Even Delta-Matroids and the Complexity of Planar Boolean CSPs

Alexandr Kazda, Vladimir Kolmogorov, Michal Rolínek

IST Austria

Abstract

The main result of our paper is a generalization of the classical blossom algorithm for finding perfect matchings that can efficiently solve Boolean CSPs where each variable appears in exactly two constraints and all constraints are even $\Delta$-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.

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 $\Gamma$. The question whether for any fixed $\Gamma$, the set of generated instances CSP($\Gamma$) forms a decision problem which is either polynomial-time solvable or NP-complete (in other words it avoids intermediate complexities assuming $P \neq NP$) 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, 17]. 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-vertex-cover, submodular minimization). The study of valued CSPs (VCSP) extended the algebraic machinery for proving NP-Hardness [21, 20, 27] and provided vast complexity classifications [18, 27, 20, 26]. Approximability of valued CSPs has also been studied extensively [23, 24, 6].
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 \[19\] 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 vertices 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 has the same complexity as its unrestricted version. Ever since there has been also progress on the algorithmic side. Several tractable classes of \(\Delta\)-matroids were identified \[12, 16, 9, 14, 10\]. 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 and for the remaining cases a reduction to Boolean edge CSP. Their results imply that completing the complexity classification of Boolean planar CSPs is equivalent to establishing the complexity of 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. Aside from providing a very broad tractable class of edge CSP, this also settles the complexity classification of planar CSP.

The paper is organized as follows. In the introductory Sections 2, 3, and 4 we formalize the frameworks, discuss how dichotomy from 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.

### 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 \(C \in C\) is a pair \((\sigma, R_C)\) where \(\sigma \subseteq V\) is a set of size \(k\) (called the scope of \(C\)) and \(R_C \subseteq \{0, 1\}^\sigma\) is a relation on \(\{0, 1\}\). A
solution to $I$ is a mapping $\hat{f}: V \to \{0, 1\}$ such that for every constraint $C = (\sigma, R_C) \in \mathcal{C}$, $\hat{f}$ restricted to $\sigma$ 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 $\text{CSP}(\Gamma)$.

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.

For Boolean CSPs (where variables are assigned Boolean values), the complexity classification has been known for a long time:

**Theorem 2** (Schaefer [25]). Let $\Gamma$ be a constraint language. Then $\text{CSP}(\Gamma)$ is tractable if

(a) all relations in $\Gamma$ admit a tuple of constant ones (resp. constant zeros) or;

(b) all relations in $\Gamma$ are equivalent to 2-SAT or;

(c) all relations in $\Gamma$ are equivalent to a conjunction of Horn clauses (resp. dual Horn clauses) or;

(d) all relations in $\Gamma$ are equivalent to linear equations over $\mathbb{Z}_2$.

Otherwise $\text{CSP}(\Gamma)$ is NP-Hard.

There exists a similar classification also 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 one is to restrict, for each variable, the number of constraints interacting with it. It turns out [12] that already restricting to at most three occurrences is as hard as the unrestricted case. 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 3** (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 vertices to constraints of $I$. Constraints (vertices) 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 vertices 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 \[28\]. In order to resolve this confusion (and for the reasons described in the previous paragraph), we propose the term “edge CSP”.

**Definition 4.** 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 set $f \oplus S = f \oplus s_1 \oplus \cdots \oplus s_k$. 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 5.** 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 6** ([12]). If $\Gamma$ is a constraint language 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 linear \[14\], co-independent \[12\], compact \[16\], 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 7.** 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.

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 vertices 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 (using SPQR-trees [22]) and hence it does not matter if we are given a planar drawing of $G$ as a part of the input or not.

Definition 9. For a tuple of Boolean variables $T = (x_1, \ldots, x_n)$, let $\bar{T} = T \oplus (1, \ldots, 1)$ and $dT = (x_1 \oplus x_2, \ldots, x_n \oplus x_1)$. A relation $R$ is called self-complementary if for all $T \in \{0, 1\}^n$ we have $T \in R$ if and only if $\bar{T} \in R$. Also let $dR = \{dT: T \in R\}$ and $d\Gamma = \{dR: R \in \Gamma\}$ when $\Gamma$ is a set of relations.

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 10. 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

\[ \text{CSP}_{\text{edge}}(d\Gamma \cup \{\text{EVEN}_1, \text{EVEN}_2, \text{EVEN}_3\}) \]

where \[ \text{EVEN}_i = \{(x_1, \ldots, x_i) : x_1 \oplus \cdots \oplus x_i = 0\}. \]

Using Theorem 7, we can finish this classification:

Theorem 11 (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 10 the only unresolved case reduces to solving

\[ \text{CSP}_{\text{edge}}(d\Gamma \cup \{\text{EVEN}_1, \text{EVEN}_2, \text{EVEN}_3\}) \]

Since the relations $\text{EVEN}_i$ are even $\Delta$-matroids for every $i$, this is polynomial-time solvable thanks to Theorem 7. \qed
Figure 1: On the left we see an instance $I$ that is equivalent to testing for perfect matching
of the given graph. On the right is an equivalent instance $I'$ with contracted “supernodes”
$X$ and $Y$.

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 12. For $n \in \mathbb{N}$ consider the “perfect matching” relation $M_n \subseteq \{0, 1\}^n$ con-
taining precisely the tuples in which exactly one coordinate is set to one and all oth-
ers to zero. Note that $M_n$ is an even $\Delta$-matroid for all $n$. Then the instance $I$ of
CSP$_{edge}(\{M_n: n \in \mathbb{N}\})$ (represented in Figure 1) is equivalent to deciding whether the
graph of the instance has a perfect matching (every vertex 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):

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

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

One message of this example is that any algorithm that solves edge CSP for the even
$\Delta$-matroid case has to be a generalization of the perfect matching algorithm. Another is
the construction of even $\Delta$-matroids $X$ and $Y$ which can be generalized as follows.

Definition 13 (Matching realizable relations). Let $G$ be a graph and let $v_1, \ldots, v_a \in V(G)$
be distinct vertices 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 vertices $v_i$ 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 vertices $v_1,\ldots,v_a \in V(G)$.

Every matching realizable relation is an even $\Delta$-matroid [10]. Also, it should be clear from the definition and the preceding example that CSP$_{edge}(\Gamma)$ is tractable if $\Gamma$ contains only matching realizable relations (assuming we know the graph $G$ and the vertices $v_1,\ldots,v_a$ 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 14.** There exists an even $\Delta$-matroid of arity 6 which is not matching realizable.

The notion of matching realizable even $\Delta$-matroids should illustrate the difficulties one needs to overcome when formulating a strengthening of Edmonds’ algorithm. The vital point is to correctly identify and also correctly contract some “blossoms-like” structures. Clearly, the sought definitions have to depend on both the graph structure and the content of the constraint nodes. Proposition 14 implies that we cannot hope to simply replace the constraint nodes by graphs.

## 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 $v \in V$ is $G_I$ 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 15.** An edge labeling of $I$ is a mapping $f : E \to \{0,1\}$. 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 any two 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 $I$, where we assume that that constraint relations are given by lists of tuples.

The main theorem we prove is the following strengthening of Theorem 7.

**Theorem 16.** 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 17.** 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 $\{v, C\} \in \mathcal{E}$ 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 $q^{-1}$ is the sequence $q_kC_k \ldots q_1C_1q_0$. Given two walks $p$ and $q$ such that the last vertex of $p$ is the first vertex of $q$, we define their concatenation $pq$ in the natural way. If $p = \alpha_1 \ldots \alpha_k$ and $q = \beta_1 \ldots \beta_\ell$ are sequences of nodes of a graph where $\alpha_k$ and $\beta_1$ are different but adjacent, we will denote the sequence $\alpha_1 \ldots \alpha_k\beta_1 \ldots \beta_\ell$ 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 18.** 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) $f \oplus q[i,j]$ 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**.
Observe that if \( p \) is an augmenting \( f \)-walk, then \( f \oplus p \) has 2 fewer inconsistent variables than \( f \).

Another structure used by the Edmonds’ algorithm for matchings is a blossom. We will adapt it to our case as follows.

**Definition 19.** Let \( f \) be a valid edge labeling of instance \( I \). A walk \( b = b_0C_1b_1C_2\ldots C_kb_k \) with \( b_0 = b_k \) will be called an \( f \)-blossom if

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 \( [i,j] \subseteq [0,k] \).

**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 \( C \) 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 forest \( T \) constructed by the algorithm will be the inconsistent vertices of the instance (for current \( f \)), and 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 \( \text{walk}(\alpha) \) be the the unique path in \( T \) from a root to \( \alpha \). Note that \( \text{walk}(\alpha) \) is a subgraph of \( T \). Sometimes we will treat walks in \( T \) as sequences of nodes in \( V \cup C \) discussed in Sec. 2 (i.e. with timestamps removed); such places should be clear from the context.

Forest \( T \) will be grown in a greedy manner as shown in Algorithm 1.

**Algorithm 1.**

**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 \( C^tw \).
Figure 2: An example run of the algorithm on the instance $T'$ from Example 12 (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 is entirely possible), we would have found an augmenting path $p = \text{walk}(C^5) \text{walk}(x)^{-1}$.

(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.

We note that the structure of the algorithm resembles that of the Edmonds’ algorithm for matchings [11], with the following distinctions:

- 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 1 some constraints $C \in \mathcal{C}$ can be added to $T$ multiple times. 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 1 will add it to $T$ at most once. (We will not use this fact, and thus omit the proof.)

Also 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.
The Edmonds’ algorithm does not impose any restrictions on the order in which the forest is grown. In contrast, 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.

### 5.3 Contracting a blossom (Step 4d)

We now elaborate step 4d of Algorithm 1. First, we describe how to obtain blossom \( b \) from Definition 19. 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 define walk \( b \) as follows: \( b = \text{walk}(C_t), \text{walk}(w)^{-1} \).

2. \( \alpha = R^s \in C(T) \). Let \( r \) be the child of \( R^s \) in \( T \) that leads to \( v \). 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).

Later we will prove the following.

**Lemma 21.** Assume that Algorithm 2 reaches step 4d and one of the cases described in the above paragraph occurs. Then:

(a) in the case 2 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 \( f^b \), in the second case).
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 $\mathcal{E}$ that are not in the digraph $T$, but will nonetheless be important later.

Figure 4: A blossom (left) and a contracted blossom (right).
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 “gadget” constraints $Z_i$ and $Z_j$ as shown in the figure. Both $Z_i$ and $Z_j$ are even $\Delta$-matroids with three tuples that depend on the values $\lambda_k = f(\{b_k, D\})$.

To summarize, at this point we have a valid edge labeling $f$ of instance $I$ and an $f$-blossom $b = b_0C_1b_1 \ldots C_kb_k$. Next, 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 $k$-ary constraint $N$ to $I$, delete the variables $b_1, \ldots, b_k$ and add new variables $v_1, v_2, \ldots, v_k$. The scope of $N$ is $\{v_1, v_2, \ldots, v_k\}$ and the matroid of $N$ consists of exactly those maps $m: \{v_1, \ldots, v_k\} \rightarrow \{0,1\}$ that send one $v_i$ to 1 and the rest to 0.

In addition to all this, we replace each constraint $D = (\sigma, M) \in \{C_1, \ldots, C_{k-1}, C_k\}$ by the constraint $D^b$ whose scope is $\sigma \setminus \{b_1, \ldots, b_k\} \cup \{v_i: 1 \leq i \leq k, \{b_{i-1}, b_i\} \subseteq \sigma\}$ (where $b_0 = b_k$) and $n$ lies in the constraint relation $M^b$ if there exists $m \in M$ that agrees with $n$ on $\sigma \setminus \{b_1, \ldots, b_k\}$ and such that whenever $v_i$ is in the scope of $D^b$ and $n(v_i) = 0$, then the mappings $m$ and $f$ agree on $\{b_{i-1}, b_i\}$, while if $n(v_i) = 1$, the mappings $m$ and $f$ differ on exactly one of $b_{i-1}, b_i$ (see also Figure 5 for more intuition about the construction).

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

**Lemma 22.** Each $C_i^b$ 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 $C_i^b$ let $f^b(C_i^b)(v) = f(C_i)(v)$ when $v$ is not one of $b_1, \ldots, b_k$, and $f^b(C_i^b)(v_i) = 0$. Finally, we let $f^b(N) = (1,0,\ldots,0)$. (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, $v_1$ is inconsistent in $f^b$ while the variables $v_2, v_3, \ldots, v_k$ are all consistent.

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

**Corollary 24.** Given an instance $I$, Algorithm 1 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 keeps the number of variables constant and adds one more constraint, it follows that there can not be a sequence of contractions longer than $2|V|$, which is $O(|V|)$.

The following two lemmas, which we prove in Section 6, 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 1

To see that Algorithm 1 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 $I$: the number of variables of $I^b$ and $I$ is the same, while $I^b$ has one more constraint $N$ than $I$ and this $N$ contains $O(|V|)$ many tuples. Therefore, we have $|I^b| \leq |I| + O(|V|)$ where $|V|$ does not change. By claim 4, 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 1 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 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(a) and Lemmas 21, 22, 23, 26 (which imply Theorem 20(b, d)). Finally, in Sec. 6.4 we prove Theorem 20(e).
Recall that if \( q = q_0 C_1 q_1 C_2 \ldots C_k q_k \) is a walk then by Definition 17 each edge \( \{ v, C \} \in E \) is traversed by \( q \) at most once. We will denote by \( E(q) \) those edges in \( E \) that are traversed by \( q \):

\[
E(q) = \{ \{ v, C \} \mid q \text{ contains either } vC \text{ or } Cv \}.
\]

For edge labelings \( f, g \), let \( f \triangle g \subseteq E \) be the set of edges in \( E \) on which \( f \) and \( g \) differ.

**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 \( |f \triangle g| \). The base case \( |f \triangle g| = 0 \) is trivial. For the induction step let us consider valid edge labelings \( f, g \) with \( |f \triangle g| \geq 1 \). Pick an edge \( \{ v, C \} \in f \triangle g \). By the property of even \( \Delta \)-matroids there exists another edge \( \{ w, C \} \in f \triangle g \) 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 \( |f^* \triangle g| = |f \triangle g| - 2 \)). 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 (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 \( f, g, x \).

**Proof.** Our algorithm will involve two stages. First, we repeatedly modify edge labeling \( g \) using the following procedure:

- 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 \( \{ v, C \} \in f \Delta g \). 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 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 \( |f \triangle g| \) by 2, 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 \( g \). 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 \( r \in V - \{ x \} \) which is consistent in \( g \) but not in \( f \), and set \( p = r \) and \( f^* = f \). We then repeatedly apply the following step:
Let $v$ be the endpoint of $p$. It is consistent in $g$ but not in $f^*$, so there must exist a unique edge $\{v,C\} \in f^* \Delta g$. Pick variable $w \neq v$ in the scope of $C$ such that $\{w,C\} \in f^* \Delta g$ and $f^*(C) \oplus w \oplus w \subseteq C$ (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 $f^* \Delta g$.

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 $|f^* \Delta g|$ by 2, so this procedure must terminate after at most $O(|E|) = O(|V|)$ 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 variable $r$ again during the procedure. □

The following lemma is at the heart of dealing with “interferences” caused by multiple visits. It shows that each interference is “compensated by a shortcut”.

**Lemma 29.** Let $p,q$ be two $f$-walks with $E(p) \cap E(q) = \emptyset$ such that $q$ is not an $(f \oplus p)$-walk. Then there exists an edge $\{w,C\} \in E(q)$ and index $i \in [0, \text{length}(p) - 1]$ with $p_{[i,i+1]} = p_iCp_{i+1}$ such that $p_{[0,i]} \cup Cw$ is an $f$-walk.

**Proof.** We will call a pair $(p,q)$ with $E(p) \cap E(q) = \emptyset$ admissible if it satisfies the precondition of the lemma, and bad if it does not satisfy the postcondition. Suppose that an admissible bad pair exists. Let $(p,q)$ be such a pair so that the sum of lengths of $p$ and $q$ is as small as possible. Write $p = p^*(xCy)$ and $q = q^*(uDv)$ where $p^*$ is a prefix of $p$ ending at $x$, and $q^*$ is a prefix of $q$ ending at $u$. Denote by $f^*$ the assignment $f \oplus p^* \oplus q^*$ (see Figure 6).

![Figure 6: The situation in Lemma 29 (after realizing that $C = D$).](image)

It is easy to check that pairs $(p^*, q)$ and $(p, q^*)$ are also bad, and so by the choice of $(p, q)$ they are not admissible. This implies that $q$ is an $(f \oplus p^*)$-walk and $q^*$ is an $(f \oplus p)$-walk. Therefore, edge labelings $f^*$, $f^* \oplus (uDv)$ and $f^* \oplus (xCy)$ are valid. However, $f^* \oplus (xCy) \oplus (uDv)$ is not valid (since $q^*$ is an $(f \oplus p)$-walk but $q$ is not an $(f \oplus p)$-walk).

We can have such situation only if $C = D$. Let us apply the even $\Delta$-matroid property for tuples $f^*(C) \oplus x \oplus y$ and $f^*(C) \oplus u \oplus v$ (which are both in $C$) and coordinate $x$. We obtain that either $f^*(C) \oplus x \oplus u \subseteq C$ or $f^*(C) \oplus x \oplus v \subseteq C$.

We will denote by $g$ the assignment $f \oplus p^*$. Obviously, $g(C) \subseteq C$. We proved that there exists a set $W$ such that (i) for each variable $w \in W$ we have $\{w,C\} \in E(q)$ and
Apply the even $\Delta$-matroid property to tuples $g(C)$ and $g(C) \oplus x \oplus W$ in coordinate $x$. We get that there exists $w \in W$ such that $g(C) \oplus x \oplus w \in C$. Therefore, $p^*(xCw)$ is an $f$-walk. This contradicts the assumption that $(p, q)$ is a bad pair.

6.2 Invariants of Algorithm $\mathbb{A}$: $f$-DAGs

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

**Definition 30.** Let $I$ be an 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 $V(T) \subseteq V$ and $C(T) \subseteq C \times \mathbb{N}$, and the following conditions hold:

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

(b) For each $\{v, C\} \in \mathcal{E}$ 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 $v$ can have at most two incident edges in $T$.)

(d) Timestamps $t$ for nodes $C^t \in C(T)$ are all distinct (and thus they give a total order on $C(T)$). Moreover, this order can be extended to a total order $\prec$ on $V(T) \cup C(T)$ such that $\alpha \prec \beta$ 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 $C^tv$, then $f(C) \oplus u \oplus v \notin C$.

(f) (“No shortcuts” property) If $T$ contains edges $uC^s$ and one of $vC^t$ or $C^tv$ 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.

The following observation will be implicitly used throughout the proof: if $C^s, C^t$ 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 $\{vC^t, C^tv\}$, then $u \neq v$.

The following lemma shows the promised invariant property:

**Lemma 31.** Let us consider the structure $T$ during the run of Algorithm $\mathbb{A}$ with the input $I$ and $f$. At any moment during the run, the forest $T$ is an $f$-DAG.

Moreover, if steps [4c] 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 $\mathbb{A}$ we need to make sure that adding $C^t$ and $vC^t$ in step [3] and adding of $C^t w$
in step 4a does not violate the properties of \( T \). Let us consider step 3 first. By the choice of \( v \) and \( C^t \), we immediately get that properties [a, b, c] and [d] 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 [e] holds as well. Finally, the only way the no shortcuts property (i.e. property [f]) could fail would be if there was 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 \( C^s \) was 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 \( C^sw \) before, a contradiction. The analysis of step 4a is similar.

Assume now that Algorithm 1 has reached one of steps 4c or 4d and consider the DAG \( T^* \) that we get from \( T \) by removing all edges of the form \( C^t \) 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 [b, c] and [e]. Were [b] violated, we would have \( C^sw \in E(T) \) already, and so step 4b would be triggered instead of steps 4c or 4d. For property [c], 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 [e] became violated after adding edge \( wC^t \) then there was a \( u \) and \( s < t \) such that \( uC^s \in E(T) \) and \( f(C) \oplus u \oplus w \in C \). Node \( C^s \) must have been added after \( w \), or else we would have \( C^sw \in E(T) \). 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 4d or 4c 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 32.** 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 two incident edges, namely incoming edge \( uC^s \) where \( u \) does not have other incident edges besides \( uC^s \), and another edge \( C^sv \) (see Figure 7). Let \( f^* = f \oplus (uCv) \) and let \( T^* \) be the DAG obtained from \( T \) by removing nodes \( u, C^s \) and the two edges incident to \( C^s \). Then \( f^* \) is a valid edge labeling of \( I \) and \( T^* \) is an \( f^* \)-DAG.

![Figure 7: An f-DAG T on the left turns into f*-DAG T* on the right; the setting from Lemma 32](image-url)
Proof. Since $T^*$ is a subgraph of $T$, it immediately follows that $T^*$ satisfies the properties $[a, b, c]$ and $[d, e, f]$ from the definition of an $f$-DAG all hold.

Let us show that $T^*$ has property $[g]$. Consider a constraint node $C^t \in \mathcal{C}(T^*)$ with $t > s$ (nothing has changed for other constraint nodes in $\mathcal{C}(T^*)$), and suppose that $T^*$ contains edges $xC^t$ and one of $yC^t$ or $C^ty$. If $x = y$, the situation is trivial, so assume that $u, v, x, y$ are all distinct variables. We need to show that $f^*(C) \oplus x \oplus y \in C$. Suppose that this is not the case. Let $p = uCv$ and $q = xCy$. Since $f^*(C) \oplus x \oplus y \notin C$, $q$ is not an $(f \oplus p)$-walk. Condition $[c]$ for graph $T$ implies that $p$ and $q$ are $f$-walks. By Lemma 29 there exists $z \in \{x, y\}$ such that $uCz$ is an $f$-walk. However, this contradicts the “no shortcuts” property for graph $T$.

Now let us prove that $T^*$ and $f^*$ have the “no shortcuts” property. Consider constraint nodes $C^k, C^\ell$ in $\mathcal{C}(T^*)$ with $s < k < \ell$ (since nothing has changed for other pairs of constraint nodes), and suppose that $T^*$ contains edges $xC^k$ and one of $yC^\ell$ or $C^\ell y$, 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 $f(C) \oplus u \oplus x \in C$, or $f(C) \oplus u \oplus y \in C$. This contradicts the “no shortcuts” property for the pair $(C^k, C^\ell)$, resp. $(C^s, C^k)$, resp. $(C^s, C^\ell)$, so we are done.

\[ \square \]

Corollary 33. Let $I$ be an edge CSP instance and $f$ be a valid assignment.

(a) Let $T$ be an $f$-DAG that consists of two directed paths $x_1C^t_1x_2\ldots x_kC^t_k$ and $y_1D^t_1\ldots y_\ell D^t_\ell$ that are disjoint everywhere except at the constraint $C^t_k = D^t_\ell$ (see Figure 8). Then $f \oplus T$ is a valid edge labeling of $I$, where $f \oplus T$ is defined in the natural way (i.e. it equals $f \oplus p$ where $p$ is the walk in $I$ that corresponds to us walking from $x_1$ to $y_1$ using the edges of $T$).

(b) Let $T$ be an $f$-DAG that consists of a single directed path $x_1C^t_1x_2\ldots x_kC^t_kx_{k+1}$. 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 $x_1C^t$ and $y_1C^t$ (where $C^t$ is an abbreviated name for $C^t_1 = D^t_1$). Then the fact that $f \oplus (x_1C^t_1)$ is a valid edge labeling follows from the property $[d]$ of $f$-DAGs.
Figure 9: A more general $f$-DAG $T$ turns into an $f^*$-DAG $T^*$ (see Lemma 34).

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 32 for $x_1 C_1^t x_2$ (there is a $x_2$ since $t_k > s_1 > t_1$), obtaining the $f \oplus (x_1 C_1 x_2)$-DAG $T^*$ that consists of two directed paths $x_2 \ldots x_k C^t u$ and $y_1 D_1^t \ldots y_\ell D_\ell^t$. Since $T^*$ is shorter than $T$, the induction hypothesis gets us that $f \oplus (x_1 C_1 x_2) \oplus T^* = f \oplus T$ is a valid edge labeling.

If $t_1 > s_1$, we do the same thing with $y_1 D_1 y_2$ instead of $x_1 C_1 x_2$.

**Lemma 34.** Let $T$ be an $f$-DAG, and $C^*$ be the constraint node in $C(T)$ with the smallest timestamp $s$. Suppose that $C^*$ has exactly one incoming edge $u C^*$, and $u$ does not have other incident edges besides $u C^*$. Suppose also that $C^*$ has an outgoing edge $C^* v$. Let $f^* = f \oplus (u C v)$, and $T^*$ be the DAG obtained from $T$ by removing the edge $u C v$ together with $u$ and reversing the orientation of edge $C^* v$ (see Figure 7).

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 $A$, 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 $C$ of Definition 30 is preserved. First, consider constraint node $C^*$. Suppose that $T^*$ contains one of $x C^* x$ or $C^* x$ with $x \neq v$. We need to show that $f^*(C) \oplus v \oplus x \in C$, 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 $C$ of Definition 30 for $T$. Now consider a constraint node $C^t \in C(T^*)$ with $t > s$, and suppose that $T^*$ contains edges $x C^t$ and one of $y C^t$ or $C^t y$. Note that $u, v, x, y$ are all distinct variables.

We need to show that $f^*(C) \oplus x \oplus y \in C$, 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 32.

Now let us prove that the “no shortcuts” property is preserved. First, consider a constraint node $C^k \in C(T^*)$ with $t > s$, and suppose that $T^*$ contains one of $x C^k$ or $C^k x$. 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 $C^k, C^t \in C(T^*)$ with $s < k < \ell$, and suppose that $T^*$ contains edges $x C^k$ and one of $y C^t$ 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 32.

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 \( wC \). By Lemma 31, \( T_2 \) is an \( f \)-DAG. Let \( T_3 \) be the subgraph of \( T_2 \) induced by the vertices in \( p \). It is easy to verify that \( T_3 \) consists of two directed paths that share their last node. Therefore, by Corollary 33, 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.

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

(a) in the case 2 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 \( \text{walk}(C) \) and \( \text{walk}(w) \), and \( T^\ast \) be the graph obtained from \( T^\dagger \) by adding the edge \( wC \). By Lemma 31 graph \( T^\ast \) is an \( f \)-DAG (we also need to observe that any subgraph of an \( f \)-DAG is again an \( f \)-DAG).

Consider case 1, i.e. when the lowest common ancestor of \( u \) and \( v \) in \( T \) is a variable node \( r \in V(T) \). By taking an appropriate subgraph of \( T^\ast \) and applying Corollary 33 we get the desired claim, i.e. that \( f \oplus b_{i,j} \) is valid for any non-empty subinterval \([i,j] \subset [0,k] \).

Now consider case 2, i.e. when the lowest common ancestor of \( u \) and \( v \) in \( T \) is a constraint node \( R \in C(T) \). Note that \( T \) has a unique source node \( u \) (that does not have incoming edges), and \( u \) has an outgoing edge \( uD \) where \( D \) is the constraint node with the smallest timestamp in \( T^\ast \). Let us repeat the following operation while \( D \neq R^\ast \): replace \( f \) with \( f \oplus (uDz) \) where \( z \) is the unique out-neighbor of \( D \) in \( T^\ast \), and simultaneously modify \( T^\ast \) by removing nodes \( u, D \) and edges \( uD, Dz \). By Lemma 32, \( f \) remains a valid edge labeling throughout this process, and \( T^\ast \) remains an \( f \)-DAG (for the latest \( f \)).

We get to the point that the unique in-neighbor \( u \) of \( R^\ast \) is the source node of \( T^\ast \). Replace \( f \) with \( f \oplus (uR^\ast r) \), and simultaneously modify \( T^\ast \) by removing node \( u \) together with the edge \( uR \) and reversing the orientation of edge \( R^\ast r \). The new \( f \) is again valid, and the new \( T^\ast \) is an \( f \)-DAG by Lemma 34.

Now we use the same argument as in the case 1 and conclude the proof.

**Lemma (Lemma 22).** Each \( C^b \) constructed in Section 5.3 is an even \( \Delta \)-matroid.
Proof. Let \( n, n' \in C^b_i \) and assume that \( n(v) \neq n'(v) \) for some \( v \). Assume first that \( v \) is not one of \( v_j \) (there can be multiple \( v_j \)'s present since \( C_i \) might appear in the blossom multiple times). From the definition of \( C^b_i \), we know that there exist \( m, m' \in C_i \) that agree with \( n, n' \) on \( \sigma \setminus \{b_1, \ldots, b_k\} \) and since \( C_i \) is an even \( \Delta \)-matroid, there exists \( w \) such that \( m \oplus v \oplus w \in C_i \). If \( w \) is not in \( \{b_1, \ldots, b_k\} \), we take \( n \oplus v \oplus w \) and win. If \( w = b_j \) for some \( j \), we take either \( n \oplus v \oplus v_{j-1} \) or \( n \oplus v \oplus v_j \) and win.

Finally, if \( v = v_j \) for some \( j \), we again consider \( m, m' \in C_i \) as above. This time, we know that \( m \) and \( m' \) disagree on one of \( b_{j-1}, b_j \), so we use the \( \Delta \)-matroid property with this variable, get some \( w \) such that \( m \oplus b_{j-1} \oplus w \) or \( m \oplus b_j \oplus w \) lies in \( C_i \) and finish the proof as above.

The last two lemmas show that if \( b \) is a blossom which we contract to obtain the instance \( I^b \) and the assignment \( f^b \), then \( f^b \) is optimal if and only if \( f^b \) is optimal for \( I^b \).

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

**Proof.** Assume that \( f \) is not optimal for \( I \). Then by Lemma 28, there exists an augmenting \( f \)-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 \( C_i \) by \( C^b_i \). Observe that if \( p \) does not contain the variables \( b_1, \ldots, b_k \), \( p \) 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 exists \( j \in \{1, 2, \ldots, k\} \) such that \( p_{[0,i]} C^b_j b_{j-1} \) or \( p_{[0,i]} C_j b_j \) is an \( f \)-walk (i.e. we can enter the blossom from \( p \)).

If \( j = 1 \), then it is straightforward to show that \( p_{[0,i]} C_1 v_1 \) is an augmenting \( f^b \)-walk in \( I^b \), while if \( j \neq 1 \), then \( p_{[0,i]} C_j v_j N v_1 \) is an augmenting \( f^b \)-walk. (This is easy to verify from the definitions of \( C^b_i \) and \( N \)). In both cases, we get an augmenting walk for \( f^b \), and so \( f^b \) was not optimal.

**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.** Observe first that if \( b = b_0 C_1 \ldots b_k \) is an \( f \)-blossom, then for any \( [i,j] \subseteq [0,k] \) the walks \( b_{[i,j]} \) and \( b_{[i,j]}^{-1} \) are in fact \( f \)-walks.

Using Lemma 28, we can use \( g^b \) and \( f^b \) to find in polynomial time an augmenting \( f^b \)-walk \( p^b \) for \( I^b \) that does not begin at \( v_1 \). If \( p^b \) does not contain any of the variables \( v_1, \ldots, v_k \), then we can just output the walk \( p \) obtained from \( p^b \) by replacing each \( C^b_i \) by \( C_i \).

If \( p^b \) enters \( \{v_1, \ldots, v_k\} \), we will follow along \( p^b \) find \( i \) minimal such that \( p_{[0,i]} C^b_j v_j \) is an \( f^b \)-walk for some \( j \). Let \( p_{[0,i]} \) be the walk obtained from \( p_{[0,i]} \) by replacing each \( C^b_i \) by \( C_i \).

As in the proof of Lemma 25, we will need to consider two cases. If \( j = 1 \) then (by the definition of \( C^b_i \)) at least one of \( p_{[0,i]} C_1 b_1 \) or \( p_{[0,i]} C_1 b_k \) is an \( f \)-walk. If it is the latter, we are done. If \( p_{[0,i]} C_1 b_1 \) is an augmenting \( f \)-walk, we consider the walk \( q = p_{[0,i]} C_1 b_{[1,k]} \).
We claim that $s$ is an $f$-walk. If that is true, $q$ will also be augmenting, so we can just output it and be done. Assume that this is not the case. If $q$ is not an $f$-walk, then $b_{[1,k]}$ is not an $(f \oplus p_{[0,0]})$-walk. But then we can apply Lemma 29 and obtain that there is an $i^* < i$ and $j^*$ such that $p_{[0,i]} C_j b_{j^*}$ or $p_{[0,i]} C_j b_{j^* - 1}$ is an $f$-walk, yielding the $f^k$-walk $p_{[0,i]} C_j [b_{j^*}, v_{j^*}]$, a contradiction with the minimality of $i$.

The case when $j \neq 1$ is similar: one of $p_{[0,i]} C_j b_j$ resp. $p_{[0,i]} C_j b_{j-1}$ is an $f$-walk and $p_{[0,i]} C_j b_{j-1}$ resp. $p_{[0,i]} C_j b_{j-1}$ is an augmenting $f$-walk.

Since the only nonobvious computational task we have to do is to find the minimal $i$, we can output the augmenting path in time linear in the size of $p$ and $I$.

\[ \square \]

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 35. Suppose that Algorithm 1 outputs “No” in step 3 without ever visiting steps 4c and 4a. Then $f$ is optimal.

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

$$E(T) = \{ C \mid C^t v \in E(T) \ \text{for some} \ t \} \cup \{ v C \mid v C^{t} \in E(T) \ \text{for some} \ t \}.$$ 

Inspecting Algorithm 1 one can check that $E(T)$ has the following properties:

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

(b) If $v C \in E(T)$ and $\{ v, D \} \in \mathcal{E}, D \neq C$, then $v D \in E(T)$.

(c) If $v C \in E(T)$, then $C v \notin E(T)$.

(d) Suppose that $v C \in E(T)$ and $f(C) \oplus v \oplus w \in C$ where $v, w$ are distinct nodes in the scope of constraint $C$. Then $C w \in E(T)$.

An $f$-walk $p$ will be called bad if it starts at a variable node which is inconsistent in $f$, and contains an edge $C v \notin E(T)$; otherwise $p$ is good. Clearly, any augmenting $f$-walk is bad: its last edge $C v$ satisfies $v C \in E(T)$ by property 1 and thus $C v \notin E(T)$. Thus, if $f$ is not optimal, then there exists at least one bad $f$-walk. Let $p$ be a shortest bad $f$-walk. Write $p = p^*(v C w)$ where $p^*$ ends at $v$. By minimality of $p$, $p^*$ is good and $C w \notin E(T)$.

Using properties 2 or 3 we obtain that $v C \in E(T)$ (and therefore $C v \notin E(T)$).

Let $q$ be the shortest prefix of $p^*$ (also an $f$-walk) such that $f \oplus q \oplus (v C w)$ is valid (at least one such prefix exists, namely $q = p^*$). The walk $q$ must be of positive length (otherwise the precondition of property 3 would hold, and we would get $C w \in E(T)$, a contradiction). Also, the last constraint node in $q$ must be $C$, otherwise we could have taken a shorter prefix. Thus, we can write $q = q^*(x C y)$ 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 $Cu \notin E(T)$ we get that $q^*(xCu)$ is a bad walk – a contradiction to the minimality of $p$.

Corollary 36. If Algorithm 7 answers “No”, then the edge labeling $f$ is optimal.

Proof. Algorithm 7 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 35. 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 $I$.

Acknowledgements

This work was supported by European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 616160.
References


A Non matching realizable even $\Delta$-matroid

Here we prove Proposition 14 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 37. 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 vertices 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 vertex of $N \setminus V$, the degree of all such vertices in $G'$ will be zero or two. Similarly, the degrees of vertices in $(V \setminus (f \Delta g))$ are either zero or two leaving $f \Delta g$ as the set of vertices of odd degree, namely of degree one. Thus $G'$ is a union of induced cycles and paths, where the paths pair up the vertices 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 38. There is an even $\Delta$-matroid of arity 6 which does not have the property from Lemma 37.

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

\begin{verbatim}
000000  101000  011011  111111
011000  100111
001100  110011
001010  110101
000101  111010
001001  001111
010001  101101
100010  101011
111100
111100
\end{verbatim}

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 $\{v_1, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_6\}$ (see the first five lines in the middle of the table above) but no three of these form a partition on $\{v_1, \ldots, v_6\}$.