Upper Bounds for Vertex Cover Further Improved

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

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

Abstract. 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 $O(kn + 1.29175^k k^2)$.

1 Introduction

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

- It was among the first *NP*-complete problems [7].
- There have been numerous efforts to design efficient approximation algorithms [3], but it is also known to be hard to approximate [1].
- It is of central importance in parameterized complexity theory and has one of the most efficient *fixed parameter algorithms* [4], which is also subject of [2, 5] 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.* [2] has been implemented as part of the DARWIN project at ETH Zürich [5, 8]. 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 $v \in C$. A straightforward greedy algorithm shows that Vertex Cover is approximable to a ratio 1 (cf. [13]). However, unless $P = NP$, Vertex Cover has no polynomial

* 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, Czech Republic.

time approximation scheme [1] and it is known to be *not* approximable to a ratio 0.1666 [9].

Vertex Cover has seen quite some history of progress with respect to fixed parameter algorithms ("fixed parameter" refers to k, see [4, 11] for details). Recently, Balasubramanian *et al.* [2] 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.* [5]. 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.* A technical report [12] contains all details that had to be omitted due to lack of space.

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 $i,j\in C$. A vertex cover is *minimal* or *optimal* if it has minimum size, i.e., if there is no vertex cover that has less vertices. 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 [10]. 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 $t_n=t_{n-d_1}+\cdots+t_{n-d_k}$ with the characteristic polynomial $z^d=z^{d-d_1}+\cdots+z^{d-d_k}$, where $d=\max\{d_1,\ldots,d_k\}$. If α is a root of the characteristic polynomial 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 $(d_1,\ldots,d_2)$. 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 $(3, 5, 8, 8)$ which occurs in Section 5 (Case 5.2.1).

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 $O(2^k n)$ [4], 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.* [2] presented an algorithm with search tree size 1.32472^k and this was improved to 1.31951^k by Downey *et al.* [5]. We further improve this size to 1.29175^k.

Before we give 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* [4], but basically the same approach can be traced back to older ideas from VLSI–theory, e.g., Evans [6]. 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.* [2] than to that of Downey *et al* [5]. 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. We refer to Downey *et al.* [5] for details. By way of contrast, Balasubramanian *et al.* [2] 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 shrinking 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 $G_1,\ldots,G_k$ 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, if 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.

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 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 $(3, 3)$.

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. Due to the lack of space, many details and proofs of correctness had to be omitted.

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 $\{x, y, z\}$. 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 $N(a)$. 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 $\{a, x, N(b), N(c)\}$. 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. 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 1 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 1 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 1. 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, 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 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. For the pathological cases we can even prove a branching vector $(3, 5, 7)$. We omit any details.

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 $a_0, \ldots, a_4$ with neighbors $b_0, \ldots, b_4$ outside the cycle, where b_2 is the neighbor with degree at least 4. We branch by either picking $\{a_1, b_2, a_3\}$ or $\{a_0, b_1, a_2, b_3, a_4\}$ or $\{a_0, b_1, N(b_2), N(b_3), b_4\}$ or $\{b_0, N(b_1), N(b_2), b_3, a_4\}$. Similar considerations as in Case 5.2.1 show that we

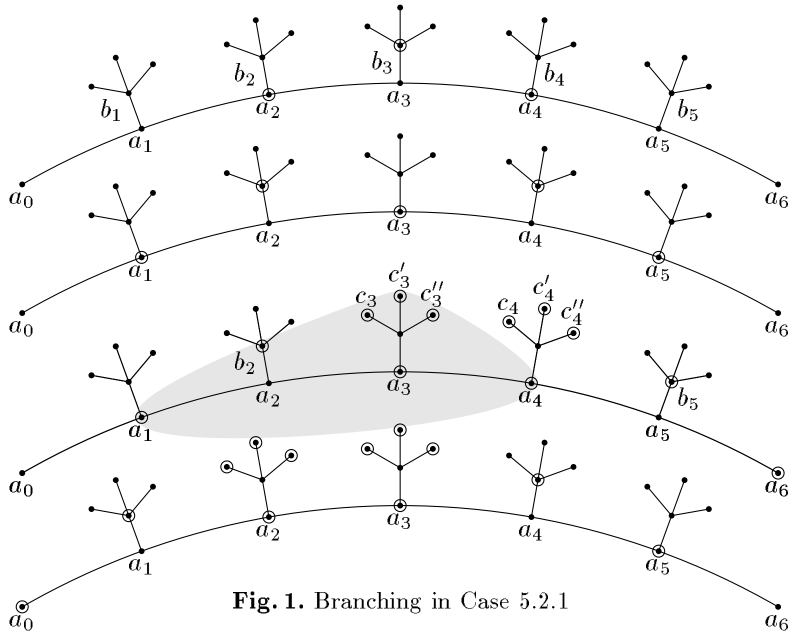

Fig. 1. Branching in Case 5.2.1

can get branching vectors $(3, 5, 8, 8)$ or $(3, 5, 7)$, in the latter case omitting the fourth branching set.

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. Again, several details had to be omitted.

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 $\{x, y, N(c)\}$: 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,

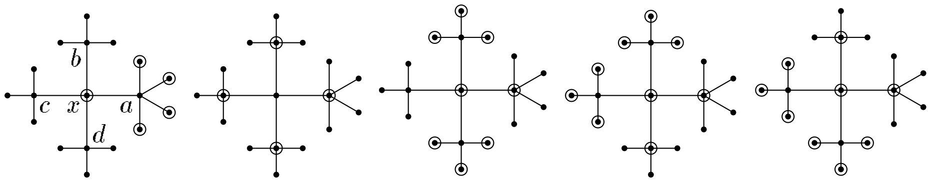

Fig. 2. All marked vertices in each of the five branches are distinct if there are no bridges that do not involve a. If there is such a bridge, we can assume it matches two marked vertices in the last branch because of symmetry.

(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 2). We get the branching vector $(5, 4, 8, 9, 9)$ if x has no bridge that a is no part of.

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 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 3: 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 [2, 5], we presented the so far best known algorithm for the *NP*-complete Vertex Cover problem, running in time $O(kn + 1.29175^k k^2)$.

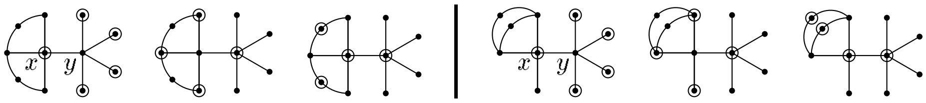

Fig. 3. How to treat two separate bridges or a double bridge.

Besides the theoretical interest, this result may also be relevant for practical applications. So, for example, the Vertex Cover algorithm of Balasubramanian *et al.* [2] has been implemented for applications in computational biology [5, 8], our algorithm now being a natural candidate to replace or complement it. Note that our as well as previous algorithms that are good in the worst case have the clear potential to perform much better on average.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33d IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.
2. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.
3. P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, April 1997.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.
5. 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, 1999.
6. R. C. Evans. Testing repairable RAMs and mostly good memories. In *Proceedings of the IEEE Int. Test Conf.*, pages 49–55, 1981.
7. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1979.
8. M. Hallett, G. Gonnet, and U. Stege. Vertex cover revisited: A hybrid algorithm of theory and heuristic. Manuscript, 1998.
9. J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.
10. O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. 1997. Submitted to *Information and Computation.* Available at http://mi.informatik.uni-frankfurt.de/people/kullmann/papers.html.
11. 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.
12. 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.
13. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.