As easy as a piece of cake
Analytically cutting infinite cakes (yum!)

*Baptiste Plaquevent-Jourdain*, with
Jean-Pierre Dussault, Université de Sherbrooke
Jean Charles Gilbert, INRIA Paris

January, 09 2024
Outline

1. My Personal Recipe
2. First part(s of the cake)
3. Formalism
4. An algorithm
5. Some improvements
Plan

1. My Personal Recipe
2. First part(s of the cake)
3. Formalism
4. An algorithm
5. Some improvements
Who are you listening to? (1)

origin

French PhD student from Brittany (sea, crêpes, galettes, Mont Saint-Michel...), then from ENSTA Paris
Who are you listening to? (1)

origin

French PhD student from Brittany (sea, crêpes, galettes, Mont Saint-Michel...), then from ENSTA Paris
Who are you listening to? (1)

origin

French PhD student from Brittany (sea, crêpes, galettes, Mont Saint-Michel...), then from ENSTA Paris
current status

- starting 3rd year, finishing on December, 31st (unless...)
- "cotutelle" France-Québec, here during winter
Who are you listening to? (3)

My (first) subject

Initially doing nonsmooth optimization (theoretically)...

(Fragments d’Optimisation Différentiable - Théorie et Algorithmes)

My (current) subject

... but today: computational/combinatorial geometry cakes!
Who are you listening to? (3)

My (first) subject

Initially doing nonsmooth optimization (theoretically)...

(Fragments d’Optimisation Différentiable - Théorie et Algorithmes)

My (current) subject

... but today: computational/combinatorial geometry cakes!
Plan

1. My Personal Recipe

2. First part(s of the cake)

3. Formalism

4. An algorithm

5. Some improvements
Cutting cakes rules

main rule

\texttt{cut:= line that completely cut the cake (no stopping in the middle)}

second rule

We also assume the cakes are infinite (see later).
Cutting cakes rules

**main rule**

\[ \text{cut} := \text{line that completely cut the cake (no stopping in the middle)} \]

WRONG!

**second rule**

We also assume the cakes are infinite (see later).
Cutting cakes rules

**main rule**

\[ \text{cut:} = \text{line that completely cut the cake (no stopping in the middle)} \]

**WRONG!**

**GOOD!!**

**second rule**

We also assume the cakes are infinite (see later).
Cutting cakes rules

**main rule**

cut: line that completely cut the cake (no stopping in the middle)

WRONG!

GOOD!!

**second rule**

We also assume the cakes are infinite (see later).
A first taste - 1

One cut, 2 slices
A first taste - 1

One cut, 2 slices

Two cuts, 4 slices
A first taste - 2

Three cuts, 6 slices

\[ p \text{ cuts, } 2p \text{ slices} \]

'Proof': every cut makes 2 previous slices becoming 4 smaller slices

\[ 2p \rightarrow (2p - 2) + 2 \times 2 = (2p - 2) + 4 = 2(p + 1). \]
A first taste - 2

Three cuts, 6 slices

us around the pizzas

\[ p \text{ cuts, } 2p \text{ slices} \]

'Proof': every cut makes 2 previous slices becoming 4 smaller slices

\[ 2p \rightarrow (2p - 2) + 2 \cdot 2 = (2p - 2) + 4 = 2(p + 1). \]
A first taste - 2

Three cuts, 6 slices

\( p \) cuts, \( 2p \) slices

'Proof': every cut makes 2 previous slices becoming 4 smaller slices

\[ 2p \rightarrow (2p - 2) + 2 \times 2 = (2p - 2) + 4 = 2(p + 1). \]
Other possibilities - 1

What about 7 parts?

Asymmetric cuts - they don’t all pass by the center/middle
Other possibilities - 2

Actually can’t (really) have 5 slices: this is cheating. This does not respect the infinite cakes assumption.

But the 7-slices one still works: the $2p$ formula isn’t valid...
Is it possible to get 8 slices in three cuts?
Other possibilities - 3
Summary

- symmetric cuts in 2D (all by the center): $p$ cuts $\Rightarrow 2p$ slices
- cutting in a "new dimension" doubles; $2^n$ slices!
- asymmetric cuts: it’s harder

But what about a cake-shaped cake?

So here, $p$ cuts mean $p + 1$ slices... because they’re all parallel!
Summary

• symmetric cuts in 2D (all by the center): $p$ cuts $\Rightarrow 2p$ slices
• cutting in a "new dimension" doubles; $2^n$ slices!
• asymmetric cuts: it’s harder

But what about a cake-shaped cake?

So here, $p$ cuts mean $p + 1$ slices… because they’re all parallel!
Summary

- symmetric cuts in 2D (all by the center): $p$ cuts $\Rightarrow 2p$ slices
- cutting in a "new dimension" doubles; $2^n$ slices!
- asymmetric cuts: it’s harder

But what about a cake-shaped cake?

So here, $p$ cuts mean $p + 1$ slices... because they’re all parallel!
Parallel sets in each dimension

But parallel set of cuts in each dimension also work:

\[ p_1, p_2 \rightarrow (p_1 + 1) \times (p_2 + 1) \]

(you can check the slices after the pizzas :3)
Conclusion

So maybe not completely a piece of cake...
Depends on: dimension $n$, number of cuts $p$, and which cuts.

Observations: new dimension means doubling the cuts, parallel cuts behave weirdly, 5 slices is hard to get...

Question
For a given set of cuts, how many slices do we get?
Conclusion

So maybe not completely a piece of cake...
Depends on: dimension $n$, number of cuts $p$, and which cuts.

Observations: new dimension means doubling the cuts, parallel cuts behave weirdly, 5 slices is hard to get...

Question
For a given set of cuts, how many slices do we get?
Conclusion

So maybe not completely a piece of cake...
Depends on: dimension $n$, number of cuts $p$, and which cuts.

Observations: new dimension means doubling the cuts, parallel cuts behave weirdly, 5 slices is hard to get...

Question

For a given set of cuts, how many slices do we get?
Plan

1. My Personal Recipe
2. First part(s of the cake)
3. Formalism
4. An algorithm
5. Some improvements
The cake $n$-dimensional, a 'cut' is an hyperplane. 

$=$ linear (affine) subspace of dimension $n - 1$ (codimension 1).

One hyperplane: $H = v^\perp = \{d \in \mathbb{R}^n : v^T d = 0\}$.

$p$ cuts: $p$ hyperplanes: $H_i = v_i^\perp, \forall i \in [1 : p], (v_i)_i =$ problem data.

<table>
<thead>
<tr>
<th>halfspaces of an hyperplane</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}^n = H_i^- \cup H_i \cup H_i^+$,</td>
</tr>
<tr>
<td>$H_i^- = {d \in \mathbb{R}^n : v_i^T d &lt; 0}$</td>
</tr>
<tr>
<td>$H_i^+ = {d \in \mathbb{R}^n : v_i^T d &gt; 0}$</td>
</tr>
</tbody>
</table>
The cake $n$-dimensional, a 'cut' is an hyperplane.

$= \text{linear (affine) subspace of dimension } n - 1 \text{ (codimension } 1)$. 

One hyperplane: $H = v^\perp = \{d \in \mathbb{R}^n : v^T d = 0\}$. 

$p$ cuts: $p$ hyperplanes: $H_i = v_i^\perp$, $\forall i \in [1 : p]$, $(v_i)_i = \text{problem data}$. 

\[
\mathbb{R}^n = H_i^- \cup H_i \cup H_i^+,
H_i^- = \{d \in \mathbb{R}^n : v_i^T d < 0\}
H_i^+ = \{d \in \mathbb{R}^n : v_i^T d > 0\}
\]
Hyperplanes - 1

The cake $n$-dimensional, a 'cut' is an hyperplane. 

$\equiv$ linear (affine) subspace of dimension $n - 1$ (codimension 1).

One hyperplane: $H = \mathbf{v}^\perp = \{ d \in \mathbb{R}^n : \mathbf{v}^T d = 0 \}$.

$p$ cuts: $p$ hyperplanes: $H_i = \mathbf{v}_i^\perp, \forall \; i \in [1 : p]$, $(\mathbf{v}_i)_i =$ problem data.

### halfspaces of an hyperplane

$$\mathbb{R}^n = H_i^- \cup H_i \cup H_i^+, \quad H_i^- = \{d \in \mathbb{R}^n : \mathbf{v}_i^T d < 0\}, \quad H_i^+ = \{d \in \mathbb{R}^n : \mathbf{v}_i^T d > 0\}$$
Each cut: $a -$ and $a +$ side: each of the $p$ cuts, intersection of each halfspaces...
$H_1 = e_1^\perp,$ $H_2 = e_2^\perp,$ $H_3 = (e_1 + e_2)^\perp.$

Actually, # of slices and on which side of each cut it is.
$H_1 = e_1^\perp$, $H_2 = e_2^\perp$, $H_3 = (e_1 + e_2)^\perp$.

Actually, # of slices and on which side of each cut it is.
There are $p$ cuts, $2^p$ potential slices ($\forall i \in [1:p], \{-1, +1\}$)
Slice $s = (s_1, \ldots, s_p) \in \{-1, +1\}^p$ exists $\iff H_1^{s_1} \cap H_2^{s_2} \cap \cdots \cap H_p^{s_p} \neq \emptyset$

$$\begin{cases} H_i^+ : v_i^T d > 0 \iff +v_i^T d > 0 \\ H_i^- : v_i^T d < 0 \iff -v_i^T d > 0 \end{cases} \iff s_i v_i^T d > 0$$

slice $s$ non-empty $\iff d_s \in$ slice $s$ $\iff \forall i \in [1:p], s_i(v_i^T d_s) > 0$
Verifying $p$ linear equations $=$ very simple...

But there are $2^p$ such systems.
Thus the interest of designing non-brute force algorithm.
Technical formalism

There are $p$ cuts, $2^p$ potential slices ($\forall \ i \in [1:p], \{-1,+1\}$)
Slice $s = (s_1, \ldots, s_p) \in \{\pm 1\}^p$ exists $\iff H_{s_1}^+ \cap H_{s_2}^+ \cap \cdots \cap H_{s_p}^+ \neq \emptyset$

\[
\left\{
\begin{array}{c}
H_i^+ : v_i^T d > 0 \iff +v_i^T d > 0 \\
H_i^- : v_i^T d < 0 \iff -v_i^T d > 0
\end{array}\right. 
\iff s_i v_i^T d > 0
\]

slice $s$ non-empty $\iff d_s \in \text{slice } s \iff \forall \ i \in [1:p], s_i (v_i^T d_s) > 0$
Verifying $p$ linear equations $= \text{very simple...}$

But there are $2^p$ such systems.
Thus the interest of designing non-brute force algorithm.
Technical formalism

There are $p$ cuts, $2^p$ potential slices ($\forall i \in [1 : p], \{-1, +1\}$)
Slice $s = (s_1, \ldots, s_p) \in \{\pm 1\}^p$ exists $\iff H_1^{s_1} \cap H_2^{s_2} \cap \cdots \cap H_p^{s_p} \neq \emptyset$

\[
\begin{align*}
H_i^+ : v_i^T d > 0 & \iff +v_i^T d > 0 \\
H_i^- : v_i^T d < 0 & \iff -v_i^T d > 0 \\
\end{align*}
\[
\iff s_i v_i^T d > 0
\]
slice $s$ non-empty $\iff d_s \in$ slice $s$ $\iff \forall i \in [1 : p], s_i (v_i^T d_s) > 0$

Verifying $p$ linear equations $=$ very simple...

But there are $2^p$ such systems.
Thus the interest of designing non-brute force algorithm.
Plan

1. My Personal Recipe
2. First part(s of the cake)
3. Formalism
4. An algorithm
5. Some improvements
Main reasoning

Algorithm from [RČ18]:

• recursive binary tree that adds hyperplanes one at a time
• each node has descendant(s) \((s, +1)\) and/or \((s, -1)\)
• checking one or two = main computational effort
Illustration of the regions and tree on the previous example
Illustration of the regions and tree on the previous example
Illustration of the regions and tree on the previous example
Illustration of the regions and tree on the previous example
Illustration of the regions and tree on the previous example
Illustration of the regions and tree on the previous example
Important property

At level $k < p$, for a slice $s \in \{\pm 1\}^k$,

$$\forall \, i \in [1 : k], \exists \, d_s, s_i v_i^T d_s > 0 \Rightarrow$$

$$\left\{ \begin{array}{l}
\forall \, i \in [1 : k], s_i v_i^T d_s > 0 \\
\quad +v_{k+1}^T d > 0 \\
\forall \, i \in [1 : k], s_i v_i^T d_s > 0 \\
\quad -v_{k+1}^T d > 0
\end{array} \right.$$  

If $v_{k+1}^T d_s > 0$, $(s, +1)$ verified with the same $d_s$ (if $< 0$, $(s, -1)$ is).

If $v_{k+1}^T d_s \simeq 0$, both for free! (formalized properly)
Important property

At level $k < p$, for a slice $s \in \{\pm 1\}^k$,

$$\forall i \in [1 : k], \exists d_s, s_i v_i^T d_s > 0 \Rightarrow \left\{ \begin{array}{l}
\forall i \in [1 : k], s_i v_i^T d > 0 \\
+v_{k+1}^T d > 0 \\
\forall i \in [1 : k], s_i v_i^T d > 0 \\
-v_{k+1}^T d > 0
\end{array} \right. \quad ?$$

If $v_{k+1}^T d_s > 0$, $(s, +1)$ verified with the same $d_s$ (if $< 0$, $(s, -1)$ is).
If $v_{k+1}^T d_s \simeq 0$, both for free! (formalized properly)
Illustration

The point is "very close" to the new hyperplane, a small simple modification suffices.
Plan

1. My Personal Recipe
2. First part(s of the cake)
3. Formalism
4. An algorithm
5. Some improvements
Reducing the node count

So \(|v_{k+1}^Td_s|\) small \(\Rightarrow\) probably 2 descendants.

idea: contrapositive

\(|v_{k+1}^Td_s|\) 'large' \(\rightarrow\) less chance of both \((s, +1)\) and \((s, -1)\).

Only a heuristic, but reasonably efficient.
Also, this order change is local - for each \(s\) it can change.
Reducing the node count

So $|v_{k+1}^T d_s| \text{ small } \Rightarrow \text{ probably 2 descendants.}$

idea: contraposition

$|v_{k+1}^T d_s| \text{ 'large' } \rightarrow \text{ less chance of both } (s, +1) \text{ and } (s, -1).$

Only a heuristic, but reasonably efficient.
Also, this order change is local - for each $s$ it can change.
Black: hyperplanes already treated, $x$ is the current point/region. Dotted and blue: remaining hyperplanes. Here, the blue hyperplanes are "far" from the point, so it's more likely there is only 1 descendant (thus less nodes and a faster algorithm).
++− (and −−+) corresponds to an empty region: + means right to \(H_1\), + over \(H_2\), − down left \(H_3\): such a point does not exist. The system is

\[+ : d_1 > 0, + : d_2 > 0, − : −d_1 − d_2 > 0\]
Infeasibility, matroids and circuits - 2

With \( p > 3 \), ++−? ? . . .? ? always infeasible, whatever the remaining signs are.

Idea

can be formalized through a (technical) recipe theorem

- before the tree, compute every "infeasible" combination
- linear optimization (≃ black-box) → linear algebra (nice!)
- but requires a lot of linear algebra
Infeasibility, matroids and circuits - 2

With $p > 3$, $++-? ? . . . ? ?$ always infeasible, whatever the remaining signs are.

**Idea**

- can be formalized through a (technical) recipe theorem

- before the tree, compute every "infeasible" combination
- linear optimization ($\simeq$ black-box) $\rightarrow$ linear algebra (nice!)
- but requires a lot of linear algebra
Infeasibility, matroids and circuits - 2

With \( p > 3 \), \( ++- \) ? ? \ldots ? ? always infeasible, whatever the remaining signs are.

**Idea**

- before the tree, compute every "infeasible" combination
- linear optimization \((\simeq \text{black-box}) \rightarrow \text{linear algebra} \) (nice!)
- but requires a lot of linear algebra
Infeasibility, matroids and circuits - 2

With $p > 3$, $++\cdots$ always infeasible, whatever the remaining signs are.

Idea

can be formalized through a (technical) recipe theorem

- before the tree, compute every "infeasible" combination
- linear optimization ($\simeq$ black-box) $\rightarrow$ linear algebra (nice!)
- but requires a lot of linear algebra
Summary

- The RC algorithm
- some improvements on the tree structure
- some improvements with duality (the linear algebra)
- best: using a little bit (using it cleverly)
Summary

- The RC algorithm
- some improvements on the tree structure
- some improvements with duality (the linear algebra)
- best: using a little bit (using it cleverly)
Summary

- The RC algorithm
- some improvements on the tree structure
- some improvements with duality (the linear algebra)
  - best : using a little bit (using it cleverly)
Summary

- The RC algorithm
- some improvements on the tree structure
- some improvements with duality (the linear algebra)
- best : using a little bit (using it cleverly)
Results; blue = times, black = time RC / time variant

<table>
<thead>
<tr>
<th>Name</th>
<th>RC</th>
<th>ABC</th>
<th>ABCD2</th>
<th>ABCD3</th>
<th>AD4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-4-8-2</td>
<td>1.70 10^{-2}</td>
<td>7.20 10^{-3}</td>
<td>2.36</td>
<td>6.53 10^{-3}</td>
<td>2.60</td>
</tr>
<tr>
<td>R-7-8-4</td>
<td>5.70 10^{-2}</td>
<td>3.38 10^{-2}</td>
<td>1.69</td>
<td>3.15 10^{-2}</td>
<td>1.81</td>
</tr>
<tr>
<td>R-7-9-4</td>
<td>9.97 10^{-2}</td>
<td>4.98 10^{-2}</td>
<td>2.00</td>
<td>4.96 10^{-2}</td>
<td>2.01</td>
</tr>
<tr>
<td>R-7-10-5</td>
<td>2.33 10^{-1}</td>
<td>1.16 10^{-1}</td>
<td>2.01</td>
<td>1.29 10^{-1}</td>
<td>1.81</td>
</tr>
<tr>
<td>R-7-11-4</td>
<td>2.36 10^{-1}</td>
<td>1.22 10^{-1}</td>
<td>1.93</td>
<td>1.20 10^{-1}</td>
<td>1.97</td>
</tr>
<tr>
<td>R-7-12-6</td>
<td>9.35 10^{-1}</td>
<td>5.05 10^{-1}</td>
<td>1.85</td>
<td>5.74 10^{-1}</td>
<td>1.63</td>
</tr>
<tr>
<td>R-7-13-5</td>
<td>9.11 10^{-1}</td>
<td>4.70 10^{-1}</td>
<td>1.94</td>
<td>5.41 10^{-1}</td>
<td>1.68</td>
</tr>
<tr>
<td>R-7-14-7</td>
<td>3.69</td>
<td>2.15</td>
<td>1.72</td>
<td>2.39</td>
<td>1.54</td>
</tr>
<tr>
<td>R-8-15-7</td>
<td>6.43</td>
<td>3.56</td>
<td>1.81</td>
<td>3.92</td>
<td>1.64</td>
</tr>
<tr>
<td>R-9-16-8</td>
<td>1.51 10^{+1}</td>
<td>8.88</td>
<td>1.70</td>
<td>1.03 10^{+1}</td>
<td>1.47</td>
</tr>
<tr>
<td>R-10-17-9</td>
<td>3.45 10^{+1}</td>
<td>2.08 10^{+1}</td>
<td>1.66</td>
<td>2.50 10^{+1}</td>
<td>1.38</td>
</tr>
<tr>
<td>2d-20-4</td>
<td>3.48 10^{-1}</td>
<td>1.76 10^{-1}</td>
<td>1.98</td>
<td>8.03 10^{-2}</td>
<td>4.33</td>
</tr>
<tr>
<td>2d-20-5</td>
<td>6.74 10^{-1}</td>
<td>3.54 10^{-1}</td>
<td>1.90</td>
<td>1.29 10^{-1}</td>
<td>5.22</td>
</tr>
<tr>
<td>2d-20-6</td>
<td>1.19</td>
<td>6.04 10^{-1}</td>
<td>1.97</td>
<td>2.23 10^{-1}</td>
<td>5.34</td>
</tr>
<tr>
<td>2d-20-7</td>
<td>2.08</td>
<td>1.45</td>
<td>1.43</td>
<td>5.40 10^{-1}</td>
<td>3.85</td>
</tr>
<tr>
<td>2d-20-8</td>
<td>3.69</td>
<td>1.85</td>
<td>1.99</td>
<td>6.36 10^{-1}</td>
<td>5.80</td>
</tr>
<tr>
<td>sR-2</td>
<td>1.71 10^{+1}</td>
<td>4.26</td>
<td>4.01</td>
<td>3.11</td>
<td>5.50</td>
</tr>
<tr>
<td>sR-4</td>
<td>8.03 10^{+1}</td>
<td>3.68 10^{+1}</td>
<td>2.18</td>
<td>4.40 10^{+1}</td>
<td>1.83</td>
</tr>
<tr>
<td>sR-6</td>
<td>1.08 10^{+2}</td>
<td>1.54 10^{+2}</td>
<td>0.70</td>
<td>7.01 10^{+1}</td>
<td>1.54</td>
</tr>
<tr>
<td>perm-5</td>
<td>6.64 10^{-1}</td>
<td>1.89 10^{-1}</td>
<td>3.51</td>
<td>6.87 10^{-2}</td>
<td>9.67</td>
</tr>
<tr>
<td>perm-6</td>
<td>5.80</td>
<td>1.32</td>
<td>4.39</td>
<td>5.19 10^{-1}</td>
<td>11.18</td>
</tr>
<tr>
<td>perm-7</td>
<td>5.70 10^{+1}</td>
<td>1.10 10^{+1}</td>
<td>5.18</td>
<td>4.16</td>
<td>13.70</td>
</tr>
<tr>
<td>perm-8</td>
<td>5.98 10^{+2}</td>
<td>1.08 10^{+2}</td>
<td>5.54</td>
<td>4.41 10^{+1}</td>
<td>13.56</td>
</tr>
<tr>
<td>r-3-7</td>
<td>5.83 10^{-1}</td>
<td>3.16 10^{-1}</td>
<td>1.84</td>
<td>2.79 10^{-1}</td>
<td>2.09</td>
</tr>
<tr>
<td>r-3-9</td>
<td>3.31 10^{-1}</td>
<td>2.92 10^{-1}</td>
<td>1.13</td>
<td>1.96 10^{-1}</td>
<td>1.69</td>
</tr>
<tr>
<td>r-4-7</td>
<td>3.13</td>
<td>1.62</td>
<td>1.93</td>
<td>1.37</td>
<td>2.28</td>
</tr>
<tr>
<td>r-4-9</td>
<td>2.76</td>
<td>1.36</td>
<td>2.03</td>
<td>1.13</td>
<td>2.44</td>
</tr>
<tr>
<td>r-5-7</td>
<td>8.92</td>
<td>4.72</td>
<td>1.89</td>
<td>3.94</td>
<td>2.26</td>
</tr>
<tr>
<td>r-5-9</td>
<td>9.02</td>
<td>4.47</td>
<td>2.02</td>
<td>3.72</td>
<td>2.42</td>
</tr>
<tr>
<td>r-6-7</td>
<td>2.18 10^{+1}</td>
<td>1.20 10^{+1}</td>
<td>1.82</td>
<td>1.14 10^{+1}</td>
<td>1.91</td>
</tr>
<tr>
<td>r-6-9</td>
<td>2.63 10^{+1}</td>
<td>1.45 10^{+1}</td>
<td>1.81</td>
<td>1.17 10^{+1}</td>
<td>2.25</td>
</tr>
<tr>
<td>r-7-7</td>
<td>5.72 10^{+1}</td>
<td>3.30 10^{+1}</td>
<td>1.73</td>
<td>3.49 10^{+1}</td>
<td>1.64</td>
</tr>
<tr>
<td>r-7-9</td>
<td>4.68 10^{+1}</td>
<td>2.58 10^{+1}</td>
<td>1.81</td>
<td>2.45 10^{+1}</td>
<td>1.91</td>
</tr>
</tbody>
</table>

median/mean | 1.93/2.23 | 2.05/3.70 | 1.93/2.48 | 1.52/1.32 |
Conclusion

- Better improvement ratios on "structured" instances
- "real-world" instances are "structured" (so good ratios!)
- next steps: articles, code details, convincing advisors of why/how it works (, writing the thesis......................)

Thanks for your attention! Some questions?
Conclusion

- Better improvement ratios on "structured" instances
- "real-world" instances are "structured" (so good ratios!)
- next steps: articles, code details, convincing advisors of why/how it works (, writing the thesis......................)

Thanks for your attention! Some questions?
Conclusion

- Better improvement ratios on "structured" instances
- "real-world" instances are "structured" (so good ratios!)
- next steps: articles, code details, convincing advisors of why/how it works (writing the thesis)

Thanks for your attention! Some questions?
Conclusion

- Better improvement ratios on "structured" instances
- "real-world" instances are "structured" (so good ratios!)
- next steps: articles, code details, convincing advisors of why/how it works (writing the thesis........................)

Thanks for your attention! Some questions?
Theoretical detour

Very well-known in algebra / combinatorics...
... but very theoretically: Möbius function, lattices, matroids.

Very impressive results / algorithms for the cardinal (number of feasible systems, number of $J \in \partial_B$)
Upper bound, formula (also combinatorial)...
Theoretical detour

Very well-known in algebra / combinatorics...
... but very theoretically: Möbius function, lattices, matroids.

Very impressive results / algorithms for the cardinal (number of feasible systems, number of $J \in \partial_B$)
Upper bound, formula (also combinatorial)...

Method - adding vectors one at a time

With one more vector

- Given \((v_1, \ldots, v_{k-1}); \ v_k; \ S_{k-1} \subseteq \{\pm 1\}^{k-1}\)
Method - adding vectors one at a time

With one more vector

- Given \((v_1, \ldots, v_{k-1}); v_k; S_{k-1} \subseteq \{\pm 1\}^{k-1}\)
- \(\forall s = (s_1, \ldots, s_{k-1}) \in S_{k-1},\) we know \(d_{s}^{k-1}\) s.t. :
  - \(\forall i \in [1 : k - 1], s_i v_i^T d_{s}^{k-1} > 0\)
Method - adding vectors one at a time

With one more vector

- **Given** \((v_1, \ldots, v_{k-1}); v_k; S_{k-1} \subseteq \{\pm 1\}^{k-1}\)
- \(\forall s = (s_1, \ldots, s_{k-1}) \in S_{k-1}, \) we know \(d_s^{k-1}\) s.t. :
  \(\forall i \in [1 : k - 1], \ s_i v_i^T d_s^{k-1} > 0\)
- \(v_k^T d_s^{k-1} > 0 \Rightarrow \left\{ \begin{array}{l} +v_k^T d_s^{k-1} > 0 \quad \checkmark \quad \left\{ \begin{array}{l} -v_k^T d > 0 \quad ? \rightarrow \text{L.O.} \\ s_i v_i^T d > 0 \end{array} \right. \\ \end{array}\right.\)
Method - adding vectors one at a time

With one more vector

- Given \((v_1, \ldots, v_{k-1}); v_k; S_{k-1} \subseteq \{\pm 1\}^{k-1}\)
- \(\forall s = (s_1, \ldots, s_{k-1}) \in S_{k-1}, \text{ we know } d_s^{k-1}\) s.t. :
  - \(\forall i \in [1 : k - 1], s_i v_i^T d_s^{k-1} > 0\)
- \(v_k^T d_s^{k-1} > 0 \Rightarrow \begin{cases} +v_k^T d_s^{k-1} > 0 \land \sqrt{s_i v_i^T d_s^{k-1} > 0} \land -v_k^T d > 0 \Rightarrow \text{L.O.} \end{cases}\)
- \(v_k^T d_s^{k-1} < 0 \Rightarrow \begin{cases} -v_k^T d_s^{k-1} > 0 \land \sqrt{s_i v_i^T d_s^{k-1} > 0} \land +v_k^T d > 0 \Rightarrow \text{L.O.} \end{cases}\)
Method - adding vectors one at a time

With one more vector

- Given \((v_1, \ldots, v_{k-1}); v_k \in S_{k-1} \subseteq \{\pm 1\}^{k-1}\)
- \(\forall s = (s_1, \ldots, s_{k-1}) \in S_{k-1}, \text{ we know } d_s^{k-1}\) s.t. :
  \(\forall i \in [1:k-1], s_i v_i^T d_s^{k-1} > 0\)
- \(v_k^T d_s^{k-1} > 0 \Rightarrow \left\{ \begin{array}{l} +v_k^T d_s^{k-1} > 0 \quad \checkmark, \quad \left\{ \begin{array}{l} -v_k^T d > 0 \quad ? \rightarrow \text{L.O.} \\
 s_i v_i^T d_s^{k-1} > 0 \end{array} \right. \right. \right. \right.
- \(v_k^T d_s^{k-1} < 0 \Rightarrow \left\{ \begin{array}{l} -v_k^T d_s^{k-1} > 0 \quad \checkmark, \quad \left\{ \begin{array}{l} +v_k^T d > 0 \quad ? \rightarrow \text{L.O.} \\
 s_i v_i^T d_s^{k-1} > 0 \end{array} \right. \right. \right. \right.
- \(v_k^T d_s^{k-1} = 0 \Rightarrow \text{both systems } \checkmark \text{ by perturbation}\)
Circuits of matroids

We look at subsets $I \subset [1 : p]$, $\dim(\mathcal{N}(V; I)) = 1$ and $\forall I' \subsetneq I$, $\dim(\mathcal{N}(V; I')) = 0$

$$\dim(\mathcal{N}(V; I)) = 1 \Rightarrow \mathcal{N}(V; I) = \text{Vect}(\eta)$$

$$\Rightarrow V; I \eta = 0 \iff \underbrace{V; I \text{sign}(\eta)}_{V; I S(I)} \underbrace{\text{sign}(\eta) \eta = 0}_{= \gamma(I) \geq 0}$$

$\mathcal{N}(V; I)$ gives ’unsigned’ $\eta$’s which define the sign $s_J = 1$ because if $\geq 2$, smaller subsets are of $\dim(\mathcal{N}) = 1$

$2^p$ LO feasibility $\leftrightarrow 2^p \mathcal{N}$ searches; subsets of size $\leq 1 + \text{rank}(V)$

Issue (unresolved): ”optimal” way to compute efficiently: if $I$ s.t. $\dim(\mathcal{N}(V; I)) = 1$, $I' \supsetneq I$ useless to check
Circuits of matroids

We look at subsets $I \subset [1 : p]$, $\dim(N(V;I)) = 1$ and $\forall I' \subset I$, $\dim(N(V;I')) = 0$

$$\dim(N(V;I)) = 1 \Rightarrow N(V;I) = \text{Vect}(\eta)$$

$$\Rightarrow V;I\eta = 0 \iff V;I\text{sign}(\eta)\text{sign}(\eta)\eta = 0$$

$N(V;I)$ gives 'unsigned' $\eta$'s which define the sign $s_J = 1$ because if $\geq 2$, smaller subsets are of $\dim(N) = 1$

$2^p$ LO feasibility $\Leftrightarrow 2^p N$ searches; subsets of size $\leq 1 + \text{rank}(V)$

Issue (unresolved): "optimal" way to compute efficiently: if $I$ s.t. $\dim(N(V;I)) = 1$, $I' \supseteq I$ useless to check
Circuits of matroids

We look at subsets \( I \subset [1 : p] \), \( \dim(\mathcal{N}(V;,I)) = 1 \)
and \( \forall \ I' \subsetneq I \), \( \dim(\mathcal{N}(V;,I')) = 0 \)
\[
\dim(\mathcal{N}(V;,I)) = 1 \Rightarrow \mathcal{N}(V;,I) = \text{Vect}(\eta)
\Rightarrow V;,I\eta = 0 \iff V;,I\text{sign}(\eta)\text{sign}(\eta)\eta = 0
\]
\[
V;,I S(I) = \gamma(I) \geq 0
\]

\( \mathcal{N}(V;,I) \) gives 'unsigned' \( \eta \)'s which define the sign \( s_I = 1 \) because if \( \geq 2 \), smaller subsets are of \( \dim(\mathcal{N}) = 1 \)

\( 2^p \) LO feasibility \( \iff 2^p \mathcal{N} \) searches; subsets of size \( \leq 1 + \text{rank}(V) \)

Issue (unresolved): "optimal" way to compute efficiently: if \( I \) s.t. \( \dim(\mathcal{N}(V;,I)) = 1 \), \( I' \supseteq I \) useless to check
Circuits of matroids

We look at subsets $I \subset [1 : p]$, $\dim(\mathcal{N}(V; I)) = 1$
and $\forall I' \subsetneq I$, $\dim(\mathcal{N}(V; I')) = 0$

$$\dim(\mathcal{N}(V; I)) = 1 \Rightarrow \mathcal{N}(V; I) = \text{Vect}(\eta)$$

$$\Rightarrow V; I\eta = 0 \iff V; I\text{sign}(\eta)\text{sign}(\eta)\eta = 0$$

$$\mathcal{N}(V; I)$$ gives 'unsigned' $\eta$'s which define the sign $s_J = 1$ because

if $\geq 2$, smaller subsets are of $\dim(\mathcal{N}) = 1$

$2^p$ LO feasibility $\leftrightarrow 2^p \mathcal{N}$ searches; subsets of size $\leq 1 + \text{rank}(V)$

Issue (unresolved): "optimal" way to compute efficiently: if $I$ s.t. $\dim(\mathcal{N}(V; I)) = 1$, $I' \supsetneq I$ useless to check
Circuits of matroids

We look at subsets $I \subset [1 : p]$, $\dim(N(V;_I)) = 1$ and $\forall I' \subsetneq I$, $\dim(N(V;_{I'})) = 0$

$$\dim(N(V;_I)) = 1 \Rightarrow N(V;_I) = \text{Vect}(\eta)$$

$$\Rightarrow V;_I \eta = 0 \iff V;_I \text{sign}(\eta) \text{sign}(\eta) \eta = 0$$

$N(V;_I)$ gives 'unsigned' $\eta$'s which define the sign $s_J = 1$ because if $\geq 2$, smaller subsets are of $\dim(N) = 1$

$2^p$ LO feasibility $\leftrightarrow 2^p N$ searches; subsets of size $\leq 1 + \text{rank}(V)$

Issue (unresolved): "optimal" way to compute efficiently: if $I$ s.t. $\dim(N(V;_I)) = 1$, $I' \supsetneq I$ useless to check
Circuits of matroids

We look at subsets $I \subset [1 : p]$, $\dim(\mathcal{N}(V_{\cdot I})) = 1$ and $\forall I' \subsetneq I$, $\dim(\mathcal{N}(V_{\cdot I'})) = 0$

\[
\dim(\mathcal{N}(V_{\cdot I})) = 1 \implies \mathcal{N}(V_{\cdot I}) = \text{Vect}(\eta)
\]

$\Rightarrow V_{\cdot I} \eta = 0 \iff \begin{array}{c}
V_{\cdot I} \text{sign}(\eta) \text{sign}(\eta) \eta = 0
\end{array}$

$\mathcal{N}(V_{\cdot I})$ gives 'unsigned' $\eta$'s which define the sign $s_J = 1$ because if $\geq 2$, smaller subsets are of $\dim(\mathcal{N}) = 1$

$2^p$ LO feasibility $\iff 2^p \mathcal{N}$ searches; subsets of size $\leq 1 + \text{rank}(V)$

Issue (unresolved): "optimal" way to compute efficiently: if $I$ s.t. $\dim(\mathcal{N}(V_{\cdot I})) = 1$, $I' \supsetneq I$ useless to check