LCNS, Vol 1672, pp. 387–397, Springer 1999

An Efficient, Exact Algorithm for Constraint Bipartite Vertex Cover

Henning Fernau and Rolf Niedermeier*

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

Abstract. The "Constraint Bipartite Vertex Cover" problem (CBVC for short) is: given a bipartite graph G with n vertices and two positive integers k_1, k_2, is there a vertex cover taking at most k_1 vertices from one and at most k_2 vertices from the other vertex set of G? CBVC is *NP*-complete. It formalizes the spare allocation problem for reconfigurable arrays, an important problem from reconfigurable VLSI manufacturing. We provide the first nontrivial so-called "fixed parameter" algorithm for CBVC, running in time $O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$. Our algorithm is efficient and practical for small values of k_1 and k_2, as occurring in applications. Despite of the seemingly great similarity between CBVC and Vertex Cover for general graphs, only little ideas developed for Vertex Cover algorithms carry over to CBVC, so several new techniques had to be developed in this paper.

1 Introduction

Nontrivial upper bounds for important *NP*-hard problems by exact algorithms have seen a continuous interest over many years till today [?,6, ?,12–16]. With the advent of parameterized complexity theory [3] a special class of exact algorithms has become more and more important [4, 9]: As an example, consider the *NP*-complete Vertex Cover problem: given an undirected graph $G = (V, E)$ and a positive integer k, is there a subset of vertices $C \subseteq V$ of size $|C| \leq k$ such that each edge in E has at least one endpoint in C? Setting $n := |V|$, the best known exact algorithm for this problem has running time $O(1.211^n)$ [13]. However, there is another exact ("fixed parameter") algorithm solving Vertex Cover in running time $O(1.29175^k + kn)$ [?]. For instance, already for $k \leq n/2$ it is much more efficient than the $O(1.211^n)$ algorithm.

The fixed parameter algorithm for Vertex Cover mentioned above is valuable from a practical point of view, where often small values of k can be assumed because of the application behind [3, 4]. In the case of Vertex Cover, for instance, these may be applications in computational biochemistry [4, 5]. In this paper,

* Partially 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.

we study an also *NP*-complete variant of the Vertex Cover problem, motivated by reconfigurable VLSI [7]: given a bipartite graph $G = (V_1, V_2, E)$ and *two* positive integers k_1 and k_2, are there two subsets $C_1 \subseteq V_1$ and $C_2 \subseteq V_2$ of sizes $|C_1| \leq k_1$ and $|C_2| \leq k_2$ such that each edge in E has at least one endpoint in $C_1 \cup C_2$? The existence of *two* parameters and two vertex sets makes this problem, called Constraint Bipartite Vertex Cover (CBVC), quite different from the original one. Thus, whereas the classical Vertex Cover problem (with only one parameter!) restricted to bipartite graphs is solvable in polynomial time (because it is equivalent to a polynomial time solvable maximal matching problem), by a reduction form Clique it has been shown that CBVC is *NP*-complete [7]. Here, to our best knowledge, we give the first nontrivial fixed parameter algorithm for the Bipartite Vertex Cover problem running in time $O(1.3999^{k_1+k_2} + (k_1+k_2)n)$. We conjecture that, due to ist different combinatorial structure in comparision with Vertex Cover, it should be very hard to get an exponential base close to the one there (1.29175^k).

Our result makes the following two contributions: First, it gives a further (of so far few) example for a problem with an efficient fixed parameter algorithm [3, 4, 9]. Second, our result is not only of theoretical interest, but is also valuable with regard to practical applications. This is also due to the fact that from the VLSI application behind it is very natural to assume small values for k_1 and k_2 compared to the total number of graph vertices n. Hence, our exact algorithm with a provable upper bound may successfully compete with heuristic algorithms in use, as first implementation tests indicate.

To achieve our result, we employ well-known methods from parameterized complexity [3]: reduction to problem kernel and bounded search trees. These have already been successfully applied for Vertex Cover [1, 4, ?], Maximum Satisfiability [8, 10], and elsewhere [3]. The main technical contribution of our work is the development of a search tree of size $1.3999^{k_1+k_2}$, which requires numerous case distinctions based on combinatorial considerations that are very different from the classical Vertex Cover case. The paper is organized as follows...

2 Preliminaries

We assume familarity with basic notations from graph theory, algorithms, and complexity. We make use of the following notation for a graph $G = (V, E)$: Writing $G - X$ means that we delete vertex X and all its incident edges from graph G. By NX we denote the neighbors of X in G. By δX we denote the degree of vertex X, that is, $|NX|$. A graph is called *r-regular* if every vertex has degree r. In this paper, we only deal with bipartite graphs. For the ease of presentation, we consider them as two-colored (black and white) graphs with each vertex having a colour opposite to all its neighbors.

Our algorithm works recursively. The number of recursions is the number of nodes in the according search tree. This number is governed by homogeneous, linear recurrences with constant coefficients (cf. [6, 11, ?]). 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 $(d_1, \ldots, d_2)$. 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 $k := k_1 + k_2$ is the parameter and α is the biggest branching number that will occur; it is about 1.3999 and belongs to the branching vector $(., ., ., ., ., ., .)$ occuring in Section **??** (Case ???).

3 The algorithm—overview

Our algorithm works in basically the same way as the fixed parameter algorithms for Vertex Cover do [1, 4, **?**]. 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 sets. 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 $2^{k_1+k_2}$. Altogether, it is easy to see that this leads to an algorithm running in time $O(2^{k_1+k_2}n)$, where n denotes the number of vertices in the graph. All results (including ours) to get more efficient algorithms are based on efforts to shrink the search tree size.

As in the classical case, we achieve a reduction of the search tree size by distinguishing between the degree of graph vertices. Since for CBVC we have to minimize with respect to two parameters, this gets significantly harder than in the classical Vertex Cover case. For instance, in the classical case for degree-1-vertices it always leads to an optimal vertex cover to take the neighbor of a degree-1-vertex. Thus, a branching in the search tree is avoided. However, this is no longer possible in the CBVC case, because the neighbor belongs to the second vertex set in the bipartite graph and we have to minimize with respect to two two vertex cover set sizes. In particular, the size of a minimal solution for CBVC is no longer uniquely determined.

Before we give an overview of our algorithm, we still have to briefly explain a technique called *reduction to problem kernel* [2, 3], which is a kind of preprocessing. This step is based on a simple observation already used by Kuo and Fuchs [7] which led to the "must-repair-analysis" pre-phase in their algorithms. Let $G = (V_1, V_2, E)$ be our given bipartite graph and k_1 and k_2 be the constraints. Clearly, if a vertex in V_1 has degree greater than k_2, then it has to be part of the vertex cover and, analogously, if a vertex in V_2 has degree greater than k_1, then it also has to be part of the vertex cover. In this way, deleting all these "high-degree-vertices" together with their incident edges, we can infer that after reduction to problem kernel the size of the graph is at most $2k_1k_2$. Obviously, reduction to problem kernel can be implemented to run in time $O((k_1 + k_2)n)$. Combining reduction to problem kernel with the trivial search tree of size $2^{k_1+k_2}$, we would end up with a time $O(2^{k_1+k_2}k_1k_2 + (k_1+k_2)n)$ algorithm. In the rest of the paper, we describe how to shrink the search tree size from $2^{k_1+k_2}$ to $1.3999^{k_1+k_2}$.

Before we describe the overall structure of our search tree algorithm, let us briefly deal with an easy special case. Clearly, we can deal with each connected component separately. So, let us assume that the graph is connected. If the maximum vertex degree of a graph is at most two, then CBVC is easy to solve: We know that, in this case, the graph is either a cycle or a path. In both cases, however, it is fairly easy to compute the linear number of possible minimal vertex covers in linear time. We omit the basically straightforward details. Furthermore, as pointed out before, here we have to take care of the fact that our given graph may be split into several connected components. Since the various components are "independent" from each other, we simply can combine them using component-wise addition and then again looking for the minimal values. Altogether, using simple data structures, this can be done in $O((k_1 + k_2)^2)$ time, because $1 + \min(k_1, k_2)$ is an upper bound for the number of minimal vertex covers belonging to a component. Furthermore, we can assume that there are at most $k_1 + k_2$ components, since otherwise the graph is not coverable and we know that each output of merging two minimal vertex covers always is bounded by $O(k_1 + k_2)$. Altogether, we have $(k_1 + k_2)$ merge steps each of time complexity $O((k_1 + k_2)^2)$. In summary, this gives:

Proposition 1. *For bipartite graphs with maximum vertex degree 2 CBVC can be solved in time $O((k_1 + k_3)^3)$.*

Because of Proposition 1, in the following description of the basic structure of our search tree algorithm we now may concentrate on graphs with maximum degree at least three:

Overall structure of the search tree algorithm. The algorithm recursively finds optimal vertex covers as follows. Given a bipartite graph G, we choose several subgraphs G_1,...,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 its incident edges. A vertex cover of G_1, together with X, is then a vertex cover of G. Moreover, if there are optimal vertex covers for G that contain X, then we can construct optimal vertex covers 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 a vertex covers of G by taking vertex covers of G_2 and adding all neighbors of X. If we start from optimal vertex covers for G_1 and G_2, then at least one of the resulting covers for G must be optimal, since either X or its neighbors must be part of any vertex cover. 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. We distinguish between eight main cases (**M1**–**M8**), some of them requiring further subcases. It is of central importance for the correctness of our algorithm to execute the various steps in the given order—that is, we always choose an applicable step with minimal number:

M1. If there is a vertex X with degree at least 4, then branch according to X and NX.
Corresponding branching vector and branching number: (1,4) and 1.3803.

M2. If the graph is 3-regular, then pick any vertex X and branch according to X and NX. (This step has to be applied at most once and thus does not significantly influence the algorithm's overall complexity.)
Corresponding branching vector and branching number:

M3. Deal with tails of size at least two, see...
Corresponding branching vector and branching number:

M4. Deal with cycles of length four, see...
Corresponding branching vector and branching number:

M5. Deal with chains of length at least three, see...
Corresponding branching vector and branching number:

M6. If there is a degree-3-vertex with three neighbors of degree 2, then proceed as described in ...
Corresponding branching vector and branching number:

M7. If there is a degree-3-vertex with two neighbors of degree 2, then proceed as described in ...
Corresponding branching vector and branching number:

M6. If there is a degree-3-vertex with one neighbor of degree 2, then proceed as described in ...
Corresponding branching vector and branching number:

It may easily be verified that the above steps provide a complete case distinction handling all cases that may occur. More precisely, from this point of view, steps **M3**–**M5** even would be superfluous—however, they are necessary in order to get a small search tree size by handling "nice special cases" in advance. The hardest cases above are **M4**, **M6**, **M7**, and **M8**. The worst case branching vector is ??? and implies a search tree size 1.3999^k.

In summary, we thus obtain an algorithm for CBVC running in time $O(1.39^{k_1+k_2}k_1k_2 + (k_1+k_2)n + (k_1+k_2)^3)$. This can be improved to $O(1.39^{k_1+k_2} + (k_1+k_2)n)$ by simple asymptotic arguments. However, the factor k_1k_2 above cannot only be omitted by "asymptotic tricks," but also by an argument due to to refined complexity analysis. The basic idea is that all nodes in the search tree are near to the leaves and we can therefore expect that the running time is the size of the search tree times a small constant instead of times k_1k_2. Observe that near to the leaves the graph to be processed has to be small. We omit any details here and just note that the argument works in complete analogy as it does for classical Vertex Cover [11, **?**].

In the rest of the paper, we provide the missing details for cases **M3**–**M8**.

4 Details of the algorithm

In the following, we present details of the main algorithm presented above, hence coming to an assessment of the running time in the worst case. To this end, we introduce a counter k which will be initialized to $k_1 + k_2$, i.e., the sum of the two input parameters bounding the size of the vertex cover. Each recursive call of the main procedure will decrement k in some way, so that k obviously bounds the depth of the search tree. Most conveniently, k is considered as a parameter of the main algorithm, which can be called like $m(G, k)$. Observe that due to the reduction to the problem kernel, G has $O(k^2)$ vertices and $O(k^2)$ edges. Since the main procedure works depth-first, the space requirement of the algorithm is only polynomial.

Case M2: 3-regularity Because of steps M1 and M2, in the following we can assume w.l.o.g. that the maximum vertex degree in the graph is three and the graph is not 3-regular.

Case M3: tails. A *tail* consists of a degree-3-vertex A, followed by a (possibly empty) sequence of degree-two-vertices, ended by a degree-1-vertex. If A is neighboured by a vertex of degree one (i.e., we have a *micro-tail*), we regard A as if it had degree two in the following analysis. Otherwise, a typical situation is depicted in Table 1, where dashed edges and vertices are optional. Here, we encounter the main trick in our time analysis for the first time: we have seen that we can cope with a graph having vertices of degree at most two in polynomial time (Proposition 1). Therefore, if we take vertex A in the present situation, we create a non-empty line component starting at the degree-2-vertex B; in order to cover that component, we need at least one vertex from that component in the cover. Although we do not know which vertex (black or white) to take into the cover, we can safely bound the search tree by calling $m(G - A, k - 2)$ and $m(G - NA, k - 3)$.

Note that in Table 1, the column labelled "Branching" contains the information necessary to understand the branching analysis. The first subcolumn lists the conditions under which the analysis is valid; $\emptyset$ indicates that no prerequisites are necessary. Then, the vertices taken in that branch are listed; here, first A and then (in the second row) its neighbours. The third subcolumn lists how many vertices will be needed at least for the cover in the final polynomial analysis of degree-2-vertices. The fourth subcolumn gives the values which can be substracted from the parameter k in the corresponding recursive call of the main procedure. Finally, the fifth subcolumn gives an upper estimate for the search tree size derived from this case, here 1.3248^k.

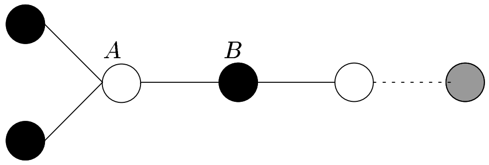
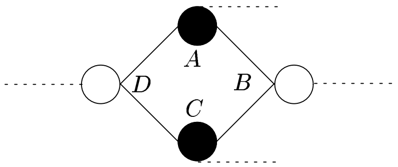
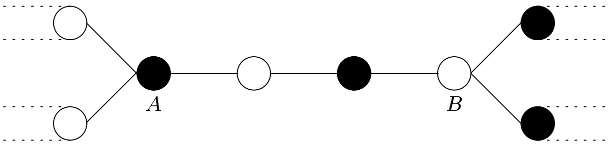
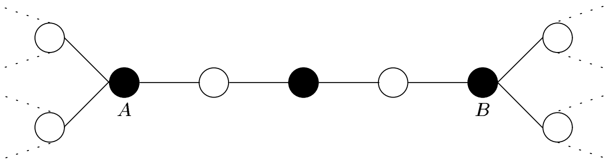
Case	Picture	Branching
M3		$\emptyset$ | A | 1 | 2 || 1.3248 $\emptyset$ | NA | 0 | 3 ||
M4		$\emptyset$ | AC | 0 | 2 || $\sqrt{2} <$ $\emptyset$ | BD | 0 | 2 || 1.4143
M5a		$A \in NB$ | see M4 || 1.4143 $\emptyset$ | AB | 1 | 3 || $A \notin NB$ | ANB | 0 | 4 || 1.3954 $\emptyset$ | NA | 0 | 3 ||
M5b		$A = B$ | see M4 || 1.4143 $A \neq B$ | AB | 1 | 3 || $\emptyset$ | ANB | 1 | 5 || 1.3640 $\emptyset$ | NA | 0 | 3 ||

Table 1. Main cases M3–M5

Case M4: cycles of length 4. Consider the corresponding picture from Table 1. Why is the given case distinction complete? Assume two neighbouring nodes, say A and B, are contained in the cover. Then, in order to cover the edge between C and D, either C or D have to be in the cover, too. Therefore, either case AC or case BD treats a sub-cover of the proposed case, and we do not exclude to take vertices others than AC or BD, resp., into the cover. A more refined analysis of 4-cycles as sketched in the appendix gives a value below $\sqrt{2}$.

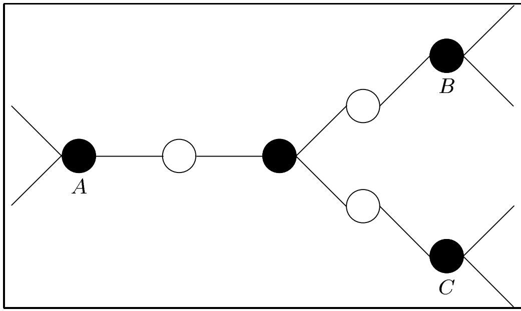
Case M5: chains of length at least 3. If two (not necessary different) degree-3-vertices A and B are connected via a path of length ℓ on which all vertices (besides A and B) have degree two, we say that A and B are conntected by a *chain* of length ℓ. Chains of length 3 and 4 are considered in Table 1 (M5a and M5b).

In case A and B are connected by a longer chain, the following branch analysis will work:

$\emptyset$	AB	2	3	1.3954
$\emptyset$	ANB	1	4	
$\emptyset$	NA	0	3	

In that analysis, we even assumed that $A = B$ or $A \in NB$ might occur.

So, we can assume from now on that two degree-3-vertices are connected by a chain of length at most two. Having this in mind, we start analysing the possible situations in the neighbourhood of a degree-3-vertex with at least one degree-2-neighbour.



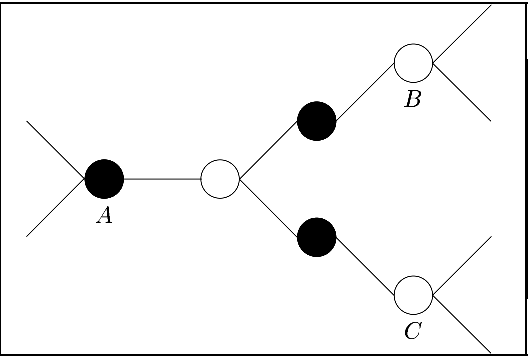
$\lvert NB \cap NC\rvert = 2$	see M4			1.4143
$\emptyset$	ABC	1	4	1.3954
$\delta C = 3$	$ABNC$	1	6	
$\delta B = 3$	$ANBC$	1	6	
$\lvert NB \cap NC\rvert < 2$	$ANBNC$	1	7	
$\delta A = 3$	NA	0	3	

Table 2. Case M6

Case M6: One degree-3-vertex with three degree-2-neighbours In Table 2, the general situation is depicted. We can assume $\delta A = \delta B = \delta C = 3$, since otherwise there is either a tail (M3) or a chain (M5).

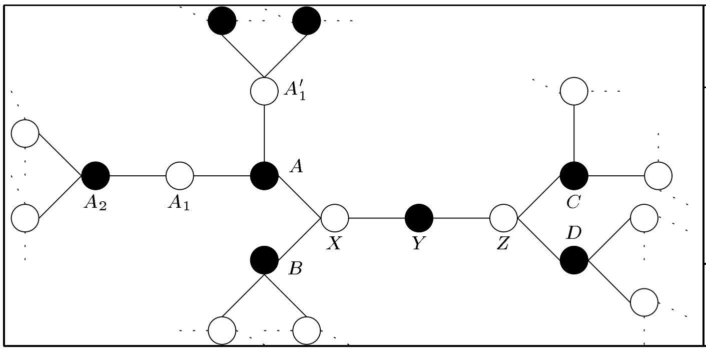
Case M7: One degree-3-vertex with two degree-2-neighbours Of course, since micro-tails at degree-3-vertices are like degree-2-vertices, the degree-3-vertex in question has one degree-3-neighbour called A. We refer to Table 3. Note that the case $\lvert NB \cap NC\rvert = 1$ is deferred, since it is treated in the section on the analysis of cycles of length 6.

Case M8: One degree-3-vertex with one degree-2-neighbour Consider the picture in Table 4. Each of both neighbours X, Z of the degree-2-vertex Y has two degree-3-neighbours (A, B resp. C, D), otherwise we are in case M6 or M7. 4-cycle-subgraphs have been treated in case M4, so that we can assume that A, B, C and D are pairwise different, and that $NA \cap NB = \emptyset$ and $NC \cap ND = \emptyset$.



$\lvert NB \cap NC\rvert = 2$	see M4			1.4143
$A \in NB \cup NC$	see M4			1.4143
$\emptyset$	ABC	1	4	1.3954
$\delta C = 3$	$ABNC$	1	6	
$\delta B = 3$	$ANBC$	1	6	
$NB \cap NC = \emptyset$	$ANBNC$	1	7	
$\delta A = 3$	NA	0	3	
$\lvert NB \cap NC\rvert = 1$	see C62			1.3954

Table 3. Case M7



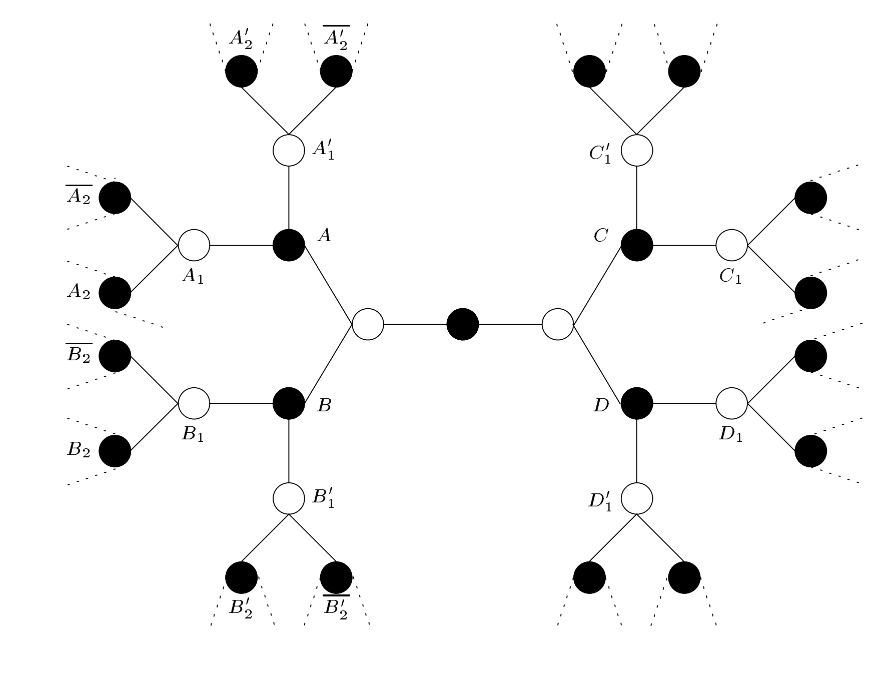
A'_1A_2BCD	2	7	1.3906
A'_1A_2BCND	2	9	
A'_1A_2BNC	2	8	
A'_1A_2BNC	1	6	
A'_1NA_2	0	4	
NA'_1	0	3	

Table 4. Case M8a: $\delta A_1 = 2$

Now, we distinguish cases making assumptions on the degree of A_1. If $\delta A_1 = 1$, we can refer to case M7, since X has "almost" two degree-2-neighbours, since there is a micro-tail at A.

Case M8a: $\delta A_1 = 2$. First, assume that $\delta A_1 = 2$. Then, we can assume that A'_1 has degree three, since otherwise A would have two degree-2-neighbours (M7). Further, we can assume $\delta A_2 = 3$, since otherwise there would be either a tail or a chain starting at A (M3 or M5). Finally, if $A'_1 \in NA_2$ or $B = A_2$ or $B \in NA'_1$, then we have a 4-cycle-substructure, see M4. The special cases $C = A_2$ or $D = A_2$ yield cycles of length 6 and are treated as C63 in the appendix. Similarly, if $C \in NA'_1$ or $D \in NA'_1$, then refer to case C64. In the main case treated in the picture, we can hence assume $\{C, D\} \cap (NA'_1 \cup NA_1) = \emptyset$. We do not repeat all these assumptions in Table 4.

Case M8b: $\delta A_1 = 3$. In fact, by symmetry, we can now assume that all neighbours of A, B, C and D have degree three. The corresponding picture is given in Table 5. Since we can again assume that a 4-cycle is no subgraph of our structure, each of the vertex sets $\{A, B, C, D\}$, $\{A_1, A'_1, B_1, B'_1\}$, $\{C_1, C'_1, D_1, D'_1\}$ contains



$A_1A'_1BCD$	2	7	$A_1NA'_1B_1NB'_1CD$	1	11	$NA_1B_1B'_1CND$	1	10	1.3976
$A_1A'_1BCND$	1	8	$A_1NA'_1B_1NB'_1CND$	1	13	$NA_1B_1B'_1NC$	1	9	
$A_1A'_1BNC$	1	7	$A_1NA'_1B_1NB'_1NC$	1	12	$NA_1B_1NB'_1CD$	1	10	
$A_1A'_1NB$	0	5	$A_1NA'_1NB_1CD$	1	10	$NA_1B_1NB'_1CND$	1	12	
$A_1NA'_1B_1B'_1CD$	2	10	$A_1NA'_1NB_1CND$	1	12	$NA_1B_1NB'_1NC$	1	11	
$A_1NA'_1B_1B'_1CND$	1	11	$A_1NA'_1NB_1NC$	1	11	NA_1NB_1CD	1	9	
$A_1NA'_1B_1B'_1NC$	1	10	$NA_1B_1B'_1CD$	2	9	NA_1NB_1CND	1	11	
						NA_1NB_1NC	1	10	

Table 5. Case M8b: $\delta A_1 = 3$

four elements. The case $\{A_1, A'_1, B_1, B'_1\} \cap \{C_1, C'_1, D_1, D'_1\} \neq \emptyset$ is treated as case C64 in the appendix, and the case $(NA_1 \cup NA'_1) \cap (NB_1 \cup NB_1) \neq \emptyset$ as C65 in the appendix. We do not repeat those assumptions in Table 5. (Horizontal lines in the branching list should help to structure that branching; they do not separate different branching lists.)

In conclusion, the analysis presented suggests that the 4- and 6-cycles remain "critical" when trying to get below 1.4^k. We deal with them in the appendix.

5 Conclusion

We presented the first nontrivial upper bound for CBVC, an algorithm running in time $O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$. Since it is only exponential in the (usually small) parameters k_1 and k_2, it is of high practical interest and contributes to the list of "efficient fixed parameter" algorithms [9]. As to future work, on the one hand, it remains to provide a competitive implementation of our algorithm. On the other hand, there is a variant of CBVC with three instead of two parameters, motivated by reconfigurable programmable logic arrays [**?**]. This problem deserves investigations as we performed for CBVC.

References

1. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.
2. 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.
3. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.
4. 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.
5. M. Hallett, G. Gonnet, and U. Stege. Vertex cover revisited: A hybrid algorithm of theory and heuristic. Manuscript, 1998.
6. E. A. Hirsch. Two new upper bounds for SAT. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 521–530, 1998.
7. S.-Y. Kuo and W. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, 4:24–31, Feb. 1987.
8. 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*.
9. 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.
10. 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.
11. 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, Nov. 1998.
12. 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.
13. J. M. Robson. Algorithms for Maximum Independent Sets. *Journal of Algorithms*, 7:425–440, 1986.
14. R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.
15. R. E. Tarjan and A. E. Trojanowski. Finding a Maximum Independent Set. *SIAM J. Comput.*, 6(3):537–550, 1977.

16. J. Wiedermann. Fast simulation of nondeterministic Turing machines with application to the Knapsack problem. *Computers and Artificial Intelligence*, 8(6):591–596, 1989.

Appendix: More details on cycles

The appendix contains all arguments missing in the main part of the paper concerning the claim that the search tree part of our CBVC algorithm runs in time $O(1.3999^{k_1+k_2})$.

Cycles of length 4 in detail I: The easy cases

As a special case, let us first treat the case C41 of two 4-cyles cycles combined as depicted in Table 6. Similar to the main case M4, we get the branching given in Table 6.

Since graph components with maximum degree two will be treated by the polynomial algorithm part following the search tree algorithm, this case is of no interest here. So, we treat the case that there exist at least two degree-2-vertices within the 4-cycle. If there are three degree-2-vertices (Case C42), we get the easy branch given in Table 6.

If there are two degree-2-vertices, we have to distinguish the cases whether those two vertices are neighboured or not (Cases C44 resp. C43). Remember that we can always presume $\delta B_1 > 1$ (in C44), $\delta A_1, \delta C_1 > 1$ (in C43) and, if necessary in C43, $\delta B_2, \delta C_2 > 1$ by the (micro-)tail-argument, referring then to cases M3 and C42.

Observe that only cases C42 and C44 have actually been used for the chain analysis in M5 of the main part of the algorithm, so that we can assume from now on that chains are not contained in the subpictures under consideration.

Case C61: A special cycle of length 6

Consider the figure given in Table 7. The branching is done using several subcases in order to get very good branching vectors. Indeed, those branchings are such good that we can afford to “create” a C61 subgraph by removing one neighbouring vertex X and all its (at least two) neighbours. In this way, we get the following branching behaviour:

Conditions	Branching vector	Basis
$\delta A_1 = 3, \delta C_1 = 3, A_1 \neq C_1$	$(6, 7, 4, 2)$	1.3993
$\delta A_1 = 3, \delta C_1 = 2, A_1 \neq C_1$	$(6, 7, 4, 2)$	1.3993
$\delta A_1 = 2, \delta C_1 = 2, A_2 \neq C_2$	$(6, 7, 7, 7, 2)$	1.3750
$A_1 = C_1$	$(5, 4, 2)$	1.3803
$A_2 = C_2$	$(5, 6, 2)$	1.3248

We will refer to this as the C61-trick in the following.

Cycles of length 4 in detail II: 4-cycle contains only one degree-2-vertex

The corresponding pictures can be seen in Table 8. Since we would like to refer to those pictures in later cases, too, we sketched an optional edge at node D. In our case here, of course, $\delta D = 2$ can be assumed.

ad C47 If $B \in NA_2$, then one can create a C61 situation using D as "X".
If $\delta A'_2 = 2$ and B (or D) equals A'_3, then we can create a C61 situation using C_1 as "X".

Cycles of length 4 in detail III: 4-cycle contains only degree-3-vertices

The first case (C48) we are going to consider is that one 4-cycle-vertex, w.l.o.g. A, has a degree-2-neighbour. Since we can exclude chains and tails, we can assume that $\delta A_2 = 3$, as depicted in Table 9. Further, we can assume $B, D \notin NA_2$, since otherwise the picture would include another 4-cycle with one degree-2-vertex.

Now, we can assume in case C49 that all 4-cycle-vertices have degree-3-neighbours. Assuming that all vertices in the corresponding picture in Table 9 are different, we obtain the branching given there, where in the first subcolumn of the third column we write the number of times symmetric cases occur; in total C49 contains a branch with sixteen different sons.

What equalities may occur? If $A_1 = C_1$ or $B_1 = D_1$, we refer to case C41.

If $A_1 \in NB_1$ (or symmetric cases), we get the situation given in case C49a in Table 10. In that picture, if $A_2 = D_1$, we can use the C61-trick by taking "$X = A_2$". Otherwise, we distinguish two cases in Table 10: $B_2 \in NA_2$ and $B_2 \notin NA_2$.

Finally, $NA_1 \cap NC_1$ might be not empty (or symmetric case $NB_1 \cap ND_1 \neq \emptyset$). If $NA_1 \cap N_C 1$ contain two common elements, the analysis given in case C46 works. If $NA_1 \cap N_C 1$ contains one element, the analysis in case $\delta A_2 = 2$ is given in Table 10. If $\delta A_2 = 3$, the analysis given in Table 8 hopefully works.

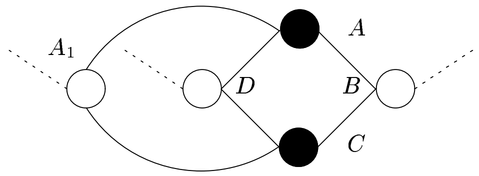
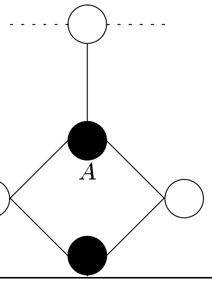
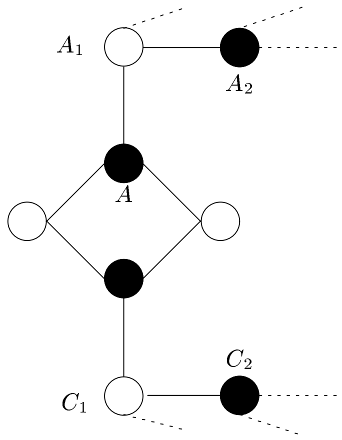
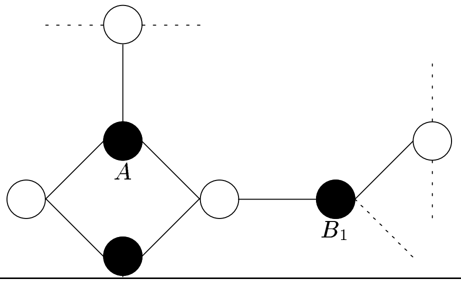
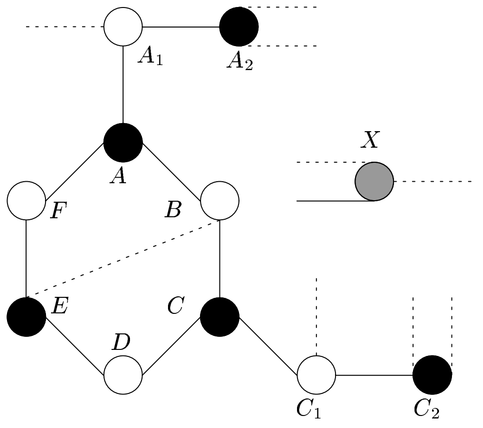
<table>
<tr><th>Case</th><th>Picture</th><th>Branching</th></tr>
<tr><td>C41</td><td> </td><td><table><tr><td>$\emptyset$</td><td>AC</td><td>0</td><td>2</td><td rowspan="2">1.3248</td></tr><tr><td>$\emptyset$</td><td>A_1DB</td><td>0</td><td>3</td></tr></table></td></tr>
<tr><td>C42</td><td> </td><td><table><tr><td>$\emptyset$</td><td>A</td><td>1</td><td>2</td><td rowspan="2">1.3248</td></tr><tr><td>$\emptyset$</td><td>NA</td><td>0</td><td>3</td></tr></table></td></tr>
<tr><td>C43</td><td> </td><td><table><tr><td>$A \in NC_1$</td><td colspan="3">see C41</td><td>1.3248</td></tr><tr><td>$\delta C_1 = 3$</td><td>AC_1</td><td>1</td><td>3</td><td rowspan="3">1.3954</td></tr><tr><td>$A \notin NC_1$</td><td>ANC_1</td><td>0</td><td>4</td></tr><tr><td>$\emptyset$</td><td>NA</td><td>0</td><td>3</td></tr><tr><td colspan="5">Now assume $\delta C_1 = \delta A_1 = 2$</td></tr><tr><td>$A_2 \neq C_2$</td><td>$A_2C_2$</td><td>2</td><td>4</td><td rowspan="3">1.3803</td></tr><tr><td>$\delta C_2 > 1$</td><td>A_2NC_2</td><td>2</td><td>5</td></tr><tr><td>$\delta A_2 > 1$</td><td>NA_2</td><td>0</td><td>2</td></tr></table></td></tr>
<tr><td>C44</td><td> </td><td><table><tr><td>$\emptyset$</td><td>AB_1</td><td>1</td><td>3</td><td rowspan="3">1.3954</td></tr><tr><td>$\delta B_1 > 1$</td><td>ANB_1</td><td>1</td><td>4</td></tr><tr><td>$\emptyset$</td><td>NA</td><td>0</td><td>3</td></tr></table></td></tr>
</table>

Table 6. Cycles of length four I



$\delta A_1 = 3$	A_1C_1	3	5	1.2786
$\delta C_1 = 3$	A_1NC_1	2	6	
$A_1 \neq C_1$	NA_1	0	3	
$\delta A_1 = 3$	A_1C_2	3	5	1.2786
$\delta C_1 = 2$	A_1NC_2	3	6	
$A_1 \neq C_1$	NC_2	0	3	
$\delta A_1 = 2$	A_2C_2	3	5	1.2740
$\delta C_1 = 2$	A_2NC_2	3	6	
$A_2 \neq C_2$	NA_2C_2	3	6	
$\emptyset$	NA_2NC_2	3	6	
$A_1 = C_1$	A_1B	2	4	1.2208
	AC	1	3	
$A_2 = C_2$	A_2	3	4	1.1673
	NA_2	3	5	

Table 7. C61: A special cycle of length 6

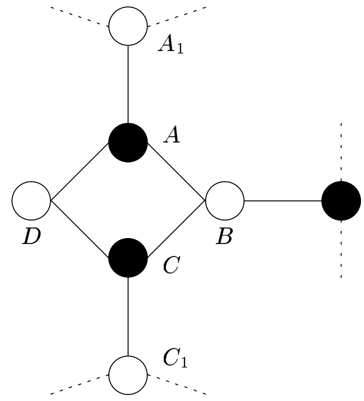
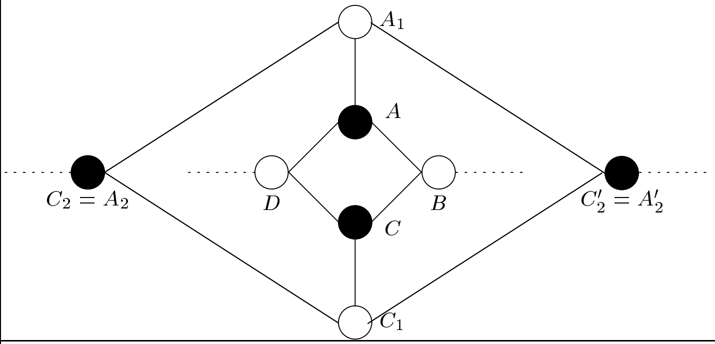
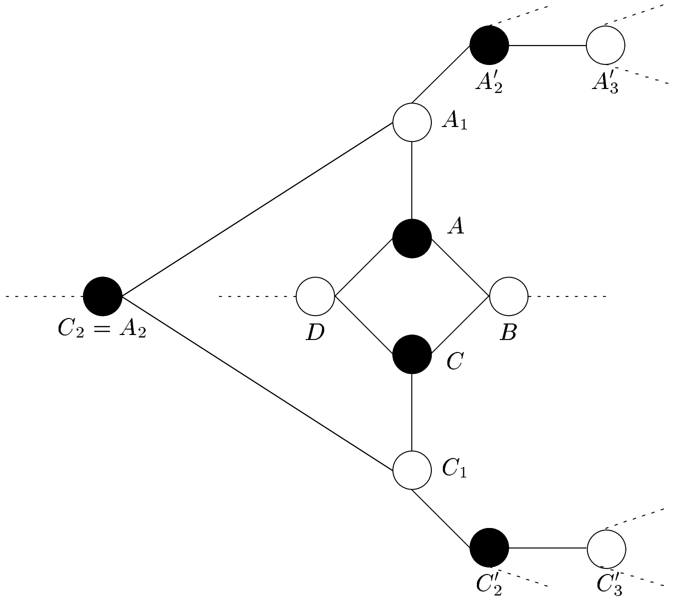
<table>
<tr><th>Case</th><th>Picture</th><th>Branching</th></tr>
<tr><td>C45</td><td>
</td><td>
<table>
<tr><td>$A_1 = C_1$</td><td colspan="3">see C41</td><td>1.3248</td></tr>
</table>
<table>
<tr><td colspan="5">Assume now $NA_1 \cap NC_1 = \emptyset$</td></tr>
<tr><td>$A_1 \neq C_1$</td><td>BA_1C_1</td><td>1</td><td>4</td><td rowspan="5">1.3954</td></tr>
<tr><td>$\emptyset$</td><td>BA_1NC_1</td><td>1</td><td>5</td></tr>
<tr><td>$\emptyset$</td><td>BNA_1C_1</td><td>1</td><td>5</td></tr>
<tr><td>$\emptyset$</td><td>BNA_1NC_1</td><td>0</td><td>7</td></tr>
<tr><td>$\emptyset$</td><td>NB</td><td>0</td><td>3</td></tr>
</table>
</td></tr>
<tr><td>C46</td><td>
</td><td>
<table>
<tr><td>$A_1 \neq C_1$</td><td>A_1C_1</td><td>0</td><td>2</td><td rowspan="3">1.3563</td></tr>
<tr><td>$\emptyset$</td><td>A_2C_2AC</td><td>0</td><td>4</td></tr>
<tr><td>$\emptyset$</td><td>A_2C_2BD</td><td>2</td><td>6</td></tr>
</table>
</td></tr>
<tr><td>C47</td><td>
</td><td>
<table>
<tr><td colspan="5">Assume $\delta A_2 = \delta A_1 = \delta C_1 = 3$:</td></tr>
<tr><td>$A'_2 \neq C'_2$</td><td>$BDA_2A'_2C'_2$</td><td>2</td><td>7</td><td rowspan="5">1.3971</td></tr>
<tr><td>$\delta C'_2 = 3$</td><td>$BDA_2A'_2NC'_2$</td><td>1</td><td>8</td></tr>
<tr><td>$\delta A'_2 = 3$</td><td>$BDA_2NA'_2$</td><td>0</td><td>6</td></tr>
<tr><td>$B \notin NA_2$</td><td>$BDNA_2$</td><td>0</td><td>5</td></tr>
<tr><td>$B \notin NC'_2$</td><td>AC</td><td>0</td><td>3</td></tr>
<tr><td>$B \in NC'_2$</td><td>$A_2A'_2C'_2$</td><td>2</td><td>5</td><td rowspan="4">1.3803</td></tr>
<tr><td>$\delta A'_2 > 1$</td><td>$A_2NA'_2C'_2$</td><td>2</td><td>6</td></tr>
<tr><td>$\delta C'_2 = 3$</td><td>$A_2NC'_2$</td><td>0</td><td>4</td></tr>
<tr><td>$\emptyset$</td><td>NA_2</td><td>0</td><td>3</td></tr>
<tr><td>$B \in NC'_2$</td><td>$A_2A'_2$</td><td>3</td><td>5</td><td rowspan="3">1.2786</td></tr>
<tr><td>$\delta A'_2 > 1$</td><td>$A_2NA'_2$</td><td>3</td><td>6</td></tr>
<tr><td>$\delta C'_2 = 2$</td><td>NA_2</td><td>0</td><td>3</td></tr>
<tr><td colspan="5">Further, let $\delta A'_2 = 2$; then $\delta A'_3 = 3$:</td></tr>
<tr><td>$A'_3 \neq B, D$</td><td>$BDA_2A'_3$</td><td>1</td><td>5</td><td rowspan="4">1.3911</td></tr>
<tr><td>$\delta A'_3 = 3$</td><td>$BDA_2NA'_3$</td><td>1</td><td>7</td></tr>
<tr><td>$\emptyset$</td><td>$BDNA_2$</td><td>0</td><td>5</td></tr>
<tr><td>$\emptyset$</td><td>AC</td><td>0</td><td>3</td></tr>
</table>
</td></tr>
</table>

Table 8. Cycles of length four II

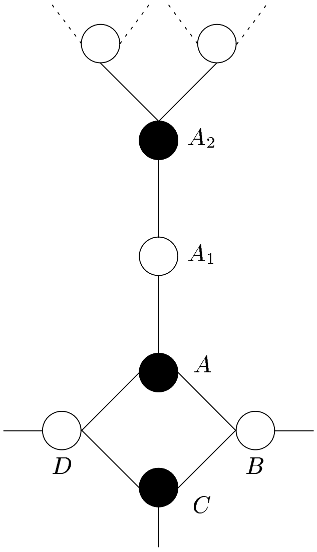
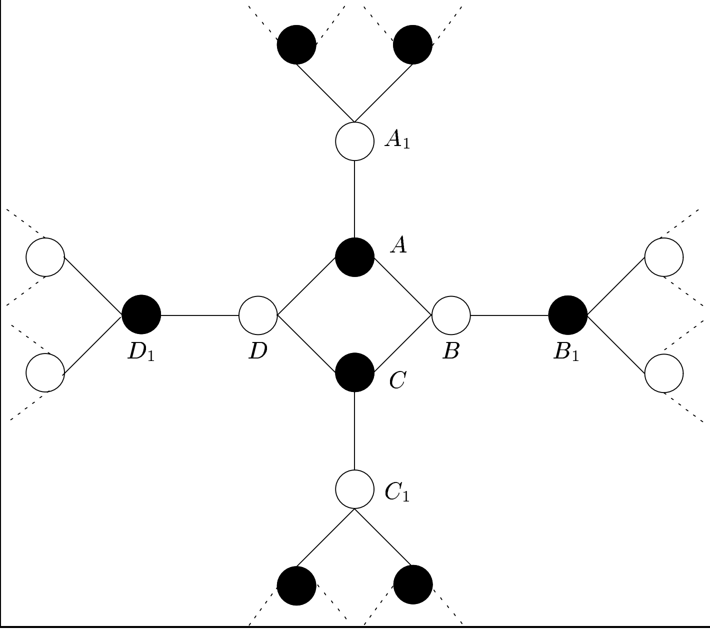
<table>
<tr><th>Case</th><th>Picture</th><th>Branching</th></tr>
<tr><td>C48</td><td>
</td><td><table>
<tr><td>∅</td><td>AC</td><td>0</td><td>2</td><td rowspan="3">1.3803</td></tr>
<tr><td>∅</td><td>BDA_2</td><td>1</td><td>4</td></tr>
<tr><td>∅</td><td>$BDNA_2$</td><td>0</td><td>5</td></tr>
</table></td></tr>
<tr><td>C49</td><td>
</td><td><table>
<tr><td>1</td><td>$A_1B_1C_1D_1$</td><td>2</td><td>6</td><td rowspan="6">1.3996</td></tr>
<tr><td>4</td><td>$A_1B_1C_1ND_1$</td><td>1</td><td>7</td></tr>
<tr><td>4</td><td>$A_1B_1NC_1ND_1$</td><td>1</td><td>9</td></tr>
<tr><td>2</td><td>$A_1NB_1C_1ND_1$</td><td>0</td><td>8</td></tr>
<tr><td>4</td><td>$A_1NB_1NC_1ND_1$</td><td>0</td><td>10</td></tr>
<tr><td>1</td><td>$NA_1NB_1NC_1ND_1$</td><td>0</td><td>12</td></tr>
</table></td></tr>
</table>

Table 9. Cycles of length four III

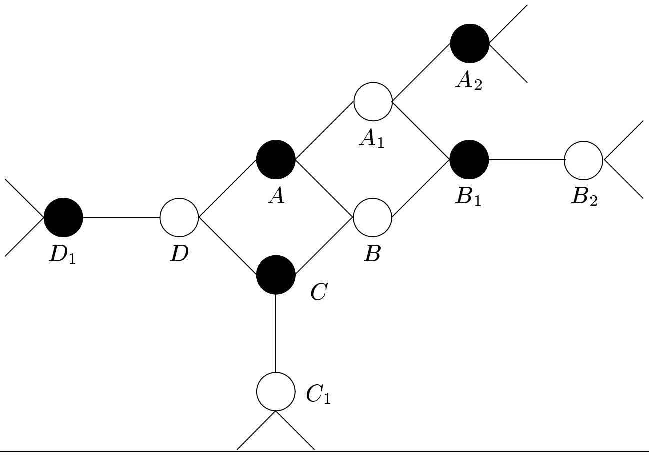
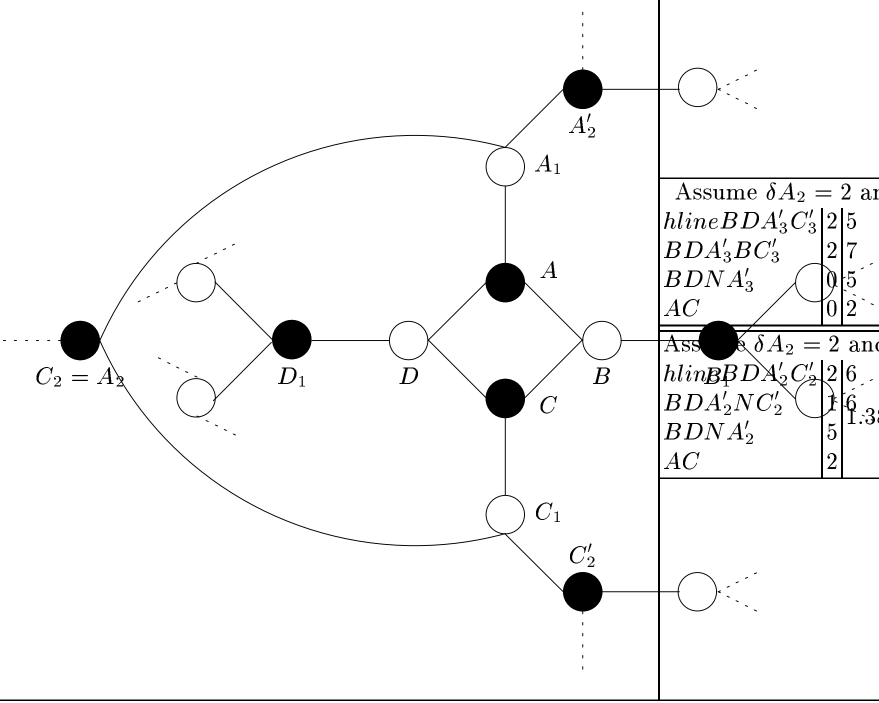
<table>
<tr><th>Case</th><th>Picture</th><th>Branching</th></tr>
<tr><td>C49a</td><td>
</td><td>
<table>
<tr><th colspan="4">Assume $B_2 \in NA_2$:</th></tr>
<tr><td>A_2B_1AC</td><td>0</td><td>4</td><td>1.3645</td></tr>
<tr><td>A_2B_1BD</td><td>1</td><td>5</td><td></td></tr>
<tr><td>A_1B_2AC</td><td>1</td><td>5</td><td></td></tr>
<tr><td>A_1B_2BD</td><td>0</td><td>4</td><td></td></tr>
</table>
<table>
<tr><th colspan="4">Assume $B_2 \notin NA_2$:</th></tr>
<tr><td>ACA_2B_2</td><td>1</td><td>5</td><td>1.3823</td></tr>
<tr><td>$ACNA_2B_2$</td><td>1</td><td>7</td><td></td></tr>
<tr><td>$ACNB_2$</td><td>0</td><td>5</td><td></td></tr>
<tr><td>BDA_2B_2</td><td>1</td><td>5</td><td></td></tr>
<tr><td>BDA_2NB_2</td><td>1</td><td>7</td><td></td></tr>
<tr><td>$BDNA_2$</td><td>0</td><td>5</td><td></td></tr>
</table>
</td></tr>
<tr><td>C49b</td><td>
</td><td>
<table>
<tr><th colspan="4">Assume $\delta A_2 = 2$ and $\delta A'_2 = \delta C'_2 = 2$:</th></tr>
<tr><td>$hlineBDA'_3C'_3$</td><td>2</td><td>5</td><td>1.3911</td></tr>
<tr><td>$BDA'_3BC'_3$</td><td>2</td><td>7</td><td></td></tr>
<tr><td>$BDNA'_3$</td><td>0</td><td>5</td><td></td></tr>
<tr><td>AC</td><td>0</td><td>2</td><td></td></tr>
</table>
<table>
<tr><th colspan="4">Assume $\delta A_2 = 2$ and $\delta A'_2 = 3, \delta C'_2 \geq 2$:</th></tr>
<tr><td>$hlineBDA'_2C'_2$</td><td>2</td><td>6</td><td></td></tr>
<tr><td>$BDA'_2NC'_2$</td><td>1</td><td>6</td><td></td></tr>
<tr><td>$BDNA'_2$</td><td>5</td><td>1.3854</td><td></td></tr>
<tr><td>AC</td><td>2</td><td></td><td></td></tr>
</table>
</td></tr>
</table>

Table 10. Cycles of length four IV