Even Delta-Matroids and the Complexity of Planar Boolean CSPs

Alexandr Kazda, Vladimir Kolmogorov, Michal Rolínek

IST Austria*

Abstract

The main result of this paper is a generalization of the classical blossom algorithm for finding perfect matchings. Our algorithm can efficiently solve Boolean CSPs where each variable appears in exactly two constraints and all constraints are even Δ-matroid relations (represented by lists of tuples). As a consequence of this, we settle the complexity classification of planar Boolean CSPs started by Dvořák and Kupec.

Furthermore, we extend the tractability result to a larger class of Δ-matroids that we call stabilizable. Our method works by calling the algorithm for even Δ-matroids polynomially many times. We then show that stabilizable Δ-matroids cover classes that were known to be tractable before, namely co-independent, compact, local and binary.

1 Introduction

The constraint satisfaction problem (CSP) has been a classical topic in computer science for decades. Aside from its indisputable practical importance, it has also heavily influenced theoretical research. The uncovered connections between CSP and areas such as graph theory, logic, group theory, universal algebra, or submodular functions provide some striking examples of the interplay between CSP theory and practice.

We can exhibit such connections especially if we narrow our interest down to fixed-template CSPs, that is, to sets of constraint satisfaction instances in which the constraints come from a fixed set of relations Γ. The question whether for any fixed Γ, the set of generated instances CSP(Γ) forms a decision problem which is either polynomial-time solvable or NP-complete (in other words it avoids intermediate complexities assuming P ≠ NP) is known as the CSP dichotomy conjecture [13] and is one of more notorious open problems in theoretical computer science.

The line of research pursuing this conjecture successfully used techniques from universal algebra and clone theory to establish some partial complexity classifications [1, 2, 3, 15] and to develop strong machinery for proving NP-hardness [4, 18]. More recently, the algebraic approach to CSP has also served as a starting point for a more general framework that includes both satisfaction problems and discrete optimization problems as well as their combinations (e.g. graph k-coloring, max-cut, min-verex-cover, submodular minimization). The study of valued CSPs (VCSP) extended the algebraic machinery for proving NP-Hardness [22, 21, 28] and provided vast complexity classifications [19, 28, 21, 27]. Approximability of valued CSPs has also been studied extensively [21, 25, 6].

*This work was supported by European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 616160.
However, since the obtained complexity classifications are dominated by hardness results, many common types of instances do not fall into any tractable class. This has sparked interest in imposing other types of restrictions. Often these concern jointly the set of allowed constraints and the structure of the constraint network (as an example see restrictions of the “microstructure” graph \[8, 7\]) of input instances – this gives rise to so-called hybrid (V)CSPs. A systematic way to generate hybrid CSPs is to fix the constraint language and the constraint network independently. However, there are few systematic results \[20\] in this direction as already very special cases cover highly non-trivial problems such as 4-colorability of planar graphs.

In this work we address two special structural restrictions for CSPs with Boolean variables. One is limiting to at most two constraints per variable and the other requires the constraint network to have a planar representation. The first type, introduced by Feder \[12\], has very natural interpretation as CSPs in which edges play the role of variables and nodes the role of constraints, which is why we choose to refer to it as edge CSP. It was Feder who showed the following hardness result: unless all relations in \(\Gamma\) are \(\Delta\)-matroids, the restricted CSP (with constant constraints allowed) has the same complexity as its unrestricted version. Since then, there has also been progress on the algorithmic side. Several tractable classes of \(\Delta\)-matroids were identified \[12, 17, 9, 14, 10\]. A recurring theme is the connection between \(\Delta\)-matroids and matching problems.

Recently, a setting for planar CSPs was formalized by Dvořák and Kupec \[10\]. In their work, they provide certain hardness results together with a reduction of the remaining cases to Boolean edge CSP. Their results imply that completing the complexity classification of Boolean planar CSPs is equivalent to establishing the complexity of (planar) Boolean edge CSP where all the constraints are even \(\Delta\)-matroids. In their paper, Dvořák and Kupec provided a tractable subclass of even \(\Delta\)-matroids along with computer-aided evidence that the subclass (matching realizable even \(\Delta\)-matroids) covers all even \(\Delta\)-matroids of arity at most 5. However, it turns out that there exist even \(\Delta\)-matroids of arity 6 that are not matching realizable; we provide an example of such a \(\Delta\)-matroid in Appendix A.

The main result of our paper is a generalization of the classical Edmonds’ blossom-shrinking algorithm for matchings \[11\] that we use to efficiently solve edge CSPs with even \(\Delta\)-matroid constraints. This settles the complexity classification of planar CSP. Moreover, we give an extension of the algorithm to cover a wider class of \(\Delta\)-matroids. This extension subsumes (to our best knowledge) all previously known tractable classes.

The paper is organized as follows. In the introductory Sections 2, 3, and 4 we formalize the frameworks, discuss how dichotomy for Boolean planar CSP follows from our main theorem and sharpen our intuition by highlighting similarities between edge-CSPs and perfect matching problems, respectively. The algorithm is described in Section 5 and the proofs required for showing its correctness are in Section 6. The extension of the algorithm is discussed in Appendices B and C.

2 Preliminaries

Definition 1. A Boolean CSP instance \(I\) is a pair \((V, C)\) where \(V\) is the set of variables and \(C\) the set of constraints of \(I\). A \(k\)-ary constraint \(C ∈ C\) is a pair \((σ, R_C)\) where \(σ ⊆ V\) is a set of size \(k\) (called the scope of \(C\)) and \(R_C ⊆ \{0, 1\}^σ\) is a relation on \(\{0, 1\}\). A solution to \(I\) is a mapping \(f : V → \{0, 1\}\) such that for every constraint \(C = (σ, R_C) ∈ C\), \(f\) restricted to \(σ\) lies in \(R_C\).

If all constraint relations of \(I\) come from a set of relations \(\Gamma\) (called the constraint language), we say that \(I\) is a \(\Gamma\)-instance. For a fixed \(\Gamma\), the set of all \(\Gamma\)-instances will be denoted as CSP(\(\Gamma\)).

Note that the above definition is not fully general in the sense that it does not allow one variable to occur multiple times in a constraint; we have chosen to define Boolean CSP in this way to make
our notation a bit simpler. This can be done without loss of generality: if a variable, say \( v \), occurs in a constraint multiple times, we can add extra copies of \( v \) to our instance and join them together by the equality constraint (i.e. \( \{(0,0),(1,1)\} \)) to obtain a slightly larger instance that satisfies our definition.

For brevity of notation, we will often not distinguish a constraint \( C \in \mathcal{C} \) from its constraint relation \( R_C \); the exact meaning of \( C \) will always be clear from the context. Even though in principle different constraints can have the same constraint relation, our notation would get cumbersome if we wrote \( R_C \) everywhere.

The main point of interest is classifying the computational complexity of \( \text{CSP}(\Gamma) \). As it is usual in the CSP world, constraints of an instance are specified by lists of tuples in the corresponding relations and thus those lists are considered to be part of the input. We will say that \( \Gamma \) contains the unary constant relations if \( \{(0)\}, \{(1)\} \in \Gamma \) (these relations allow us to fix the value of a certain variable to 0 or 1). Having unary constants is a common assumption on \( \Gamma \).

For Boolean CSPs (where variables are assigned Boolean values), the complexity classification of \( \text{CSP}(\Gamma) \) due to Schaefer has been known for a long time [26]. There also exists a classification for the three-element domain [2] and there is also substantial progress on the general case (see [3, 1, 5]). However, in this work we concentrate on Boolean domains only.

Our main focus is on restricted forms of the CSP. In particular, we are interested in structural restriction, i.e. in restriction on the constraint network. A natural such restriction would be to limit the number of constraints in whose scope a variable can lie. When \( k \geq 3 \) and \( \Gamma \) contains all unary constants, then \( \text{CSP}(\Gamma) \) with each variable in at most \( k \) constraints is polynomial time equivalent to unrestricted \( \text{CSP}(\Gamma) \), see [9, Theorem 2.3]. This leaves instances with at most two occurrences per variable in the spotlight. To make our arguments clearer, we will assume that each variable occurs exactly in two constraints in our paper (following [12], we can reduce decision CSP instances with at most two appearances of each variable to instances with exactly two appearances by taking two copies of the instance and adding equality constraints between both copies of variables that appear only in a single constraint).

**Definition 2** (Edge CSP). Let \( \Gamma \) be a constraint language. Then \( \text{CSP}_{\text{edge}}(\Gamma) \) is the set of \( \Gamma \)-instances in which every variable is present in exactly two constraints.

Perhaps a more natural way to look at an instance \( I \) of an edge CSP is to consider a graph whose edges correspond to variables of \( I \) and nodes to constraints of \( I \). Constraints (nodes) are incident with variables (edges) they interact with. In this (multi)graph, we are looking for a satisfying Boolean edge labeling. Viewed like this, edge CSP becomes a counterpart to the usual CSP where variables are typically identified with nodes and constraints with (hyper)edges.

This type of CSP is sometimes called “binary CSP” in the literature [10]. However, this term is very commonly used for CSPs whose all constraints have arity at most two [29]. In order to resolve this confusion (and for the reasons described in the previous paragraph), we propose the term “edge CSP”.

In this paper, we will only consider Boolean edge CSP, often omitting the word “Boolean” for space reasons. The following Boolean-specific definitions will be useful for us:

**Definition 3.** Let \( f : V \to \{0,1\} \) and \( v \in V \). We will denote by \( f \oplus v \) the mapping \( V \to \{0,1\} \) that agrees with \( f \) on \( V \setminus \{v\} \) and has value \( 1 - f(v) \) on \( v \). For a set \( S = \{s_1, \ldots, s_k\} \subseteq V \) we let \( f \oplus S = f \oplus s_1 \oplus \cdots \oplus s_k \). Also for \( f, g : V \to \{0,1\} \) let \( f \Delta g \subseteq V \) be the set of variables \( v \) for which \( f(v) \neq g(v) \).

**Definition 4.** Let \( V \) be a set. A nonempty subset \( M \) of \( \{0,1\}^V \) is called a \( \Delta \)-matroid if whenever \( f, g \in M \) and \( v \in f \Delta g \), then there exists \( u \in f \Delta g \) such that \( f \oplus \{u,v\} \in M \). If moreover, the
parity of the number of ones over all tuples of $M$ is constant, we have an even $\Delta$-matroid (note that in that case we never have $u = v$ so $f \oplus \{u, v\}$ reduces to $f \oplus u \oplus v$).

The strongest hardness result on edge CSP is from Feder.

**Theorem 5** ([12]). If $\Gamma$ is a constraint language containing unary constant relations such that $\text{CSP}(\Gamma)$ is NP-Hard and there is $R \in \Gamma$ which is not a $\Delta$-matroid, then $\text{CSP}_{\text{edge}}(\Gamma)$ is NP-Hard.

Tractability was shown for special classes of $\Delta$-matroids, namely binary [14, 9], co-independent [12], compact [17], and local [9] (see the definitions in the respective papers). All the proposed algorithms are based on variants of searching for augmenting paths.

In this work we propose a more general algorithm that involves both augmentations and contractions. In particular, we prove the following.

**Theorem 6.** If $\Gamma$ contains only even $\Delta$-matroid relations, then $\text{CSP}_{\text{edge}}(\Gamma)$ can be solved in polynomial time.

Our algorithm will in fact be able to solve even a certain optimization version of the edge CSP (corresponding to finding a maximum matching). This is discussed in detail in Section 5.

In Appendix B we also define what does it mean for a class of $\Delta$-matroids to be what we call stabilizable, and prove that stabilizable $\Delta$-matroids are tractable:

**Theorem 7.** If $\Gamma$ is a stabilizable class of $\Delta$-matroid relations, then $\text{CSP}_{\text{edge}}(\Gamma)$ can be solved in polynomial time (by calling the algorithm for even $\Delta$-matroids polynomially many times).

As we show in Appendix C, stabilizable $\Delta$-matroids cover several known tractable classes of $\Delta$-matroids, namely co-independent [12], compact [17], local [9], and binary [14, 9] $\Delta$-matroids. To our best knowledge these are all the known tractable classes and according to [9] they are pairwise incomparable.

### 3 Implications

In this section we explain how our result implies full complexity classification of planar Boolean CSPs.

**Definition 8.** Let $\Gamma$ be a constraint language. Then $\text{CSP}_{\text{planar}}(\Gamma)$ is the set of $\Gamma$-instances for which there exists a planar graph $G(V,E)$ such that $v_1, \ldots, v_k$ is a face of $G$ (with nodes listed in counter-clockwise order) if and only if there is a unique constraint imposed on the tuple of variables $(v_1, \ldots, v_k)$.

It is also noted in [10] that checking whether an instance has a planar representation can be done efficiently (see eg. [16]) and hence it does not matter if we are given a planar drawing of $G$ as a part of the input or not. The planar restriction does lead to new tractable cases, for example planar NAE-3-SAT (Not-All-Equal 3-Satisfiability) [23].

**Definition 9.** A relation $R$ is called self-complementary if for all $T \in \{0,1\}^n$ we have $T \in R$ if and only if $T \oplus (1, \ldots, 1) \in R$.

**Definition 10.** For a tuple of Boolean variables $T = (x_1, \ldots, x_n)$, let $dT = (x_1 \oplus x_2, \ldots, x_n \oplus x_1)$. For $R$ relation and $\Gamma$ set of relations, let $dR = \{dT : T \in R\}$ and $d\Gamma = \{dR : R \in \Gamma\}$.
Since self-complementary relations don’t change when we flip all their coordinates, we can describe a self-complementary relation by looking at the differences of neighboring coordinates; this is exactly the meaning of \( dR \). Note that these differences are realized over edges of the given planar graph.

Knowing this, it is not so difficult to imagine that via switching to the planar dual of \( G \), one can reduce a planar CSP instance to some sort of edge CSP instance. This is in fact part of the following theorem from [10]:

**Theorem 11.** Let \( \Gamma \) be such that \( \text{CSP}(\Gamma) \) is NP-Hard. Then:

(a) If there is \( R \in \Gamma \) that is not self-complementary, then \( \text{CSP}_{\text{PLANAR}}(\Gamma) \) is NP-Hard.

(b) If every \( R \in \Gamma \) is self-complementary and there exists \( R \in \Gamma \) such that \( dR \) is not even \( \Delta \)-matroid, then \( \text{CSP}_{\text{PLANAR}}(\Gamma) \) is NP-Hard.

(c) If every \( R \in \Gamma \) is self-complementary and \( dR \) is an even \( \Delta \)-matroid, then \( \text{CSP}_{\text{PLANAR}}(\Gamma) \) is polynomial-time reducible to

\[
\text{CSP}_{\text{EDGE}}(d\Gamma \cup \{EVEN_1, EVEN_2, EVEN_3\})
\]

where \( EVEN_i = \{(x_1, \ldots, x_i) : x_1 \oplus \cdots \oplus x_i = 0\} \).

Using Theorem 7 we can finish this classification:

**Theorem 12** (Dichotomy for planar Boolean CSP). Let \( \Gamma \) be a constraint language. Then \( \text{CSP}_{\text{PLANAR}}(\Gamma) \) is tractable if either

(a) \( \text{CSP}(\Gamma) \) is tractable or;

(b) \( \Gamma \) contains only self-complementary relations \( R \) such that \( dR \) is an even \( \Delta \)-matroid.

Otherwise, \( \text{CSP}_{\text{PLANAR}}(\Gamma) \) is NP-Hard.

**Proof.** By Theorem 11 the only unresolved case reduces to solving

\[
\text{CSP}_{\text{EDGE}}(d\Gamma \cup \{EVEN_1, EVEN_2, EVEN_3\}).
\]

Since the relations \( EVEN_i \) are even \( \Delta \)-matroids for every \( i \), this is polynomial-time solvable thanks to Theorem 7.

4 **Even \( \Delta \)-matroids and Matchings**

In this section we highlight the similarities and dissimilarities between even \( \Delta \)-matroid CSPs and matching problems. These similarities will guide us on our way through the rest of the paper.

**Example 13.** For \( n \in \mathbb{N} \) consider the “perfect matching” relation \( M_n \subseteq \{0, 1\}^n \) containing precisely the tuples in which exactly one coordinate is set to one and all others to zero. Note that \( M_n \) is an even \( \Delta \)-matroid for all \( n \). Then the instance \( I \) of \( \text{CSP}_{\text{EDGE}}(\{M_n : n \in \mathbb{N}\}) \) (represented in Figure 1) is equivalent to deciding whether the graph of the instance has a perfect matching (every node is adjacent to precisely one edge with label 1).

One may also construct an equivalent instance \( I' \) by “merging” some parts of the graph (in the figure those are \( X \) and \( Y \)) to single constraint nodes. The constraint relations imposed on the “supernodes” record sets of outgoing edges which can be extended to a perfect matching on the
subgraph induced by the “supernode”. For example, in the instance $I'$ the constraints imposed on $X$ and $Y$ would be (with variables ordered as in Figure 1):

$$X = \{10000, 01000, 00100, 00010, 10011, 11001, 10101\}, \quad Y = \{001, 010, 100, 111\}.$$  

It is easy to check that both $X$ and $Y$ are even $\Delta$-matroids.

One takeaway from this example is that any algorithm that solves edge CSP for the even $\Delta$-matroid case has to work for perfect matchings in graphs as well. Another is the construction of even $\Delta$-matroids $X$ and $Y$ which can be generalized as follows.

**Definition 14** (Matching realizable relations). Let $G$ be a graph and let $v_1, \ldots, v_a \in V(G)$ be distinct nodes of $G$. For an $a$-tuple $T = (x_1, \ldots, x_a) \in \{0, 1\}^a$, we denote by $G_T$ the graph obtained from $G$ by deleting all nodes $v_i$ such that $x_i = 1$. Then we can define

$$M(G,v_1,\ldots,v_a) = \{T \in \{0, 1\}^a : G_T \text{ has a perfect matching}\}.$$  

We say that a relation $R \in \{0, 1\}^a$ is matching realizable if $R = M(G,v_1,\ldots,v_a)$ for some graph $G$ and nodes $v_1,\ldots,v_a \in V(G)$.

Every matching realizable relation is an even $\Delta$-matroid [10]. Also, it should be clear from the definition and the preceding example that CSP$_{\text{EDGE}}(\Gamma)$ is tractable if $\Gamma$ contains only matching realizable relations (assuming we know the graph $G$ and the nodes $v_1,\ldots,v_a$ for each relation): One can simply replace each constraint node with the corresponding graph and then test for existence of perfect matching.

The authors of [10] also verify that every even $\Delta$-matroid of arity at most 5 is matching realizable. However, as we prove in Appendix A this is not true for higher arities.

**Proposition 15.** There exists an even $\Delta$-matroid of arity 6 which is not matching realizable.

Proposition [15] shows that we cannot hope to simply replace the constraint nodes by graphs and run the Edmonds’ algorithm.
5 Algorithm

5.1 Setup

We can draw edge CSP instances as constraint graphs: The constraint graph $G_I = (V \cup C, E)$ of $I$ is a bipartite graph with partitions $V$ and $C$. There is an edge $\{v, C\} \in E$ if and only if $v$ belongs to the scope of $C$. Throughout the rest of the paper we use lower-case letters for variable nodes in $V$ ($u, v, x, y, \ldots$) and upper-case letters for constraint nodes in $C$ ($A, B, C, \ldots$). Since we are dealing with edge CSP, the degree of each node $v \in V$ in $G_I$ is exactly two and since we don’t allow a variable to appear in a constraint twice, $G_I$ has no multiple edges. For such instances $I$ we introduce the following terminology and notation.

Definition 16. An edge labeling of $I$ is a mapping $f : E \rightarrow \{0, 1\}$. For a constraint $C \in C$ with the scope $\sigma$ we will denote by $f(C)$ the tuple in $\{0, 1\}^\sigma$ such that $f(C)(v) = f(\{v, C\})$ for all $v \in \sigma$. Edge labeling $f$ will be called valid if $f(C) \in C$ for all $C \in C$.

Variable $v \in V$ is called consistent in $f$ if $f(\{v, A\}) = f(\{v, B\})$ for the two distinct edges $\{v, A\}, \{v, B\} \in E$ of $G_I$. Otherwise, $v$ is inconsistent in $f$.

A valid edge labeling $f$ is optimal if its number of inconsistent variables is minimal among all valid edge labelings of $I$. Otherwise $f$ is called non-optimal.

Note that $I$ has a solution if and only if an optimal edge labeling $f$ of $I$ has no inconsistent variables.

Let $|I|$ be the size of input instance $I$, where we assume that the constraint relations are given by lists of tuples. The main theorem we prove is the following strengthening of Theorem 7.

Theorem 17. Given an edge CSP instance $I$ with even $\Delta$-matroid constraints, an optimal edge labeling $f$ of $I$ can be found in time polynomial in $|I|$.

Note that if the $\Delta$-matroids in $I$ were given by oracles then our algorithm (in particular our method of contracting blossoms) would not be polynomial.

Walks and blossoms When studying matchings in a graph, augmenting paths are important. We will use analogous objects, called $f$-walks resp. augmenting $f$-walks.

Definition 18. A walk $q$ of length $k$ in the instance $I$ is a sequence $q_0C_1q_1C_2\ldots C_kq_k$ where the variables $q_{i-1}, q_i$ lie in the scope of the constraint $C_i$, and each edge $\{v, C\} \in E$ is traversed at most once: $vC$ and $Cv$ occur in $q$ at most once, and they do not occur simultaneously.

Note that $q$ can be viewed as a walk in the graph $G_I$ that starts and ends at nodes in $V$. Since each node $v \in V$ has degree two in $G_I$, the definitions imply that $v$ can be visited by $q$ at most once, with a single exception: we may have $q_0 = q_k = v$, with $q = vC\ldots Dv$ where $C \neq D$. We allow walks of length 0 for formal reasons.

A subwalk of $q$, denoted by $q[i,j]$, is the walk $q_iC_{i+1}\ldots C_jq_j$ (again, we need to start and end in a variable). The inverse walk to $q$, denoted by $q^{-1}$, is the sequence $q_kC_k\ldots q_1C_1q_0$. Given two walks $p$ and $q$ such that the last node of $p$ is the first node of $q$, we define their concatenation $pq$ in the natural way. If $p = \alpha_1\ldots \alpha_k$ and $q = \beta_1\ldots \beta_l$ are sequences of nodes of a graph where $\alpha_k$ and $\beta_1$ are different but adjacent, we will denote the sequence $\alpha_1\ldots \alpha_k\beta_1\ldots \beta_l$ also by $pq$ (or sometimes as $p, q$).

If $f$ is an edge labeling of $I$ and $q$ a walk in $I$, we denote by $f \oplus q$ the mapping that takes $f$ and flips the values on all variable-constraint edges encountered in $q$, i.e.

\[
(f \oplus q)(\{v, C\}) = \begin{cases} 1 - f(\{v, C\}) & \text{if } q \text{ contains } vC \text{ or } Cv \\ f(\{v, C\}) & \text{otherwise} \end{cases}
\]
Definition 19. Let \( f \) be a valid edge labeling of instance \( I \). A walk \( q = q_0C_1q_1C_2\ldots C_kq_k \) with \( q_0 \neq q_k \) will be called an \( f \)-walk if

(a) variables \( q_1, \ldots, q_{k-1} \) are consistent in \( f \), and

(b) \( f \oplus q_{[0,i]} \) is a valid edge labeling for any \( i \in [1,k] \).

If in addition variables \( q_0 \) and \( q_k \) are inconsistent in \( f \) then \( q \) will be called an augmenting \( f \)-walk.

Later we will show that a valid edge labeling \( f \) is non-optimal if and only if there exists an augmenting \( f \)-walk. Note that one direction is straightforward: if \( p \) is an augmenting \( f \)-walk, then \( f \oplus p \) is valid and has 2 fewer inconsistent variables than \( f \).

Another structure used by the Edmonds’ algorithm for matchings is a blossom. The precise definition of a blossom in our setting (Definition [B4]) is a bit technical. Informally, an \( f \)-blossom is a walk \( b = b_0C_1b_1C_2\ldots C_kb_k \) with \( b_0 = b_k \) such that:

(a) variable \( b_0 = b_k \) is inconsistent in \( f \) while variables \( b_1, \ldots, b_{k-1} \) are consistent, and

(b) \( f \oplus b_{[i,j]} \) is a valid edge labeling for any non-empty proper subinterval \([i,j] \subseteq [0,k]\),

(c) there are not too many shortcuts inside \( b \) (we will make this precise later).

5.2 Algorithm description

Our algorithm will explore the graph \((V \cup C, E)\) building a directed forest \( T \). Each variable node \( v \in V \) will be added to \( T \) at most once. Constraint nodes \( C \in C \), however, can be added to \( T \) multiple times. To tell the copies of \( f \) apart (and to keep track of the order in which we built \( T \)), we will mark each \( C \) with a timestamp \( t \in \mathbb{N} \); the resulting node of \( T \) will be denoted as \( C^t \in C \times \mathbb{N} \). Thus, the forest will have the form \( T = (V(T) \cup C(T), E(T)) \) where \( V(T) \subseteq V \) and \( C(T) \subseteq C \times \mathbb{N} \).

The roots of the forest \( T \) constructed by the algorithm will be the inconsistent nodes of the instance (for current \( f \)); all non-root nodes in \( V(T) \) will be consistent. The edges of \( T \) will be oriented towards the leaves. Thus, each non-root node \( \alpha \in V(T) \cup C(T) \) will have exactly one parent \( \beta \in V(T) \cup C(T) \) with \( \beta \alpha \in E(T) \). For a node \( \alpha \in V(T) \cup C(T) \) let walk(\( \alpha \)) be the the unique path in \( T \) from a root to \( \alpha \). Note that walk(\( \alpha \)) is a subgraph of \( T \). Sometimes we will treat walks in \( T \) as sequences of nodes in \( V \cup C \) discussed in Sec. 5.1 (i.e. with timestamps removed); such places should be clear from the context.

We will grow the forest \( T \) in a greedy manner as shown in Algorithm [I]. The structure of the algorithm resembles that of the Edmonds’ algorithm for matchings [II], with the following important distinctions: First, in the Edmonds’ algorithm each “constraint node” (i.e. each node of the input graph) can be added to the forest at most once, while in Algorithm [I] some constraints \( C \in C \) can be added to \( T \) and “expanded” multiple times (i.e. \( E(T) \) may contain edges \( C^s u \) and \( C^t w \) added at distinct timestamps \( s \neq t \)). This is because we allow more general constraints. In particular, if \( C \) is a “perfect matching” constraint (i.e. \( C = \{(a_1, \ldots, a_k) \in \{0,1\}^k : a_1 + \ldots + a_k = 1\} \) then Algorithm [I] will expand it at most once. (We will not use this fact, and thus omit the proof.)

Note that even when we enter a constraint node for the second or third time, we “branch out” based on transitions \( vCw \) available before the first visit, even though it is not clear these are preserved. We will have to show in later sections that our algorithm avoids such “disappearing transitions”, which otherwise do exist as one may see for example by studying the non matching realizable even \( \Delta \)-matroid from Appendix [A].
Algorithm 1 Improving a given edge labeling

Input: Instance $I$, valid edge labeling $f$ of $I$.

Output: A valid edge labeling $g$ of $I$ with fewer inconsistent variables than $f$, or “No” if no such $g$ exists.

1. Initialize $T$ as follows: set timestamp $t = 1$, and for each inconsistent variable $v \in V$ of $I$ add $v$ to $T$ as an isolated root.

2. Pick an edge $\{v, C\} \in E$ such that $v \in V(T)$ but there is no $s$ such that $vC^s \in E(T)$ or $C^sv \in E(T)$. (If no such edge exists, then output “No” and terminate.)

3. Add new node $C^t$ to $T$ together with the edge $vC^t$.

4. Let $W$ be the set of all variables $w \neq v$ in the scope of $C$ such that $f(C) \oplus v \oplus w \in C$. For each $w \in W$ do the following (see Figure 2):
   
   (a) If $w \notin V(T)$, then add $w$ to $T$ together with the edge $C^tw$.
   
   (b) Else if $w$ has a parent of the form $C^s$ for some $s$, then do nothing.
   
   (c) Else if $v$ and $w$ belong to different trees in $T$ (i.e. originate from different roots), then we have found an augmenting path. Let $p = \text{walk}(C^t), \text{walk}(w)^{-1}$, output $f \oplus p$ and exit.
   
   (d) Else if $v$ and $w$ belong to the same tree in $T$, then we have found a blossom. Form a new instance $I^b$ and new valid edge labeling $f^b$ of $I^b$ by contracting this blossom. Solve this instance recursively, use the resulting improved edge labeling for $I^b$ (if it exists) to compute an improved valid edge labeling for $I$, and terminate. All details are given in Sec. 5.3.

5. Increase the timestamp $t$ by 1 and goto step 2.

A second distinction is that while the Edmonds’ algorithm does not impose any restrictions on the order in which the forest is grown, we require that all valid children $w \in W$ are added to $T$ simultaneously when exploring edge $\{v, C\}$ in step 4. Informally speaking, this will guarantee that forest $T$ does not have “shortcuts”, which will be essential in the proofs.

The correctness of Algorithm 1 will follow from the results below.

Theorem 20. If $I$ is a CSP instance, $f$ a valid edge labeling of $I$ and we run Algorithm 1, then the following is true:

(a) The mapping $f \oplus p$ from step 4c is a valid edge labeling of $I$ with fewer inconsistencies than $f$.

(b) When contracting a blossom, as described Section 5.3, $I^b$ is an edge CSP instance with even $\Delta$-matroid constraints and $f^b$ is a valid edge labeling to $I^b$.

(c) The recursion in 4d will occur at most $O(|V|)$ many times.

(d) In step 4d, $f^b$ is optimal for $I^b$ if and only if $f$ is an optimal for $I$. Moreover, given a valid edge labeling $g^b$ of $I^b$ with fewer inconsistent variables than $f^b$, we can in polynomial time output a valid edge labeling $g$ of $I$ with fewer inconsistent variables than $f$.

(e) If the algorithm answers “No” then $f$ is optimal.
Figure 2: A possible run of Algorithm 1 on the instance $I'$ from Example 13 (with renamed constraint nodes) where the edge labeling $f$ is marked by thick (1) and thin (0) half-edges. We see that the algorithm finds a blossom when it hits the variable $v$ the second time in the same tree. However, had we first processed the transition $Cx$ (which we could have done), we would have found an augmenting path $p = \text{walk}(C^5) \text{walk}(x)^{-1}$ (where $\text{walk}(x)^{-1}$ ends in $z$).

5.3 Contracting a blossom (Step 4d)

We now elaborate step 4d of Algorithm 1. First, we describe how to obtain a blossom $b$. Let $\alpha \in V(T) \cup C(T)$ be the lowest common ancestor of nodes $v$ and $w$ in $T$. Two cases are possible.

1. $\alpha = r \in V(T)$. Variable node $r$ must be inconsistent in $f$ because it has outdegree two. We let $b = \text{walk}(C^t), \text{walk}(w)^{-1}$ in this case.

2. $\alpha = R^s \in C(T)$. Let $r$ be the child of $R^s$ in $T$ that is an ancestor of $v$. Replace edge labeling $f$ with $f \oplus \text{walk}(r)$ (variable $r$ then becomes inconsistent). Now define walk $b = p, q^{-1}, r$ where $p$ is the walk from $r$ to $C^t$ in $T$ and $q$ is the walk from $R^s$ to $w$ in $T$ (see Figure 3).

Lemma 21 (To be proved in Section 6.3). Assume that Algorithm 1 reaches step 4d and one of the cases described in the above paragraph occurs. Then:

(a) in the case $\square$ the edge labeling $f \oplus \text{walk}(r)$ is valid, and

(b) in both cases the walk $b$ is an $f$-blossom (for the new edge labeling $f$, in the second case). (Note that we have not formally defined $f$-blossoms yet; they require some machinery that will come later – see Definition 34.)

To summarize, at this point we have a valid edge labeling $f$ of instance $I$ and an $f$-blossom $b = b_0C_1b_1 \ldots C_kb_k$. Let us denote by $L$ the set of constraints in the blossom, i.e. $L = \{C_1, \ldots, C_k\}$.

We construct a new instance $I^b$ and its valid edge labeling $f^b$ by contracting the blossom $b$ as follows: we take $I$, add one $|L|$-ary constraint $N$ to $I$, delete the variables $b_1, \ldots, b_k$, and add new variables $\{v_C : C \in L\}$ (see Figure 4). The scope of $N$ is $\{v_C : C \in L\}$ and the matroid of $N$ consists of exactly those maps $\alpha \in \{0,1\}^L$ that send one $v_C$ to 1 and the rest to 0.

In addition to all this, we replace each blossom constraint $D \in L$ by the constraint $D^b$ whose scope is $\sigma \setminus \{b_1, \ldots, b_k\} \cup \{v_D\}$ where $\sigma$ is the scope of $D$. The constraint relation of $D^b$ consists of all maps $\beta$ for which there exists $\alpha \in D$ such that $\alpha$ agrees with $\beta$ on $\sigma \setminus \{b_1, \ldots, b_k\}$ and one of the following occurs (see Figure 5):

10
Figure 3: The two cases of Step 4d. On the left, $\alpha = r$ is a variable, while on the right $\alpha = R^s$ is a constraint and the thick edges denote $p = \text{walk}(r)$. The dashed edges are orientations of edges from $E$ that are not in the digraph $T$, but belong to the blossom.

Figure 4: A blossom (left) and a contracted blossom (right) in the case when all constraints $C_1, \ldots, C_k$ are distinct. If some constraints appear in the blossom multiple times then the number of variables $v_D$ will be smaller than $k$ (see Figure 5).
Figure 5: Modification of a constraint node $D$ that appears in a blossom $b$ twice, i.e. when $b = \ldots b_{i-1}Db_i\ldots b_{j-1}Db_j\ldots$ (and so $D = C_i = C_j$). Variables $y$ and $z$ are not part of the walk. The construction of $D^b$ described in the text can be alternatively viewed as attaching a “gadget” constraint $Z_D$ as shown in the figure. Here $Z_D$ is an even $\Delta$-matroid with five tuples that depend on the values $\lambda_k = f(\{b_k, D\})$ and $\bar{\lambda}_k = 1 - \lambda_k$.

(a) $\beta(v_D) = 0$ and $\alpha$ agrees with $f(D)$ on all variables in $\{b_1, \ldots, b_k\} \cap \sigma$, or

(b) $\beta(v_D) = 1$ and there is exactly one variable $z \in \{b_1, \ldots, b_k\} \cap \sigma$ such that $\beta(z) \neq f(D)(z)$.

The lemma below follows from a more general result shown e.g. in [12, Theorem 4].

**Lemma 22.** Each $C^b_D$ is an even $\Delta$-matroid.

We define the edge labeling $f^b$ of $I^b$ as follows: for constraints $A \notin \{C_1, \ldots, C_k, N\}$ we set $f^b(A) = f(A)$. For each $C \in L$, we let $f^b(C^b)(v) = f(C)(v)$ when $v \neq v_C$, and $f^b(C^b)(v_C) = 0$. Finally, we let $f^b(N)(v_C) = 1$ for $C = C_1$ and $f^b(N)(v_C) = 0$ for all other $C$s. (The last choice is arbitrary; initializing $f^b(N)$ with any other tuple in $N$ would work as well).

It is easy to check that $f^b$ is valid for $I^b$. Furthermore, $v_{C_1}$ is inconsistent in $f^b$ while for each $C \in L \setminus \{C_1\}$ the variable $v_C$ is consistent.

**Observation 23.** In the situation described above, the instance $I^b$ will have at most as many variables as $I$ and one constraint more than $I$. Edge labelings $f$ and $f^b$ have the same number of inconsistent variables.

**Corollary 24** (Theorem 20(c)). Given an instance $I$, Algorithm 7 will recursively call itself $O(|V|)$ many times.

**Proof.** Since $C$ and $V$ are partitions of $G_I$ and the degree of each $v \in V$ is two, the number of edges of $G_I$ is $2|V|$. From the other side, the number of edges of $G_I$ is equal to the sum of arities of all constraints in $I$. Since we never consider constraints with empty scopes, the number of constraints of an instance is at most double the number of variables of the instance.

Since each contraction adds one more constraint and never increases the number of variables, it follows that there can not be a sequence of consecutive contractions longer than $2|V|$, which is $O(|V|)$.

The following two lemmas, which we prove in Section 5, show why the procedure works. In both lemmas, we let $(I, f)$ and $(I^b, f^b)$ denote the instance and the valid edge labeling before and after the contraction, respectively.
Lemma 25. In the situation described above, if \( f^b \) is optimal for \( I^b \), then \( f \) is optimal for \( I \).

Lemma 26. In the situation described above, if we are given a valid edge labeling \( g^b \) of \( I^b \) with fewer inconsistencies than \( f^b \), then we can find in polynomial time a valid edge labeling \( g \) of \( I \) with fewer inconsistencies than \( f \).

5.4 Time complexity of Algorithm

To see that Algorithm runs in time polynomial in the size of \( I \), consider first the case when step 4d does happen. In this case, the algorithm runs in time polynomial in the size of \( I \), since it essentially just searches through the graph \( G_I \).

Moreover, from the description of contracting a blossom in part 5.3, it is easy to see that one can compute \( I^b \) and \( f^b \) from \( I \) and \( f \) in polynomial time and that \( I^b \) is not significantly larger than \( I \): \( I^b \) has at most as many variables as \( I \) and the contracted blossom constraints \( C^b \) are not larger than the original constraints \( C \). Finally, \( I^b \) does have one brand new constraint \( N \), but \( N \) contains only \( O(|V|) \) many tuples. Therefore, we have \( |I^b| \leq |I| + O(|V|) \) where \( |V| \) does not change. By Corollary 24, there will be at most \( O(|V|) \) contractions in total, so the size of the final instance \( I^* \) is at most \( |I| + O(|V|^2) \), which is easily polynomial in \( |I| \).

All in all, Algorithm will give its answer in time polynomial in \( I \).

6 Proofs

In this section, we flesh out detailed proofs of the statements we gave above. In the whole section, \( I \) will be an instance of a Boolean edge CSP whose constraints are even \( \Delta \)-matroids.

In Sec. 6.1, we establish some properties of \( f \)-walks, and show in particular that a valid edge labeling \( f \) of \( I \) is non-optimal if and only if there exists an augmenting \( f \)-walk in \( I \). In Sec. 6.2, we introduce the notion of an \( f \)-DAG, prove that the forest \( T \) constructed during the algorithm is indeed an \( f \)-DAG, and describe some tools for manipulating \( f \)-DAGs. Then in Sec. 6.3 we analyze augmentation and contraction operations, namely prove Theorem 20 and Lemmas 21, 22, 25, 26 (which imply Theorem 22 and 23). Finally, in Sec. 6.4, we prove Theorem 24.

For edge labelings \( f, g \), let \( f \Delta g \subseteq E \) be the set of edges in \( E \) on which \( f \) and \( g \) differ. We open the proof section with an easy parity observation.

Observation 27. If \( f \) and \( g \) are valid edge labelings of instance \( I \) then they have the same number of inconsistencies modulo 2.

Proof. We use induction on \( |f \Delta g| \). The base case \( |f \Delta g| = 0 \) is trivial. For the induction step let us consider valid edge labelings \( f, g \) with \( |f \Delta g| \geq 1 \). Pick an edge \( \{v, C\} \in f \Delta g \). By the property of even \( \Delta \)-matroids there exists another edge \( \{w, C\} \in f \Delta g \) with \( w \neq v \) such that \( f(C) \oplus v \oplus w \in C \). Thus, edge labeling \( f^* = f \oplus (vCw) \) is valid. Clearly, \( f \) and \( f^* \) have the same number of inconsistencies modulo 2. By the induction hypothesis, the same holds for edge labelings \( f^* \) and \( g \) (since \( |f^* \Delta g| = |f \Delta g| - 2 \) ). This proves the claim.

6.1 The properties of \( f \)-walks

Let us begin with some results on \( f \)-walks that will be of use later. The following lemma is a (bit more technical) variant of the well known property of matchings in graphs:

Lemma 28. Let \( f, g \) be valid edge labelings of \( I \) such that \( g \) has fewer inconsistencies than \( f \), and \( x \) be an inconsistent variable in \( f \). Then there exists an augmenting \( f \)-walk that begins in a variable different from \( x \). Moreover, such a walk can be computed in polynomial time given \( I, f, g, \) and \( x \).
Proof. Our algorithm will proceed in two stages. First, we repeatedly modify the edge labeling $g$ using the following procedure:

1. Pick a variable $v \in V$ which is consistent in $f$, but not in $g$. (If no such $v$ exists then go to the next paragraph). By the choice of $v$, there exists a unique edge $\{v, C\} \in \Delta g$. Pick a variable $w \neq v$ in the scope of $C$ such that $\{w, C\} \in \Delta g$ and $g(C) \oplus v \oplus w \in C$ (it exists since $C$ is an even $\Delta$-matroid). Replace $g$ with $g \oplus (vCw)$, then go to the beginning and repeat.

It can be seen that $g$ remains a valid edge labeling, and the number of inconsistencies in $g$ never increases. Furthermore, each step decreases $|\Delta g|$ by 2, so this procedure must terminate after at most $O(|I|) = O(|V|)$ steps.

We now have valid edge labelings $f, g$ such that $f$ has more inconsistencies than $g$, and variables consistent in $f$ are also consistent in $g$. By Observation 27, $f$ has at least two more inconsistent variables than $g$; one of them must be different from $x$.

In the second stage we will maintain an $f$-walk $p$ and the corresponding valid edge labeling $f^* = f \oplus p$. To initialize, pick a variable $r \in V \setminus \{x\}$ which is consistent in $g$ but not in $f$, and set $p = r$ and $f^* = f$. We then repeatedly apply the following step:

2. Let $v$ be the endpoint of $p$. The variable $v$ is consistent in $g$ but not in $f^*$, so there is a unique edge $\{v, C\} \in \Delta g$. Pick a variable $w \neq v$ in the scope of $C$ such that $\{w, C\} \in \Delta g$ and $f^*(C) \oplus v \oplus w \in C$ (it exists since $C$ is an even $\Delta$-matroid). Append $vCw$ to the end of $p$, and accordingly replace $f^*$ with $f^* \oplus (vCw)$ (which is valid by the choice of $w$). As a result of this update, edges $\{v, C\}$ and $\{w, C\}$ are removed from $\Delta g$.

If $w$ is inconsistent in $f$, then output $p$ (which is an augmenting $f$-walk) and terminate. Otherwise $w$ is consistent in $f$ (and thus in $g$) but not in $f^*$; in this case, go to the beginning and repeat.

Each step decreases $|\Delta g|$ by 2, so this procedure must terminate after at most $O(|I|) = O(|V|)$ steps. It can also be seen that $p$ is indeed a walk. In particular, the starting node $r$ has exactly one incident edge in the graph $(V \cup C, f^* \Delta g)$. Since this edge is immediately removed from $f^* \Delta g$, we will never encounter the variable $r$ again during the procedure. $\square$

6.2 Invariants of Algorithm 1: $f$-DAGs

In this section we examine the properties of the forest $T$ as generated by Algorithm 1. For future comfort, we will actually allow $T$ to be a bit more general than what appears in Algorithm 1 – our $T$ can be a directed acyclic digraph (DAG):

**Definition 29.** Let $I$ be a Boolean edge CSP instance and $f$ a valid edge labeling of $I$. We will call a directed graph $T$ an $f$-DAG if $T = (V(T) \cup C(T), E(T))$ where $V(T) \subseteq V$, $C(T) \subseteq C \times \mathbb{N}$, and the following conditions hold:

(a) Edges of $E(T)$ have the form $vC^t$ or $C^tv$ where $\{v, C\} \in I$ and $t \in \mathbb{N}$.

(b) For each $\{v, C\} \in I$ there is at most one $t \in \mathbb{N}$ such that $vC^t$ or $C^tv$ appears in $E(T)$. Moreover, $vC^t$ and $C^tv$ are never both in $E(T)$.

(c) Each node $v \in V(T)$ has at most one incoming edge. (Note that by the previous properties, the node $v$ can have at most two incident edges in $T$.)
(d) Timestamps $t$ for nodes $C^t \in \mathcal{C}(T)$ are all distinct (and thus give a total order on $\mathcal{C}(T)$). Moreover, this order can be extended to a total order $< \mathcal{C}(T)$ such that $\alpha < \beta$ for each edge $\alpha \beta \in E(T)$. (So in particular the digraph $T$ is acyclic.)

(e) If $T$ contains edges $uC^t$ and one of $vC^t$ or $C^t v$, then $f(C) \oplus u \oplus v \in C$.

(f) (“No shortcuts” property) If $T$ contains edges $uC^s$ and one of $vC^t$ or $C^t v$ where $s < t$, then $f(C) \oplus u \oplus v \notin C$.

It is easy to verify that any subgraph of an $f$-DAG is also an $f$-DAG. If $T$ is an $f$-DAG, then we denote by $f \oplus T$ the edge labeling we obtain from $f$ by flipping the value of any $f(\{v, C\})$ such that $vC^t \in E(T)$ for some timestamp $t$. We will need to show that $f \oplus T$ is a valid edge labeling for nice enough $f$-DAGs $T$.

The following observation will be implicitly used throughout the proof: if $C^*, C^t$ are distinct constraint nodes in an $f$-DAG $T$ and $T$ contains one of the edges $\{uC^s, C^s u\}$ and one of the edges $\{vC^t, C^t v\}$, then $u \neq v$.

The following lemma shows the promised invariant property:

**Lemma 30.** Let us consider the structure $T$ during the run of Algorithm 4 with the input $I$ and $f$. At any moment during the run, the forest $T$ is an $f$-DAG.

Moreover, if steps 4a or 4d are reached, then the digraph $T^*$ obtained from $T$ by removing all edges outgoing from $C^t$ and adding the edge $wC^t$ is also an $f$-DAG.

**Proof.** Obviously, an empty $T$ is an $f$-DAG, as is the initial $T$ consisting of inconsistent variables and no edges. To verify that $T$ remains an $f$-DAG during the whole run of Algorithm 4, we need to make sure that neither adding $vC^t$ in step 3 nor adding $C^t w$ in step 4a violates the properties of $T$. Let us consider step 3 first. By the choice of $v$ and $C^t$, we immediately get that properties 1a, 1b, 1c, and 1d all hold even after we have added $vC^t$ to $T$ (we can order the nodes by the order in which they were added to $T$). Since there is only one edge incident with $C^t$, property 1 holds as well. Finally, the only way the no shortcuts property (i.e. property 1f) could fail be if there were some $u$ and $s$ such that $uC^s \in E(T)$ and $f(C) \oplus u \oplus v \in C$. But then, after the node $C^s$ got added to $T$, we should have computed the set $W$ of variables $w$ such that $f(C) \oplus v \oplus w$ (step 4) and $u$ should have been in $W \setminus V(T)$ at that time, i.e. we should have added the edge $C^s w$ before, a contradiction. The analysis of step 4a is similar.

Assume now that Algorithm 4 has reached one of steps 4c or 4d and consider the DAG $T^*$ we get from $T$ by removing all edges of the form $C^t z$ and adding the edge $wC^t$. Note that the node $C^t$ is the only node with two incoming edges. The only three properties that this could possibly affect are 2a, 2b, and 2c. Were 2b violated, we would have $C^s w \in E(T)$ already, and so step 4b would be triggered instead of steps 4c or 4d. For property 2a, the only new pair of edges to consider is $vC^t$ and $wC^t$ for which we have $f(C) \oplus v \oplus w \in C$. Finally, if property 2c became violated after adding the edge $wC^t$ then there were a $u$ and $s < t$ such that $uC^s \in E(T)$ and $f(C) \oplus u \oplus v \in C$. Node $C^s$ must have been added after $w$, or else we would have $C^s w \in E(T)$. Also, $w$ cannot have a parent of the form $C^k$ (otherwise step 4b would be triggered for $w$ when expanding $C^t$). But then one of steps 4c or 4d would be triggered at timestamp $s$ already when we tried to expand $C^s$, a contradiction.

We will use the following two lemmas to prove that $f \oplus p$ is a valid edge labeling of $I$ for various paths $p$ that appear in steps 4c and 4d.

**Lemma 31.** Let $T$ be an $f$-DAG, and $C^s$ be the constraint node in $\mathcal{C}(T)$ with the smallest timestamp $s$. Suppose that $C^s$ has exactly two incident edges, namely incoming edge $uC^s$ where $u$ does not
Let \( f \) be an edge CSP instance and \( T \) be a valid edge labeling of \( I \) and \( T^* \) is an \( f^* \)-DAG.

Proof. Since \( T^* \) is a subgraph of \( T \), it immediately follows that \( T^* \) satisfies the properties (a), (b), (c), and (d) from the definition of an \( f \)-DAG all hold.

Let us show that \( T^* \) has property (e). Consider a constraint node \( (u,C^s) \in \mathcal{C}(T^*) \) with \( t > s \) (nothing has changed for other constraint nodes in \( \mathcal{C}(T^*) \)), and suppose that \( T^* \) contains edges \( uC^s \) and one of \( yC^t \) or \( C^t y \). If \( x = y \), the situation is trivial, so assume that \( u,v,x,y \) are all distinct variables. We need to show that \( f^*(C) \oplus x \oplus y \in C \). The constraint \( C \) contains the tuples \( f(C) \oplus u \oplus v \) and \( f(C) \oplus x \oplus y \) (by condition (c) for \( T \)), but the no shortcuts property prohibits the tuples \( f(C) \oplus u \oplus x \) and \( f(C) \oplus u \oplus y \) from lying in \( C \). Therefore, applying the even \( \Delta \)-matroid property on \( f(C) \oplus u \oplus v \) and \( f(C) \oplus x \oplus y \) in the variable \( u \) we get that \( C \) must contain \( f(C) \oplus u \oplus v \oplus x \oplus y \), so we have \( f^*(C) \oplus x \oplus y \in C \).

Now let us prove that \( T^* \) and \( f^* \) have the “no shortcuts” property. Consider constraint nodes \( (u,C^s) \in \mathcal{C}(T^*) \) with \( s < k < \ell \) (since nothing has changed for other pairs of constraint nodes), and suppose that \( T^* \) contains edges \( uC^k \) and one of \( yC^t \) or \( C^t y \), where again \( u,v,x,y \) are all distinct variables. We need to show that \( f^*(C) \oplus x \oplus y \notin C \), or equivalently that \( f(C) \oplus u \oplus v \oplus x \oplus y \notin C \).

Assume that it is not the case. Apply the even \( \Delta \)-matroid property to tuples \( f(C) \oplus u \oplus v \oplus x \oplus y \) and \( f(C) \) (which are both in \( C \)) in coordinate \( v \). We get that either \( f(C) \oplus x \oplus y \in C \), or \( f(C) \oplus u \oplus v \oplus x \oplus y \in C \). This contradicts the “no shortcuts" property for the pair \( (C^k,C^t) \), resp. \( (C^s,C^t) \), resp. \( (C^s,C^k) \), so we are done.

\[ \square \]

Corollary 32. Let \( I \) be an edge CSP instance and \( f \) be a valid edge labeling.

(a) Let \( T \) be an \( f \)-DAG that consists of two directed paths \( x_0C^t_1x_1 \ldots x_{k-1}C^k_k \) and \( y_0D^{s_1}_1 \ldots y_{\ell-1}D^{s_\ell}_\ell \) that are disjoint everywhere except at the constraint \( C^k_k = D^{s_\ell}_\ell \) (see Figure 7). Then \( f \oplus T \) is a valid edge labeling of \( I \).

(b) Let \( T \) be an \( f \)-DAG that consists of a single directed path \( x_0C^t_1x_1 \ldots x_{k-1}C^k_k \). Then \( f \oplus T \) is a valid edge labeling of \( I \).

Proof. We will prove only part (a); the proof of part (b) is completely analogous. We proceed by induction on \( k + \ell \). If \( k = \ell = 1 \), \( T \) consists only of the two edges \( x_0C^t_1 \) and \( y_0C^s_1 \) (where \( C^t_1 \) is an
Suppose that $T$ can use Lemma 31 for $v$ element in this order (just take the linear order on nodes of $T$).

Proof. It is easy to verify that $f \oplus (x_0 C y_0)$ is a valid edge labeling follows from the property (e) of $f$-DAGs.

If we are now given an $f$-DAG $T$ of the above form, then we compare $t_1$ and $s_1$. If $s_1 > t_1$, we can use Lemma 31 for $x_1 C_{t_1} x_2$ (there is a $x_2$ since $t_k > s_1 > t_1$), obtaining the $f \oplus (x_1 C_{x_2})$-DAG $T^*$ that consists of two directed paths $x_2 \ldots x_k C^{t_k}$ and $y_1 D^{s_1}_1 \ldots y_i D^{s_i}_i$. Since $T^*$ is shorter than $T$, the induction hypothesis gets us that $f \oplus (x_1 C_{x_2}) \oplus T^* = f \oplus T$ is a valid edge labeling.

If $t_1 > s_1$, we do the same thing with $y_1 D_{1 y_2}$ instead of $x_1 C_{x_2}$.

Lemma 33. Let $T$ be an $f$-DAG, and $C^s$ be the constraint node in $C(T)$ with the smallest timestamp $s$. Suppose that $C^s$ has exactly one incoming edge $u C^s$, and $u$ does not have other incident edges besides $u C^s$. Suppose also that $C^s$ has an outgoing edge $C^s v$. Let $f^* = f \oplus (u C^s v)$, and $T^*$ be the DAG obtained from $T$ by removing the edge $u C^s$ together with $u$ and reversing the orientation of edge $C^s v$ (see Figure 8).

Then $f^*$ is a valid edge labeling of $I$ and $T^*$ is an $f^*$-DAG.

Proof. It is easy to verify that $T^*$ satisfies the properties (a), (b) and (c). To see property (d), just take the linear order on nodes of $T$ and change the position of $v$ so that it is the new minimal element in this order ($v$ has no incoming edges in $T^*$).

Let us prove that property (e) of Definition 29 is preserved. First, consider constraint node $C^s$. Suppose that $T^*$ contains one of $x C^s$ or $C^s x$ with $x \neq v$. We need to show that $f^*(C) \oplus u \oplus x \in C$, or equivalently $f(C) \oplus u \oplus x \in C$ (since $f^*(C) \oplus v = f(C) \oplus (u \oplus v) \oplus v = f(C) \oplus u$). This claim holds by property (e) of Definition 29 for $T$.

Now consider a constraint node $C^{t'}$ in $C(T^*)$ with $t > s$, and suppose that $T^*$ contains edges $xC^{t'}$ and one of $y C^{t'}$ or $C^{t'} y$. We need to show that $f^*(C) \oplus x \oplus y \in C$, or equivalently that $f(C) \oplus u \oplus v \oplus x \oplus y \in C$. For that we can simply repeat word-by-word the argument used in the proof of Lemma 31.
Now let us prove that the “no shortcuts” property is preserved. First, consider a constraint node $C^t$ in $\mathcal{C}(T^*)$ with $t > s$, and suppose that $T^*$ contains one of $xC^t$ or $C^t x$. We need to show that $f^*(C) \oplus v \oplus x \notin C$, or equivalently $f(C) \oplus u \oplus x \notin C$. This claim holds by the “no shortcuts” property for $T$. Now consider constraint nodes $C^k, C^t$ in $\mathcal{C}(T^*)$ with $s < k < t$, and suppose that $T^*$ contains edges $xC^k$ and one of $yC^t$ or $C^t y$. Note that $u, v, x, y$ are all distinct variables. We need to show that $f^*(C) \oplus x \oplus y \notin C$, or equivalently that $f(C) \oplus u \oplus v \oplus x \oplus y \notin C$. For that we can simply repeat word-by-word the argument used to show the no shortcuts property in the proof of Lemma 31.

### 6.3 Analysis of augmentations and contractions

First, we prove the correctness of the augmentation operation, i.e. that edge labeling $f \oplus p$ in step 4c is valid.

**Proof of Theorem 20(a).** Let $T_1$ be the $f$-DAG constructed during the run of Algorithm 1, let $T_2$ be the DAG obtained from $T_1$ by adding the edge $uC^t$. By Lemma 30 $T_2$ is an $f$-DAG. Let $T_3$ be the subgraph of $T_2$ induced by the nodes in $p$. It is easy to verify that $T_3$ consists of two directed paths that share their last node. Therefore, by Corollary 32 we get that $f \oplus T_3 = f \oplus p$ is a valid edge labeling of $I$.

In the remainder of this section we show the correctness of the contraction operation by proving Lemmas 21, 22, 25, 26. Let us begin by giving a full definition of a blossom:

**Definition 34.** Let $f$ be a valid edge labeling. An $f$-blossom is any walk $b = b_0C_1b_1C_2...b_k$ with $b_0 = b_k$ such that:

(a) variable $b_0 = b_k$ is inconsistent in $f$ while variables $b_1, ..., b_{k-1}$ are consistent, and

(b) there exists $\ell \in [1, k]$ and timestamps $t_1, ..., t_k$ such that the DAG consisting of two directed paths $b_0C_{t_1}b_1C_{t_2}...b_{\ell-1}C_{t_\ell}$ and $b_kC_{t_{k-1}}b_{k-1}C_{t_{k-2}}...b_{\ell}C_{t_\ell}$ is an $f$-DAG.

**Lemma 35.** Let $b$ be a blossom. Then $b_{[i,j]}$ is an $f$-walk for any non-empty proper subinterval $[i, j) \subseteq [0, k]$.

**Proof.** Let us denote the $f$-DAG from the definition of a blossom by $B$. By taking an appropriate subgraph of $B$ and applying Corollary 32 we get that $f \oplus b_{[i,j]}$ is valid for any non-empty subinterval $[i, j) \subseteq [0, k]$. Since the set of these intervals is downward closed, $b_{[i,j]}$ is in fact an $f$-walk.

**Lemma (Lemma 21).** Assume that Algorithm 1 reaches step 4a and one of the cases described at the beginning of Section 5.3 occurs. Then:

(a) in the case 3 the edge labeling $f \oplus \text{walk}(r)$ is valid, and

(b) in both cases the walk $b$ is an $f$-blossom (for the new edge labeling $f$, in the second case).

**Proof.** Let $T$ be the forest at the moment of contraction, $T^\dagger$ be the subgraph of $T$ containing only paths walk($C^t$) and walk($w$), and $T^*$ be the graph obtained from $T^\dagger$ by adding the edge $uC^t$. By Lemma 30 graph $T^*$ is an $f$-DAG (we also need to observe that any subgraph of an $f$-DAG is again an $f$-DAG).

If the lowest common ancestor of $u$ and $v$ in $T$ is a variable node $r \in V(T)$ (i.e. we have case 1 from Section 5.3), then the $f$-DAG $T^*$ consists of two directed paths from $r$ to the constraint $C$. 

18
and it is easy to verify that when we let \( b \) to be one of these paths followed by the other in reverse, we get a blossom.

Now consider case 2, i.e. when the lowest common ancestor of \( u \) and \( v \) in \( T \) is a constraint node \( R^* \in \mathcal{C}(T) \). Note that \( T^* \) has a unique source node \( u \) (that does not have incoming edges), and \( v \) has an outgoing edge \( uD^t \) where \( D^t \) is the constraint node with the smallest timestamp in \( T^* \). Let us repeat the following operation while \( u \) has an outgoing edge.

**Lemma (Lemma 25).** In the situation described above, if \( f^b \) is optimal for \( I^b \), then \( f^b \) is optimal if and only if \( f^b \) is optimal for \( I \).

**Proof.** Assume that \( f \) is not optimal for \( I \), so there exists a valid edge labeling \( g \) with fewer inconsistencies than \( f \). Then by Lemma 23, there exists an augmenting walk \( p \) in \( I \) that starts at some node other than \( b_k \). Denote by \( p^b \) the sequence obtained from \( p \) by replacing each \( C_i \) from the blossom by \( C^b_i \). Observe that if \( p \) does not contain the variables \( b_1, \ldots, b_k \), then \( p \) is an \( f \)-walk if and only if \( p^b \) is an \( f^b \)-walk, so the only interesting case is when \( p \) enters the set \( \{b_1, \ldots, b_k\} \).

We will proceed along \( p \) and consider the first \( i \) such that there is a blossom constraint \( D \) and an index \( j \) for which \( p_{[0,i]} Db_j \) is an \( f \)-walk (i.e. we can enter the blossom from \( p \)).

If \( D = C_1 \), then it follows from the definition of \( C^b_1 \) that \( p_{[0,i]} C^b_1 v_{C_1} \) is an augmenting \( f^b \)-walk in \( I^b \), while if \( D \neq C_1 \), then \( p_{[0,i]} D^b v_{C_D} N_{V_{C_1}} \) is an augmenting \( f^b \)-walk. In both cases, we get an augmenting walk for \( f^b \), and so \( f^b \) were not optimal.

To show the other direction, we will first prove the following result.

**Lemma 36.** Let \( q \) be an \( f \)-walk and \( T \) an \( f \)-DAG such that there is no proper prefix \( q^* \) of \( q \) and no edge \( vC^* \) or \( C^*v \) of \( T \) such that \( q^*Cv \) would be an \( f \)-walk. Then \( T \) is a \( (f \oplus q) \)-DAG.
Lemma (Lemma 26). In the situation described above, if we are given a valid edge labeling \( g^b \) of \( I^b \) with fewer inconsistencies than \( f^b \), then we can find in polynomial time a valid edge labeling \( g \) of \( I \) with fewer inconsistencies than \( f \).

Proof. Our overall strategy here is to take an inconsistency from the outside of the blossom \( b \) and bring it into the blossom. We begin by showing how to get a valid edge labeling \( f' \) for \( I \) with an inconsistent variable just one edge away from \( b \).

Using Lemma 28, we can use \( g^b \) and \( f^b \) to find in polynomial time an augmenting \( f^b \)-walk \( p^b \) that does not begin at the inconsistent variable \( v_C \). If \( p^b \) does not contain any of the variables \( v_{C_1}, \ldots, v_{C_k} \), then we can just output the walk \( p \) obtained from \( p^b \) by replacing each \( C_i^b \) by \( C_i \) and be done. Assume now that some \( v_C \) appears in \( p^b \). We choose the \( f^b \)-walk \( r^b \) so that \( r^b v_C \) is the shortest prefix of \( p^b \) that ends with some blossom variable \( v_C \). By renaming all \( C_i^b \)'s in \( r^b \) to \( C_s \), we get the walk \( r \). It is straightforward to verify that \( r \) is an \( f \)-walk and that \( rC_ib_i \) or \( rC_ib_i-1 \) is an \( f \)-walk for some \( i \in [1, k] \). Let \( q \) be the shortest prefix of \( r \) such that one of \( qC_ib_i \) or \( qC_ib_i-1 \) is an \( f \)-walk for some \( i \in [1, k] \).

Recall that the blossom \( b \) originates from an \( f \)-DAG \( B \). The minimality of \( q \) allows us to apply Lemma 36 and obtain that \( B \) is also an \( (f \oplus q) \)-DAG. Let \( f' = f \oplus q \) and let \( x \) be the last variable in \( q \). It is easy to see that \( f' \) is a valid edge labeling with exactly as many inconsistent variables as \( f \). Moreover \( x \) is inconsistent in \( f' \) and there is an index \( i \) such that at least one of \( xC_i b_i \) or \( xC_i b_i-1 \) is an \( f' \)-walk. We will now show how to improve \( f' \).

If the constraint \( C_i \) appears only once in the blossom \( b \), it is easy to verify (using Lemma 35) that one of \( xC_i b_{i[k]} \) or \( xC_i b_{[0,i-1]}^{-1} \) is an augmenting \( f' \)-walk. However, since the constraint \( C_i \) might appear in the blossom several times, we have to come up with a more elaborate scheme to handle that. The blossom \( b \) comes from an \( f \)-DAG \( B \) in which some node \( C_i^\ell \) is the node with the maximal timestamp (for a suitable \( \ell \in [1, k] \)). Assume first that there is a \( j \in [\ell, k] \) such that \( xC_j b_j \) is an \( f' \)-walk. In that case, we take maximal such \( j \) and consider the DAG \( B' \) we get by adding the edge \( C_j^\ell x \) to the subgraph of \( B \) induced by the nodes \( C_j^\ell, b_j, C_j^{\ell+1}, \ldots, C_k^\ell, b_k \).

It is routine to verify that \( B' \) is an \( f' \)-DAG; the only thing that could possibly fail is the no shortcuts property involving \( C_j^\ell \). However, \( C_j^\ell \) has maximal timestamp in \( B' \) and there is no \( i > j \) such that \( f'(C_j) \oplus x \oplus b_i \in C_j \).

Using Corollary 32, we get that \( f' \oplus B' \) is a valid edge labeling which has fewer inconsistencies than \( f' \), so we are done. In a similar way, we can improve \( f' \) when there exists a \( j \in [1, \ell] \) such that \( xC_j b_j \) is an \( f' \)-walk.

If neither of the above cases occurs, then we take \( j \) such that the timestamp \( t_j \) is maximal and either \( xC_j b_j \) or \( xC_j b_{j-1} \) is an \( f' \)-walk. Without loss of generality, let \( xC_j b_j \) be an \( f' \)-walk. Then \( j <
ℓ and we consider the DAG B′ we get from the subgraph of B induced by 𝐶𝑡ℓ, 𝑗, 𝐶𝑡j+1, ..., 𝐶𝑡k, 𝑘 by adding the edge 𝑥𝐶𝑡𝑗 (see Figure 9). As before, the only way B′ can not be an 𝑓′-DAG is if the no shortcuts property fails, but that is impossible: we chose 𝑗 so that 𝑡𝑗 is maximal, so an examination of the makeup of B′ shows that the only bad thing that could possibly happen is if there were an index 𝑖 ≥ ℓ such that 𝐶𝑖 = 𝐶𝑗, we had in B the edge 𝑏𝑖𝐶𝑡𝑖, and 𝑓′(𝐶𝑖) ⊕ 𝑏𝑖 ⊕ 𝑥 ∈ 𝐶𝑖. But then we would have the 𝑓′-walk 𝑥𝐶𝑡𝑖𝑏𝑖 for 𝑖 ≥ ℓ and the procedure from the previous paragraph would apply. Using Corollary 32, we again see that 𝑓′ ⊕ B′ is a valid edge labeling with fewer inconsistencies than 𝑓.

It is easy to verify that finding 𝑞, calculating 𝑓′ = 𝑓 ⊕ 𝑞, finding an appropriate 𝑗 and augmenting 𝑓′ can all be done in time polynomial in the size of the instance.

6.4 Proof of Theorem 20(e)

In this section we will prove that if the algorithm answers “No” then 𝑓 is an optimal edge labeling.

Lemma 37. Suppose that Algorithm 1 outputs “No” in step 2, without ever visiting steps 4c and 4d. Then 𝑓 is optimal.

Proof. Let T be the forest upon termination, and denote

\[ \mathcal{E}(T) = \{ C v | C^t v ∈ E(T) \text{ for some } t \} \cup \{ v C | v C^t ∈ E(T) \text{ for some } t \}. \]

Inspecting Algorithm 1, one can check that \( \mathcal{E}(T) \) has the following properties:

(a) If 𝑣 is an inconsistent variable in 𝑓 and \{ 𝑣, C \} ∈ 𝐸, then 𝑣C ∈ \( \mathcal{E}(T) \).

(b) If \( v C ∈ \mathcal{E}(T) \) and \{ 𝑣, 𝐷 \} ∈ 𝐸, 𝐷 ≠ 𝐶, then 𝑣D ∈ \( \mathcal{E}(T) \).

(c) If 𝑣C ∈ \( \mathcal{E}(T) \), then 𝑣C ∉ \( \mathcal{E}(T) \).

(d) Suppose that 𝑣C ∈ \( \mathcal{E}(T) \) and \( f(C) ⊕ v ⊕ w ∈ C \) where 𝑣, 𝑤 are distinct nodes in the scope of constraint 𝐶. Then 𝑤C ∈ \( \mathcal{E}(T) \).

An 𝑓-walk 𝑝 will be called bad if it starts at a variable node which is inconsistent in 𝑓, and contains an edge 𝐶v ∉ \( \mathcal{E}(T) \); otherwise 𝑝 is good. Clearly, any augmenting 𝑓-walk is bad: its last edge 𝐶v satisfies 𝑣C ∈ \( \mathcal{E}(T) \) by property a, and thus 𝑣C ∉ \( \mathcal{E}(T) \). Thus, if 𝑓 is not optimal, then
there exists at least one bad $f$-walk. Let $p$ be a shortest bad $f$-walk. Write $p = p^*(vCw)$ where $p^*$ ends at $v$. By minimality of $p$, $p^*$ is good and $Cw \notin \overline{E}(T)$. Using properties (a) or (b), we obtain that $vC \in \overline{E}(T)$ (and therefore $Cv \notin \overline{E}(T)$).

Let $q$ be the shortest prefix of $p^*$ (also an $f$-walk) such that $f \oplus q \oplus (vCw)$ is valid (at least one such prefix exists, namely $q = p^*$). The walk $q$ must be of positive length (otherwise the precondition of property (d) would hold, and we would get $Cw \in \overline{E}(T)$, a contradiction). Also, the last constraint node in $q$ must be $C$, otherwise we could have taken a shorter prefix. Thus, we can write $q = q^*(xCy)$ where $q^*$ ends at $x$. Note that, since $p$ is a walk, the variables $x, y, v, w$ are (pairwise) distinct.

We shall write $g = f \oplus q^*$. Let us apply the even $\Delta$-matroid property to tuples $g(C) \oplus x \oplus y \oplus v \oplus w$ and $g(C)$ (which are both in $C$) in coordinate $y$. We get that either $g(C) \oplus v \oplus w \in C$, or $g(C) \oplus x \oplus v \in C$, or $g(C) \oplus x \oplus w \in C$. In the first case we could have chosen $q^*$ instead of $q$ – a contradiction to the minimality of $q$. In the other two cases $q^*(xCu)$ is an $f$-walk for some $u \in \{v, w\}$. But then from $Cu \notin \overline{E}(T)$ we get that $q^*(xCu)$ is a bad walk – a contradiction to the minimality of $p$.

**Corollary 38** (Theorem 20(c)). If Algorithm 2 answers “No”, then the edge labeling $f$ is optimal.

**Proof.** Algorithm 1 can answer “No” for two reasons: either the forest $T$ can not be grown further and neither an augmenting path nor a blossom are found, or the algorithm finds a blossom $b$, contracts it and then concludes that $f^b$ is optimal for $I^b$. We proceed by induction on the number of contractions that have occurred during the run of the algorithm.

The base case, when there were no contractions, follows from Lemma 37. The induction step is an easy consequence of Lemma 25. If we find $b$ and the algorithm answers “No” when run on $f^b$ and $I^b$, then, by the induction hypothesis, $f^b$ is optimal for $I^b$, and by Lemma 25 $f$ is optimal for $I$. 

**Appendices**

**A. Non matching realizable even $\Delta$-matroid**

Here we prove Proposition 15 which says that not every even $\Delta$-matroid of arity six is matching realizable. We do it by first showing that matching realizable even $\Delta$-matroids satisfy certain decomposition property and then we exhibit an even $\Delta$-matroid of arity six which does not posses this property and thus is not matching realizable.

**Lemma 39.** Let $M$ be a matching realizable even $\Delta$-matroid and let $f, g \in M$. Then $f \Delta g$ can be partitioned into pairs of variables $P_1, \ldots P_k$ such that $f \oplus P_i \in M$ and $g \oplus P_i \in M$ for every $i = 1, \ldots k$.

**Proof.** Fix a graph $G = (N, E)$ that realizes $M$ and let $V = \{v_1, \ldots, v_n\} \subseteq N$ be the nodes corresponding to variables of $M$. Let $E_f$ and $E_g$ be the edge sets from matchings that correspond to tuples $f$ and $g$. Now consider the graph $G' = (N, E_f \Delta E_g)$ (symmetric difference of matchings). Since both $E_f$ and $E_g$ cover each node of $N \setminus V$, the degree of all such nodes in $G'$ will be zero or two. Similarly, the degrees of nodes in $(V \setminus (f \Delta g))$ are either zero or two leaving $f \Delta g$ as the set of nodes of odd degree, namely of degree one. Thus $G'$ is a union of induced cycles and paths, where the paths pair up the nodes in $f \Delta g$. Let us use this pairing as $P_1, \ldots, P_k$.

Each such path is a subset of $E$ and induces an alternating path with respect to both $E_f$ and $E_g$. After altering the matchings accordingly, we obtain new matchings that witness $f \oplus P_i \in M$ and $g \oplus P_i \in M$ for every $i$. 


Lemma 40. There is an even $\Delta$-matroid of arity 6 which does not have the property from Lemma 39.

Proof. Let us consider the set $M$ with the following tuples:

\[
\begin{array}{cccc}
000000 & 100100 & 011011 & 111111 \\
011000 & 100111 & & \\
001100 & 110011 & & \\
001010 & 110101 & & \\
000101 & 111010 & & \\
001001 & 001111 & & \\
010001 & 101101 & & \\
100010 & 101011 & & \\
111100 & & & \\
\end{array}
\]

With enough patience or with computer aid one can verify that this is indeed an even $\Delta$-matroid. Also for tuples $f = 000000$, and $g = 111111$ it is not so hard to see that no pairing $P_1, P_2, P_3$ exists. In fact the set of pairs $P$ for which both $f \oplus P \in M$ and $g \oplus P \in M$ is $\{v_1, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_6\}$ (see the first five lines in the middle of the table above) but no three of these form a partition on $\{v_1, \ldots, v_6\}$.

B Extending our algorithm to stabilizable $\Delta$-matroids

In this section we extend the algorithm from even $\Delta$-matroids to a wider class. The definitions of valid, optimal, and non-optimal edge labeling may remain intact but we need to adjust our definition of a walk. Now it will be allowed to end in a constraint.

Definition 41 (Walk for general $\Delta$-matroids). A walk $q$ of length $k$ or $k + 1/2$ in the instance $I$ is a sequence $q_0C_1q_1C_2 \ldots C_kq_k$ or $q_0C_1q_1C_2 \ldots C_{k+1}$, respectively, where the variables $q_{i-1}, q_i$ lie in the scope of the constraint $C_i$, and each edge $\{v, C\} \in E$ is traversed at most once: $vC$ and $Cv$ occur in $q$ at most once, and they do not occur simultaneously.

Given an edge labeling $f$ and a walk $q$, we define the edge labeling $f \oplus q$ in the same way as before (see eq. 1). We also extend in a natural way the definitions of an $f$-walk and an augmenting $f$-walk for a valid edge labeling $f$. Namely, a walk $q$ is an $f$-walk if $f \oplus q$ is a valid edge labeling, and so are $f \oplus q^*$ for any prefix $q^*$ of $q$ that ends at a variable. An $f$-walk is called augmenting if it starts at an inconsistent variable in $f$, ends either at an inconsistent variable or at a constraint, and all variables in $q$ different from the endpoints are consistent. Note that if $f$ is a valid edge labeling for which there is an augmenting $f$-walk, then $f$ is non-optimal (since $f \oplus q$ is a valid edge labeling with 1 or 2 fewer inconsistent variables).

The rough intuition of the algorithm is the following. When dealing with general $\Delta$-matroids, augmenting $f$-walks may also end in a constraint $C$. But in that case, the parity of all other constraints remains unchanged. If we knew the correct $C$ (in fact, we will try all options) and flipped its parity, we might be able to find this augmentation via the algorithm for even $\Delta$-matroids.

Definition 42. For a $\Delta$-matroid $M$, let Odd($M$) be the set of $M$-tuples with an odd number of ones (odd tuples) and Even($M$) the $M$-tuples with an even number of ones (even tuples). We say that $M$ is parity stable if each of Odd($M$) and Even($M$) is empty or an (even) $\Delta$-matroid.
Definition 43. For a $\Delta$-matroid $M$ and a tuple $\alpha \notin M$ we say that $\alpha$ is a hole in $M$ if whenever $\alpha \oplus u \oplus v \in M$ for some $u \neq v$, then $\alpha \oplus u \in M$ and $\alpha \oplus v \in M$. Let $\text{Holes}(M)$ be the set of holes in $M$.

Definition 44. We say that a class of $\Delta$-matroids $M$ is stabilizable if for every $M \in M$ we can in polynomial time find a set $H \subseteq \text{Holes}(M)$ such that $S = M \cup H$ is a parity stable $\Delta$-matroid. In that case we say that $S$ is a stabilizer of $M$ and write $S = \text{Stab}(M)$.

Unfortunately, there are $\Delta$-matroids $M$ such that $M \cup H$ is not parity stable for any $H \subseteq \text{Holes}(M)$. However, we will show below how to stabilize some previously considered classes of $\Delta$-matroids. These would be co-independent [12], compact [17], local [9], and binary [14, 9] $\Delta$-matroids. According to [9], these classes are pairwise incomparable.

The main result of this section is tractability of stabilizable $\Delta$-matroids.

Theorem 45. Given an edge CSP instance $I$ with stabilizable $\Delta$-matroid constraints, an optimal edge labeling $f$ of $I$ can be found in time polynomial in $|I|$.

Proposition 46. The classes of co-independent, compact, local and binary $\Delta$-matroids are stabilizable.

The proof of this proposition as well as (some of) the definitions can be found at the end of this Section.

B.1 The algorithm

The following lemma is a straightforward generalization of an analogous result given in Lemma 28.

Lemma 47. Let $f, g$ be valid edge labelings of instance $I$ (with general $\Delta$-matroid constraints) such that $g$ has fewer inconsistencies than $f$. Then there exists an augmenting $f$-walk (possibly ending in a constraint, in the sense of Definition 41) and it can be computed in polynomial time given $f$ and $g$.

Proof. Our process is a variant of the one from Lemma 28. First, we repeatedly modify the edge labeling $g$ using the following procedure:

1. Pick a variable $v \in V$ which is consistent in $f$, but not in $g$. (If no such $v$ exists then go to the next paragraph). By the choice of $v$, there exists a unique edge $\{v, C\} \in f \Delta g$. If $g(C) \oplus v \in C$, replace $g$ with $g \oplus vC$, then go to the beginning and repeat. Otherwise, pick variable $w \neq v$ in the scope of $C$ such that $\{w, C\} \in f \Delta g$ and $g(C) \oplus v \oplus w \in C$ (it exists since $C$ is a $\Delta$-matroid and $g(C) \oplus v \notin C$). Replace $g$ with $g \oplus (vCw)$ and then also go to the beginning and repeat.

It can be seen that $g$ remains a valid edge labeling, and the number of inconsistencies in $g$ never increases. Furthermore, each step decreases $|f \Delta g|$, so this procedure must terminate after at most $O(|E|) = O(|V|)$ steps.

We now have valid edge labelings $f, g$ such that $f$ has more inconsistencies than $g$, and variables consistent in $f$ are also consistent in $g$. In the second stage we will maintain an $f$-walk $p$ and the corresponding valid edge labeling $f^* = f \oplus p$. To initialize, pick a variable $r \in V$ which is consistent in $g$ but not in $f$, and set $p = r$ and $f^* = f$. We then repeatedly apply the following step:
Lemma 49. optimal edge labeling by Algorithm 1 in polynomial time.

Lemma 50. Let \( I \) be a valid edge labeling of instance \( I \) with stabilizable \( \Delta \)-matroid constraints. Let \( C \in \mathcal{C} \), denote by \( E(D) \) the \( \Delta \)-matroid among \( \text{Odd}(\text{Stab}(D)) \) and \( \text{Even}(\text{Stab}(D)) \) that contains \( f(D) \). For a constraint \( C \in \mathcal{C} \) and a \( \Delta \)-matroid \( C' \subseteq C \), we will denote by \( I(f,C,C') \) the instance obtained from \( I \) by replacing the constraint relation of \( C \) by \( C' \) and the constraint relation of each \( D \in \mathcal{C} \setminus \{C\} \) by the even \( \Delta \)-matroid \( E(D) \).

Observe that \( f \) induces a valid edge labeling for \( I(f,C,C) \). Moreover, if we choose \( \alpha \in C \), then \( I(f,C,\{\alpha\}) \) is an edge CSP instance with even \( \Delta \)-matroid constraints and hence we can find its optimal edge labeling by Algorithm 1 in polynomial time.

Lemma 49. Let \( f \) be a non-optimal valid edge labeling of instance \( I \) with stabilizable \( \Delta \)-matroid constraints. Then there exist \( C \in \mathcal{C} \) and \( \alpha \in C \) such that the optimal edge labeling for \( I(f,C,\{\alpha\}) \) has fewer inconsistencies than \( f \).

Proof. If \( f \) is non-optimal for \( I \), then by Lemma 47 there exists an augmenting \( f \)-walk \( q \). Let \( C \) be the last constraint in the walk and \( \alpha = (f \oplus q)(C) \). Now we only need to argue that \( f \oplus q \) is also a valid edge labeling for the instance \( I(f,C,\alpha) \). The agreement on the constraint \( C \) was already taken care of and for any other constraint \( D \) we have that \( f(D) \) has the same parity as \( (f \oplus q)(D) \) and thus certainly \( (f \oplus q)(D) \in E(D) \).

Lemma 50. Let \( f \) be a valid assignment for instance \( I \) with stabilizable \( \Delta \)-matroid constraints and let \( C \in \mathcal{C} \) and \( \alpha \in C \) be such that there exists a valid edge labeling \( g \) for the instance \( I(f,C,\{\alpha\}) \) with fewer inconsistencies than \( f \). Then there exists an augmenting \( f \)-walk for \( I \) and it can be computed in polynomial time given \( g \).

Proof. We begin by noticing that both \( f \) and \( g \) are valid edge labelings for the instance \( I(f,C,C) \) (still with \( g \) having fewer inconsistencies than \( f \)). Hence by Lemma 47 we can compute an \( f \)-walk \( q \) which is augmenting for \( I(f,C,C) \). We will show that either \( q \) is already an augmenting \( f \)-walk for instance \( I \) or \( q \) has a non-integral prefix which is.

First assume that every prefix of \( q \) with integral length is an \( f \)-walk in \( I \). Then if \( q \) weren’t an \( f \)-walk in \( I \), it would have to end in a constraint (have non-integral length) and flipping the last half-edge of \( q \) would be invalid in \( I \). But the only constraint with a relation allowing to possibly switch parity is \( C \) (all other are even \( \Delta \)-matroids) and valid transitions over \( C \) in \( I(f,C,C) \) are also valid in \( I \).
In the other case let us take the shortest prefix $p$ of $q$ with integral length which is not an $f$-walk in $I$. Write $p = p^*xDy$. Thus we have $\beta = (f \oplus p)(D) \not\in D$ but $\beta \oplus x \oplus y = (f \oplus p^*)(D) \in D$. Thus we have $D \neq C$ and thus we know that $\beta \in E(D)$. Since $E(D) \setminus D \subseteq \text{Holes}(D)$ we learn that $\beta$ is a hole of $D$. Since $\beta \oplus x \oplus y \in D$, we get that $\beta \oplus x \in D$ and thus the walk $p^*xD$ is an augmenting $f$-walk in $I$.

It is easy to see that all steps of the proof can be made algorithmic.

Now the algorithm is very simple to describe. Set some valid edge labeling $f$ and repeat the following procedure. For all pairs $(C, \alpha)$ with $\alpha \in C$ and $C \in \mathcal{C}$, call Algorithm 1 on the instance $I(f, C, \alpha)$ (computing $I(f, C, \alpha)$ can be done in polynomial time because all constraints of $I$ come from a stabilizable class). If for some $(C, \alpha)$ we obtained an edge labeling with fewer inconsistencies use Lemma 50 to get an augmenting $f$-walk, perform the augmentation and update $f$. Otherwise output (current) $f$ as the optimal edge labeling.

The algorithm is correct due to Lemma 49 and is polynomial simply because there are at most $|I|$ pairs $(C, \alpha)$ with $\alpha \in C$ and at most $|I|$ inconsistencies in the initial edge labeling so the (polynomial) Algorithm 1 is called at most $|I|^2$ times.

### C Classes of $\Delta$-matroids that are stabilizable

As we promised, here we will show that all classes of $\Delta$-matroids that were previously known to be tractable are stabilizable.

#### C.1 Co-independent $\Delta$-matroids

**Definition 51.** A $\Delta$-matroid $M$ is co-independent if whenever $\alpha \not\in M$, then $\alpha \oplus u \in M$ for every $u$ in the scope of $M$.

The definition implies that every missing tuple in $M$ is a hole. Thus $M \cup \text{Holes}(M)$ contains every possible tuple and is easily seen to be parity stable. Moreover, if $M$ is a $k$-ary co-independent $\Delta$-matroid, then a straightforward double counting argument gives us that $M \geq 2^k - 1$, so the set of holes is $O(|M|)$ large and listing it can be done in polynomial time.

#### C.2 Compact $\Delta$-matroids

We present the definition of compact $\Delta$-matroids in an alternative form compared to [17].

**Definition 52.** We say that $F: \{0, 1\}^V \to \{0, \ldots, |V|\}$ is a *generalized counting function* (gc-function) if

1. for each $\alpha \in \{0, 1\}^V$ and $v \in V$ we have $F(\alpha \oplus v) = F(\alpha) \pm 1$ and;

2. if $F(\alpha) > F(\beta)$ for some $\alpha, \beta \in \{0, 1\}^V$, then there exist $u, v \in \alpha \Delta \beta$ such that $F(\alpha \oplus u) = F(\alpha) - 1$ and $F(\beta \oplus v) = F(\beta) + 1$

An example of such function is the function which simply counts the number of ones in a tuple.

**Definition 53.** We say that a $S \subseteq \{0, 1, \ldots, n\}$ is *2-gap free* if whenever $x \not\in S$ and $\min S < x < \max S$, then $x + 1, x - 1 \in S$.

**Definition 54.** A set of tuples $M$ is *compact-like* if $\alpha \in M$ if and only if $F(\alpha) \in S$ for some gc-function $F$ and a 2-gap free subset $S$ of $\{0, 1, \ldots, |V|\}$.
The difference to the presentation in [17] is that they give an explicit set of possible gc-functions (without using the term gc-function). However, we decided for more brevity and omit the description of the set.

**Lemma 55.** Each compact-like set of tuples $M$ is a $\Delta$-matroid.

*Proof.* Let the gc-function $F$ and the 2-gap free set $S$ witness that $M$ is compact-like. Take $\alpha, \beta \in M$ and $u \in \alpha \Delta \beta$. If $F(\alpha \oplus u) \in S$, then $\alpha \oplus u \in M$ and we are done. Thus we have $F(\alpha \oplus u) \neq F(\beta)$.

Let us assume $F(\alpha \oplus u) > F(\beta)$. Since $F$ is a gc-function we can find $v \in (\alpha \oplus u) \Delta \beta$ (note that $u \neq v$) such that $F(\alpha \oplus u \oplus v) = F(\alpha \oplus u) - 1$. Now we have either $F(\alpha) = F(\alpha \oplus u \oplus v) \in S$, or $F(\alpha) > F(\alpha \oplus u) > F(\alpha \oplus u \oplus v) \geq F(\beta)$, which means $F(\alpha \oplus u \oplus v) \in S$ because $S$ does not have 2-gaps.

The case when $F(\alpha \oplus u) < F(\beta)$ is handled analogously. \hfill \qed

**Lemma 56.** Let $M$ be a compact-like $\Delta$-matroid for some gc-function $F$ and a 2-gap free set $S$. Let $\alpha \notin M$ be such that $\min S < F(\alpha) < \max S$. Then $\alpha$ is a hole in $M$.

*Proof.* Pick any $v \in V$. Then $F(\alpha \oplus v) = F(\alpha) \pm 1$ and since $S$ is 2-gap free, we have $F(\alpha \oplus v) \in S$ and hence $\alpha \oplus v \in M$. \hfill \qed

Clearly, after filling all such holes we obtain a compact-like $\Delta$-matroid for which the set $S$ is an interval. We now show that these compact-like $\Delta$-matroids are good for us:

**Lemma 57.** Let $M$ be a compact-like $\Delta$-matroid with a gc-function $F$ whose set $S$ is an interval. Then $M$ is parity stable.

*Proof.* Denote by $S_{EVEN}$ and $S_{ODD}$ then sets of even (odd) elements of $S$ and observe that both are 2-gap free. Also note that if $\alpha, \beta \in M$ have the same parity, then $F(\alpha)$ and $F(\beta)$ have the same parity. Thus the odd and even parts of $M$ are compact $\Delta$-matroids given by gc-function $F$ and 2-gap free sets $S_{EVEN}$ and $S_{ODD}$ in some order. \hfill \qed

We conclude that any practical class of compact-like $\Delta$-matroids is stabilizable:

**Corollary 58.** If $M$ is a class of compact-like $\Delta$-matroids where the description of each $M \in M$ includes a set $S_M$ and a function $F_M$ witnessing that $M$ is compact-like and there is a polynomial $p$ such that the time to compute $F_M(\alpha)$ is at most $p(|M|)$, then $M$ is stabilizable.

*Proof.* Given $M \in M$, we find the set $H$ by trying all $\alpha \in M, v \in V$. For each such choice, we calculate $F(\alpha \oplus v)$. If $\min S < F(\alpha \oplus v) < \max S$ and $F(\alpha \oplus v) \notin S_M$, we add $\alpha \oplus v$ to $H$. Since $F$ is a gc-function, at the end of this procedure we get $M \cup H$ equal to the compact-like $\Delta$ matroid with the interval set $[\max S, \min S]$ and are done. \hfill \qed

### C.3 Local and binary $\Delta$-matroids

We will avoid giving the definitions of local and binary $\Delta$-matroids. Instead, we will rely on a result from [9] saying that both of these classes avoid a certain substructure. This will be enough to show that both binary and local matroids are all parity stable (and thus clearly stabilizable).

**Definition 59.** Let $M, N$ be two $\Delta$-matroids where $M \subseteq \{0, 1\}^V$. We say that $M$ contains $N$ as a minor if we can get $N$ from $M$ by a sequence of the following operations:
(a) Choose \( v \in V \) and take the \( \Delta \)-matroid we obtain by fixing the value at \( v \) to 0 and deleting \( v \):

\[
M_{v=0} = \{ \beta \in \{0,1\}^V \setminus \{ v \} : \exists \alpha \in M, \alpha(v) = 0 \land \forall u \neq v, \alpha(u) = \beta(u) \}.
\]

(b) Choose \( v \in V \) and take the \( \Delta \)-matroid we obtain by fixing the value at \( v \) to 1 and deleting \( v \):

\[
M_{v=1} = \{ \beta \in \{0,1\}^V \setminus \{ v \} : \exists \alpha \in M, \alpha(v) = 1 \land \forall u \neq v, \alpha(u) = \beta(u) \}.
\]

**Definition 60.** The interference \( \Delta \) matroid is the ternary \( \Delta \)-matroid given by the tuples

\[
\{(0,0,0), (1,1,0), (1,0,1), (0,1,1), (1,1,1)\}.
\]

We say that a \( \Delta \)-matroid \( M \) is **interference free** if it does not contain any minor isomorphic (via renaming variables or flipping the values 0 and 1 of some variables) to the interference \( \Delta \)-matroid.

**Lemma 61.** If \( M \) is an interference-free \( \Delta \)-matroid and \( \alpha, \beta \in M \) are such that \( |\alpha \Delta \beta| \) is odd, then we can find \( v \in \alpha \Delta \beta \) so that \( \alpha \oplus v \in M \).

**Proof.** Let us take \( \beta' \in M \) so that \( \alpha \Delta \beta' \subseteq \alpha \Delta \beta \) and \( |\alpha \Delta \beta'| \) is odd and minimal possible. If \( |\alpha \Delta \beta'| \) is odd then we are done. Assume thus that \( |\alpha \Delta \beta'| = 2k + 3 \) for some \( k \in \mathbb{N}_0 \). Applying the \( \Delta \)-matroid property on \( \alpha \) and \( \beta' \) (with \( \alpha \) being the tuple changed) \( k \) times, we get a set of \( 2k \) variables \( U \subseteq \alpha \Delta \beta' \) such that \( \alpha \oplus U \in M \) (since \( \beta' \) is at minimal odd distance from \( \alpha \), in each step we need to switch exactly two variables of \( \alpha \)). Let the three variables in \( \alpha \Delta \beta' \setminus U \) be \( x, y, z \) and consider the matroid \( P \) on \( x, y, z \) we get from \( M \) by fixing the values of all \( v \in \{ x, y, z \} \) to those of \( \alpha \oplus U \) and deleting these variables afterward. Moreover, we switch 0s and 1s so that the triple corresponding to \( (\alpha(x), \alpha(y), \alpha(z)) \) is \( (0,0,0) \).

We claim that \( P \) is the interference matroid: It contains the triple \( (0,0,0) \) (because of \( \alpha \oplus U \)) and \( (1,1,1) \) (as witnessed by \( \beta' \)) and does not contain any of the triples \( (1,0,0), (0,1,0), (0,0,1) \) (for then \( \beta' \) would not be at minimal odd distance from \( \alpha \)). Applying the \( \Delta \)-matroid property on \( (1,1,1) \) and \( (0,0,0) \) in each of the three variables then necessarily gives us the tuples \( (0,1,1), (1,0,1), \) and \( (1,1,0) \in P \).

**Corollary 62.** Let \( M \) be an interference-free \( \Delta \)-matroid. If \( M \) contains at least one even tuple then the set \( \text{Even}(M) \) of all even tuples of \( M \) forms a \( \Delta \)-matroid. The same holds for \( \text{Odd}(M) \), so any interference-free \( \Delta \)-matroid is parity stable.

**Proof.** Take \( \alpha, \beta \in \text{Even}(M) \) and pick a variable \( v \) so that \( \alpha(v) \neq \beta(v) \). We want \( u \neq v \) so that \( \alpha(u) \neq \beta(u) \) and \( \alpha \oplus u \oplus v \in M \). Apply the delta-matroid property of \( M \) to \( \alpha \) and \( \beta \), changing the tuple \( \alpha \). If we get \( \alpha \oplus u \oplus v \in M \) for some \( u \), we are done, so let us assume that we get \( \alpha \oplus v \in M \) instead. But then we recover as follows: The tuples \( \alpha \oplus v \) and \( \beta \) have different parity, so by Lemma 61 there exists a variable \( u \) so that \( (\alpha \oplus v)(u) \neq \beta(u) \) (ie. \( u \in \alpha \Delta \beta \setminus \{ v \} \)) and \( \alpha \oplus v \oplus u \in M \).

It is mentioned in [9] (Section 4) that the interference \( \Delta \)-matroid is among the forbidden minors for both local and binary (minors B1 and L2) \( \Delta \)-matroids. Thus both of those classes are parity stable.
References

[1] L. Barto, M. Kozik, and T. Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (A positive answer to a conjecture of Bang-Jensen and Hell). SIAM Journal on Computing, 38(5):1782–1802, 2009.

[2] A. Bulatov. A dichotomy theorem for constraints on a 3-element set. In FOCS’02, pages 649–658, 2002.

[3] A. Bulatov. Tractable conservative constraint satisfaction problems. In LICS’03, pages 321–330, 2003.

[4] A. Bulatov, P. Jeavons, and A. Krokhin. Classifying complexity of constraints using finite algebras. SIAM Journal on Computing, 34(3):720–742, 2005.

[5] A. Bulatov, A. Krokhin, and P. Jeavons. Constraint satisfaction problems and finite algebras. In ICALP’00, volume 1853 of LNCS, pages 272–282, 2000.

[6] S. O. Chan, J. R. Lee, P. Raghavendra, and D. Steurer. Approximate constraint satisfaction requires large LP relaxations. In FOCS, pages 350–359. IEEE Computer Society, 2013.

[7] D. A. Cohen, M. C. Cooper, P. Jeavons, and S. Živný. Tractable classes of binary CSPs defined by excluded topological minors. In Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI’15), pages 1945–1951. AAAI Press, 2015.

[8] M. C. Cooper and S. Živný. Hybrid tractability of valued constraint problems. Artificial Intelligence, 175(9-10):1555–1569, 2011.

[9] V. Dalmau and D. K. Ford. Mathematical Foundations of Computer Science 2003: 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003. Proceedings, chapter Generalized Satisfiability with Limited Occurrences per Variable: A Study through Delta-Matroid Parity, pages 358–367. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[10] Z. Dvořák and M. Kupec. On planar Boolean CSP. In ICALP ’15, volume 9134 of Lecture Notes in Computer Science, pages 432–443. Springer Berlin Heidelberg, 2015.

[11] J. Edmonds. Path, trees, and flowers. Canadian J. Math., 17:449–467, 1965.

[12] T. Feder. Fanout limitations on constraint systems. Theoretical Computer Science, 255(1–2):281–293, 2001.

[13] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. SIAM Journal on Computing, 28(1):57–104, 1999.

[14] J. F. Geelen, S. Iwata, and K. Murota. The linear delta-matroid parity problem. Journal of Combinatorial Theory, Series B, 88(2):377 – 398, 2003.

[15] P. Hell and J. Nešetřil. On the complexity of $H$-coloring. Journal of Combinatorial Theory, Ser. B, 48:92–110, 1990.

[16] John Hopcroft and Robert Tarjan. Efficient planarity testing. J. ACM, 21(4):549–568, October 1974.
[17] G. Istrate. Looking for a version of Schaefer’s dichotomy theorem when each variable occurs at most twice. Technical report, University of Rochester, Rochester, NY, USA, 1997.

[18] P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[19] V. Kolmogorov, A. Krokhin, and M. Rolínek. The complexity of general-valued CSPs. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS*, pages 1246–1258, 2015.

[20] V. Kolmogorov, M. Rolínek, and R. Takhanov. Effectiveness of structural restrictions for hybrid CSPs. In *ISAAC’15*, 2015. arXiv1504.07067.

[21] V. Kolmogorov and S. Živný. The complexity of conservative valued CSPs. *Journal of the ACM*, 60(2):Article 10, 2013.

[22] M. Kozik and J. Ochremiak. Algebraic properties of valued constraint satisfaction problem. ICALP’15, 2015.

[23] B. M. E. Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, June 1988.

[24] P. Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *STOC’08*, pages 245–254, 2008.

[25] P. Raghavendra and N. Tan. Approximating CSPs with global cardinality constraints using SDP hierarchies. In *SODA*, pages 373–387. SIAM, 2012.

[26] T.J. Schaefer. The complexity of satisfiability problems. In *STOC’78*, pages 216–226, 1978.

[27] R. Takhanov. A dichotomy theorem for the general minimum cost homomorphism problem. In *STACS’10*, pages 657–668, 2010.

[28] J. Thapper and S. Živný. The complexity of finite-valued CSPs. *Journal of the ACM*, 2015. to appear.

[29] E. P. K. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.