Solving edge CSP with even delta-matroid constraints

Alexander Kazda, Vladimir Kolmogorov, and Michal Rolínek

June 15th, 2017
AAA94 + NSAC 2017
A finite set of variables $V$ . . .

. . . to which we want to assign values 0 or 1 . . .

. . . so that a set $C$ of constraints is satisfied.

Examples: Graph 2-coloring, linear equations over $\mathbb{Z}_2$, 3-SAT, finding a perfect matching in a graph, . . .
Boolean Constraint Satisfaction Problem

- A finite set of variables $V$...
- ...to which we want to assign values 0 or 1...
- ...so that a set $C$ of constraints is satisfied.
- Examples: Graph 2-coloring, linear equations over $\mathbb{Z}_2$, 3-SAT, finding a perfect matching in a graph, ...
Boolean Constraint Satisfaction Problem

- A finite set of variables $V$...
- ...to which we want to assign values 0 or 1...
- ...so that a set $C$ of constraints is satisfied.
- Examples: Graph 2-coloring, linear equations over $\mathbb{Z}_2$, 3-SAT, finding a perfect matching in a graph, ...
Boolean Constraint Satisfaction Problem

- A finite set of variables $V$ . . .
- . . .to which we want to assign values 0 or 1 . . .
- . . .so that a set $C$ of constraints is satisfied.
- Examples: Graph 2-coloring, linear equations over $\mathbb{Z}_2$, 3-SAT, finding a perfect matching in a graph, . . .
A finite set of variables $V$...

...to which we want to assign values 0 or 1...

...so that a set $C$ of constraints is satisfied.

Examples: Graph 2-coloring, linear equations over $\mathbb{Z}_2$, 3-SAT, finding a perfect matching in a graph, ...
Boolean Constraint Satisfaction Problem

- A finite set of variables $V$...
- ...to which we want to assign values 0 or 1...
- ...so that a set $C$ of constraints is satisfied.
- Examples: Graph 2-coloring, linear equations over $\mathbb{Z}_2$, 3-SAT, finding a perfect matching in a graph, ...
Perfect matchings

Given a set of edges $V$ and a set of vertices $C$

Goal: Find $f : V \rightarrow \{0, 1\}$ that is a perfect matching:

$\forall C \in C$ we have


Strategy: Start with an empty matching and keep improving it.
Given a set of edges $V$ and a set of vertices $C$

Goal: Find $f : V \rightarrow \{0, 1\}$ that is a **perfect matching**:

$\forall C \in C$ we have

- Strategy: Start with an empty matching and keep improving it.
Perfect matchings

- Given a set of edges $V$ and a set of vertices $C$
- Goal: Find $f : V \rightarrow \{0, 1\}$ that is a perfect matching:

  $\forall C \in C$ we have

- Strategy: Start with an empty matching and keep improving it.
Perfect matchings

- Given a set of edges \( V \) and a set of vertices \( C \)
- Goal: Find \( f : V \rightarrow \{0, 1\} \) that is a perfect matching:

\[ \forall C \in C \text{ we have } \]

- Strategy: Start with an empty matching and keep improving it.
Boolean edge CSP

- Boolean CSP where each variable appears in exactly two constraints.
- Constraints = vertices, variables = edges:

\[ C = \{ \times, \times, \times, \times \} \]
Boolean edge CSP

- Boolean CSP where each variable appears in exactly two constraints.
- Constraints = vertices, variables = edges:

\[ C = \{ \times, \times, \times, \times \} \]
Boolean edge CSP

- Boolean CSP where each variable appears in exactly two constraints.
- Constraints = vertices, variables = edges:

\[ C = \{ \times \times, \times \times, \times \times, \times \times \} \]
Boolean edge CSP

- Boolean CSP where each variable appears in exactly two constraints.
- Constraints = vertices, variables = edges:

\[ C = \{ \text{constraint 1}, \text{constraint 2}, \text{constraint 3}, \text{constraint 4} \} \]
T. Feder, 2001: Edge CSP is only interesting when all constraint relations are \( \Delta \)-matroids (if we have constants).

A (nonempty) relation \( M \subset \{0, 1\}^n \) is a \( \Delta \)-matroid if it satisfies a certain exchange axiom.

Previous algorithms for special classes of \( \Delta \)-matroids: co-independent (Feder, 2001), compact (Istrate, 1997), local (Dalmau and Ford, 2003), binary (Geelen, Iwata and Murota, 2003; Dalmau and Ford, 2003).

Our algorithm will work for the natural class of even \( \Delta \)-matroids.
T. Feder, 2001: Edge CSP is only interesting when all constraint relations are \(\Delta\)-matroids (if we have constants).

A (nonempty) relation \(M \subset \{0, 1\}^n\) is a \(\Delta\)-matroid if it satisfies a certain exchange axiom.

Previous algorithms for special classes of \(\Delta\)-matroids: co-independent (Feder, 2001), compact (Istrate, 1997), local (Dalmau and Ford, 2003), binary (Geelen, Iwata and Murota, 2003; Dalmau and Ford, 2003).

Our algorithm will work for the natural class of even \(\Delta\)-matroids.
T. Feder, 2001: Edge CSP is only interesting when all constraint relations are $\Delta$-matroids (if we have constants).

A (nonempty) relation $M \subset \{0, 1\}^n$ is a $\Delta$-matroid if it satisfies a certain exchange axiom.

Previous algorithms for special classes of $\Delta$-matroids: co-independent (Feder, 2001), compact (Istrate, 1997), local (Dalmau and Ford, 2003), binary (Geelen, Iwata and Murota, 2003; Dalmau and Ford, 2003).

Our algorithm will work for the natural class of even $\Delta$-matroids.
- T. Feder, 2001: Edge CSP is only interesting when all constraint relations are \( \Delta \)-matroids (if we have constants).

- A (nonempty) relation \( M \subset \{0, 1\}^n \) is a \( \Delta \)-matroid if it satisfies a certain exchange axiom.

- Previous algorithms for special classes of \( \Delta \)-matroids: co-independent (Feder, 2001), compact (Istrate, 1997), local (Dalmau and Ford, 2003), binary (Geelen, Iwata and Murota, 2003; Dalmau and Ford, 2003).

- Our algorithm will work for the natural class of even \( \Delta \)-matroids.
T. Feder, 2001: Edge CSP is only interesting when all constraint relations are \( \Delta \)-matroids (if we have constants).

A (nonempty) relation \( M \subset \{0,1\}^n \) is a \( \Delta \)-matroid if it satisfies a certain exchange axiom.

Previous algorithms for special classes of \( \Delta \)-matroids: co-independent (Feder, 2001), compact (Istrate, 1997), local (Dalmau and Ford, 2003), binary (Geelen, Iwata and Murota, 2003; Dalmau and Ford, 2003).

Our algorithm will work for the natural class of even \( \Delta \)-matroids.
**Even Δ-matroids**

- $M$ is an even Δ-matroid if $M \subset \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

- The result of switching the two positions in the second tuple needs to stay within $M$.
- Example: $M = \{(1000), (0100), (0010), (0001)\}$. 
$M$ is an even $\Delta$-matroid if $M \subset \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

- The result of switching the two positions in the second tuple needs to stay within $M$.
- Example: $M = \{(1000), (0100), (0010), (0001)\}$. 
**Even $\Delta$-matroids**

- $M$ is an even $\Delta$-matroid if $M \subseteq \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
\end{array} \in M
\]

- The result of switching the two positions in the second tuple needs to stay within $M$.
- Example: $M = \{(1000), (0100), (0010), (0001)\}$.
$M$ is an even $\Delta$-matroid if $M \subset \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

\[
\begin{align*}
0 & \quad 0 \quad 0 \quad 1 \quad 1 \quad \in M \\
1 & \quad 1 \quad 1 \quad 0 \quad 1 \quad \in M
\end{align*}
\]

- The result of switching the two positions in the second tuple needs to stay within $M$.
- Example: $M = \{(1000), (0100), (0010), (0001)\}$. 
Even \( \Delta \)-matroids

- \( M \) is an even \( \Delta \)-matroid if \( M \subset \{0, 1\}^n \), all members of \( M \) have same parity and \( M \) satisfies this exchange axiom:

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1
\end{array} \in M
\]

- The result of switching the two positions in the second tuple needs to stay within \( M \).
- Example: \( M = \{(1000), (0100), (0010), (0001)\} \).
Even $\Delta$-matroids

- $M$ is an even $\Delta$-matroid if $M \subset \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- The result of switching the two positions in the second tuple needs to stay within $M$.
- Example: $M = \{(1000), (0100), (0010), (0001)\}$. 
Even $\Delta$-matroids

- $M$ is an even $\Delta$-matroid if $M \subset \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

  \[
  \begin{array}{cccc}
  0 & 0 & 0 & 1 & 1 \\
  1 & 1 & 1 & 0 & 1 \\
  0 & 0 & 1 & 0 & 1 \\
  \end{array} \in M
  \]

- The result of switching the two positions in the second tuple needs to stay within $M$.

- Example: $M = \{(1000), (0100), (0010), (0001)\}$.
Even $\Delta$-matroids

- $M$ is an even $\Delta$-matroid if $M \subset \{0, 1\}^n$, all members of $M$ have same parity and $M$ satisfies this exchange axiom:

  \[
  \begin{array}{cccc}
  0 & 0 & 0 & 1 & 1 \\
  1 & 1 & 1 & 0 & 1 \\
  0 & 0 & 1 & 0 & 1 \\
  \end{array}
  \in M
  \]

- The result of switching the two positions in the second tuple needs to stay within $M$.

- Example: $M = \{(1000), (0100), (0010), (0001)\}$. 
Solving edge CSP for even $\Delta$-matroids

- Similar to perfect matchings in graphs, but much more delicate.
- Label variables with 0s and 1s, some variables inconsistent.
- Exchange axiom $\implies$ we can walk.
- Want: Augmenting walk from one inconsistent variable to another.
- Unlike in matchings, we can visit a constraint multiple times.
Solving edge CSP for even $\Delta$-matroids

- Similar to perfect matchings in graphs, but much more delicate.
- Label variables with 0s and 1s, some variables inconsistent.
- Exchange axiom $\Rightarrow$ we can walk.
- Want: Augmenting walk from one inconsistent variable to another.
- Unlike in matchings, we can visit a constraint multiple times.
Solving edge CSP for even $\Delta$-matroids

- Similar to perfect matchings in graphs, but much more delicate.
- Label variables with 0s and 1s, some variables inconsistent.
  - Exchange axiom $\Rightarrow$ we can walk.
  - Want: Augmenting walk from one inconsistent variable to another.
  - Unlike in matchings, we can visit a constraint multiple times.
Solving edge CSP for even $\Delta$-matroids

- Similar to perfect matchings in graphs, but much more delicate.
- Label variables with 0s and 1s, some variables inconsistent.
- Exchange axiom $\Rightarrow$ we can walk.
- Want: Augmenting walk from one inconsistent variable to another.
- Unlike in matchings, we can visit a constraint multiple times.
Similar to perfect matchings in graphs, but much more delicate.

- Label variables with 0s and 1s, some variables inconsistent.
- Exchange axiom $\Rightarrow$ we can walk.
- Want: Augmenting walk from one inconsistent variable to another.
- Unlike in matchings, we can visit a constraint multiple times.
Solving edge CSP for even Δ-matroids

- Similar to perfect matchings in graphs, but much more delicate.
- Label variables with 0s and 1s, some variables inconsistent.
- Exchange axiom $\Rightarrow$ we can walk.
- Want: Augmenting walk from one inconsistent variable to another.
- Unlike in matchings, we can visit a constraint multiple times.
- Explore the instance starting from inconsistent variables.
- If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \times, \times, \times, \times \} \]
Explore the instance starting from inconsistent variables.

If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \times, \times, \times, \times \} \]
Example

- Explore the instance starting from inconsistent variables.
- If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \color{red}{\times}, \color{green}{\times}, \color{red}{\times}, \color{green}{\times}, \color{green}{\times} \} \]
Explore the instance starting from inconsistent variables.
If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \text{\textbullet}, \text{\textbullet}, \text{\textbullet}, \text{\textbullet} \} \]
- Explore the instance starting from inconsistent variables.
- If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \times, \times, \times, \times \} \]
Explore the instance starting from inconsistent variables.
If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \times, \times, \times, \times \} \]
Example

- Explore the instance starting from inconsistent variables.
- If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \text{cross}, \text{cross}, \text{cross}, \text{cross} \} \]
Explore the instance starting from inconsistent variables.
If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \quad , \quad , \quad , \quad \} \]
Example

- Explore the instance starting from inconsistent variables.
- If we don’t reach any variable from both directions, everything is easy.

\[
C = \{ \times, \times, \times, \times, \times \}
\]
Explore the instance starting from inconsistent variables.
If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \times, \times, \times, \times \} \]
Explore the instance starting from inconsistent variables.
If we don’t reach any variable from both directions, everything is easy.

\[ C = \{ \, \, , \, \, , \, \, \} \]
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a “smaller” edge CSP instance.
- Example:

- Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.

  Example:

  - Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.
- Example:

  Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.
- Example:

![Graph with nodes and edges]

- Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.
- Example:

![Diagram of a blossom in an edge CSP instance](attachment:diagram.png)

- Proving correctness requires work (e.g. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a “smaller” edge CSP instance.
- Example:

```
+---+---+---+---+
|   |   |   |   |
+---+---+---+---+
|   |   |   |   |
+---+---+---+---+
|   |   |   |   |
+---+---+---+---+
```

- Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.
- Example:

Proving correctness requires work (e.g. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.
- Example:

![Diagram of blossoms](image)

- Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we **contract the blossom** and recursively solve a “smaller” edge CSP instance.
- Example:

![Diagram](image)

- Proving correctness requires work (eg. keeping track of the order in which we visited variables).
Contracting blossoms

- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a “smaller” edge CSP instance.
- Example:

Proving correctness requires work (e.g. keeping track of the order in which we visited variables).
Planar Boolean CSP

- Boolean CSP instances having planar drawings.
- Dvořák and Kupec: All interesting cases of planar CSP can be reduced to edge CSP with even $\Delta$-matroid constraints.
- Our algorithm $\Rightarrow$ Dichotomy for planar Boolean CSP.

Boolean CSP instances having planar drawings.

Dvořák and Kupec: All interesting cases of planar CSP can be reduced to edge CSP with even $\Delta$-matroid constraints.

Our algorithm $\Rightarrow$ Dichotomy for planar Boolean CSP.

Boolean CSP instances having planar drawings.

Dvořák and Kupec: All interesting cases of planar CSP can be reduced to edge CSP with even \( \Delta \)-matroid constraints.

Our algorithm \( \Rightarrow \) Dichotomy for planar Boolean CSP.
Planar Boolean CSP

- Boolean CSP instances having planar drawings.
- Dvořák and Kupec: All interesting cases of planar CSP can be reduced to edge CSP with even $\Delta$-matroid constraints.
- Our algorithm $\Rightarrow$ Dichotomy for planar Boolean CSP.
- Boolean CSP instances having planar drawings.
- Dvořák and Kupec: All interesting cases of planar CSP can be reduced to edge CSP with even Δ-matroid constraints.
- Our algorithm $\Rightarrow$ Dichotomy for planar Boolean CSP.
Open problems

- How to find a solution of minimal cost?
- We can handle effectively coverable \( \Delta \)-matroids \( \supseteq \) previously known tractable classes.
- Algorithm for general \( \Delta \)-matroids?
- Generalization to value sets larger than 2?
- Where is the algebraic approach hiding?!
Open problems

- How to find a solution of minimal cost?
- We can handle effectively coverable $\Delta$-matroids $\supseteq$ previously known tractable classes.
- Algorithm for general $\Delta$-matroids?
- Generalization to value sets larger than 2?
- Where is the algebraic approach hiding?!
Open problems

- How to find a solution of minimal cost?
- We can handle **effectively coverable** $\Delta$-matroids $\supseteq$ previously known tractable classes.
- Algorithm for general $\Delta$-matroids?
- Generalization to value sets larger than 2?
- Where is the algebraic approach hiding?!
Open problems

- How to find a solution of minimal cost?
- We can handle **effectively coverable** Δ-matroids ⊇ previously known tractable classes.
- Algorithm for general Δ-matroids?
  - Generalization to value sets larger than 2?
  - Where is the algebraic approach hiding?!
Open problems

- How to find a solution of minimal cost?
- We can handle **effectively coverable** $\Delta$-matroids $\supseteq$ previously known tractable classes.
- Algorithm for general $\Delta$-matroids?
- Generalization to value sets larger than 2?
- Where is the algebraic approach hiding?!
Open problems

- How to find a solution of minimal cost?
- We can handle **effectively coverable** Δ-matroids ⊇ previously known tractable classes.
- Algorithm for general Δ-matroids?
- Generalization to value sets larger than 2?
- Where is the algebraic approach hiding?!
Thank you for your attention.