Journal of Algorithms, Vol. 38(2), pp. 374–410, 2001

An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover*

Henning Fernau† and Rolf Niedermeier‡
Wilhelm-Schickard-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 VLSI manufacturing.

We provide a 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. The analysis of the search tree is based on a novel bonus point system: after the processing of the search tree (which takes time exponential in k), a polynomial-time final analysis follows. Parts of the computation which would be normally done within the search-tree phase can such be postponed; nevertheless, the knowledge about the size of those parts can be used to reduce the length of the search paths (and hence the depth of the search tree as a whole) by sort of bonus points.

*An extended abstract of this paper was presented at the 24th *International Symposium on Mathematical Foundations of Computer Science* (MFCS'99), Springer-Verlag, *LNCS* 1672, pages 387–397, held in Szklarska Poręba, Poland, September 6-10, 1999.

†Partially supported by Deutsche Forschungsgemeinschaft grant DFG La 618/3-2.

‡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.

Abbreviated title. Constraint Bipartite Vertex Cover
Key words. *NP*-complete problem, graph algorithm, VLSI, vertex cover, fixed parameter algorithm
AMS subject classification. 05C85, 68Q35, 68R10, 05C70

1 Introduction

Nontrivial upper bounds for important *NP*-hard problems by exact algorithms have excited a continuous interest for many years [2, 6, 12, 24, 35, 39, 40, 42, 41, 47, 48]. With the advent of parameterized complexity theory [16] a special class of exact algorithms has become more and more important [3, 17]: As an example, consider the *NP*-complete Vertex Cover problem: given a 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 a running time of $O(1.211^n)$ [40]. There are, however, other exact ("fixed parameter") algorithms solving Vertex Cover in running time $O(1.29175^k + kn)$ [38] or even $O(1.271^k + kn)$ [11]. For instance, already for $k \leq n/2$ these are 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, since small values of k can often be assumed in applications [16, 17]. In this paper, we study another *NP*-complete variant of the Vertex Cover problem, motivated by reconfigurable VLSI [28]: 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 there is a close relation to the polynomial time solvable maximum matching problem for bipartite graphs [14, 28]), by a reduction from Clique it has been shown that CBVC is *NP*-complete [28]. Here, to our best knowledge, we give the first nontrivial fixed parameter algorithm for the Constraint Bipartite Vertex Cover problem running in time $O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$. We conjecture that, due to the different combinatorial structure in comparison with Vertex Cover, it should be very hard to get an exponential base close to the one there.

Our result makes the following two contributions: First, it provides a further example (of which there are still few) for a problem with an *effi-*

cient fixed parameter algorithm [3, 16, 17]. Second, our result is not only of theoretical interest, but it is also valuable with regard to practical applications. This is also due to the fact that, for the VLSI application which will be explained below, 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 its proven upper bound may successfully compete with heuristic algorithms in use, as first implementation tests indicate.

To achieve our result, we combine well-known methods from parameterized complexity [16], namely reduction to problem kernel and bounded search trees (these two methodologies can also be seen as the first two phases of our algorithm), with a new technique for restricting the size of the search tree by deferring some work to a third, polynomial-time phase, but nevertheless counting the parameter reduction as kind of bonus when processing the search tree and hence reducing its size. The techniques of reduction to problem kernel and of bounded search tree have already been successfully applied for Vertex Cover [5, 11, 17, 38], Maximum Satisfiability [33, 36], and elsewhere [16]. The main technical contribution of our work is the development of a search tree of size $1.3999^{k_1+k_2}$ by making use of the bonus point system mentioned above, 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. In Section 2, we review some previous work and give more details about the practical applications behind CBVC. In Section 3, we give some basic definitions and results. The heart of the paper is Section 4, where we explain the basic parts and the overall structure of our algorithm. In Section 5, we present details of our algorithm. However, additional details needed to achieve the exponential base 1.3999 are deferred to a clearly marked appendix (parts A, B, and C). In Section 6, we draw some final conclusions and make suggestions for future research.

2 Previous Work

Kuo and Fuchs [28] provide a fundamental study of the *spare allocation problem*. Put concisely, this "most widely used approach to reconfigurable VLSI" uses spare rows and columns to tolerate failures in rectangular arrays of identical computational elements, which may be as simple as memory cells or as complex as processor units (see [28] for details). If a faulty cell is detected, the entire row or column is replaced by a spare one. The underlying problem can be easily seen to be CBVC by identifying one vertex set of the bipartite graph with the rows, the other vertex set with the columns of the given ar-

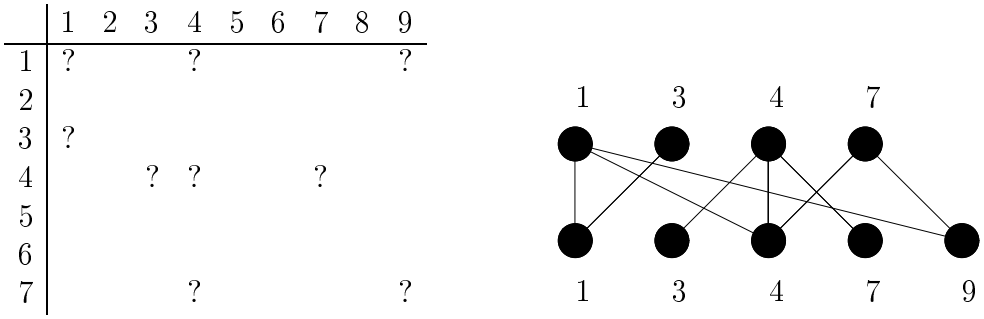


Figure 1: A 7×9 array with faults "?" and the bipartite graph model

ray, and each edge with a faulty cell. Kuo and Fuchs emphasize the practical relevance of the problem and the lack of efficient algorithms for it. Consider the following example of a reconfigurable array of size 7×9 with 2 spare rows and 3 spare columns in Fig. 1, where faulty cells are marked with a "?", together with its corresponding bipartite graph, where, e.g., an edge between 3 and 1 indicates the faulty cell in row 3 and column 1. Herein, we omit isolated vertices. Obviously, the array is repairable using one spare row (replacing row 4) and 3 spare columns (replacing columns 1, 4, 9). This corresponds to a solution of the CBVC problem with $k_1 = 1$ and $k_2 = 3$, as one can easily see. Typical examples in [28] are arrays with up to 1024 rows and 1024 columns (nowadays possibly even more) and with up to 20 additional spare rows and columns each time (cf. their Table 1), making this problem a natural candidate for a "fixed parameter approach" [3, 16, 17]. Kuo and Fuchs obtained the following results. First, they showed NP-completeness for CBVC by reducing Clique to it. Then they present two heuristic algorithms, first a branch-and-bound approach which always delivers optimal results, but is only "efficient for arrays with a moderate number of faulty cells" [28, page 29]. The second one is a polynomial time approximation algorithm. By way of contrast, our approach always delivers optimal results. Furthermore, its complexity basically depends on the number of available spare rows and columns.

It should be noted here that people developing algorithms for VLSI design actually discovered the FPT concept in the analysis of their algorithms, coming up with $O(2^{k_1+k_2}k_1k_2 + (k_1 + k_2)|G|)$ algorithms in [31, 21]. They observed that "if k_1 and k_2 are small, for instance $O(\log(|G|))$, then this may be adequate." [21, p. 157].

CBVC has important applications especially in the fabrication of high-end VLSI chips: With increasing integration in chip technology, a fault-free chip fabrication can no longer be assumed. So, fault-tolerance has to be taken

into consideration within the very fabrication process to obtain reasonable production yields [1, 10]. These ideas are already quite old, see [43] for an early treatment of the topic, but have continued to persist into the most recent developments in VLSI technology. Interestingly, the most challenging (parts of) high-end processors of today are the seemingly simplest of all possible VLSI chips, namely memories due to their rapidly expanding needs; moreover, e.g., they account for approximately 44% of transistors of the UltraSparc processor, so that they are used as technology and yield drivers [30, 49]. One common solution to increase yield in memory fabrication is to supply a few spare rows and columns to each memory array. These can be used in an internal (on-chip) or external reconfiguration phase (using lasers to "program" fuses) where faulty array rows or columns are replaced by spare ones, cf. the recent reports on DEC Alpha and UltraSparc processor manufacturing [7, 30, 49].

More work on the CBVC problem and related topics, mainly with respect to heuristic and average case aspects of the problem can be found in the papers [8, 9, 10, 13, 18, 20, 21, 22, 23, 25, 26, 27, 29, 31, 32, 44, 45, 46, 49].

3 Preliminaries

We assume familiarity with fundamentals from graph theory, algorithms, and complexity. It should be especially remembered that, given a graph in adjacency list representation, finding all components of the graph can be done in time linear to the representation size [34, Thm. IV.6.1], i.e., in time $O(n+m)$, where n is the number of vertices and m is the number of edges of the graph. A maximal matching in a bipartite graph is obtainable in time $O(\sqrt{n}m)$ [34, Thm. IV.9.10] (for more sophisticated algorithms, see [4, 19]). 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 G. By NX we denote the set of 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. If X and Y are vertices, then $\overline{XY}$ denotes the undirected edge between X and Y. In this paper, we only deal with bipartite graphs. For the ease of presentation, we treat them as two-colored (black and white) graphs with each vertex having a color opposite to all its neighbors.

Our algorithm works recursively. The number of recursions is the number of nodes in the corresponding search tree. This number is governed by homogeneous, linear recurrences with constant coefficients (cf. [24, 38]). If the algorithm solves a problem of size k and calls itself recursively for problems of sizes $k - d_1, \ldots, k - d_i$, then $(d_1, \ldots, d_i)$ is called the *branching vector* of

this recursion. It corresponds to the recurrence

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

The characteristic polynomial of this recurrence is

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

where $d = \max\{d_1, \ldots, d_i\}$. If α is a root of (2) with maximum absolute value, then t_k is α^k up to a polynomial factor. We call $|\alpha|$ the *branching number* which corresponds to the branching vector $(d_1, \ldots, d_i)$. Moreover, if α is a single root, then even $t_k = O(\alpha^k)$ and since we are only interested in estimating α, we can assume that branching numbers that 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 $(8, 9, 8, 10, 11, 10, 10, 11, 10, 10, 12, 11, 9, 10, 9, 10, 12, 11, 7, 9, 8, 9, 10, 9)$.

4 The algorithm—overview

Our algorithm works in basically the same way as the fixed parameter algorithms for Vertex Cover do [5, 17, 38]. 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 $2^{k_1+k_2}$. As a consequence, it is easy to see that this leads to an algorithm running in time $O(2^{k_1+k_2}(n+m))$, where n denotes the number of vertices and m the number of edges in the graph. All results (including ours) to get more efficient algorithms are based on efforts to shrink the search tree size. A special ingredient, however, of our algorithm is the use of bonus points to reduce the search tree size, i.e., we measure the expected parameter reduction for certain graph components which are analyzed in detail later with a polynomial-time algorithm. How this is done exactly, can be best clarified by the detailed description of the algorithm shown below.

The algorithm as a whole consists of three pieces:

1. A reduction to problem kernel;

2. a search tree processing;

3. a special treatment of graphs with maximum degree two and some larger graphs.

Only the second part of the algorithm is exponential.

As is usually the 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 instance, taking the neighbor of a degree-1-vertex will always lead to an optimal vertex cover. 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 vertex cover set sizes. In particular, the size of a minimal solution for CBVC is no longer uniquely determined, because the *signature* $s := (|C_1|, |C_2|)$ of a vertex cover C_1 and C_2 is a vector of numbers instead of simply a number. Our algorithm provides, however, for each minimal s a corresponding minimal solution.

4.1 Reduction to problem kernel

Before we give an overview of our algorithm, we still have to briefly explain a technique called *reduction to problem kernel* [15, 16], which is a kind of preprocessing. This step is based on a simple observation already used by Kuo and Fuchs [28], 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 a greater degree than k_2, then it has to be part of the vertex cover and, analogously, if a vertex in V_2 has a greater degree than k_1, then it also has to be part of the vertex cover. In this way, deleting all the "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 $2k_1k_2$. By assuming appropriate input data structures, reduction to problem kernel can be implemented to run in time $O((k_1+k_2)n)$, where $n = |V_1 \cup V_2|$ is the number of vertices in G. Combining this 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 [31, 21]. In the rest of the paper, we describe how to shrink the search tree size from $2^{k_1+k_2}$ to $1.3999^{k_1+k_2}$, a key issue in the development of efficient fixed parameter algorithms [3, 16].

4.2 Graphs with maximum degree two

Before we describe the overall structure of our search tree algorithm, let us briefly deal with an easy special case, namely graphs with maximum degree two. 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 lesser details. In addition, as previously mentioned, here we have to take into consideration that our given graph may be split into several connected components. Since the various components are "independent" of each other, we can simply combine them using component-wise addition and then again looking for the minimal values. Consequently, by 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 signatures 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 $O(k_1 + k_2)$. As a result of this, we have $(k_1 + k_2)$ merge steps each of time complexity $O((k_1 + k_2)^2)$. Summarized, this gives:

Proposition 4.1 For bipartite graphs with a maximum vertex degree of 2, CBVC can be solved in time $O((k_1 + k_2)^3)$.

Because of Proposition 4.1, in the following description of the basic structure of our search tree algorithm we may concentrate on graphs with a maximum degree of at least three.

In fact, there are certain (but only constantly many) graphs containing degree-3-vertices which are left over for the third, polynomial-time part of the whole algorithm by the search-tree part of the algorithm. Those cases will be treated in the first part of the Appendix.

4.3 Overall structure of the search tree algorithm

The rest of the paper proves the following theorem:

Theorem 4.2 CBVC can be solved in running time

$$O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$$

and polynomial space.

Our algorithm recursively finds optimal vertex covers as follows. Given a bipartite graph G, we choose several subgraphs $G_1,\ldots,G_i$ and compute optimal vertex covers for all of them. From these covers 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, then is 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 cover 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, this is how our algorithm works. We do however select the subgraphs $G_1,\ldots,G_i$ in a more complex manner and branch according to much more complicated sets. The rules how to select those branching sets are as follows. W.l.o.g. we assume that the graph is connected. We distinguish between eight main cases (M1–M8), some of them requiring further subcases. More details on these appear in Section 5 and the Appendix. 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.
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 once at most and thus does not influence the algorithm's asymptotic complexity. Similarly, if G contains only a small constant number of vertices of degree three, such a branching does not affect the overall time analysis.)

M3. Deal with tails of size at least two.
Branching vector and branching number: $(2, 3)$ and 1.3248.

M4. Deal with 4-cycles.[1]
Branching vector and branching number: $(2, 2)$ and 1.4143. Improved

[1] In the main part of the paper, we only give the result obtained from a very simple analysis. A refined one is deferred to the Appendix and yields the improved branching vector.

values (see Appendix): $(6,7,7,7,7,9,9,9,9,8,8,10,10,10,10,12)$ and 1.3996.

M5. Deal with chains of length at least three.
Branching vector and branching number: $(3,4,3)$ and 1.3954.

M6. Deal with degree-3-vertices with three neighbors of degree 2.
Branching vector and branching number: $(4,6,6,7,3)$ and 1.3954.

M7. Deal with degree-3-vertices with two neighbors of degree 2.
Branching vector and branching number: $(6,7,4,2)$ and 1.3993.

M8. Deal with degree-3-vertices with one neighbor of degree 2.
Branching vector and branching number:
$(8,9,8,10,11,10,10,11,10,10,12,11,9,10,9,10,12,11,7,9,8,9,10,9)$
and 1.3999.

The above steps can be easily shown to provide a complete case distinction handling all cases that may occur.[2] More specifically, from this point of view, steps M3–M5 would even be superfluous—they are, however, necessary in order to get a small search tree size by handling "nice special cases" in advance. The harder cases shown above are M4, M6, M7, and M8. The worst case branching vector occurs in M8 and implies a search tree size $1.3999^{k_1+k_2}$.

In total, we obtain a CBVC algorithm running in time

$$O(1.3999^{k_1+k_2}k_1k_2 + (k_1+k_2)n + (k_1+k_2)^3).$$

This can be improved to $O(1.3999^{k_1+k_2} + (k_1+k_2)n)$ by simple asymptotic arguments. However, the factor k_1k_2 above can be removed by means other than omission through "asymptotic tricks," such as refined complexity analysis, resulting in a slightly modified algorithm: repeated applications of the reduction to the problem kernel during the search tree processing are recommended by this sort of analysis, see [37].

In the rest of the paper, we provide the missing details for cases M3–M8. Such detailed study relies on the scrutinized analysis of 4-cycles and 6-cycles, so that the complete succession of cases is more complicated and given in Table 1. We will defer the concise description of the 4-cycle (C4x) and 6-cycle (C6x) cases to the Appendix.

Table 1 summarizes the complete case sequence and the corresponding (worst case) recursion bases. Observe that M4 has been replaced by the

[2] We will see in the following that degree-3-vertices with (at least) one degree-1-neighbor can be treated as if they were degree-2-vertices, so that we do not have to consider them in this main case distinction.

Case	Branching vector	Branching number
M1	$(1,4)$	1.3803
M2	(Can be ignored)	
M3	$(2,3)$	1.3248
C41	$(2,3)$	1.3248
C42	$(2,3)$	1.3248
C43	$(3,4,3)$	1.3954
C44	$(3,4,3)$	1.3954
M5	$(3,4,3)$	1.3954
C61	$(5,6,3)$	1.2786
C45	$(4,6,6,7,3)$	1.3954
C46	$(2,4,6)$	1.3563
C47	$(6,7,4,2)$	1.3993
C48	$(2,4,5)$	1.3803
C49	number: 6 7 9 8 10 12 times: 1 4 4 2 4 1	1.3996
M6	$(4,6,6,7,3)$	1.3954
M7	$(4,6,6,7,3)$	1.3954
C62	$(6,7,4,2)$	1.3993
M8a	$(7,9,8,6,4,3)$	1.3906
C63	$(6,7,6,9,6,8,3)$	1.3930
C64	$(6,7,7,9,6,8,3)$	1.3829
M8b	see Table 6	1.3976
C65	$(5,7,5,6,7,6,8,9,9)$	1.3982
C66	see Table 23	1.3999

Table 1: The sequence of cases with their branching vectors and branching numbers. For C49 the vector has 16 entries.

sequence of cases C41 - C49, and those cases are "interrupted" by the chain case M5. The worst case appears in case C66, but observe that lots of cases do not differ greatly from that branching number, leading us to believe that it would be hard to get close to an overall branching number of 1.3803 (the simple branching of case M1).

5 Details of the algorithm

Next, we present details of the main algorithm shown above. 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 to be a parameter of the main algorithm, which can be called $m(G, k)$. 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.

Because of steps M1 and M2 (cf. Section 4), 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.

5.1 Case M3: tails.

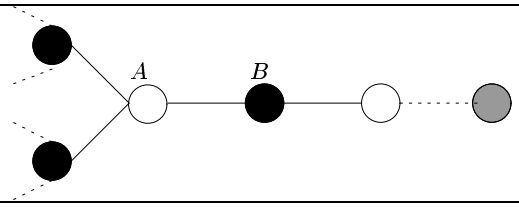
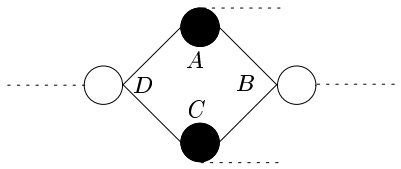
Picture	Branching
	A | 1 | 2 ; NA | 0 | 3 — 1.3248
	AC | 0 | 2 ; BD | 0 | 2 — 1.4143

Table 2: Cases M3 and M4

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 neighbor of a degree-1-vertex (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 2, where dashed edges and grey vertices are optional. Here, we encounter the *main trick* in our time analysis for the first time: we have seen that we can cope very easily with a graph having vertices of a degree at most two (Proposition 4.1). Therefore, if we take vertex A in the upper picture of Table 2, we create a non-empty path component starting at the degree-2-vertex B; in order to cover that component, we need at least one vertex from it 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 $m(G-NA, k-3)$. In other words: the exact analysis of the path component starting at vertex A is deferred to the third polynomial-time phase of the algorithm which mainly deals with all remaining degree-two components as a whole, see Section 4.2. This is the first time that this kind of bonus point system has been used and it will be employed frequently in the following. Observe that we could have collected even more bonus points by taking the length of the split-off path component into consideration, too, and such considerations are highly recommended for implementations of our algorithm. However, as we are only interested in worst-case analysis here, we can only count on one bonus point.

Note that in Table 2, the column labeled "Branching" contains the information necessary to understand the branching analysis. The first subcolumn lists the conditions under which the analysis is valid. Then, the vertices taken in that branch are listed; here, first A and then (in the second row) its neighbors. The third subcolumn lists how many vertices will be needed at least for the cover in the final (after the complete search tree has been constructed) polynomial time analysis of degree-2-vertices, i.e., here we count the bonus points. The fourth subcolumn gives the values which can be subtracted from the parameter k in the corresponding recursive call of the main procedure. Finally, the fifth subcolumn gives an upper estimate for the base of the search tree size derived from this case, implying search tree size 1.3248^k.

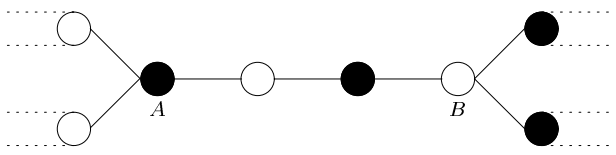
5.2 Case M4: 4-cycles (simple analysis).

Consider the corresponding picture from Table 2. Why is the given case distinction complete? Assume two neighboring vertices, 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 other than AC or BD, resp., into the cover. A much more detailed analysis of 4-cycles, given in the Appendix, allows for a branching vector $(6, 7, 7, 7, 7, 9, 9, 9, 9, 8, 8, 10, 10, 10, 10, 12)$ and branching number 1.3996.

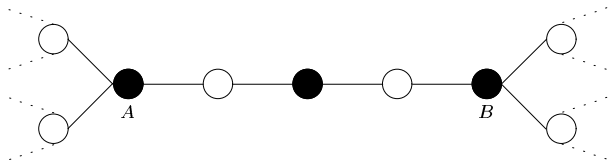
5.3 Case M5: chains of length at least 3.

If two (not necessarily 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 can say that A and B are connected by a *chain* of length ℓ. Chains of length 3 and 4 are considered in Table 3 (M5a and M5b).

Picture



Branching			
$A \in NB$	see M4		
$A \notin NB$	AB	1	3
$\delta A = 3$	ANB	0	4
	NA	0	3
1.3954			



$A = B$	see M4		
$A \neq B$	AB	1	3
$\delta B = 3$	ANB	1	5
$\delta A = 3$	NA	0	3
1.3640			

Table 3: Cases M5a and M5b

Should A and B be connected by a longer chain, the following branch analysis will occur:

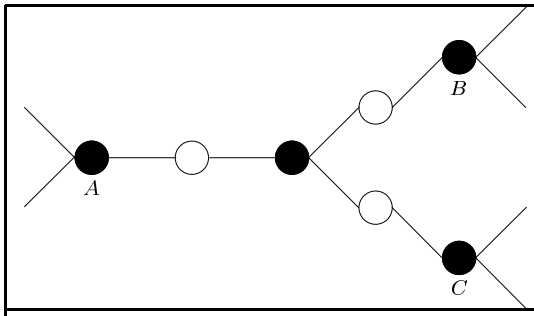
	AB	2	3	
$\delta B = 3$	ANB	1	4	1.3954
$\delta A = 3$	NA	0	3	

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

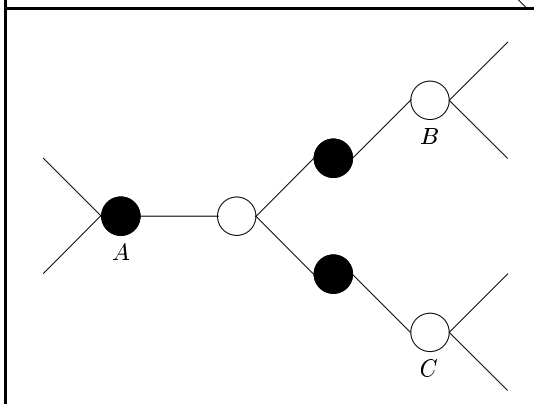
So, we can assume from now on that two degree-3-vertices are connected by a chain of length at most two. Bearing this in mind, we start analyzing the possible situations in the neighborhood of a degree-3-vertex with at least one degree-2-neighbor.

5.4 Case M6: one degree-3-vertex with three degree-2-neighbors.

In Table 4, the general situation is depicted in the first picture. We can assume $\delta A = \delta B = \delta C = 3$, since otherwise there is either a tail (M3) or a chain (M5).



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



$\lvert NB \cap NC \rvert = 2$	see M4		
$A \in NB \cup NC$	see M4		
	ABC	1	4
$\delta C = 3$	$ABNC$	1	6
$\delta B = 3$	$ANBC$	1	6
$NB \cap NC = \emptyset$	$ANBNC$	0	7
$\delta A = 3$	NA	0	3
1.3954			
$\lvert NB \cap NC \rvert = 1$	see App., C62		
1.3993			

Table 4: Cases M6 and M7

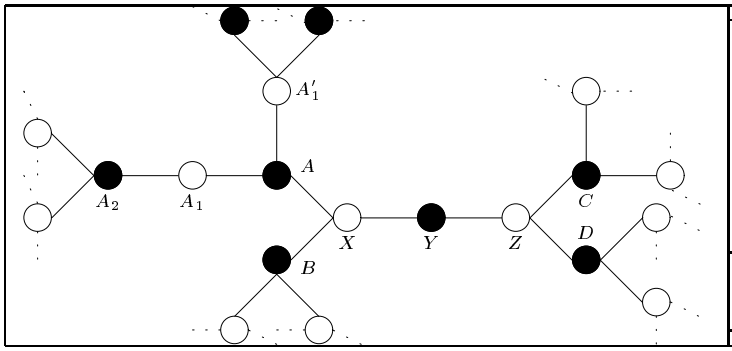
5.5 Case M7: one degree-3-vertex with two degree-2-neighbors.

Of course, since micro-tails at degree-3-vertices are like degree-2-vertices, the degree-3-vertex in question has one degree-3-neighbor called A. We refer to the second picture in Table 4. As to the case $|NB \cap NC| = 1$, we refer to the Appendix (Case C62) and just mention here that a worst case branching vector $(6, 7, 4, 2)$ with branching number 1.3993 is attainable.

5.6 Case M8: one degree-3-vertex with one degree-2-neighbor.

Consider the picture in Table 5. Each of both neighbors X, Z of the degree-2-vertex Y has two degree-3-neighbors (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 $NC \cap ND = \emptyset$.

Now, we can distinguish cases making assumptions on the degree of A_1.



$A_1'A_2BCD$	2	7
$A_1'A_2BCND$	2	9
$A_1'A_2BNC$	2	8
$A_1'A_2NB$	1	6
$A_1'NA_2$	0	4
NA_1'	0	3
Branching number: 1.3906		

Table 5: Case M8a: $\delta A_1 = 2$

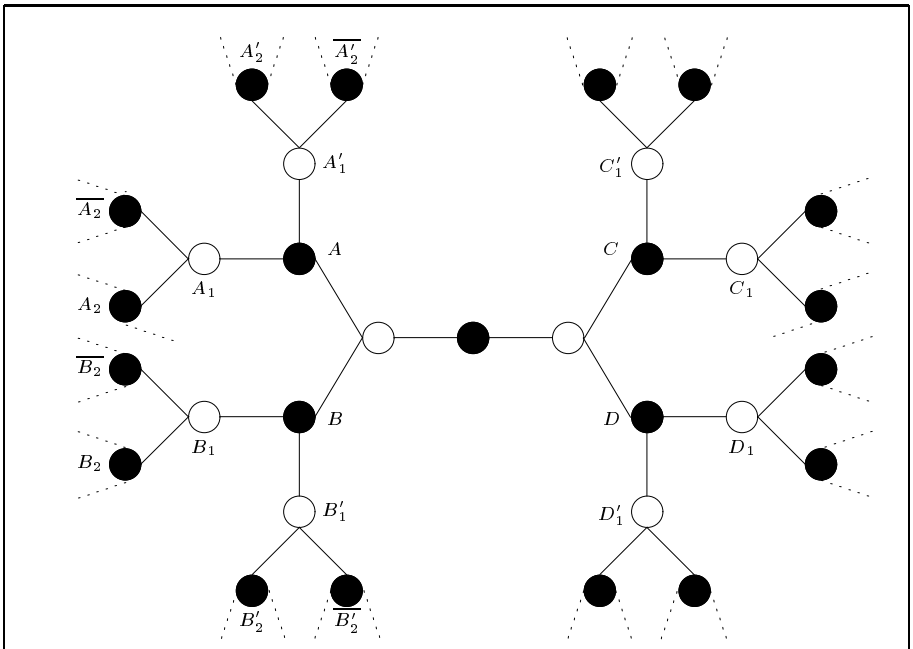
If $\delta A_1 = 1$, we can refer to case M7, since X has "almost" two degree-2-neighbors, due to the fact that there is a micro-tail at A.

Case M8a: $\delta A_1 = 2$**.** Here, we can assume that A_1' has degree three, since otherwise A would have two degree-2-neighbors (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. Two special cases occur: If $C = A_2$ or $D = A_2$, then we have a 6-cycle with two degree-2-vertices. If $C \in NA_1'$ or $D \in NA_1'$, then we have a 6-cycle with one degree-2-vertex. We defer the proofs to the Appendix (cases C63 and C64) and wish only to mention that branching vectors $(6,7,6,9,6,8,3)$ (branching number 1.3930) and $(6,7,7,9,6,8,3)$ (branching number 1.3829) can be achieved. In the main case dealt with in the picture, we can hence assume $\{C,D\} \cap (NA_1' \cup NA_1) = \emptyset$. These assumptions are only in part repeated in Table 5.

Case M8b: $\delta A_1 = 3$**.** In fact, through symmetry, we can now assume that all neighbors of A, B, C, and D have degree three. The corresponding picture is given in Table 6. Since we can again assume that a 4-cycle is not a 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 four elements. The special case $\{A_1,A_1',B_1,B_1'\} \cap \{C_1,C_1',D_1,D_1'\} \neq \emptyset$ (C65 in the Appendix) yields as branching vector $(5,7,5,6,7,6,8,9,9)$ (branching number 1.3982) and the special case $(NA_1 \cup NA_1') \cap (NB_1 \cup NB_1') \neq \emptyset$ (C66 in the Appendix) yields as branching vector

$$(8,9,8,10,11,10,10,11,10,10,12,11,9,10,9,10,12,11,7,9,8,9,10,9)$$

(branching number 1.3999). These assumptions are not repeated in Table 6. (Horizontal lines in the branching list should help to structure that branching;



$A_1A_1'BCD$	2	7	$A_1NA_1'NB_1CND$	1	12	
$A_1A_1'BCND$	1	8	$A_1NA_1'NB_1NC$	1	11	
$A_1A_1'BNC$	1	7	$NA_1B_1B_1'CD$	2	9	
$A_1A_1'NB$	0	5	$NA_1B_1B_1'CND$	1	10	
$A_1NA_1'B_1B_1'CD$	2	10	$NA_1B_1B_1'NC$	1	9	
$A_1NA_1'B_1B_1'CND$	1	11	$NA_1B_1NB_1'CD$	1	10	1.3976
$A_1NA_1'B_1B_1'NC$	1	10	$NA_1B_1NB_1'CND$	1	12	
$A_1NA_1'B_1NB_1'CD$	1	11	$NA_1B_1NB_1'NC$	1	11	
$A_1NA_1'B_1NB_1'CND$	1	13	NA_1NB_1CD	1	9	
$A_1NA_1'B_1NB_1'NC$	1	12	NA_1NB_1CND	1	11	
$A_1NA_1'NB_1CD$	1	10	NA_1NB_1NC	1	10	

Table 6: Case M8b: $\delta A_1 = 3$

they do not separate different branching lists.)

6 Conclusion

We have presented a nontrivial upper bound for CBVC, namely an algorithm running in time $O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$. Since it is exponential only 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 [3]. A first, still unsophisticated version of our algorithm has been implemented and delivers promising results. As to future work, however, a competitive implementation of our algorithm still remains to be achieved.

Let us point to three further things which are important for such a competitive implementation:

1. In the literature, various heuristics to prune the search tree at an early stage are presented (see the literature list at the end of Section 2). As a rule, however, these methods only cause average case improvements. The worst case remains unchanged.

2. Shi and Fuchs presented in [44] a polynomial-time algorithm for CBVC when the graph is a tree and gave results from the theory of random graphs indicating that tree components are very frequent. This can also be a starting point for the average-case analysis of such algorithmic types.

3. It is possible that horizontal and vertical movements of the repair laser are not equally fast, see [13]. Looking for the quickest repair solution corresponds to solving a restricted weighted version of CBVC. Since our algorithm yields all minimal signatures (and one solution per signature), a cover of minimal weight can easily be deduced from these signatures.

   Another criterion for a "good repair" may be to leave approximately as much spare rows as spare columns in order to simplify another later repair phase; again, such a solution can be deduced from the signatures.

For technological reasons, a memory board can be split into several regions each of which may have individual and/or common (shared) spare rows or columns with respect to other regions (for details, see [22, 29]). Trying to repair each such region independently and combining the signatures in an obvious manner leads to quite efficient reconfiguration algorithms, since the total number of spare rows and columns is not influencing the repair process as severely as in our seemingly special mode: only part of the spare rows and columns are available for each region, thus we each time have to deal with smaller parameter values each time, yielding smaller exponential factors.

Moreover, our approach can also be used when we drop the assumption that the spare rows and columns are always fault-free, see [25, 27].

Finally, there is a variant of CBVC with three instead of two parameters, motivated by reconfigurable programmable logic arrays [23]. This problem deserves investigation similar to that undertaken for CBVC.

Acknowledgments:

We are grateful to Peter Kadau, Klaus Reinhardt, and Peter Rossmanith for valuable discussions and to Mark Dokoupil and Martin Stark for helping us with the picture drawing. In addition, we are grateful to Jarik Nešetřil and DIMATIA, Prague, for making possible a stay of Henning Fernau in Prague, where this work was started.

Appendix A: Some special graphs with degree-3-vertices

Unfortunately, the search tree algorithm presented above does not leave only graphs—or maybe better components—with maximum degree two for the subsequent polynomial-time algorithm, but also some graphs with degree-3-vertices. As long as there is only a constant number of such exceptions, this is of course no problem. We will argue here that this is the case in our algorithm. As the reader may verify by closely analyzing the search-tree part of our algorithm, the only critical thing are path- or cycle-'like' subgraphs, where at least one of the vertices, which are "usually" of degree two, has three neighbors but one of them has degree one.

The shortest possible path of interest is depicted in Fig. 2.

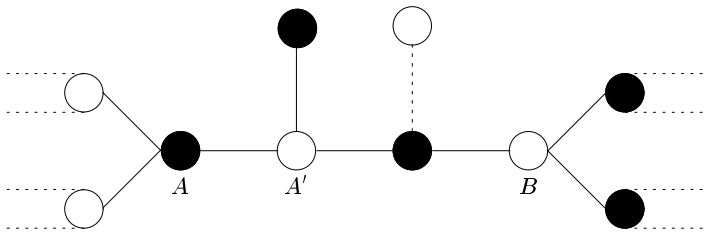


Figure 2: Case M5a with micro-tails

Taking into consideration the branch analysis which is given for case M5a (the case M4 works in analogy), where we branched according to the three cases AB, ANB, NA, one sees that only two components with maximum degree two remain for the polynomial-time part (namely in the first two cases AB and ANB).

Consider now a "path" of length four, cf. Fig. 3: namely, if, e.g., in Fig. 3 A' had degree 3, then it would be possible to consider the (shorter)

path between A' and B instead of the path between A and B, hence leading to case M5a above. Therefore, the most critical case is the one depicted in Fig. 3.

In case of a "path" of length greater than four, any degree-3-vertex with one degree-1-neighbor on that "path" would imply the existence of a shorter path, so that we can assume without loss of generality that "paths" of length greater than four do not contain degree-3-vertices.

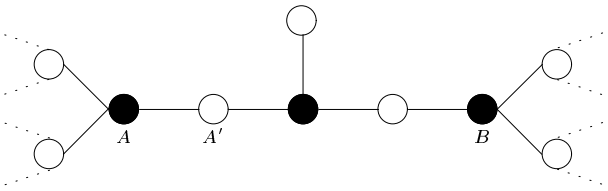


Figure 3: Case M5b with micro-tail

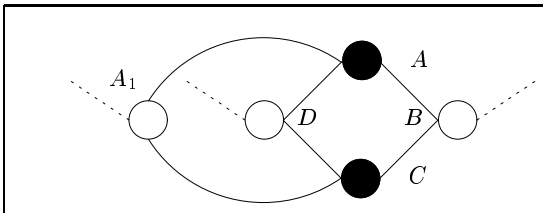
Similarly, looking at the branch analysis of case M5b, only one star-component remains for the polynomial-time section.

In conclusion, only a finite number of graph components containing degree-3-vertices actually remain for the polynomial-time section of the algorithm. We have refrained from listing all those cases in detail here. Of course, we can assume to know all minimal solutions and the corresponding signatures of those constantly many exception graphs, so that we can combine them to obtain the solutions for the whole graph as described above.

Appendix B: A detailed analysis of 4-cycles

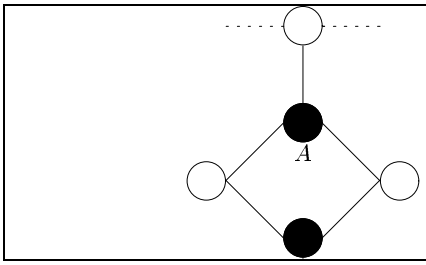
4-Cycles in detail I: The easy cases

As a special case, let us first treat the case C41 of two 4-cycles combined as depicted in Table 7. Similar to M4, we eperience the branching given in Table 7: In short, assume that both A and B are in the cover of that subgraph. Then, in order to cover edge $\overline{DC}$, either D or C must be in the cover. If C is in the cover, then the subcase $\{A, C\}$ from Table 7 will deal with it. If D is in the cover, then either A_1 or C is in the cover (for edge $\overline{A_1C}$). Both cases are handled by either the subcase $\{A_1, B, D\}$ or the subcase $\{A, C\}$ given in Table 7.



	AC	0	2	1.3248
	A_1DB	0	3	

Table 7: C41: A double 4-cycle



	A	1	2	1.3248
$\delta A = 3$	NA	0	3	

Table 8: C42: A 4-cycle with three degree-2-vertices

Since graph components with maximum degree two will be treated by the polynomial algorithm section following the search tree algorithm, this case is of no interest here.

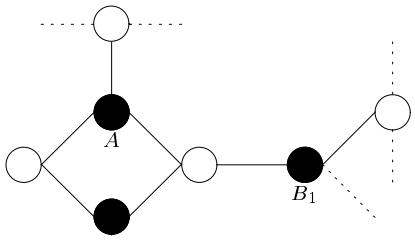
If there are three degree-2-vertices (Case C42), the easy branch given in Table 8 develops.

If there are two degree-2-vertices, we have to investigate the cases according to whether those two vertices are neighbors or not (Cases C43 resp. C44, cf. Table 9). Remember that we can always presume $\delta B_1 > 1$ (in C43), $\delta C_1 > 1$ (in C44) and, if necessary in C44, $\delta C_2 > 1$ by the (micro-)tail-argument, referring then to cases M3 and C42.

Observe that only cases C42 and C43 have actually been used for the chain analysis in M5 of the main part of the algorithm, so that we can in future assume that chains are not contained in the sub-pictures under consideration. In other words, in the following we can always assume that every degree-2-vertex we consider has two degree-3-neighbors.

Case C61: A very special 6-cycle

Consider the figure given in Table 10. Although the depicted figure is obviously a 6-cycle, it should be clear that it becomes a combination of two



	AB_1	1	3
$\delta B_1 > 1$	ANB_1	1	4
$\delta a = 3$	NA	0	3
1.3954			

$A \in NC_1$	see C41
Assume $A \notin NC_1$:	
Subcase $\delta C_1 = 3$:	

	AC_1	1	3
	ANC_1	0	4
$\delta A = 3$	NA	0	3
1.3954			

Subcase $\delta C_1 = 2$:

$A \neq C_2$	AC_2	1	3
$\delta C_2 > 1$	ANC_2	1	4
$\delta A = 3$	NA	0	3
1.3954			

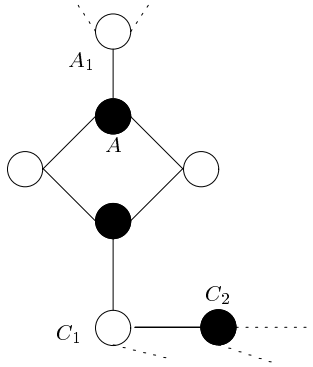


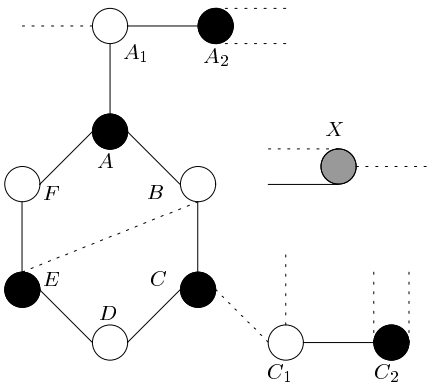
Table 9: 4-Cycles with two degree-2-vertices

4-cycles by inserting the optional edge $\overline{BE}$; this special case will be of importance in the subsequent analysis of 4-cycles.

As a result of the chain and tail argument, $\delta A_2 = 3$ could be assumed if $\delta A_1 = 2$, and similar $\delta C_2 = 3$ if $\delta C_1 = 2$. For reasons that will later become clear, we will use only the assumption that $\delta A_2 \geq 2$ in case $\delta A_1 = 2$, and similarly $\delta C_2 \geq 2$ if $\delta C_1 = 2$.

The branching is done using several subcases in order to get particularly good branching vectors. Additional explanatory remarks on the case distinctions given in Table 10 are as follows. First, we distinguish the case whether $\delta C = 2$ or $\delta C = 3$. In case $\delta C = 2$, we can assume that $\delta A = 3$; were it otherwise we would have a small graph component which we could afford to treat separately in the final polynomial-time algorithm. Observe that the case $\delta A = 2$ is analogous to the one discussed above.

We can as a result assume that both A and C are degree-3-vertices. We call the neighbors of A and C which are not part of the 6-cycle under



$\delta C = 2$ $\delta A = 3$	A NA	2 1	3 4	1.2208
Assume now $\delta A = \delta C = 3$:				
$A_1 = C_1$ $\delta A_1 > 1$	A_1 NA_1	3 2	4 4	1.1893
$A_1 \neq C_1$ $\delta C_1 = 3$ $\delta A_1 = 3$	A_1C_1 A_1NC_1 NA_1	3 2 0	5 6 3	1.2786
 $\delta C_1 = 2$ $\delta A_1 = 3$	A_1C_2 A_1NC_2 NA_1	3 3 0	5 6 3	1.2786
Assume now $\delta A_1 = \delta C_1 = 2$:				
$A_2 = C_2$ $\delta A_2 > 1$	A_2 NA_2	3 3	4 5	1.1673
$\delta A_1 = 2$ $\delta C_1 = 2$ $A_2 \neq C_2$ $A_1 \neq C_1$	A_2C_2 A_2NC_2 NA_2C_2 NA_2NC_2	3 3 3 3	5 6 6 6	1.2740

Table 10: C61: A very special 6-cycle

consideration A_1 and C_1, respectively. The special case $A_1 = C_1$ is quite easy, since $A_1 = C_1$ is the only possible contact to the rest of the graph. Trivially, $\delta A_1 \geq 2$, since A and C are then two different neighbors of $A_1 = C_1$. A similar argument applies in case $\delta A_1 = 1$ or $\delta C_1 = 1$ (and, *a fortiori*, in case $\delta A_1 = 2$ and $\delta A_2 = 1$, or $\delta C_1 = 2$ and $\delta C_2 = 1$); further details have been omitted here.

In case $A_1 \neq C_1$, we distinguish the following cases. First, it may be that A_1 and C_1 are of degree three. Secondly, we consider $\delta A_1 = 3$, $\delta C_1 = 2$. (Obviously, the case $\delta A_1 = 2$, $\delta C_1 = 3$ is symmetric and hence omitted.) For the second line of this case in Table 10 we have to argue that A_1NC_2 yields at least three vertices. If $\delta C_2 = 3$, this is trivial. If $\delta C_2 = 2$, we can assume $A_1 \notin NC_2$, since otherwise we would have a small component. In subcase $\delta C_2 = 3$, trivially three vertices are deleted in the mentioned second line. In conclusion, the case analysis of Table 10 is valid. Thirdly, consider $\delta A_1 = \delta C_1 = 2$. The case $A_2 = C_2$ is again an easy one. In case $A_2 \neq C_2$, we can assume $\delta A_2, \delta C_2 \geq 2$ by our remarks above. This concludes our case analysis.

Due to the development of especially good branching vectors, we are now

Conditions	Branching vector	Basis
$\delta C = 2, \delta A = 3$	$(4, 5, 2)$	1.3803
$\delta C = 3, A_1 = C_1$	$(5, 5, 2)$	1.3479
$\delta C = 3, \delta A_1 = 3, \delta C_1 = 3, A_1 \neq C_1$	$(6, 7, 4, 2)$	1.3993
$\delta C = 3, \delta A_1 = 3, \delta C_1 = 2, A_1 \neq C_1$	$(6, 7, 4, 2)$	1.3993
$\delta C = 3, \delta A_1 = 2, \delta C_1 = 2, A_2 = C_2$	$(5, 6, 2)$	1.3248
$\delta C = 3, \delta A_1 = 2, \delta C_1 = 2, A_2 \neq C_2$	$(6, 7, 7, 7, 2)$	1.3750

Table 11: C61: A very special 6-cycle plus a vertex X

capable of dealing with the following scenario. Assume that we do not yet have a C61 situation, but we could generate one by removing one other vertex X (which may be linked to vertices B, D, E or F). By trivial branching on X and NX, in the first case we now get a C61 situation and can do the analysis given in Table 10. Clearly, we may assume $\delta X \geq 2$. Of course, we must analyze the branching situation generated in this manner. This is done in Table 11. The case distinctions shown there are listed in the same order as in Table 10. The branching vectors given in Table 11 are obtained from the ones from Table 10 by adding 1 component-wise (namely, in case we take X) and appending one component 2 (for NX).

Observe that in the analysis in Table 10, in order to avoid unnecessary case distinctions when creating Table 11, we only assumed $\delta A_2 \geq 2$ in case $\delta A_1 = 2$ and $\delta C_2 \geq 2$ in case $\delta C_1 = 2$, as deleting X may create such a long chain or tail. Observe that if X equals A_1, A_2, C_1 or C_2, deleting X can only improve the analysis.

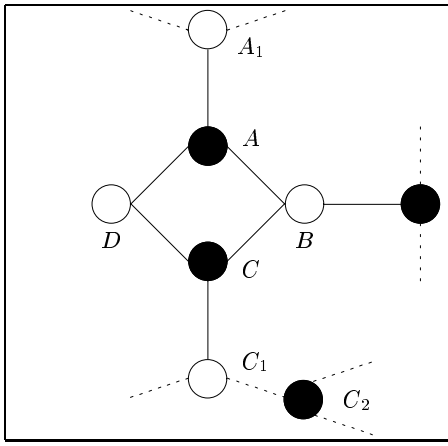
We will refer to this as the *C61-trick* in the following. We will always give the vertex we use as X and the 6-cycle (where the order of vertices would be $ABCDEF$ in the standard picture, i.e., the only vertices possibly having degree three are A and C which are mentioned in the first and third place in that sequence).

4-Cycles in detail II: C45, a 4-cycle which contains only one degree-2-vertex

Should a 4-cycle have one degree-2-vertex (w.l.o.g. at D), we distinguish three main cases:

C45: $|NA_1 \cap NC_1| = 0$;

C46: $|NA_1 \cap NC_1| = 2$;



$A_1 = C_1$	see C41			
Assume now $NA_1 \cap NC_1 = \emptyset$				
$A_1 \neq C_1$	BA_1C_1	1	4	
$\delta C_1 = 3$	BA_1NC_1	1	6	
$\delta A_1 = 3$	BNA_1C_1	1	6	1.3954
	BNA_1NC_1	0	7	
	NB	0	3	
Further, let $\delta C_1 = 2$				
	AC	0	2	
	BDC_2	1	4	1.3803
$\delta C_2 = 3$	$BDNC_2$	0	5	

Table 12: C45, a 4-cycle with one degree-2-vertex

C47: $|NA_1 \cap NC_1| = 1$.

The corresponding pictures can be seen in Tables 12, 13 and 14. The sequence chosen has algorithmic reasons; C47 is the most involved case.

The analysis given in Table 12 for the general case without interfering vertices should be quite clear. Observe that $\delta A_1, \delta C_1 > 1$ can be assumed by the (micro-)tail argument, possibly referring to the analysis of cases C42 or C43. We can distinguish between two basic subcases in Table 12, namely (1) $\delta C_1 = \delta A_1 = 3$ and (2) $\delta C_1 = 2$, as the case $\delta A_1 = 2$ is symmetric to (2). Since we can assume that long paths and tails are not present, we can infer that $\delta C_2 = 3$ in subcase (2). Moreover, observe that $\{B, D\} \cap NC_2 \neq \emptyset$ would lead to case C61, so that, in the last line of the analysis of subcase (2), $BDNC_2$ are actually 5 different vertices.

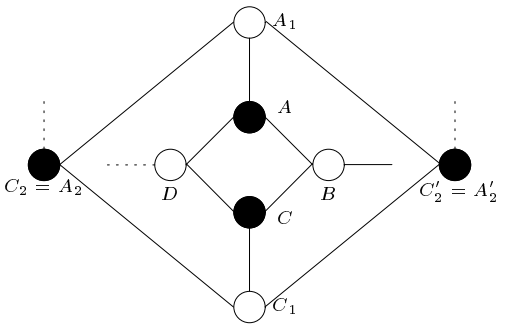
Since we would also like later to refer to the two cases C46 and C47, we sketched an optional edge at node D in Tables 13, 14. In this instance, of course, $\delta D = 2$ can be assumed.

In cases C46 and C47 (Tables 13, 14), we often employ the rather trivial 4-cycle-branch with branching vector $(2, 2)$ as explained in the rough analysis of M4 above. This branch does not assume that D has degree two, so that this is the case where we can re-use the present analysis later on.

Observe that due to the assumption $|NA_1 \cap NC_1| = 2$, in case C46 we can assume $\delta A_1 = \delta C_1 = 3$.

We will now explain the cases of C47 in more detail. Since we have assumed that $|NA_1 \cap NC_1| = 1$, we know that $\delta A_1, \delta C_1 > 1$.

Let us first discuss what might happen if $\delta A_1 = 2$. (The case $\delta C_1 = 2$ is again symmetric.) Now, if $\delta A_2 = 2$, then we have a chain between A and C_1



$A_1 = C_1$	see C41			
Assume now $\lvert NA_1 \cap NC_1 \rvert = 2$				
$A_1 \neq C_1$	A_1C_1	0	2	
	$A_2A_2'AC$	0	4	1.3563
	$A_2A_2'BD$	2	6	

Table 13: C46, a special case of the picture given in Table 12

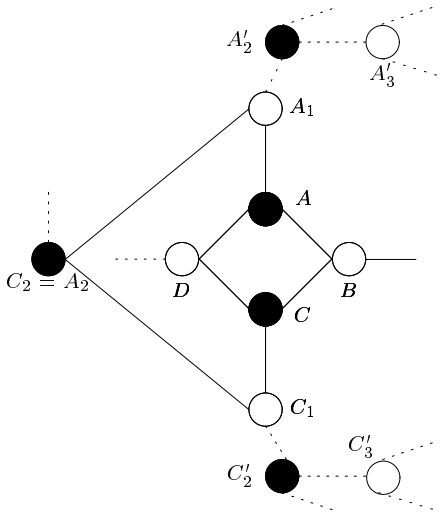


Table 14: C47, a special case of the picture given in Table 12

or C (depending on whether $\delta C_1 = 3$ or $\delta C_1 = 2$, respectively), a case which has already been covered by case M5. So, we can assume $\delta A_2 = 3$ whenever $\delta A_1 = 2$. Then, if $B \in NA_2$ (or symmetrically if $D \in NA_2$), we can create a C61 situation as explained under item 1. below. The case $B, D \notin NA_2$ is analyzed in Table 14.

In the following, we assume that $\delta A_1 = \delta C_1 = 3$. We start with five simple observations:

1. If $B \in NA_2$, then one can create a C61 situation $A_1A_2C_1CBA$ using D as "X". The case $D \in NA_2$ is symmetric.

2. If $\delta A'_2 = 2$ and B equals A'_3, then we can create a C61 situation $A_1ADCBA'_2$ using C_1 as "X". Again, $D \in NA'_2$ is symmetric.

3. By C46, $A'_2 \neq C'_2$ can be assumed.

4. If $\delta A_2 = 2$ and $B \in NA'_2$, we create a C61 situation $C_1CDAA_1A_2$ using A'_2 as "X". (Actually, there is an additional vertex B connected to A and C, but B is not connected to anywhere else after deleting A'_2, so that the C61 analysis could only improve.) Symmetrically, the case $\delta A_2 = 2$ and $B \in NC'_2$ can be treated.

5. If $\delta D = 2$ and $B \in NA'_2$, we create a C61 situation A'_2BCDAA_1 using A_2 as "X".

Observe that it is only in the last item listed above that D has degree two. By using these preliminary observations, the following four cases remain:

1. In case $\delta A_2 = 3$ and $\delta A'_2 = \delta C'_2 = 3$, we can assume the following:
   - $B, D \notin NA_2$ (by the first observation),
   - $B, D \notin NA'_2 \cup NC'_2$ (by the fifth observation),
   - $A'_2 \neq C'_2$ (by the third observation).

2. In case $\delta A_2 = 3$ and $\delta A'_2 = 2$, we can assume:
   - By the (micro-)tail argument, we can assume $\delta A'_3 = 3$.
   - Further, $B, D \notin NA'_2$ (by the second observation).

3. In case $\delta A_2 = 2$ and $\delta A'_2 = \delta C'_2 = 2$, we can assume the following:
   - $B, D \notin NA'_2 \cup NC'_2$ (by the fourth observation).

4. In case $\delta A_2 = 2$, $\delta A'_2 = 3$ and $\delta C'_2 \geq 2$, we can assume:
   - $A'_2 \neq C'_2$ (by the third observation),
   - $B, D \notin NA'_2 \cup NC'_2$ (by the fourth observation).

The detailed analysis of these four cases in contained in the list in Table 15.

Table 14 shows the picture. In none of those studies, $\delta D = 2$ is used.

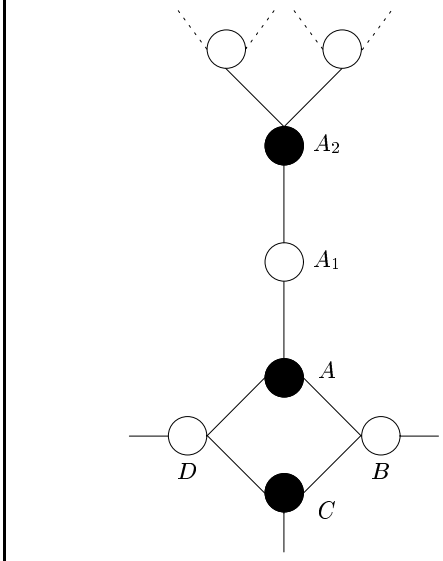
Assume $\delta A_2 = 3$ and $\delta A_1 = 2$:				
$D \notin NA_2$	BDA_2	1	4	1.3803
$B \notin NA_2$	$BDNA_2$	0	5	
	AC	0	2	
Assume now $\delta A_1 = \delta C_1 = 3$:				
Case 1: $\delta A_2 = \delta A'_2 = \delta C'_2 = 3$:				
$A'_2 \neq C'_2$	$BDA_2A'_2C'_2$	2	7	1.3971
$\delta C'_2 = 3$	$BDA_2A'_2NC'_2$	1	8	
$\delta A'_2 = 3$	$BDA_2NA'_2$	0	6	
$B, D \notin NA_2$	$BDNA_2$	0	5	
$B, D \notin NC'_2$	AC	0	2	
Case 2: $\delta A_2 = 3$, $\delta A'_2 = 2$:				
$A'_3 \notin \{B, D\}$	$BDA_2A'_3$	1	5	1.3911
$\delta A'_3 = 3$	$BDA_2NA'_3$	1	7	
$B, D \notin NA_2$	$BDNA_2$	0	5	
	AC	0	2	
Case 3: $\delta A_2 = \delta A'_2 = \delta C'_2 = 2$:				
$B \notin \{A'_3, C'_3\}$	$BDA'_3C'_3$	2	5	1.3911
$D \notin \{A'_3, C'_3\}$	$BDA'_3NC'_3$	2	7	
$\delta A'_3 = 3$	$BDNA'_3$	0	5	
$\delta C'_3 = 3$	AC	0	2	
Case 4: $\delta A_2 = 2$, $\delta A'_2 = 3$, $\delta C'_2 \geq 2$:				
$A'_2 \neq C'_2$	$BDA'_2C'_2$	2	6	1.3854
$B, D \notin NC'_2$	$BDA'_2NC'_2$	1	6	
$B, D \notin NA'_2$	$BDNA'_2$	0	5	
	AC	0	2	

Table 15: The branching in case C47

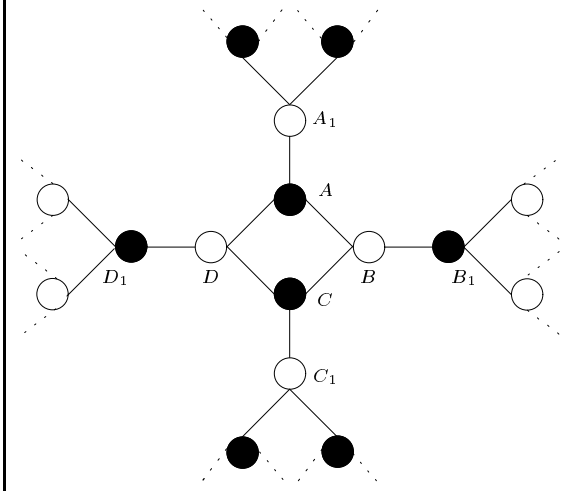
4-Cycles in detail III: the 4-cycle contains only degree-3-vertices

The first case (C48) we are going to consider possesses one 4-cycle-vertex, w.l.o.g. A, has a degree-2-neighbor. Since we can exclude chains and tails, we can assume that $\delta A_2 = 3$, as depicted in Table 16. Further, we can assume $B, D \notin NA_2$, since otherwise the picture would include another 4-cycle with one degree-2-vertex, a case which has just been dealt with.

We can now assume in case C49 that all 4-cycle-vertices have degree-3-neighbors. Assuming that all vertices in the corresponding picture in Table 16 are different, we obtain the branching given there, where in the first



	AC	0	2	
	BDA_2	1	4	1.3803
	$BDNA_2$	0	5	



1	$A_1B_1C_1D_1$	2	6
4	$A_1B_1C_1ND_1$	1	7
4	$A_1B_1NC_1ND_1$	1	9
2	$A_1NB_1C_1ND_1$	0	8
4	$A_1NB_1NC_1ND_1$	0	10
1	$NA_1NB_1NC_1ND_1$	0	12

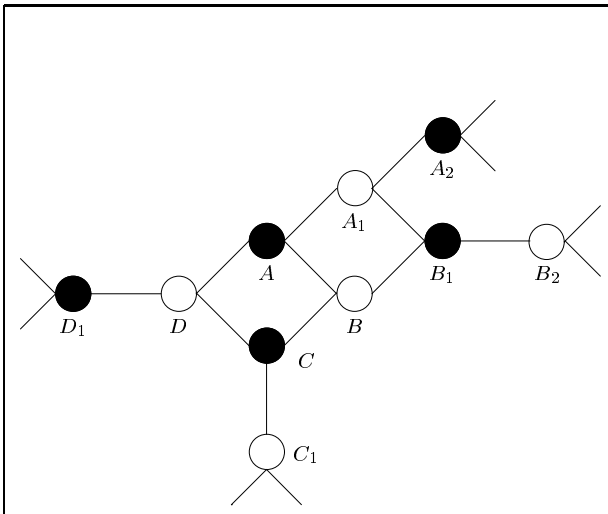
Basis: 1.3996

Table 16: 4-Cycles III: all 4-cycle-vertices have degree three: C48 and C49

subcolumn of the third column we write the number of times symmetric cases occur; in total C49 contains a branch with sixteen different children. This case gives the worst branch of the whole 4-cycle-analysis.

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), the situation given in case C49a



Assume $B_2 \in NA_2$:		
A_2B_1AC	0	4
A_2B_1BD	1	5
A_1B_2AC	1	5
A_1B_2BD	0	4
1.3645		
Assume $B_2 \notin NA_2$:		
ACA_2B_2	1	5
$ACNA_2B_2$	1	7
$ACNB_2$	0	5
BDA_2B_2	1	5
BDA_2NB_2	1	7
$BDNA_2$	0	5
1.3823		

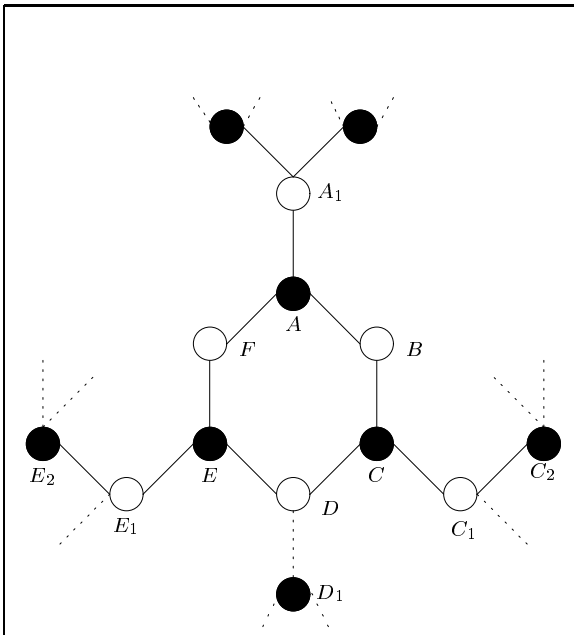
Table 17: C49a: a special 4-cycle

in Table 17 occurs. Observe that we can assume that all vertices part of or neighboring any 4-cycle have degree three in previous case studies. In case C49a, if $A_2 = D_1$, we can use the C61-trick by taking "$X = A_2$" and the 6-cycle CBB_1A_1AD. Otherwise, we distinguish two cases in Table 17: $B_2 \in NA_2$ 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 $|NA_1 \cap NC_1| = 2$, the analysis given in case C46 works. If $|NA_1 \cap NC_1| = 1$, the analysis given for case C47 works with one exception: that is, only in the fifth observation there we assumed that $\delta D = 2$ could be assumed in case C47. That in turn has only been used in case $\delta A_2 = 3$, $\delta A_2' = \delta C_2' = 3$ together with the further assumptions $B, D \notin NA_2$, $B, D \notin NA_2' \cup NC_2'$ (by the fifth observation), $A_2' \neq C_2'$, see Table 14. In the notation of Table 16, the assumptions $B, D \notin NA_2$ as well as $B, D \notin NA_2' \cup NC_2'$ correspond to the case "$A_1 \in NB_1$" treated in Table 17, so that we were actually able to complete our analysis of case C49 and hence of all possible 4-cycles.

Appendix C: A detailed analysis of 6-cycles

In the analysis of the main course of the algorithm, several 6-cycles have not been analyzed. We will do this here. In contrast to the analysis of cycles of length 4, as shown above, this analysis does not treat all possible 6-cycles



$\delta D = 2$	Use C61-trick with "$X = A_1$".		
1.3993			
Assume now $\delta D = 3$:			
$\delta E_1 \geq 2$ $\delta C_1 = 3$	$A_1DC_1E_1$	2	6
	$A_1DC_1NE_1$	2	7
	A_1DNC_1	0	5
	A_1ND	1	5
	NA_1	0	3
1.3889			
$\delta C_1 = \delta E_1 = 2$ $\delta C_2 = \delta E_2 = 3$ $\{A_1\} = NC_2 \cap NE_2$			
	DC_2E_2	3	6
	DC_2NE_2	3	8
	DNC_2	0	4
	ND	0	3
1.3384			

Table 18: C62: A 6-cycle for case M7

but only those which actually occur as special cases in our algorithm.

C62: A 6-cycle for case M7

In case M7, the structure given in Table 18 had yet to be analyzed. Note that we can now assume that 4-cycles can be analyzed efficiently, so that especially all white vertices occurring in the picture of Table 18 are pairwise different. We made the following distinction of subcases in Table 18:

1. If $\delta D = 2$, then we can create a C61 situation using A_1 as X with respect to the 6-cycle $CDEFAB$. In the course of the subsequent analysis, we can assume $\delta D = 3$.

2. Similarly, if $\delta E_1 = 1$, then either we already have a C61 situation "with micro-tails" (in case $\delta D_1 = 1$) or we can create one "with micro-tails" if $\delta D_1 > 1$ by taking D_1 as X with respect to the 6-cycle $ABCDEF$. The case $\delta C_1 = 1$ is symmetric and $\delta A_1 = 1$ is analogous. In the following, we can therefore assume that A_1, C_1 and E_1 each have degree of at least two.

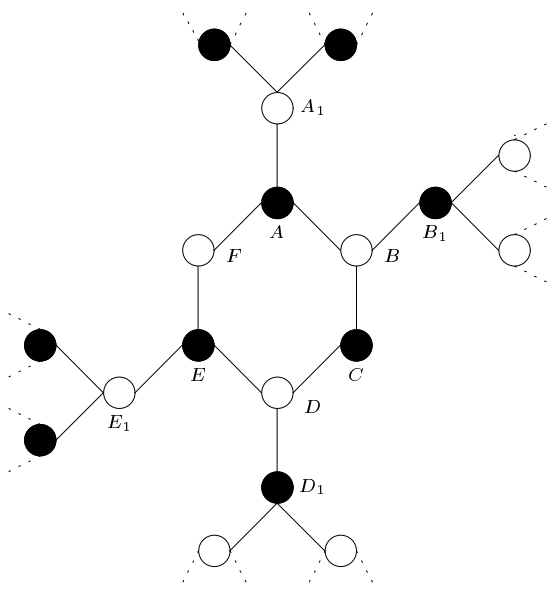
3. If $\delta E_1 \geq 2$ and $\delta C_1 = 3$, we refer to the analysis given in Table 18. Obviously, the case $\delta E_1 = 3$ and $\delta C_1 \geq 2$ is symmetric. Therefore, the following case remains:

4. If $\delta E_1 = \delta C_1 = 2$, then we can assume that $\delta C_2 = \delta E_2 = 3$, since otherwise we would have tails or chains, which are done already. Furthermore, if $NA \cap NC_2 = \emptyset$, we are in the same situation as that of the main case M7 which can be assumed to have already occurred: namely, we have one degree-3-vertex C neighbored by two degree-2-vertices. The case $NA \cap NE_2 = \emptyset$ is treated symmetrically.

   Therefore, the only subcase that remains to be analyzed is: $NA \cap NC_2 \neq \emptyset$ and $NA \cap NE_2 \neq \emptyset$. Obviously, this means that $A_1 \in NC_2 \cap NE_2$. No further vertex is in this intersection, since we can assume the absence of 4-cycles.

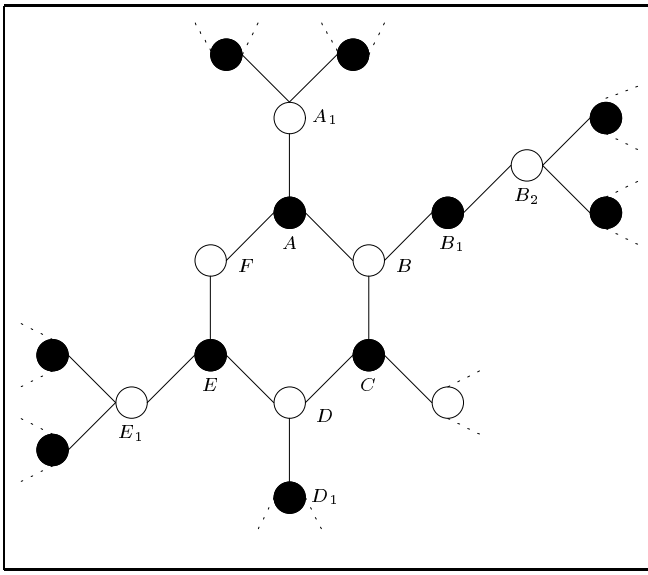
Note that now (having carried out M7) we can assume that not only neighbors and neighbors of neighbors, but also neighbors of neighbors of neighbors of degree-2-vertices are degree-3-vertices.

Two 6-cycles for case M8a



$A_1DB_1E_1$	2	6	
$A_1DB_1NE_1$	2	7	
$A_1DNB_1E_1$	1	6	
$A_1DNB_1NE_1$	1	9	1.3930
A_1NDB_1	1	6	
A_1NDNB_1	1	8	
NA_1	0	3	

Table 19: C63: A special 6-cycle for case M8a



A_1CB_2E	2	6
A_1CB_2NE	1	7
A_1CNB_2E	1	7
A_1CNB_2NE	1	9
A_1NCE_1	1	6
A_1NCNE_1	1	8
NA_1	0	3
1.3829		

Table 20: C64: A special 6-cycle for case M8a

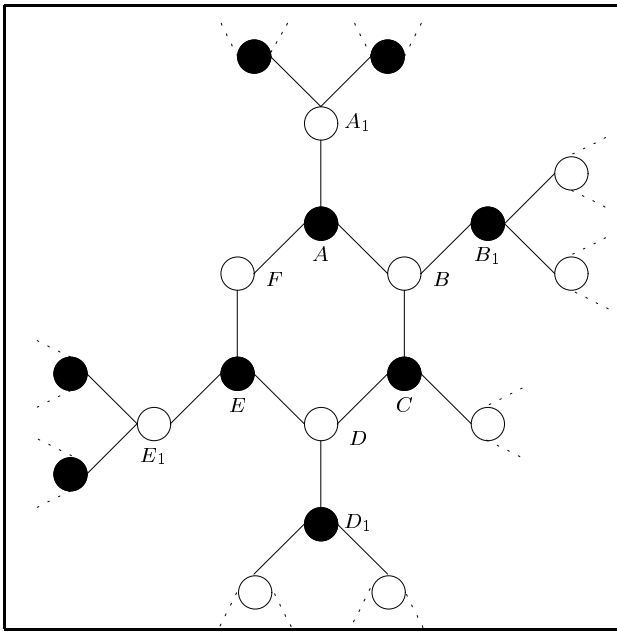
In case M8a, two 6-cycle cases remained to be analyzed (in the given order): the case C63 of a 6-cycle with two degree-2-vertices (which have to be arranged as in Table 19 since M7 has been done before; moreover, $\delta A_1 = \delta B_1 = \delta D_1 = \delta E_1 = 3$ can be assumed because of M7; this implies that for 6-cycles with two degree-2-vertices, the analysis is complete) and a special case C64 of a 6-cycle with one degree-2-vertex, see Table 20.

As regards case C63, we can suppose that no 4-cycles are present, which means that $\{A_1, E_1\}$ and $\{B_1, D_1\}$ each contain two elements, and $A_1 \notin NB_1$. Note that possibly $B_1 \in NE_1$.

Regarding case C64, observe that if $B_2 = E_1$, we find a structure already treated in C63, namely the 6-cycle ABB_1E_1EF, so that we can assume $B_2 \neq E_1$ (i.e., $B_2 \notin NE$) in the analysis given in Table 20. Further, recall that the main case M8a forces B_1 to be of degree two in case C64. (A similar situation where B_1 has degree three occurs as a subcase for M8b, see below.)

Two 6-cycles for case M8b

In case M8b, two other 6-cycle cases are to be analyzed. The first (C65) is quite similar to C64; now, $\delta B_1 = 3$ is assumed, see Table 21. In that table, we assumed that, if $A_1 \in ND_1$, then $B_1 \notin NE_1$. Therefore, in the second line, we subtracted one from the best possible 9-branch (assuming $A_1 \in ND_1$), but nothing in the third line of the analysis. In the fourth line,



$CA_1E_1B_1D_1$	2	7	
$CA_1E_1B_1ND_1$	2	8	
$CA_1E_1NB_1D_1$	2	9	
$CA_1E_1NB_1ND_1$	1	8	
$CA_1NE_1B_1$	1	7	
$CA_1NE_1NB_1$	1	9	
NA_1	0	3	
NCA_1E_1	1	6	
NCA_1NE_1	1	8	
			1.3969

Table 21: C65: A special 6-cycle for case M8b

we even subtracted two, assuming $A_1 \in ND_1$ and $NB_1 \cap ND_1 \neq \emptyset$. Note that the roles of A_1 and D_1 as well as E_1 and B_1 are interchangeable.

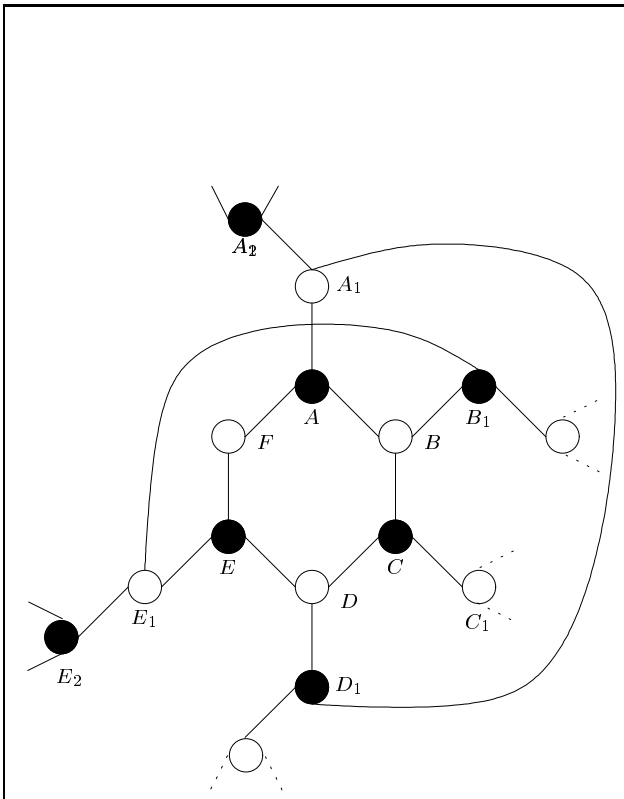
By this sort of analysis, the instance where $A_1 \in ND_1$ and $B_1 \in NE_1$ remains. That is shown in Table 22. Observe that we can assume $\delta A_2 = \delta E_2 = 3$, since A_2 and E_2 are each at distance three from a degree-2-vertex and because we have completed the analysis of case M7. Moreover, we can assume $\delta C_1 > 1$ by the micro-tail argument, possibly referring to the analysis in case C63. If, in the case $A_2 \in NE_1$, we consider the subcase $\delta C_1 = 2$ and $NC_1 \cap NE_1 \neq \emptyset$ (which means that $\{A_2\} = NC_1 \cap NE_1$ and $NA_2 = \{E_1, C_1, A_1\}$) we get a large structure with only two outlets at B_1 and D_1 which easily yields the following branch:

B_1D_1	4	6	
B_1ND_1	3	7	1.2457
NB_1	0	3	

Therefore, we can assume that either $\delta C_1 = 3$ or $\delta C_1 = 2$ and $NC_1 \cap NE_1 = \emptyset$ in the last line of the analysis given in Table 22.

Finally, the 6-cycle depicted in Table 23 remains to be analyzed for case M8b. By the previous case C65, we can assume that

$$(NG_1 \cup NG_1') \cap \{B_1, D_1\} = \emptyset \tag{3}$$



Assume $A_2 \notin NE_1$:		
$A_2D_1BE_1C$	2	7
$A_2D_1BE_1NC$	2	8
$A_2D_1BNE_1$	1	7
$A_2D_1NBE_2$	1	7
$A_2D_1NBNE_2$	1	9
$A_2ND_1E_1B$	1	7
$A_2ND_1E_1NB$	1	9
$A_2ND_1NE_1C$	1	9
$A_2ND_1NE_1NC$	1	10
NA_2EB	1	6
NA_2ENB	0	7
NA_2NE	0	6
1.3976		

Assume $A_2 \in NE_1$:		
D_1E_1AC	1	5
D_1E_1ANC	1	7
D_1E_1NA	0	5
D_1NE_1C	1	6
D_1NE_1NC	1	7
ND_1E_1B	1	6
ND_1E_1NB	1	8
$ND_1NE_1C_1$	2	9
$ND_1NE_1NC_1$	1	9
1.3982		

Table 22: C65a: A special triple 6-cycle for case C65

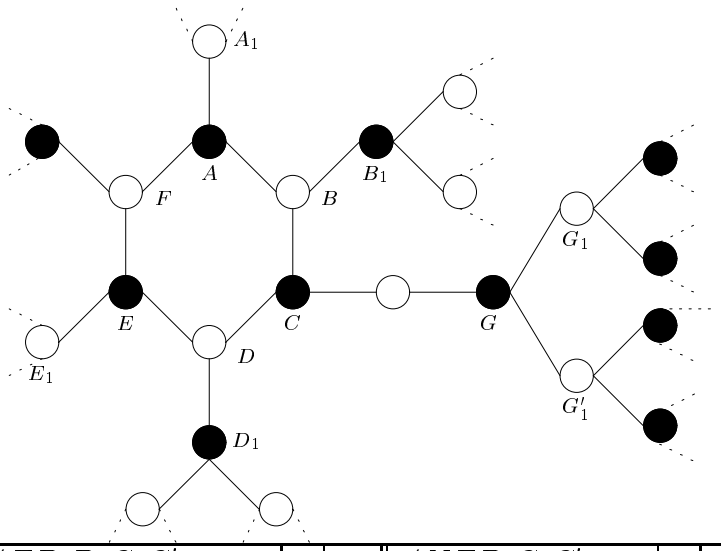
as well as

$$\{G_1, G_1'\} \cap (NA \cup ND_1) = \emptyset. \tag{4}$$

As a special case, let us first consider the triple-6-cycle given in Table 24, where $B_1 \in NE_1$ and $A_1 \in ND_1$ is assumed.

(a) We can assume that NF can have at most one common element F_1 within the neighborhood of either G_1 or G_1' but not in both, otherwise there would be a 4-cycle $GG_1'F_1G_1$. (Of course, neither A nor E is in $NG_1 \cup NG_1'$ by condition Eq.(4).)

Observe that since $\delta B = \delta D = 3$ by assumption and because of condition Eq. (3), we can assume that (b) the neighbor of A_1, which is neither A nor D_1, is not neighbor of both G_1 and G_1'.



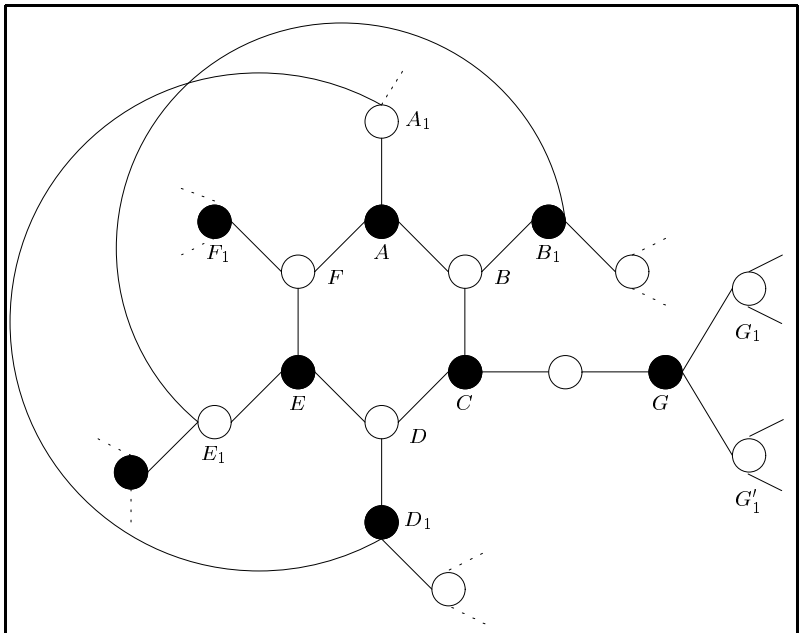
$AEB_1D_1G_1G'_1$	2	8	$ANEB_1G_1G'_1$	2	9
$AEB_1D_1G_1NG'_1$	1	9	$ANEB_1G_1NG'_1$	1	10
$AEB_1D_1NG_1$	1	8	$ANEB_1NG_1$	1	9
$AEB_1ND_1G_1G'_1$	2	10	$ANENB_1G_1G'_1$	1	10
$AEB_1ND_1G_1NG'_1$	1	11	$ANENB_1G_1NG'_1$	1	12
$AEB_1ND_1NG_1$	1	10	$ANENB_1NG_1$	1	11
$AENB_1D_1G_1G'_1$	2	10	$NADG_1G'_1$	1	7
$AENB_1D_1G_1NG'_1$	1	11	$NADG_1NG'_1$	1	9
$AENB_1D_1NG_1$	1	10	$NADNG_1$	1	8
$AENB_1ND_1G_1G'_1$	1	10	$NANDG_1G'_1$	1	9
$AENB_1ND_1G_1NG'_1$	1	12	$NANDG_1NG'_1$	0	10
$AENB_1ND_1NG_1$	1	11	$NANDNG_1$	0	9
1.3999					

Table 23: C66: A special 6-cycle for case M8b

In calculating the branching vectors, we assumed that one neighboring situation in each of those cases (a) and (b) might occur.

To be more precise using one example, observe that we assumed that the case $A_1B_1DNFNG_1$ contains nine vertices, i.e., we assumed that $NF \cap NG_1 = \emptyset$, while we only counted nine vertices in case $A_1B_1DNFG_1NG'_1$, assuming here $NF \cap NG'_1 \neq \emptyset$. Since the vertices G_1 and G'_1 are completely symmetric, such an assumption can be made, since the roles of G_1 and G'_1 are interchangeable.

By reasons of symmetry, in the main analysis of case C66 we can now



$A_1B_1DFG_1G'_1$	2	8	$NA_1B_1NEG_1G'_1$	2	11
$A_1B_1DFG_1NG'_1$	2	10	$NA_1B_1NEG_1NG'_1$	1	11
$A_1B_1DFNG_1$	2	9	$NA_1B_1NENG_1$	1	11
$A_1B_1DNFG_1G'_1$	2	10	$NB_1DG_1G'_1$	1	7
$A_1B_1DNFG_1NG'_1$	1	10	$NB_1DG_1NG'_1$	1	9
$A_1B_1DNFNG_1$	1	10	NB_1DNG_1	1	8
$NA_1B_1EG_1G'_1$	2	9	NB_1ND	0	6
$NA_1B_1EG_1NG'_1$	1	10	A_1B_1ND	0	5
$NA_1B_1ENG_1$	1	9			

1.3989

Table 24: C66a: A triple 6-cycle for case C66

assume that

$$|NE \cap NB_1| = 0 \quad \wedge \quad |NA \cap ND_1| \leq 1. \tag{5}$$

Observe that ND_1 is never mentioned in the branch given in Table 23 whenever neighbors of A are considered. Since we did not consider the case where $NB_1 \cap ND_1 \neq \emptyset$, we assumed $|NB_1 \cap ND_1| = 1$ in the case analysis of Table 23.

References

[1] J. A. Abraham et al. Fault tolerance techniques for systolic arrays. *IEEE Computer*, pages 65–75, July 1987.

[2] J. Alber, H. L. Bodlaender, H. Fernau and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, number 1851 in Lecture Notes in Computer Science, pages 97–110. Springer-Verlag, July 2000.

[3] J. Alber, J. Gramm, and R. Niedermeier. Faster exact solutions for hard problems: a parameterized point of view. Accepted for *Discrete Mathematics*, August 2000.

[4] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1.5}\sqrt{m/\log n})$. *Inform. Process. Lett.*, 37:237–240, 1991.

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

[6] R. Beigel and D. Eppstein. 3-Coloring in time $o(1.3446^n)$: a no MIS algorithm. In *Proceedings of the 36th IEEE Conference on Foundations of Computer Science*, pages 444–452, 1995.

[7] D. K. Bhavsar and J. H. Edmondson. Alpha 21164 testability strategy. *IEEE Design and Test*, 14(1):25–33, 1997.

[8] D. M. Blough. On the reconfiguration of memory arrays containing clustered faults. In *Fault Tolerant Computing*, pages 444–451, Los Alamitos, Ca., USA, June 1991. IEEE Computer Society Press.

[9] D. M. Blough and A. Pelc. Complexity of fault diagnosis in comparison models. *IEEE Trans. Comput.*, 41(3):318–323, 1992.

[10] M. Chean and J. A. B. Fortes. A taxonomy of reconfiguration techniques for fault-tolerant processor arrays. *IEEE Computer*, pages 55–69, Jan. 1990.

[11] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, number 1665

in Lecture Notes in Computer Science, pages 313–324. Springer-Verlag, June 1999.

[12] E. Dantsin, A. Goerdt, E.A. Hirsch, and U. Schöning. Deterministic algorithms for k-SAT based on covering codes and local search. In *Proceedings of the 27th International Conference on Automata, Languages, and Programming*, number 1853 in Lecture Notes in Computer Science, pages 236–247. Springer-Verlag, 2000.

[13] J. Day. A fault-driven comprehensive redundancy algorithm. *IEEE Design and Test*, 2(3):35–44, 1985.

[14] R. Diestel. *Graph Theory*. Springer-Verlag, New York, 1997.

[15] 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.

[16] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[17] 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, pages 49–99, AMS, 1999.

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

[19] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *STOC'91*, pages 123–133, 1991.

[20] T. Fuja and C. Heegard. Row/column replacement for the control of hard defects in semicondutor RAMs. *IEEE Trans. Comput.*, 35:996–1000, Nov. 1986.

[21] R. W. Haddad, A. T. Dahburaa, and A. B. Sharma. Increased throughput for the testing and repair of RAMs with redundancy. *IEEE Trans. Comput.*, 40(2):154–166, Feb. 1991.

[22] N. Hasan and C. L. Liu. Minimum fault coverage in reconfigurable arrays. In *International Symposium on Fault-Tolerant Computing (FTCS '88)*, pages 348–353, Washington, D.C., USA, June 1988. IEEE Computer Society Press.

[23] N. Hasan and C. L. Liu. Fault covers in reconfigurable PLAs. In *20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS'90)*, pages 166–173. IEEE Computer Society Press, 1990.

[24] E. A. Hirsch. New Worst-Case Upper Bounds for SAT. *Journal of Automated Reasoning* , 24(4), pages 397–420, 2000.

[25] J. H. Kim and S. M. Reddy. On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement. *IEEE Trans. Comput.*, 38(4):515–525, Apr. 1989.

[26] I. Koren and M. A. Breuer. On area and yield considerations for fault-tolerant VLSI processor arrays. *IEEE Trans. Comput.*, C-33(1):21–27, 1984.

[27] S.-Y. Kuo and I.-Y. Chen. Efficient reconfiguration algorithms for degradable VLSI/WSI arrays. *IEEE Trans. Computer-Aided Design*, 11(10):1289–1300, 1992.

[28] S.-Y. Kuo and W. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, 4:24–31, Feb. 1987.

[29] S.-Y. Kuo and W. K. Fuchs. Fault diagnosis and spare allocation for yield enhancement in large reconfigurable PLAs. In *International Test Conference*, pages 944–953. IEEE Computer Society Press, Sept. 1987.

[30] M. E. Levitt. Designing UltraSparc for testability. *IEEE Design and Test*, 14(1):10–17, 1997.

[31] F. Lombardi and W. K. Huang. Approaches to the repair of VLSI/WSI PRAMs by row/column deletion. In *International Symposium on Fault-Tolerant Computing (FTCS '88)*, pages 342–347, Washington, D.C., USA, June 1988. IEEE Computer Society Press.

[32] C. P. Low and H. W. Leong. A new class of efficient algorithms for reconfiguration of memory arrays. *IEEE Trans. Comput.*, 45(5):614–618, 1996.

[33] M. Mahajan and V. Raman. Parametrizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31:335–354, 1999.

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

[35] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Appl. Math.*, 10:287–295, 1985.

[36] R. Niedermeier and P. Rossmanith. New upper bounds for Maximum Satisfiability. *J. Algorithms*, 36: 63–88, 2000.

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

[38] R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.

[39] 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*, pages 628–637, 1998.

[40] J. M. Robson. Algorithms for Maximum Independent Sets. *J. Algorithms*, 7:425–440, 1986.

[41] 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.

[42] U. Schöning. A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems In *Proceedings of the 40 IEEE Conference on Foundations of Computer Science*, pages 410–414, 1999.

[43] S. E. Schuster. Multiple word/bit line redundancy for semiconductor memories. *IEEE J. Solid-State Circuits*, 13(5):698–703, 1978.

[44] W. Shi and W. K. Fuchs. Probabilistic analysis and algorithms for reconfiguration of memory arrays. *IEEE Trans. Computer-Aided Design*, 11(9):1153–1160, 1992.

[45] M. D. Smith and P. Mazumder. Generation of minimal vertex covers for row/column allocation in self-repairable arrays. *IEEE Trans. Comput.*, 45(1):109–115, 1996.

[46] C. H. Stapper. Large-area fault clusters and fault tolerance in VLSI circuits: a review. *IBM J. Research and Development*, 33(2):162–173, 1989.

[47] R. E. Tarjan and A. E. Trojanowski. Finding a Maximum Independent Set. *SIAM J. Comput.*, 6(3):537–550, 1977.

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

[49] L. Youngs and S. Paramanandam. Mapping and repairing embedded-memory defects. *IEEE Design and Test*, 14(1):18–24, 1997.