PhD Pizza Seminar

#1

09/01/2024

Learn

Eat

Make Friends ♡

PhD Pizza Seminar

#1

09/01/2024

presents

Baptiste Plaquevent-Jourdain
SERENA

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

Plan

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



Who are you listening to? (2)

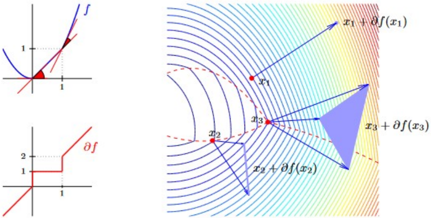
current status

- 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)

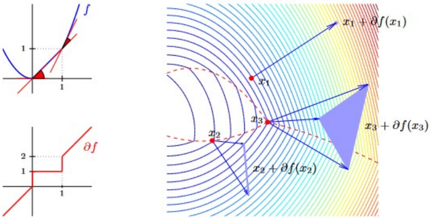
My (current) subject

... 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

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

cut:= 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

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).

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 cuts, $2p$ slices

'Proof': every cut makes 2 previous slices becoming 4 smaller slices
$2p \to (2p-2)+2*2=(2p-2)+4=2(p+1)$.

A first taste - 2

Three cuts, 6 slices

us around the pizzas

p cuts, $2p$ slices

'Proof': every cut makes 2 previous slices becoming 4 smaller slices
$2p \to (2p-2)+2*2=(2p-2)+4=2(p+1)$.

A first taste - 2

Three cuts, 6 slices

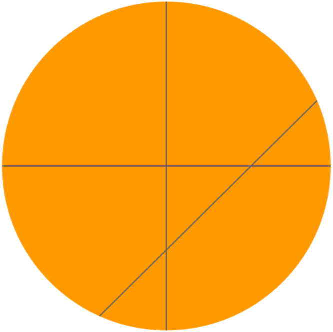
us around the pizzas

$$p \text{ cuts}, 2p \text{ slices}$$

'Proof': every cut makes 2 previous slices becoming 4 smaller slices
$2p \rightarrow (2p - 2) + 2 * 2 = (2p - 2) + 4 = 2(p + 1)$.

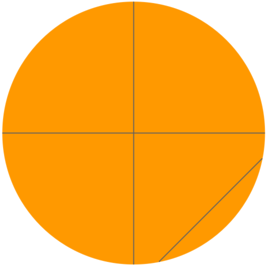
Other possibilities - 1

What about 7 parts ?



Asymmetric cuts - they don't all pass by the center/middle

Other possibilities - 2



Acually 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...

Other possibilities - 3

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

Hyperplanes - 1

The cake n-dimensional, a 'cut' is an hyperplane.
$=$ linear (affine) subspace of dimension $n-1$ (codimension 1).
One hyperplane: $H = v^{\perp} = \{d \in \mathbb{R}^n : v^\top d = 0\}$.

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

halfspaces of an hyperplane

$$\mathbb{R}^n = H_i^- \cup H_i \cup H_i^+, \qquad \begin{array}{l} H_i^- = \{d \in \mathbb{R}^n : v_i^\top d < 0\} \\ H_i^+ = \{d \in \mathbb{R}^n : v_i^\top d > 0\} \end{array}$$

Hyperplanes - 1

The cake n-dimensional, a 'cut' is an hyperplane.
= linear (affine) subspace of dimension $n-1$ (codimension 1).
One hyperplane: $H = v^\perp = \{d \in \mathbb{R}^n : v^\top d = 0\}$.

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

halfspaces of an hyperplane

$$\mathbb{R}^n = H_i^- \cup H_i \cup H_i^+, \qquad \begin{matrix} H_i^- = \{d \in \mathbb{R}^n : v_i^\top d < 0\} \\ H_i^+ = \{d \in \mathbb{R}^n : v_i^\top d > 0\} \end{matrix}$$

Hyperplanes - 1

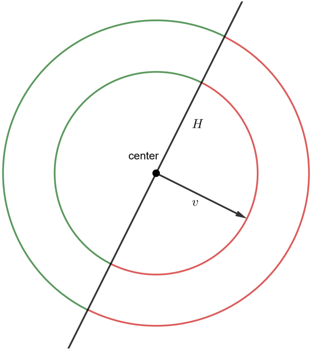
The cake n-dimensional, a 'cut' is an hyperplane.
$=$ linear (affine) subspace of dimension $n-1$ (codimension 1).
One hyperplane: $H = v^{\perp} = \{d \in \mathbb{R}^n : v^\top d = 0\}$.

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

halfspaces of an hyperplane

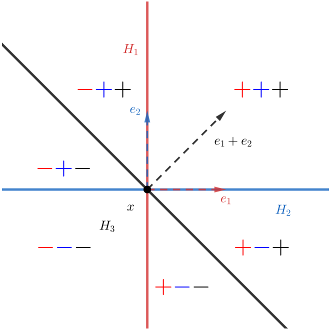
$$\mathbb{R}^n = H_i^- \cup H_i \cup H_i^+, \qquad \begin{aligned} H_i^- &= \{d \in \mathbb{R}^n : v_i^\top d < 0\} \\ H_i^+ &= \{d \in \mathbb{R}^n : v_i^\top d > 0\} \end{aligned}$$

Hyperplanes - 2



Each cut: a − and a + side: each of the p cuts, intersection of each halfspaces...

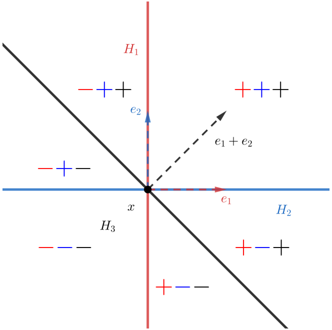
Illustration



$H_1 = e_1^\perp, H_2 = e_2^\perp, H_3 = (e_1 + e_2)^\perp.$

Actually, # of slices and on which side of each cut it is.

Illustration



$$H_1 = e_1^\perp, H_2 = e_2^\perp, H_3 = (e_1 + e_2)^\perp.$$

Actually, # of slices and on which side of each cut it is.

Technical formalism

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

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

slice s non-empty $\Leftrightarrow d_s \in$ slice $s \Leftrightarrow \forall\ i \in [1:p], s_i(v_i^\top d_s) > 0$
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 $s = (s_1, \dots, s_p) \in \{\pm 1\}^p$ exists $\Leftrightarrow H_1^{s_1} \cap H_2^{s_2} \cap \dots \cap H_p^{s_p} \neq \varnothing$

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

slice s non-empty $\Leftrightarrow d_s \in$ slice $s \Leftrightarrow \forall\ i \in [1:p], s_i(v_i^\mathsf{T} d_s) > 0$
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 $\Leftrightarrow H_1^{s_1} \cap H_2^{s_2} \cap \cdots \cap H_p^{s_p} \neq \varnothing$

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

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

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

Plan

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

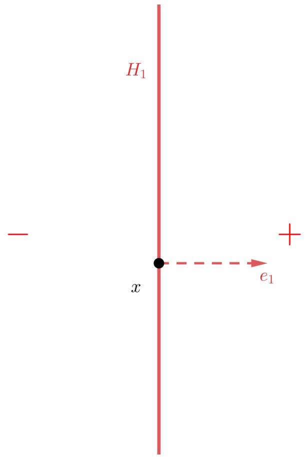


Illustration of the regions and tree on the previous example

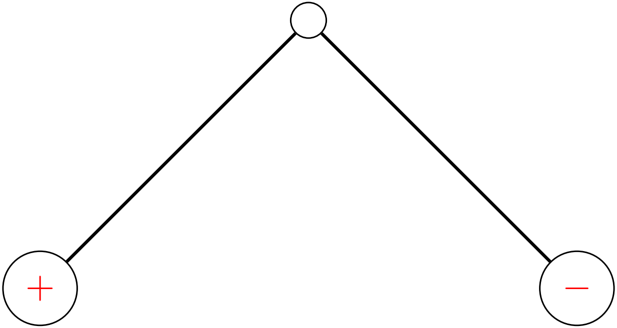


Illustration of the regions and tree on the previous example

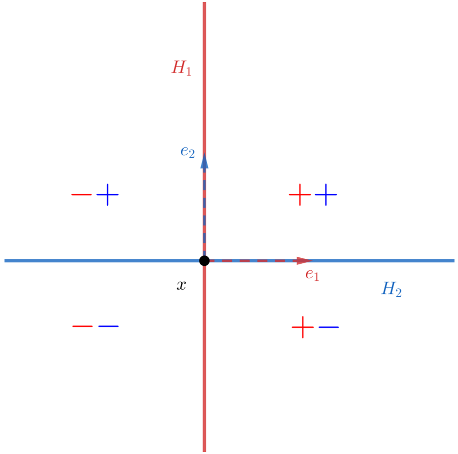


Illustration of the regions and tree on the previous example

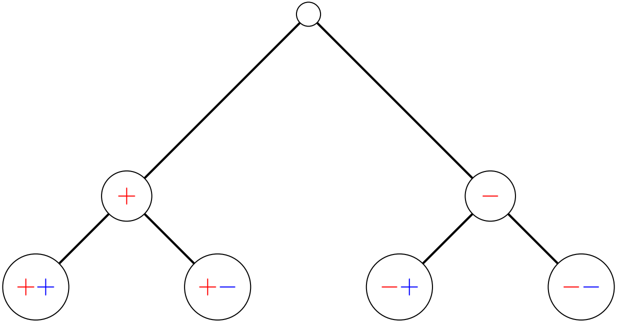


Illustration of the regions and tree on the previous example

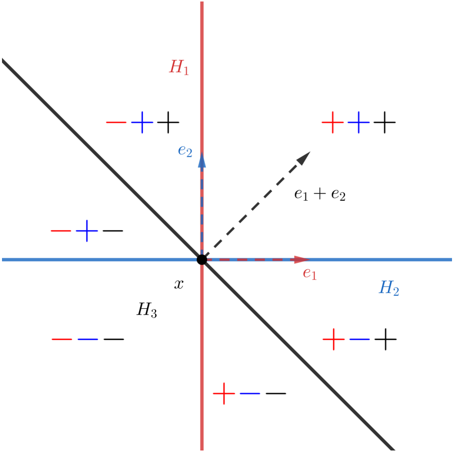
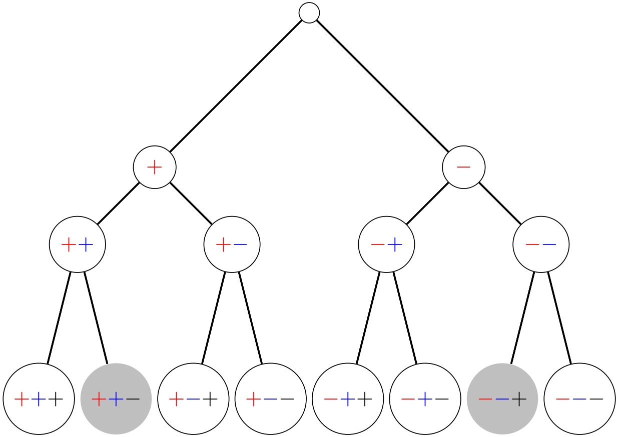


Illustration of the regions and tree on the previous example



Important property

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

$$\forall\, i \in [1:k], \exists\, d_s, s_i v_i^\top d_s > 0 \Rightarrow \begin{cases} \forall\, i \in [1:k], s_i v_i^\top d > 0 \\ \qquad +v_{k+1}^\top d > 0 \\ \forall\, i \in [1:k], s_i v_i^\top d > 0 \\ \qquad -v_{k+1}^\top d > 0 \end{cases} \quad \begin{matrix} ? \\ ? \end{matrix}$$

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

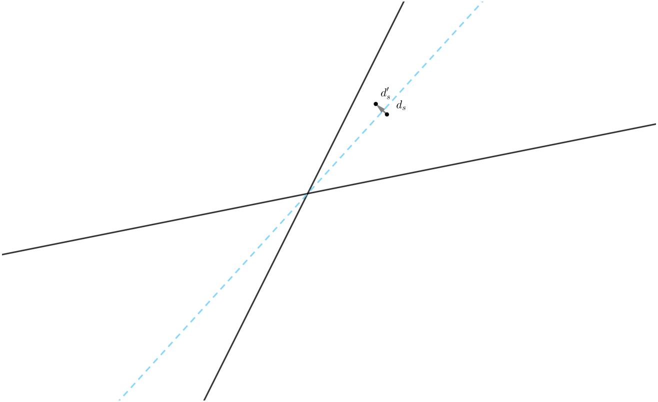
Important property

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

$$\forall\ i \in [1:k], \exists\ d_s, s_i v_i^\top d_s > 0 \Rightarrow \begin{cases} \forall\ i \in [1:k], s_i v_i^\top d > 0 \\ \qquad +v_{k+1}^\top d > 0 & ? \\ \forall\ i \in [1:k], s_i v_i^\top d > 0 \\ \qquad -v_{k+1}^\top d > 0 & ? \end{cases}$$

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

Illustration



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

Plan

Reducing the node count

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

idea: contrapositive

$|v_{k+1}^T d_s|$ '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

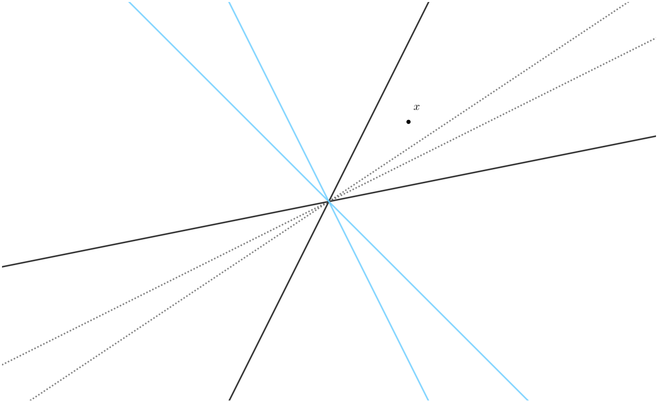
So $|v_{k+1}^{\mathsf{T}} d_s|$ small $\Rightarrow$ probably 2 descendants.

idea: contrapositive

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

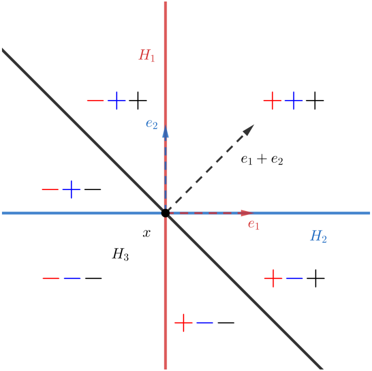
Only a heuristic, but reasonably efficient.
Also, this order change is local - for each s it can change.

Illustration



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).

Infeasibility, matroids and circuits - 1



$++-$ (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 $p > 3$, $++-??\dots??$ 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

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

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 <u>lot</u> of linear algebra

Infeasibility, matroids and circuits - 2

With $p > 3$, $++-??\ldots??$ 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

Name	RC	ABC		ABCD2		ABCD3		AD4	
R-4-8-2	1.70 10^{-2}	7.20 10^{-3}	2.36	6.53 10^{-3}	2.60	3.13 10^{-3}	**5.43**	8.03 10^{-3}	2.12
R-7-8-4	5.70 10^{-2}	3.38 10^{-2}	1.69	3.15 10^{-2}	1.81	2.24 10^{-2}	**2.54**	2.79 10^{-2}	2.04
R-7-9-4	9.97 10^{-2}	4.98 10^{-2}	2.00	4.96 10^{-2}	2.01	3.43 10^{-2}	**2.91**	5.16 10^{-2}	1.93
R-7-10-5	2.33 10^{-1}	1.16 10^{-1}	2.01	1.29 10^{-1}	1.81	1.05 10^{-1}	**2.22**	1.22 10^{-1}	1.91
R-7-11-4	2.36 10^{-1}	1.22 10^{-1}	1.93	1.20 10^{-1}	1.97	8.49 10^{-2}	**2.78**	1.32 10^{-1}	1.79
R-7-12-6	9.35 10^{-1}	5.05 10^{-1}	**1.85**	5.74 10^{-1}	1.63	5.13 10^{-1}	1.82	5.65 10^{-1}	1.65
R-7-13-5	9.11 10^{-1}	4.70 10^{-1}	**1.94**	5.41 10^{-1}	1.68	4.71 10^{-1}	1.93	5.33 10^{-1}	1.71
R-7-14-7	3.69	2.15	**1.72**	2.39	1.54	2.42	1.52	2.42	1.52
R-8-15-7	6.43	3.56	**1.81**	3.92	1.64	4.30	1.50	4.57	1.41
R-9-16-8	1.51 10^{+1}	8.88	**1.70**	1.03 10^{+1}	1.47	1.34 10^{+1}	1.13	1.41 10^{+1}	1.07
R-10-17-9	3.45 10^{+1}	2.08 10^{+1}	**1.66**	2.50 10^{+1}	1.38	4.04 10^{+1}	0.85	3.53 10^{+1}	0.98
2d-20-4	3.48 10^{-1}	1.76 10^{-1}	1.98	8.03 10^{-2}	4.33	6.96 10^{-2}	**5.00**	1.73 10^{-1}	2.01
2d-20-5	6.74 10^{-1}	3.54 10^{-1}	1.90	1.29 10^{-1}	**5.22**	1.32 10^{-1}	5.11	3.59 10^{-1}	1.88
2d-20-6	1.19	6.04 10^{-1}	1.97	2.23 10^{-1}	**5.34**	2.70 10^{-1}	4.41	6.52 10^{-1}	1.83
2d-20-7	2.08	1.45	1.43	5.40 10^{-1}	**3.85**	6.21 10^{-1}	3.35	1.11	1.87
2d-20-8	3.69	1.85	1.99	6.36 10^{-1}	**5.80**	7.95 10^{-1}	4.64	1.92	1.92
sR-2	1.71 10^{+1}	4.26	4.01	3.11	**5.50**	4.14	4.13	1.05 10^{+1}	1.63
sR-4	8.03 10^{+1}	3.68 10^{+1}	2.18	4.40 10^{+1}	**1.83**	1.41 10^{+2}	0.57	2.02 10^{+2}	0.40
sR-6	1.08 10^{+2}	1.54 10^{+2}	0.70	7.01 10^{+1}	**1.54**	2.58 10^{+2}	0.42	4.04 10^{+2}	0.27
perm-5	6.64 10^{-1}	1.89 10^{-1}	3.51	6.87 10^{-2}	**9.67**	8.53 10^{-2}	7.78	3.75 10^{-1}	1.77
perm-6	5.80	1.32	4.39	5.19 10^{-1}	**11.18**	1.03	5.63	3.81	1.52
perm-7	5.70 10^{+1}	1.10 10^{+1}	5.18	4.16	**13.70**	2.12 10^{+1}	2.69	6.37 10^{+1}	0.89
perm-8	5.98 10^{+2}	1.08 10^{+2}	5.54	4.41 10^{+1}	**13.56**	6.46 10^{+2}	0.93	1.59 10^{+3}	0.38
r-3-7	5.83 10^{-1}	3.16 10^{-1}	1.84	2.79 10^{-1}	2.09	2.27 10^{-1}	**2.57**	3.64 10^{-1}	1.60
r-3-9	3.31 10^{-1}	2.92 10^{-1}	1.13	1.96 10^{-1}	1.69	1.41 10^{-1}	**2.35**	1.77 10^{-1}	1.87
r-4-7	3.13	1.62	1.93	1.37	**2.28**	2.21	1.42	3.01	1.04
r-4-9	2.76	1.36	2.03	1.13	**2.44**	1.85	1.49	2.87	0.96
r-5-7	8.92	4.72	1.89	3.94	**2.26**	8.64	1.03	1.26 10^{+1}	0.71
r-5-9	9.02	4.47	2.02	3.72	**2.42**	7.92	1.14	1.06 10^{+1}	0.85
r-6-7	2.18 10^{+1}	1.20 10^{+1}	1.82	1.14 10^{+1}	**1.91**	2.89 10^{+1}	0.75	4.03 10^{+1}	0.54
r-6-9	2.63 10^{+1}	1.45 10^{+1}	1.81	1.17 10^{+1}	**2.25**	3.39 10^{+1}	0.78	4.89 10^{+1}	0.54
r-7-7	5.72 10^{+1}	3.30 10^{+1}	**1.73**	3.49 10^{+1}	1.64	1.17 10^{+2}	0.49	1.60 10^{+2}	0.36
r-7-9	4.68 10^{+1}	2.58 10^{+1}	1.81	2.45 10^{+1}	**1.91**	7.30 10^{+1}	0.64	8.74 10^{+1}	0.54
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?

Bibliographic elements I

[RČ18] Miroslav Rada and Michal Černý. "A New Algorithm for Enumeration of Cells of Hyperplane Arrangements and a Comparison with Avis and Fukuda's Reverse Search". In: SIAM Journal on Discrete Mathematics 32 (Jan. 2018), pp. 455–473. DOI: 10.1137/15M1027930.

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 ; $\mathcal{S}_{k-1} \subseteq \{\pm 1\}^{k-1}$

Method - adding vectors one at a time

With one more vector

- Given $(v_1, \dots, v_{k-1})$; v_k ; $\mathcal{S}_{k-1} \subseteq \{\pm 1\}^{k-1}$
- $\forall\ s = (s_1, \dots, s_{k-1}) \in \mathcal{S}_{k-1}$, we know d_s^{k-1} s.t. :
  $\forall\ i \in [1 : k-1],\ s_i v_i^\top d_s^{k-1} > 0$

Method - adding vectors one at a time

With one more vector

- Given $(v_1, \ldots, v_{k-1})$; v_k ; $\mathcal{S}_{k-1} \subseteq \{\pm 1\}^{k-1}$
- $\forall\ s = (s_1, \ldots, s_{k-1}) \in \mathcal{S}_{k-1}$, we know d_s^{k-1} s.t. : $\forall\ i \in [1 : k-1],\ s_i v_i^\mathsf{T} d_s^{k-1} > 0$
- $v_k^\mathsf{T} d_s^{k-1} > 0 \Rightarrow \begin{cases} +v_k^\mathsf{T} d_s^{k-1} > 0 \\ s_i v_i^\mathsf{T} d_s^{k-1} > 0 \end{cases} \checkmark, \begin{cases} -v_k^\mathsf{T} d > 0 \\ s_i v_i^\mathsf{T} d > 0 \end{cases} ? \rightarrow$ L.O.

Method - adding vectors one at a time

With one more vector

- Given $(v_1, \dots, v_{k-1})$; v_k ; $\mathcal{S}_{k-1} \subseteq \{\pm 1\}^{k-1}$
- $\forall\ s = (s_1, \dots, s_{k-1}) \in \mathcal{S}_{k-1}$, we know d_s^{k-1} s.t. :
  $\forall\ i \in [1 : k-1],\ s_i v_i^\top d_s^{k-1} > 0$
- $v_k^\top d_s^{k-1} > 0 \Rightarrow \begin{cases} +v_k^\top d_s^{k-1} > 0 \\ s_i v_i^\top d_s^{k-1} > 0 \end{cases} \checkmark, \begin{cases} -v_k^\top d > 0 \\ s_i v_i^\top d > 0 \end{cases} ? \rightarrow$ L.O.
- $v_k^\top d_s^{k-1} < 0 \Rightarrow \begin{cases} -v_k^\top d_s^{k-1} > 0 \\ s_i v_i^\top d_s^{k-1} > 0 \end{cases} \checkmark, \begin{cases} +v_k^\top d > 0 \\ s_i v_i^\top d > 0 \end{cases} ? \rightarrow$ L.O.

Method - adding vectors one at a time

With one more vector

- Given $(v_1, \ldots, v_{k-1})$; v_k ; $\mathcal{S}_{k-1} \subseteq \{\pm 1\}^{k-1}$
- $\forall\ s = (s_1, \ldots, s_{k-1}) \in \mathcal{S}_{k-1}$, we know d_s^{k-1} s.t. :
  $\forall\ i \in [1 : k-1],\ s_i v_i^\top d_s^{k-1} > 0$
- $v_k^\top d_s^{k-1} > 0 \Rightarrow \begin{cases} +v_k^\top d_s^{k-1} > 0 \\ s_i v_i^\top d_s^{k-1} > 0 \end{cases} \checkmark, \begin{cases} -v_k^\top d > 0 \\ s_i v_i^\top d > 0 \end{cases} ? \rightarrow$ L.O.
- $v_k^\top d_s^{k-1} < 0 \Rightarrow \begin{cases} -v_k^\top d_s^{k-1} > 0 \\ s_i v_i^\top d_s^{k-1} > 0 \end{cases} \checkmark, \begin{cases} +v_k^\top d > 0 \\ s_i v_i^\top d > 0 \end{cases} ? \rightarrow$ L.O.
- $v_k^\top d_s^{k-1} = 0 \Rightarrow$ both systems $\checkmark$ by perturbation

Circuits of matroids

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

$$\dim(\mathcal{N}(V_{:,I})) = 1 \Rightarrow \mathcal{N}(V_{:,I}) = \mathrm{Vect}(\eta)$$
$$\Rightarrow V_{:,I}\eta = 0 \Leftrightarrow \underbrace{V_{:,I}\mathrm{sign}(\eta)}_{V_{(:,I)}S_{(I)}}\underbrace{\mathrm{sign}(\eta)\eta}_{=\gamma_{(I)}\geq 0} = 0$$

$\mathcal{N}(V_{:,I})$ gives 'unsigned' η's which define the sign $s_J = 1$ because

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

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

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

$$\dim(\mathcal{N}(V_{:,I})) = 1 \Rightarrow \mathcal{N}(V_{:,I}) = \mathrm{Vect}(\eta)$$
$$\Rightarrow V_{:,I}\eta = 0 \Leftrightarrow \underbrace{V_{:,I}\mathrm{sign}(\eta)}_{V_{(:,I)}S_{(I)}}\underbrace{\mathrm{sign}(\eta)\eta}_{=\gamma_{(I)}\geq 0} = 0$$

$\mathcal{N}(V_{:,I})$ gives 'unsigned' η's which define the sign $s_J = 1$ because

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

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

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

$$\dim(\mathcal{N}(V_{:,I})) = 1 \Rightarrow \mathcal{N}(V_{:,I}) = \mathrm{Vect}(\eta)$$
$$\Rightarrow V_{:,I}\eta = 0 \Leftrightarrow \underbrace{V_{:,I}\mathrm{sign}(\eta)}_{V_{(:,I)}S_{(I)}}\underbrace{\mathrm{sign}(\eta)\eta}_{=\gamma_{(I)}\geq 0} = 0$$

$\mathcal{N}(V_{:,I})$ gives 'unsigned' η's which define the sign $s_J = 1$ because

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

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

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

$$\begin{aligned}\dim(\mathcal{N}(V_{:,I})) = 1 &\Rightarrow \mathcal{N}(V_{:,I}) = \mathrm{Vect}(\eta)\\ &\Rightarrow V_{:,I}\eta = 0 \Leftrightarrow \underbrace{V_{:,I}\mathrm{sign}(\eta)}_{V_{(:,I)}S_{(I)}}\underbrace{\mathrm{sign}(\eta)\eta}_{=\gamma_{(I)}\geq 0} = 0\end{aligned}$$

$\mathcal{N}(V_{:,I})$ gives 'unsigned' η's which define the sign $s_J = 1$ because

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

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

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

$$\dim(\mathcal{N}(V_{:,I})) = 1 \Rightarrow \mathcal{N}(V_{:,I}) = \mathrm{Vect}(\eta)$$
$$\Rightarrow V_{:,I}\eta = 0 \Leftrightarrow \underbrace{V_{:,I}\mathrm{sign}(\eta)}_{V_{(:,I)}S_{(I)}}\underbrace{\mathrm{sign}(\eta)\eta}_{=\gamma_{(I)}\geq 0} = 0$$

$\mathcal{N}(V_{:,I})$ gives 'unsigned' η's which define the sign $s_J = 1$ because

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

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

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

$$\dim(\mathcal{N}(V_{:,I})) = 1 \Rightarrow \mathcal{N}(V_{:,I}) = \mathrm{Vect}(\eta)$$
$$\Rightarrow V_{:,I}\eta = 0 \Leftrightarrow \underbrace{V_{:,I}\mathrm{sign}(\eta)}_{V_{(:,I)}S_{(I)}}\underbrace{\mathrm{sign}(\eta)\eta}_{=\gamma_{(I)}\geq 0} = 0$$

$\mathcal{N}(V_{:,I})$ gives 'unsigned' η's which define the sign $s_J = 1$ because

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

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

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