An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover

Henning Fernau[1] and Rolf Niedermeier[2]
Wilhelm-Schickhard-Institut für Informatik,
Universität Tübingen,
Sand 13, D-72076 Tübingen,
Fed. Rep. of Germany
email: `fernau,niedermr@informatik.uni-tuebingen.de`

[1]Partially supported by Deutsche Forschungsgemeinschaft grant DFG La 618/3-2.
[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.

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 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.

Chapter 1

Introduction

Nontrivial upper bounds for important *NP*-hard problems by exact algorithms have seen a continuous interest over many years till today [4, 20, 31, 36, 37, 38, 43, 44]. With the advent of parameterized complexity theory [12] a special class of exact algorithms has become more and more important [13, 32]: 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 running time $O(1.211^n)$ [37]. There is, however, another exact ("fixed parameter") algorithm solving Vertex Cover in running time $O(1.29175^k + kn)$ [35]. 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 [12, 13]. In this paper, we study an also *NP*-complete variant of the Vertex Cover problem, motivated by reconfigurable VLSI [24]: 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 [10, 24]), by a reduction from Clique it has been shown that CBVC is *NP*-complete [24]. 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 (1.29175^k) [35].

Our result makes the following two contributions: First, it gives a further (of so far still few) example for a problem with an *efficient* fixed parameter algorithm [12, 13, 32]. 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 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 its proven 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 [12]: reduction to problem kernel and bounded search trees. These have already been successfully applied for Vertex Cover [3, 13, 35], Maximum Satisfiability [29, 33], and elsewhere [12]. 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. In Chapter 2, we review some previous work and give more details about the practical applications behind CBVC. In Chapter 3, we provide some basic definitions and results. The heart of the paper is Chapter 4, where we explain the basic parts and the overall structure of our algorithm. In Chapter 5, we present details of our algorithm. However, some more details needed to achieve the exponential base 1.3999 are deferred to a clearly marked appendix (parts A, B, and C). In Chapter 6, we draw some final conclusions and give prospects for future work.

Chapter 2

Previous Work

Kuo and Fuchs [24] provide a fundamental study of the *spare allocation problem*. In few words, this "most widely used approach to reconfigurable VLSI" uses spare rows and columns for tolerating failures in rectangular arrays of identical computational elements, which may be as simple as memory cells or as complex as processor units (see [24] 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 formalized as CBVC by identifying one vertex set of the bipartite graph with the rows, the other vertex set with the columns of the given array, and each edge represents 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. 2.1, 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 [24] are arrays with up to 1024 rows and 1024 columns (today 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" [12, 13, 32]. 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" [24, page 29]. The second one is a polynomial time approximation algorithm. By way of contrast, our approach always delivers optimal results *and* its complexity mainly depends on the number of available spare rows and columns.

	1	2	3	4	5	6	7	8	9
1	?			?					?
2									
3	?								
4			?	?			?		
5									
6									
7				?					?

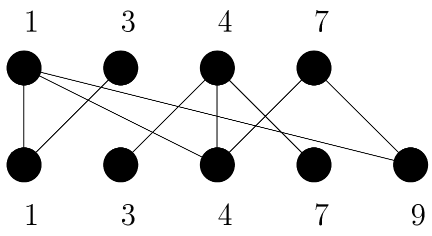


Figure 2.1: A 7×9 array with faults "?" and the bipartite graph model

It should be noted in this place 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 [27, 17]. They observed that "if k_1 and k_2 are small, for instance $O(\log(|G|))$, then this may be adequate." [17, 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 cannot be assumed anymore. So, fault-tolerance has to be considered within the very fabrication process to obtain reasonable production yields [1, 8]. These ideas are already quite old, see [39] for an early treatment of the topic, but persist until 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 [26, 45]. One common solution to increase yield in memory fabrication is to supply a few spare rows and columns to each memory array which 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 [5, 26, 45].

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 [6, 7, 8, 9, 14, 16, 17, 18, 19, 21, 22, 23, 25, 27, 28, 40, 41, 42, 45].

Chapter 3

Preliminaries

We assume familiarity with fundamentals from graph theory, algorithms, and complexity. Especially recall that, given a graph in adjacency list representation, finding all components of the graph can be done in time linear to the representation size [30, 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 $O(\sqrt{n}m)$ [30, Thm. IV.9.10] (for more sophisticated algorithms, see [2, 15]). 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 consider 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 according search tree. This number is governed by homogeneous, linear recurrences with constant coefficients (cf. [20, 34, 35]). 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

$$t_k = t_{k-d_1} + \cdots + t_{k-d_i}. \tag{3.1}$$

The characteristic polynomial of this recurrence is

$$z^d = z^{d-d_1} + \cdots + z^{d-d_i}, \tag{3.2}$$

where $d = \max\{d_1, \ldots, d_i\}$. If α is a root of (3.2) with maximum absolute value, then t_k is α^k up to a polynomial factor. We call $|\alpha|$ the *branching* *number* that 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 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 $(8, 9, 8, 10, 11, 10, 10, 11, 10, 10, 12, 11, 9, 10, 9, 10, 12, 11, 7, 9, 8, 9, 10, 9)$.

Chapter 4

The algorithm—overview

Our algorithm works in basically the same way as the fixed parameter algorithms for Vertex Cover do [3, 13, 35]. 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 $O(2^{k_1+k_2}(n+m))$, 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.

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 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 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* [11, 12], which is a kind of preprocessing. This step is based on a simple observation already used by Kuo and Fuchs [24], 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 $2k_1k_2$. 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 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 [27, 17]. 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 [12, 32].

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 basically straightforward details. In addition, 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 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 $O(k_1 + k_2)$. Altogether, we have $(k_1 + k_2)$ merge steps each of time complexity $O((k_1 + k_2)^2)$. In summary, this gives:

Proposition 4.1 For bipartite graphs with maximum vertex degree 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 maximum degree at least three.

In fact, there are certain (but only constantly many) graphs containing degree-3-vertices which are left over to the 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, that is the way our algorithm works, but we select the subgraphs $G_1,\ldots,G_i$ in a more complicated way 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 at most once 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 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.

[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.

It may easily be verified that the above steps provide a complete case distinction handling all cases that may occur.[2] More precisely, from this point of view, steps M3–M5 even would 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 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 summary, 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 cannot only be omitted by "asymptotic tricks," but also by an argument due to refined complexity analysis. The basic idea is that nearly 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 the details and just note that the argument works in analogy as it does for classical Vertex Cover [34, 35].

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 4.1. We will defer the concise description of the 4-cycle (C4x) and 6-cycle (C6x) cases to the Appendix.

Table 4.1 summarizes the complete case sequence and the corresponding (worst case) recursion bases. Observe that M4 has been replaced by the 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 are not far off from that branching number, so that we think it would be hard to get close to an overall branching number of 1.3803 (the simple branching of case M1).

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

<table>
<tr><th>Case</th><th colspan="7">Branching vector</th><th>Branching number</th></tr>
<tr><td>M1</td><td colspan="7">$(1, 4)$</td><td>1.3803</td></tr>
<tr><td>M2</td><td colspan="8">(Can be ignored)</td></tr>
<tr><td>M3</td><td colspan="7">$(2, 3)$</td><td>1.3248</td></tr>
<tr><td>C41</td><td colspan="7">$(2, 3)$</td><td>1.3248</td></tr>
<tr><td>C42</td><td colspan="7">$(2, 3)$</td><td>1.3248</td></tr>
<tr><td>C43</td><td colspan="7">$(3, 4, 3)$</td><td>1.3954</td></tr>
<tr><td>C44</td><td colspan="7">$(3, 4, 3)$</td><td>1.3954</td></tr>
<tr><td>M5</td><td colspan="7">$(3, 4, 3)$</td><td>1.3954</td></tr>
<tr><td>C61</td><td colspan="7">$(5, 6, 3)$</td><td>1.2786</td></tr>
<tr><td>C45</td><td colspan="7">$(4, 6, 6, 7, 3)$</td><td>1.3954</td></tr>
<tr><td>C46</td><td colspan="7">$(2, 4, 6)$</td><td>1.3563</td></tr>
<tr><td>C47</td><td colspan="7">$(6, 7, 4, 2)$</td><td>1.3993</td></tr>
<tr><td>C48</td><td colspan="7">$(2, 4, 5)$</td><td>1.3803</td></tr>
<tr><td rowspan="2">C49</td><td>number</td><td>6</td><td>7</td><td>9</td><td>8</td><td>10</td><td>12</td><td rowspan="2">1.3996</td></tr>
<tr><td>times</td><td>1</td><td>4</td><td>4</td><td>2</td><td>4</td><td>1</td></tr>
<tr><td>M6</td><td colspan="7">$(4, 6, 6, 7, 3)$</td><td>1.3954</td></tr>
<tr><td>M7</td><td colspan="7">$(4, 6, 6, 7, 3)$</td><td>1.3954</td></tr>
<tr><td>C62</td><td colspan="7">$(6, 7, 4, 2)$</td><td>1.3993</td></tr>
<tr><td>M8a</td><td colspan="7">$(7, 9, 8, 6, 4, 3)$</td><td>1.3906</td></tr>
<tr><td>C63</td><td colspan="7">$(6, 7, 6, 9, 6, 8, 3)$</td><td>1.3930</td></tr>
<tr><td>C64</td><td colspan="7">$(6, 7, 7, 9, 6, 8, 3)$</td><td>1.3829</td></tr>
<tr><td>M8b</td><td colspan="7">see Table 5.5</td><td>1.3976</td></tr>
<tr><td>C65</td><td colspan="7">$(5, 7, 5, 6, 7, 6, 8, 9, 9)$</td><td>1.3982</td></tr>
<tr><td>C66</td><td colspan="7">see Table C.6</td><td>1.3999</td></tr>
</table>

Table 4.1: The sequence of cases with their branching vectors and branching numbers. For C49 the vector has 16 entries.

Chapter 5

Details of the main cases

In the following, we present details of the main algorithm presented 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 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.

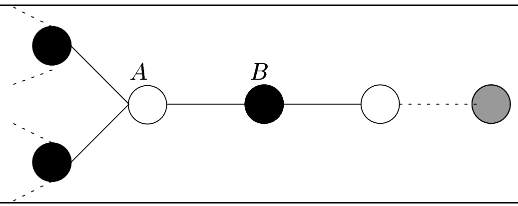
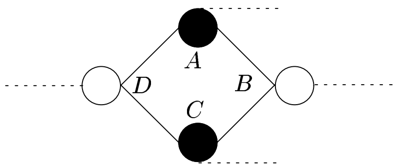
Picture	Branching
	A 1 2 NA 0 3 1.3248
	AC 0 2 BD 0 2 1.4143

Table 5.1: Cases M3 and M4

Because of steps M1 and M2 (cf. Chapter 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.

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 5.1, 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 with a graph having vertices of degree at most two very easily (Proposition 4.1). Therefore, if we take vertex A in the present situation, 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 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 5.1, 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. 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 5.1. 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 as 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 say that A and B are connected by a *chain* of length ℓ. Chains of length 3 and 4 are considered in Table 5.2 (M5a and M5b).

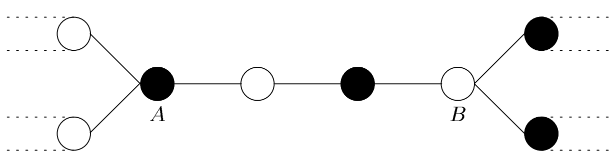
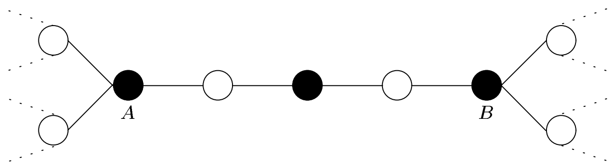
<table>
<tr><th>Picture</th><th>Branching</th></tr>
<tr><td></td><td>
<table>
<tr><td>$A \in NB$</td><td colspan="3">see M4</td></tr>
<tr><td rowspan="3">$A \notin NB$
$\delta A = 3$</td><td>AB</td><td>1</td><td>3</td></tr>
<tr><td>ANB</td><td>0</td><td>4</td></tr>
<tr><td>NA</td><td>0</td><td>3</td></tr>
<tr><td colspan="4">1.3954</td></tr>
</table>
</td></tr>
<tr><td></td><td>
<table>
<tr><td>$A = B$</td><td colspan="3">see M4</td></tr>
<tr><td>$A \neq B$</td><td>AB</td><td>1</td><td>3</td></tr>
<tr><td>$\delta B = 3$</td><td>ANB</td><td>1</td><td>5</td></tr>
<tr><td>$\delta A = 3$</td><td>NA</td><td>0</td><td>3</td></tr>
<tr><td colspan="4">1.3640</td></tr>
</table>
</td></tr>
</table>

Table 5.2: Cases M5a and M5b

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

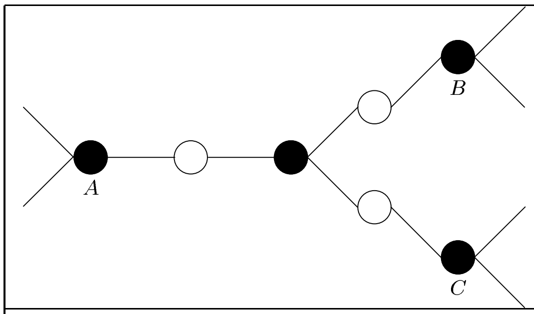
	AB	2	3	
$\delta B = 3$	ANB	1	4	1.3954
$\delta A = 3$	NA	0	3	

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

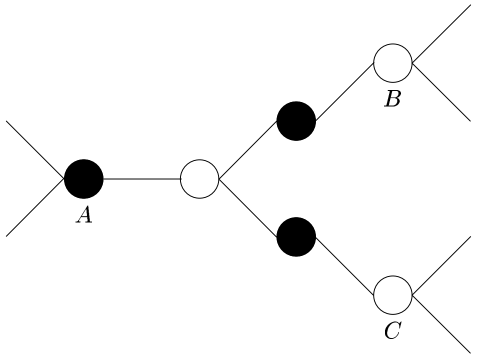
So, we can assume from now on that two degree-3-vertices are connected by a chain of length at most two. Having this in mind, we start analyzing the possible situations in the neighborhood of a degree-3-vertex with at least one degree-2-neighbor.

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

In Table 5.3, 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



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



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

Table 5.3: Cases M6 and M7

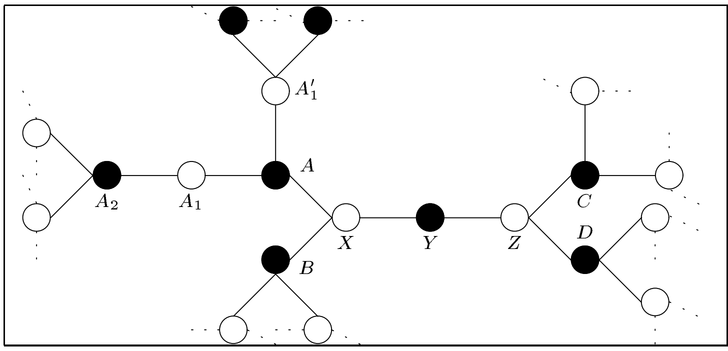
chain (M5).

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 5.3. 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 5.4. 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



A'_1A_2BCD	2	7
A'_1A_2BCND	2	9
A'_1A_2BNC	2	8
A'_1A_2NB	1	6
A'_1NA_2	0	4
NA'_1	0	3
Branching number: 1.3906		

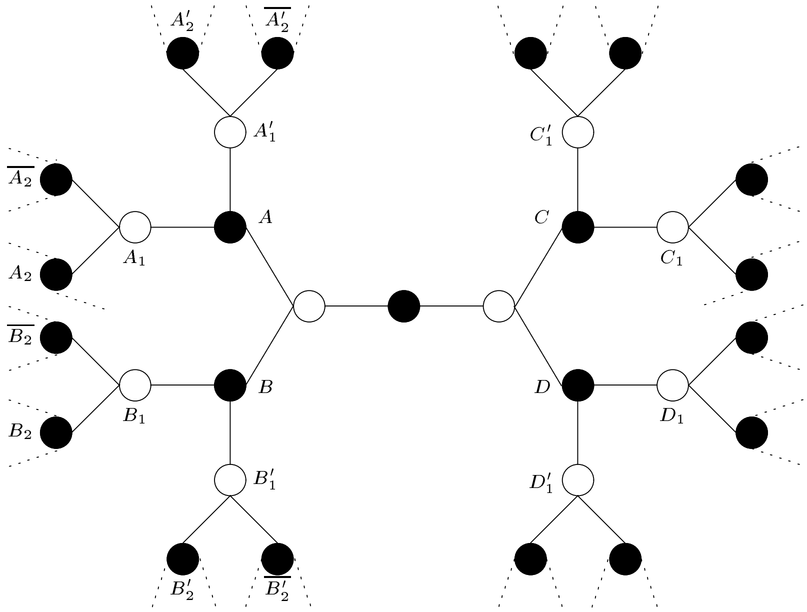
Table 5.4: Case M8a: $\delta A_1 = 2$

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 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-neighbors, since there is a micro-tail at A.

Case M8a: $\delta A_1 = 2$**.** Then, 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 just 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 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 5.4.

Case M8b: $\delta A_1 = 3$**.** In fact, by symmetry, we can now assume that all neighbors of A, B, C, and D have degree three. The corresponding picture is given in Table 5.5. 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



$A_1A_1'BCD$	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 5.5: Case M8b: $\delta A_1 = 3$

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). We do not repeat those assumptions in Table 5.5. (Horizontal lines in the branching list should help to structure that branching; they do not separate different branching lists.)

Chapter 6

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 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 [32]. A first, still non-sophisticated version of our algorithm has been implemented and delivers promising results. As to future work, on the one hand, it remains to provide a competitive implementation of our algorithm.

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 Chapter 2). As a rule, however, these methods only cause average case improvements. The worst case remains unchanged.

2. Shi and Fuchs presented in [40] 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 be also a starting point for the average-case analysis of such type of algorithms.

3. It is possible that horizontal and vertical movements of the repair lazer are not equally fast, see [9]. 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 [18, 25]). Trying to repair each such region independently and combining the signatures in an obvious way leads to quite efficient reconfiguration algorithms, since the total number of spare rows and columns is not influencing the repair process such badly 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, yielding smaller exponential factors.

Moreover, our approach can be used also when we drop the assumption that the spare rows and columns are always fault-free, see [21, 23].

Finally, there is a variant of CBVC with three instead of two parameters, motivated by reconfigurable programmable logic arrays [19]. This problem deserves investigations as we performed 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 only leave over 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 are only a constant number of such exception graphs, this is of course no problem. We will argue here that this is the case in our algorithm indeed. As the reader may verify by analyzing the search-tree part of our algorithm in details, 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. A.1.

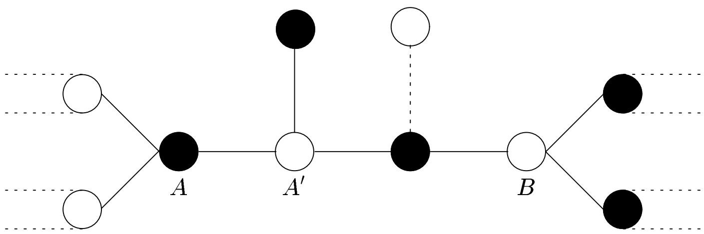


Figure A.1: Case M5a with micro-tails

When one considers now 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. A.2: namely, if, e.g., in Fig. A.2 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. A.2.

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.

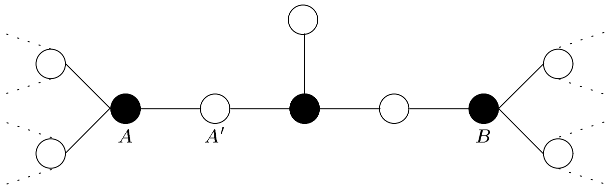


Figure A.2: Case M5b with micro-tail

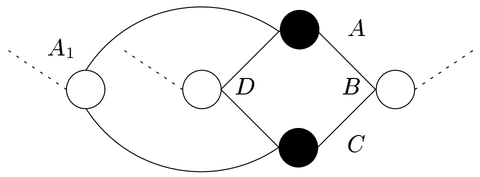
Similarly, looking at the branch analysis of case M5b, only one star-component remains for the polynomial-time part.

In conclusion, actually only a finite number of graph components containing degree-3-vertices remain for the polynomial-time part of the algorithm. We refrain 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

B.1 4-Cycles in detail I: the easy cases



AC	0	2	1.3248
A_1DB	0	3	

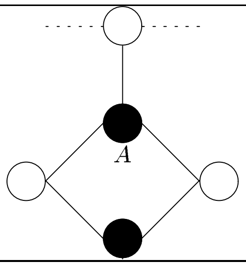
Table B.1: C41: A double 4-cycle

As a special case, let us first treat the case C41 of two 4-cycles cycles combined as depicted in Table B.1. Similar to M4, we get the branching given in Table B.1: Namely, 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 is in the cover. If C is in the cover, then the subcase $\{A, C\}$ from Table B.1 will handle 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 B.1.

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.

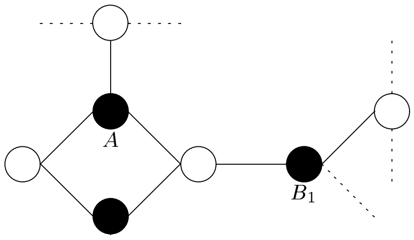
If there are three degree-2-vertices (Case C42), we get the easy branch given in Table B.2.

If there are two degree-2-vertices, we have to distinguish the cases whether those two vertices are neighbors or not (Cases C43 resp. C44, cf. Table B.3).

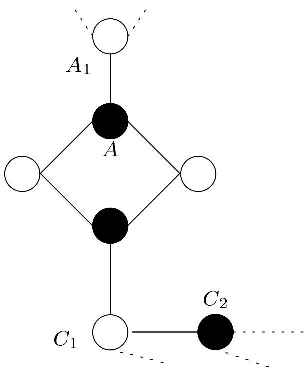


	A	1	2	1.3248
$\delta A = 3$	NA	0	3	

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



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



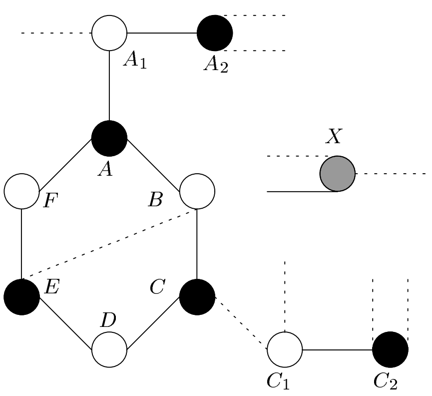
$A \in NC_1$	see C41		
Assume $A \notin NC_1$:			
Subcase $\delta C_1 = 3$:			
	AC_1	1	3
	ANC_1	0	4
$\delta A = 3$	NA	0	3
1.3954			
Subcase $\delta C_1 = 2$:			
$A \neq C_2$	AC_2	1	3
$\delta C_2 > 1$	ANC_2	1	4
$\delta A = 3$	NA	0	3
1.3954			

Table B.3: 4-Cycles with two degree-2-vertices

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 assume from now on 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.

B.2 Case C61: A very special 6-cycle



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

Table B.4: C61: A very special 6-cycle

Consider the figure given in Table B.4. Although the depicted figure is obviously a 6-cycle, it should be clear that it becomes a combination of two 4-cycles by inserting the optional edge $\overline{BE}$; this special case will be of importance in the subsequent analysis of 4-cycles.

By 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 become clear later, we will only exploit that we can assume $\delta A_2 \geq 2$ in case $\delta A_1 = 2$, and similar $\delta C_2 \geq 2$ if $\delta C_1 = 2$.

The branching is done using several subcases in order to get very good branching vectors. We now give some additional explanatory remarks on the case distinctions given in Table B.4. First, we distinguish the case whether

$\delta C = 2$ or $\delta C = 3$. In case $\delta C = 2$, we can assume that $\delta A = 3$, since otherwise we would have a small graph component which we can afford to treat separately in the final polynomial-time algorithm. Observe that the case $\delta A = 2$ is analogous to the one discussed above.

Now, we can 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 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 $\delta C_2 = 1$); we omit details 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 B.4 we have to argue that $A_1 N C_2$ 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 B.4 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.

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 B.5: C61: A very special 6-cycle plus a vertex X

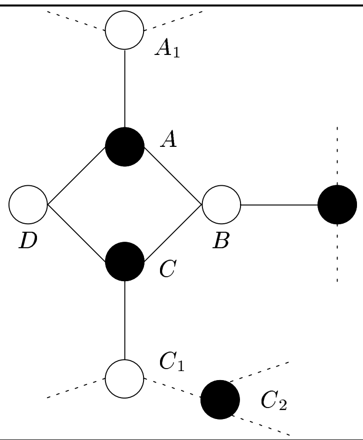
Now, because we were able to get such good branching vectors, we can even handle the following situation. Assume that we do not yet have a C61 situation, but we could generate one by removing one other vertex X (which may be linked to vertices B, D, E or F). By trivial branching on X and NX, in the first case we now get a C61 situation and can do the analysis given in Table B.4. Clearly, we may assume $\delta X \geq 2$. Of course, we must analyze the branching situation generated in that way. This is done in Table B.5. The case distinctions given there are listed in the same order as in Table B.4.

The branching vectors given in Table B.5 are obtained from the ones from Table B.4 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 B.4, in order to avoid unnecessary case distinctions when creating Table B.5, we only assumed $\delta A_2 \geq 2$ in case $\delta A_1 = 2$ and $\delta C_2 \geq 2$ in case $\delta C_1 = 2$, since deleting X might 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).

B.3 4-Cycles in detail II: C45, a 4-cycle which contains only one degree-2-vertex



$A_1 = C_1$	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 B.6: C45, a 4-cycle with one degree-2-vertex

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

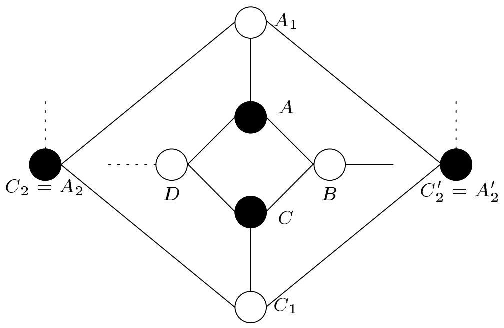
C45: $|NA_1 \cap NC_1| = 0$;

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

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

The corresponding pictures can be seen in Tables B.6, B.7 and B.8. The sequence chosen has algorithmic reasons; C47 is the most involved case.

The analysis given in Table B.6 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 distinguish basically two subcases in Table B.6, namely (1) $\delta C_1 = \delta A_1 = 3$ and (2) $\delta C_1 = 2$, since 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.



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

Table B.7: C46, a special case of the picture given in Table B.6

Since we would like to refer to the two cases C46 and C47 later, too, we sketched an optional edge at node D in Tables B.7, B.8. In our case here, of course, $\delta D = 2$ can be assumed.

In cases C46 and C47 (Tables B.7, B.8), 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 exploit 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 explain the cases of C47 in more detail now. 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 or C (depending on whether $\delta C_1 = 3$ or $\delta C_1 = 2$, respectively), a case which

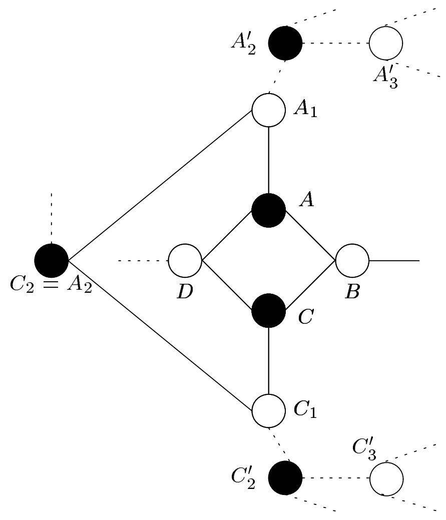


Table B.8: C47, a special case of the picture given in Table B.6

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 B.8.

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 $A_2'BCDAA_1$ using A_2 as "X".

Observe that only in the last item listed above we used that D has degree two in our case. 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 B.9.

Table B.8 shows the picture. In none of those studies, $\delta D = 2$ is used.

B.4 4-Cycles in detail III: the 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-neighbor. Since we can exclude chains and tails, we can assume that $\delta A_2 = 3$, as depicted in Table B.10. 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 been treated just before.

Now, we can assume in case C49 that all 4-cycle-vertices have degree-3-neighbors. Assuming that all vertices in the corresponding picture in Table B.10 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 children. This gives also the worst case 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.

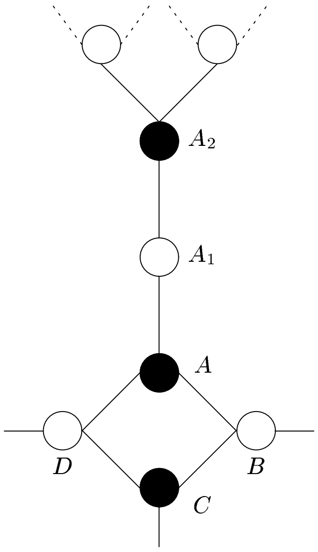
If $A_1 \in NB_1$ (or symmetric cases), we get the situation given in case C49a in Table B.11. Observe that we can assume that all vertices part of or

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: $\delta A_2 = \delta A_2' = \delta C_2' = 3$:				
$A_2' \neq C_2'$	$BDA_2A_2'C_2'$	2	7	1.3971
$\delta C_2' = 3$	$BDA_2A_2'NC_2'$	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: $\delta A_2 = \delta A_2' = \delta C_2' = 2$:				
$B \notin \{A_3', C_3'\}$	$BDA_3'C_3'$	2	5	1.3911
$D \notin \{A_3', C_3'\}$	$BDA_3'NC_3'$	2	7	
$\delta A_3' = 3$	$BDNA_3'$	0	5	
$\delta C_3' = 3$	AC	0	2	
Case 4: $\delta A_2 = 2$, $\delta A_2' = 3$, $\delta C_2' \geq 2$:				
$A_2' \neq C_2'$	$BDA_2'C_2'$	2	6	1.3854
$B, D \notin NC_2'$	$BDA_2'NC_2'$	1	6	
$B, D \notin NA_2'$	$BDNA_2'$	0	5	
	AC	0	2	

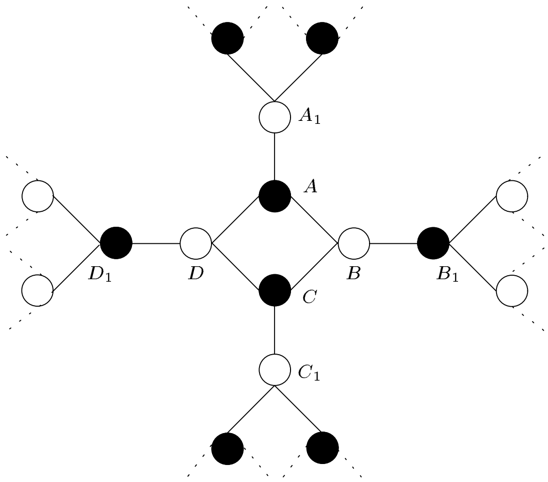
Table B.9: The branching in case C47

neighboring any 4-cycle have degree three by our 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 B.11: $B_2 \in NA_2$ 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: namely, only in the fifth observation there we exploited 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 B.8.



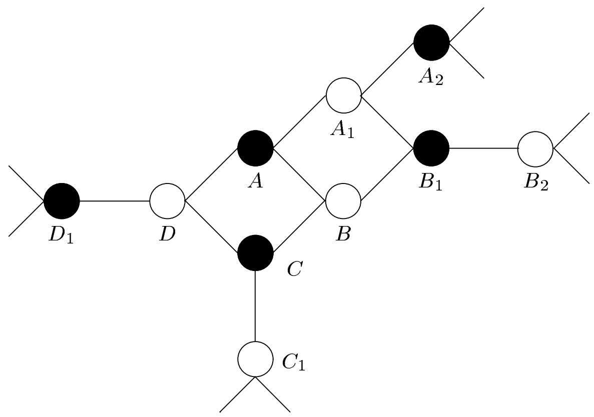
	AC	0	2	1.3803
	BDA_2	1	4	
	$BDNA_2$	0	5	



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 B.10: 4-Cycles III: all 4-cycle-vertices have degree three: C48 and C49

In the notation of Table B.10, 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 B.11, so that we actually completed our analysis of case C49 and hence of all possible 4-cycles.



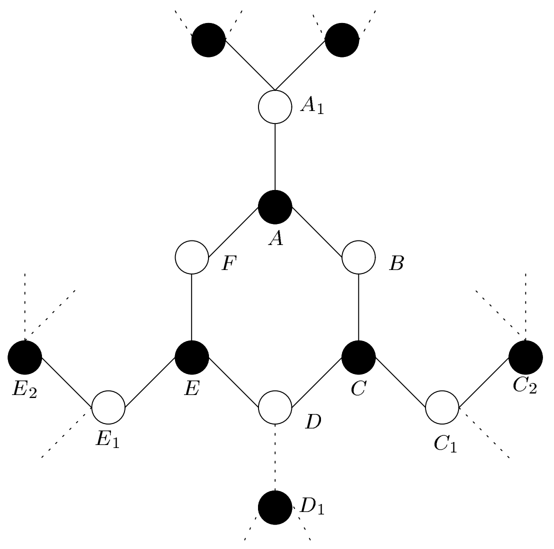
Assume $B_2 \in NA_2$:		
A_2B_1AC	0	4
A_2B_1BD	1	5
A_1B_2AC	1	5
A_1B_2BD	0	4
1.3645		

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 B.11: C49a: a special 4-cycle

Appendix C

A detailed analysis of 6-cycles



$\delta D = 2$	Use C61-trick with "$X = A_1$".		
1.3993			
Assume now $\delta D = 3$:			
$\delta E_1 \geq 2$ $\delta C_1 = 3$	$A_1 D C_1 E_1$	2	6
	$A_1 D C_1 N E_1$	2	7
	$A_1 D N C_1$	0	5
	$A_1 N D$	1	5
	$N A_1$	0	3
1.3889			
$\delta C_1 = \delta E_1 = 2$ $\delta C_2 = \delta E_2 = 3$ $\{A_1\} = N C_2 \cap N E_2$			
	$D C_2 E_2$	3	6
	$D C_2 N E_2$	3	8
	$D N C_2$	0	4
	$N D$	0	3
1.3384			

Table C.1: C62: A 6-cycle for case M7

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 done above, this analysis does not treat all possible 6-cycles but only those which actually occur as special cases in our algorithm.

C.1 C62: A 6-cycle for case M7

In case M7, the structure given in Table C.1 has remained 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 C.1 are pairwise different. We made the following distinction of subcases in Table C.1:

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 have already 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 at least two.

3. If $\delta E_1 \geq 2$ and $\delta C_1 = 3$, we refer to the analysis given in Table C.1. Obviously, the case $\delta E_1 = 3$ and $\delta C_1 \geq 2$ is symmetric. Therefore, the following case remains:

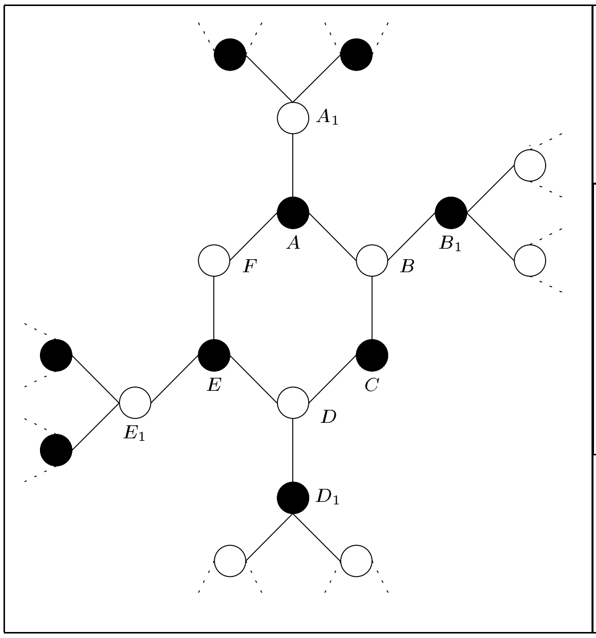
4. If $\delta E_1 = \delta C_1 = 2$, then we can assume that $\delta C_2 = \delta E_2 = 3$, since otherwise we would have tails or chains which are already done. Furthermore, if $NA \cap NC_2 = \emptyset$, we are in the situation of the main case M7 which we can assume to be done already: namely, we 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 $A_1 \in NC_2 \cap NE_2$, and no further vertex is in that intersection, since we can assume the absence of 4-cycles.

Note that now (after having completed with 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.

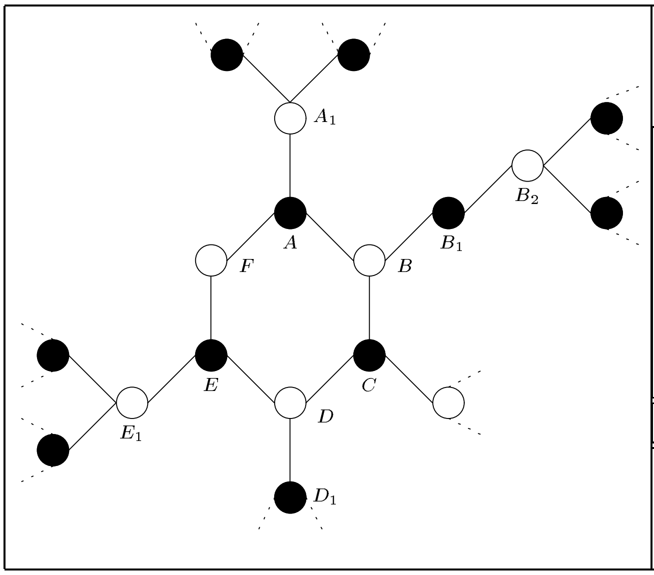
C.2 Two 6-cycles for case M8a

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 C.2 since M7 has been done before; moreover,



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

Table C.2: 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		

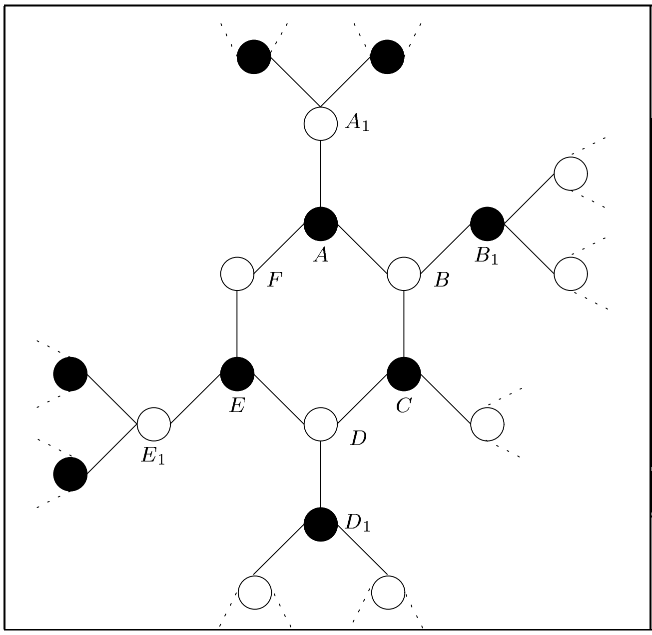
Table C.3: C64: A special 6-cycle for case M8a

$\delta A_1 = \delta B_1 = \delta D_1 = \delta E_1 = 3$ 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 C.3.

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 C.3. 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.)

C.3 Two 6-cycles for case M8b

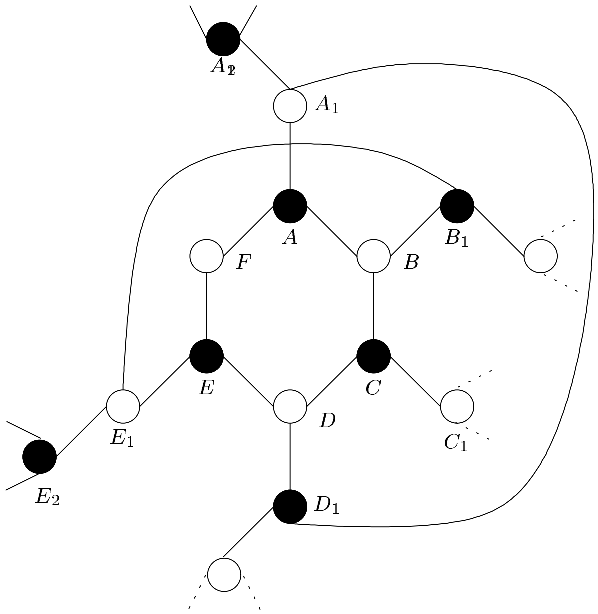


$CA_1E_1B_1D_1$	2	7
$CA_1E_1B_1ND_1$	2	8
$CA_1E_1NB_1D_1$	2	9
$CA_1E_1NB_1ND_1$	1	8
$CA_1NE_1B_1$	1	7
$CA_1NE_1NB_1$	1	9
NA_1	0	3
NCA_1E_1	1	6
NCA_1NE_1	1	8
1.3969		

Table C.4: C65: A special 6-cycle 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 C.4. 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, we subtracted even 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 special case $A_1 \in ND_1$ and $B_1 \in NE_1$ remains. That case is treated in Table C.5. Observe that we can assume



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		

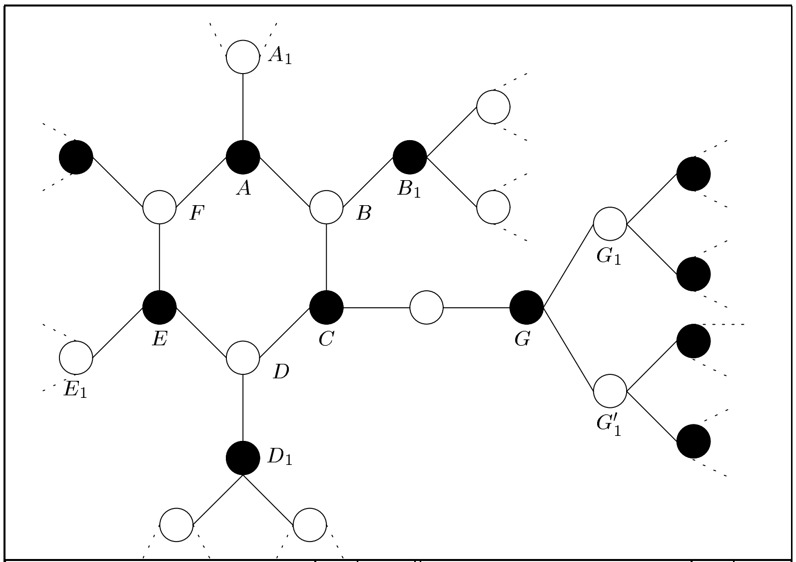
Assume $A_2 \in NE_1$:		
D_1E_1AC	1	5
D_1E_1ANC	1	7
D_1E_1NA	0	5
D_1NE_1C	1	6
D_1NE_1NC	1	7
ND_1E_1B	1	6
ND_1E_1NB	1	8
$ND_1NE_1C_1$	2	9
$ND_1NE_1NC_1$	1	9
1.3982		

Table C.5: C65a: A special triple 6-cycle for case C65

$\delta A_2 = \delta E_2 = 3$, since A_2 and E_2 are each at distance three from a degree-2-vertex and 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	1.2457
B_1ND_1	3	7	
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 C.5.



$AEB_1D_1G_1G'_1$	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					

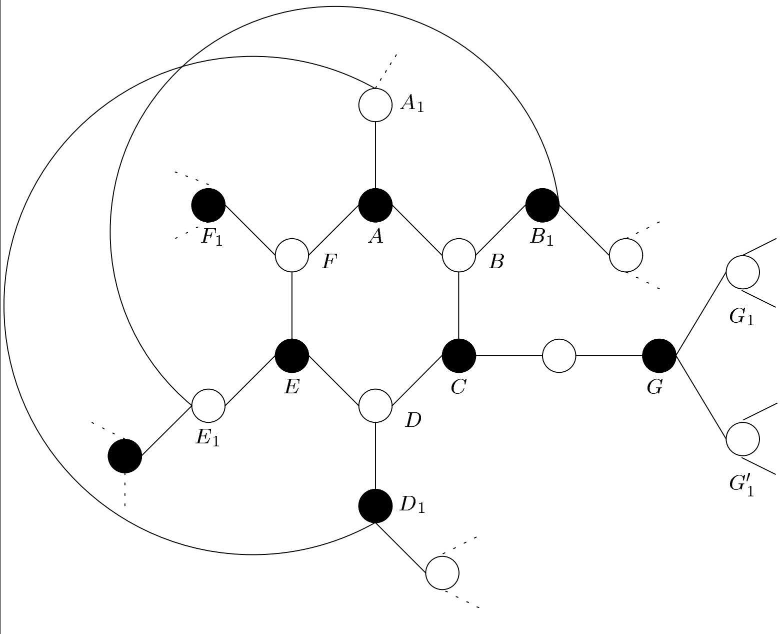
Table C.6: C66: A special 6-cycle for case M8b

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

$$(NG_1 \cup NG'_1) \cap \{B_1, D_1\} = \emptyset \tag{C.1}$$

as well as

$$\{G_1, G'_1\} \cap (NA \cup ND_1) = \emptyset. \tag{C.2}$$



$A_1B_1DFG_1G'_1$	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 C.7: C66a: A triple 6-cycle for case C66

As specialization, let us first consider the triple-6-cycle given in Table C.7, 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, since else there would be a 4-cycle $GG'_1F_1G_1$. (Of course, neither A nor E is in $NG_1 \cup NG'_1$ by condition Eq.(C.2).)

Observe that since $\delta B = \delta D = 3$ by assumption and because of condition Eq. (C.1), we can assume that (b) the neighbor of A_1 which is neither A nor D_1 is not both neighbor of G_1 and G'_1.

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 assume that

$$|NE \cap NB_1| = 0 \quad \wedge \quad |NA \cap ND_1| \leq 1. \tag{C.3}$$

Observe that ND_1 is never mentioned in the branch given in Table C.6 whenever neighbors of A are considered. Since we did not consider the special case $NB_1 \cap ND_1 \neq \emptyset$, we assumed $|NB_1 \cap ND_1| = 1$ in the case analysis of Table C.6.

Bibliography

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

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

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

[4] R. 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.

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

[6] 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.

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

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

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

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

[11] 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.

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

[13] 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.

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

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

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

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

[18] 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.

[19] 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.

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

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

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

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

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

[25] 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.

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

[27] 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.

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

[29] 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*.

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

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

[32] 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.

[33] 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. Extended abstract to appear at ICALP'99, Prague, July 1999.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

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

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

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

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

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

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

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