Solving edge CSP with even delta-matroid constraints

Alexandr Kazda, Vladimir Kolmogorov, and Michal Rolínek

June 15th, 2017
AAA94 + NSAC 2017

Boolean Constraint Satisfaction Problem

- A finite set of variables V ...
- ... to which we want to assign values 0 or 1 ...
- ... so that a set $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathcal{C}$
- Goal: Find $f\colon V \to \{0,1\}$ that is a perfect matching:

$\forall C \in \mathcal{C}$ we have

- Poly-time algorithm by Jack Edmonds (1965).
- Strategy: Start with an empty matching and keep improving it.

Perfect matchings

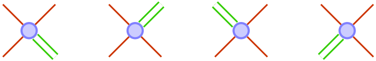
- Given a set of edges V and a set of vertices $\mathcal{C}$
- Goal: Find $f\colon V \to \{0,1\}$ that is a perfect matching:

$\forall C \in \mathcal{C}$ we have 

- Poly-time algorithm by Jack Edmonds (1965).
- Strategy: Start with an empty matching and keep improving it.

Perfect matchings

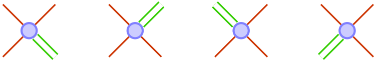
- Given a set of edges V and a set of vertices $\mathcal{C}$
- Goal: Find $f\colon V \to \{0,1\}$ that is a perfect matching:

$\forall C \in \mathcal{C}$ we have 

- Poly-time algorithm by Jack Edmonds (1965).
- Strategy: Start with an empty matching and keep improving it.

Perfect matchings

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

$\forall C \in \mathcal{C}$ we have 

- Poly-time algorithm by Jack Edmonds (1965).
- 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:

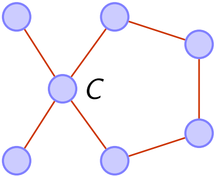
$C = \{$ $\}$

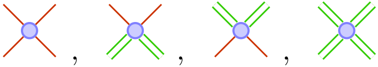
Boolean edge CSP

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

$C =$

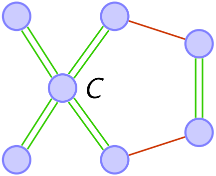
- Boolean CSP where each variable appears in exactly two constraints.
- Constraints = vertices, variables = edges:

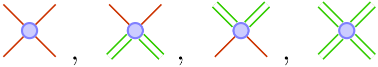


$C = \Big\{$  $\Big\}$

Boolean edge CSP

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



$C = \{$  $\}$

Δ-matroids

- T. Feder, 2001: Edge CSP is only interesting when all constraint relations are Δ-matroids (if we have constants).
- A (nonempty) relation $M \subset \{0,1\}^n$ is a Δ-matroid if it satisfies a certain exchange axiom.
- Previous algorithms for special classes of Δ-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 Δ-matroids.

Δ-matroids

- T. Feder, 2001: Edge CSP is only interesting when all constraint relations are Δ-matroids (if we have constants).
- A (nonempty) relation $M \subset \{0,1\}^n$ is a Δ-matroid if it satisfies a certain exchange axiom.
- Previous algorithms for special classes of Δ-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 Δ-matroids.

Δ-matroids

- T. Feder, 2001: Edge CSP is only interesting when all constraint relations are Δ-matroids (if we have constants).
- A (nonempty) relation $M \subset \{0,1\}^n$ is a Δ-matroid if it satisfies a certain exchange axiom.
- Previous algorithms for special classes of Δ-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 Δ-matroids.

Δ-matroids

- T. Feder, 2001: Edge CSP is only interesting when all constraint relations are Δ-matroids (if we have constants).
- A (nonempty) relation $M \subset \{0,1\}^n$ is a Δ-matroid if it satisfies a certain exchange axiom.
- Previous algorithms for special classes of Δ-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 Δ-matroids.

Δ-matroids

- T. Feder, 2001: Edge CSP is only interesting when all constraint relations are Δ-matroids (if we have constants).
- A (nonempty) relation $M \subset \{0,1\}^n$ is a Δ-matroid if it satisfies a certain exchange axiom.
- Previous algorithms for special classes of Δ-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 Δ-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: $M = \{(1000), (0100), (0010), (0001)\}$.

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: $M = \{(1000), (0100), (0010), (0001)\}$.

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:

$$\boxed{0} \quad 0 \quad 0 \quad 1 \quad 1 \quad \in M$$

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

Even 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:

$$\begin{array}{cccccl} \boxed{0} & 0 & 0 & 1 & 1 & \in M \\ \boxed{1} & 1 & 1 & 0 & 1 & \in M \end{array}$$

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

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:

$$\begin{array}{cccccc} \boxed{0} & \boxed{0} & 0 & 1 & 1 & \in M \\ \boxed{1} & \boxed{1} & 1 & 0 & 1 & \in M \end{array}$$

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

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:

$$\begin{array}{cccccc} \boxed{0} & \boxed{0} & 0 & 1 & 1 & \in M \\ \boxed{1} & \boxed{1} & 1 & 0 & 1 & \in M \\ \boxed{0} & \boxed{0} & 1 & 0 & 1 & \in M \end{array}$$

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

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:

$$\begin{array}{cccccc} \boxed{0} & \boxed{0} & 0 & 1 & 1 & \in M \\ \boxed{1} & \boxed{1} & 1 & 0 & 1 & \in M \\ \boxed{0} & \boxed{0} & 1 & 0 & 1 & \in M \end{array}$$

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

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:

$$\begin{array}{cccccc} \boxed{0} & \boxed{0} & 0 & 1 & 1 & \in M \\ \boxed{1} & \boxed{1} & 1 & 0 & 1 & \in M \\ \boxed{0} & \boxed{0} & 1 & 0 & 1 & \in M \end{array}$$

- 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 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.

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.

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.

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.

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.

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.

Example

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

Example

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

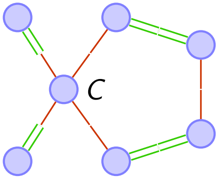
$C = \{$ $\}$

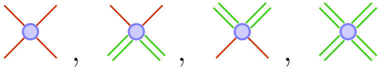
Example

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

Example

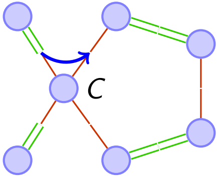
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

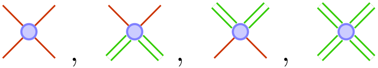


$C = \{$  $\}$

Example

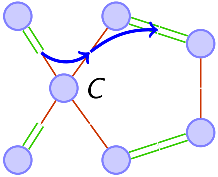
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

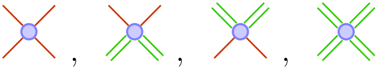


$C = \{$  $\}$

Example

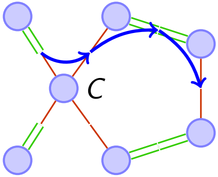
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

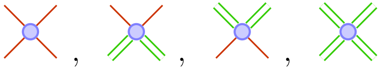


$C = \{$  $\}$

Example

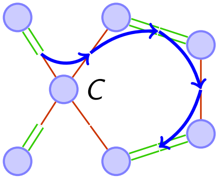
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

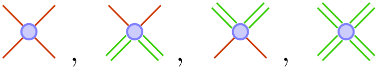


$C = \{$  $\}$

Example

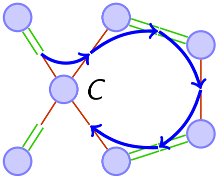
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

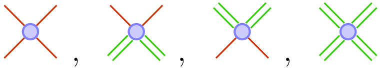


$C = \{$  $\}$

Example

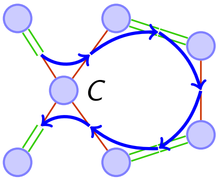
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

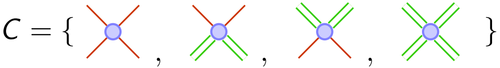


$C = \{$  $\}$

Example

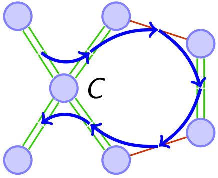
- Explore the instance starting from inconsistent variables.
- If we don't reach any variable from both directions, everything is easy.

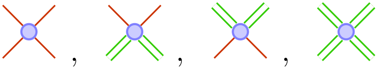




Example

- 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).

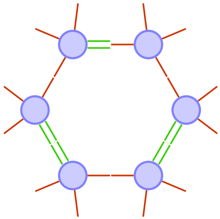
Contracting blossoms

- 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).

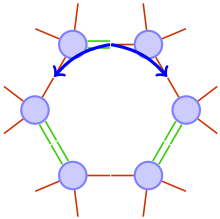
Contracting blossoms

- 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:



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

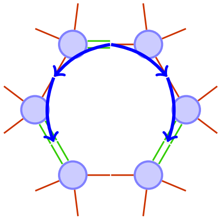
- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a "smaller" edge CSP instance.
- Example:



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

Contracting blossoms

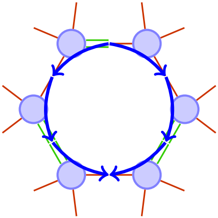
- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a "smaller" edge CSP instance.
- Example:



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

Contracting blossoms

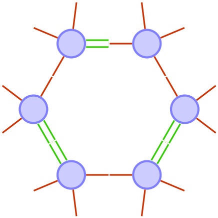
- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a "smaller" edge CSP instance.
- Example:



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

Contracting blossoms

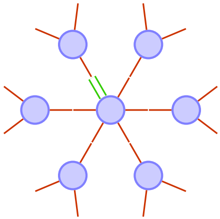
- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a "smaller" edge CSP instance.
- Example:



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

Contracting blossoms

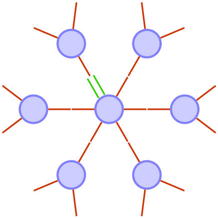
- If we can reach a variable from both sides, we have a blossom.
- If that happens we contract the blossom and recursively solve a "smaller" edge CSP instance.
- Example:



- 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).

Planar Boolean CSP

- Z. Dvořák and M. Kupec: On Planar Boolean CSP, 2015.
- 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.

Planar Boolean CSP

- Z. Dvořák and M. Kupec: On Planar Boolean CSP, 2015.
- 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.

Planar Boolean CSP

- Z. Dvořák and M. Kupec: On Planar Boolean CSP, 2015.
- 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.

Planar Boolean CSP

- Z. Dvořák and M. Kupec: On Planar Boolean CSP, 2015.
- 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.

Planar Boolean CSP

- Z. Dvořák and M. Kupec: On Planar Boolean CSP, 2015.
- 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 Δ-matroids $\supseteq$ 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 Δ-matroids $\supseteq$ 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 Δ-matroids $\supseteq$ 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 Δ-matroids $\supseteq$ 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 Δ-matroids $\supseteq$ 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 Δ-matroids $\supseteq$ 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.