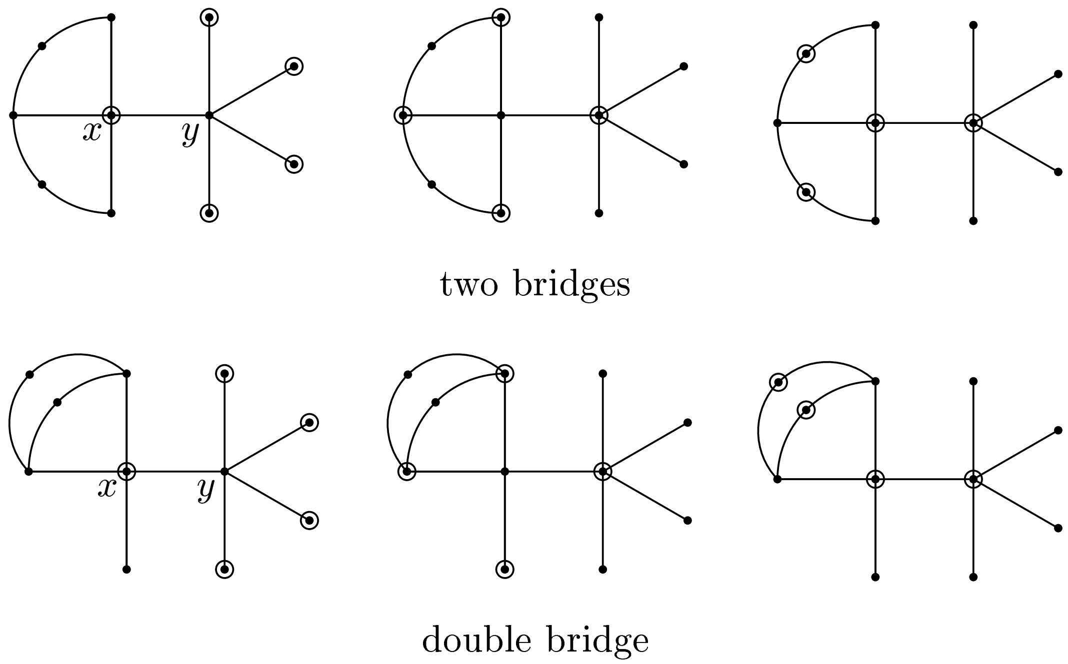

Figure 8: How to treat two separate bridges or a double bridge—the special case that was forbidden in Case 2: The branching vector is at least $(5, 4, 4)$.

tive implementation of our algorithm and to compare its performance with existing algorithms. Clearly, it is also an open question whether the exponential term of now 1.29175^k can be further decreased. Observe that even seemingly minor improvements of the base of the exponential term can mean decisive progress in practical applications [12]. A significant hurdle in improving our upper bounds seems to be the great number of case distinctions. Perhaps, some kind of mechanized case analysis could help?

References

[1] K. A. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73:235–276, 1995.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33d IEEE Conference on Foundations of Computer Science*, pages 14–23, 1992.

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

[4] 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.

[5] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[6] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, April 1997.

[7] R. G. Downey and M. R. Fellows. Fixed parameter tractability and completeness III. In *Complexity Theory: Current Research, Edited by K. Ambos-Spies, S. Homer, and U. Schöning, Cambridge University Press*, pages 191–225. 1993.

[8] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, August 1995.

[9] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141:109–131, 1995.

[10] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.

[11] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

[12] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In F. Roberts,

J. Kratochvíl, and J. Nešetřil, editors, *The Future of Discrete Mathematics: Proceedings of the First DIMATIA Symposium, June 1997*, AMS-DIMACS Proceedings Series. AMS, 1998. To appear. Available through http://www.inf.ethz.ch/personal/stege.

[13] R. C. Evans. Testing repairable RAMs and mostly good memories. In *Proceedings of the IEEE Int. Test Conf.*, pages 49–55, 1981.

[14] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, 26:157–162, 1987.

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

[16] M. Hallett, G. Gonnet, and U. Stege. Vertex cover revisited: A hybrid algorithm of theory and heuristic. Manuscript, 1998.

[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. Two new upper bounds for SAT. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 521–530, 1998.

[19] O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. 1997. Submitted to *Information and Computation*.

[20] M. Mahajan and V. Raman. Parametrizing above guaranteed values: MaxSat and MaxCut. Technical Report TR97-033, ECCC Trier, 1997. To appear in *Journal of Algorithms*.

[21] K. Mehlhorn. *Graph algorithms and NP-completeness*. Heidelberg: Springer, 1984.

[22] B. Monien and E. Speckenmeyer. Upper bounds for covering problems. Technical Report Reihe Theoretische Informatik, Bericht Nr. 7/1980, Universität Gesamthochschule Paderborn, 1980.

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

[24] R. Niedermeier. Some prospects for efficent fixed parameter algorithms (invited paper). In B. Rovan, editor, *Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, number 1521 in Lecture Notes in Computer Science, pages 168–185. Springer-Verlag, 1998.

[25] R. Niedermeier and P. Rossmanith. New upper bounds for MaxSat. Technical Report KAM-DIMATIA Series 98-401,

Faculty of Mathematics and Physics, Charles University, Prague, July 1998. Submitted for publication.

[26] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[27] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *Journal of Computer and System Sciences*, 53:161–170, 1996.

[28] R. Paturi, P. Pudlák, M. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th IEEE Conference on Foundations of Computer Science*, 1998.

[29] P. Pudlák. Satisfiability—algorithms and logic (invited paper). In *Proceedings of the 23d Conference on Mathematical Foundations of Computer Science*, number 1450 in Lecture Notes in Computer Science, pages 129–141, Brno, Czech Republic, August 1998. Springer-Verlag.

[30] V. Raman. Parameterized complexity. In *Proceedings of the 7th National Seminar on Theoretical Computer Science (Chennai, India)*, pages I–1–I–18, June 1997.

[31] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7:425–440, 1986.