Integer Programming: Techniques and Applications

Martin Koutecký

TECHNION
Israel Institute
of Technology

Prague, March 27th, 2018

Preview

1. *Intro:* Integer Programming

Preview

1. *Intro:* Integer Programming
2. *IP big picture:* $\overbrace{\text{Fixed dim} \qquad\qquad \text{Variable dim}}$

Preview

1. Intro: Integer Programming

2. IP big picture: Fixed dim $\boxed{\text{Variable dim}}$

$\Downarrow$

3. Zoom: Unifying theory

Preview

1. *Intro:* Integer Programming
2. *IP big picture:* Fixed dim $\boxed{\text{Variable dim}}$
3. *Zoom:* Unifying theory
4. *Applications:* Computational Social Choice (and a few others)

Preview

1. *Intro:* Integer Programming
2. *IP big picture:* Fixed dim $\quad$ $\boxed{\text{Variable dim}}$ $\Downarrow$
3. *Zoom:* Unifying theory
4. *Applications:* Computational Social Choice (and a few others)
5. *Outlook:* Can we make it practical? & other outlook questions

Preview

1. *Intro:* Integer Programming
2. *IP big picture:* Fixed dim | $\boxed{\text{Variable dim}}$ $\Downarrow$
3. *Zoom:* Unifying theory
4. *Applications:* Computational Social Choice (and a few others)
5. *Outlook:* Can we make it practical? & other outlook questions

My goal:

perspective

flavor

Intro: Integer Programming

$$\min \mathbf{w}\mathbf{x} \; : \; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Intro: Integer Programming

$$\min \mathbf{w}\mathbf{x} \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Fundamental for theory $\Leftrightarrow$ Highly successful in practice

Intro: Integer Programming

$$\min \mathbf{wx} \;:\; A\mathbf{x} = \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n$$

Fundamental for theory $\Leftrightarrow$ Highly successful in practice

Intro: Integer Programming

$$\min \mathbf{w}\mathbf{x} \; : \; A\mathbf{x} = \mathbf{b},\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n$$

Fundamental for theory $\Leftrightarrow$ Highly successful in practice

Majority of solver speedup in last 30+ years comes from ***theory****, not* ***hardware****.*
—Bob Bixby, CPLEX & Gurobi founder

Fixed Dimension:
Volume & Flatness

Fixed Dimension: Volume & Flatness

Theorem (Lenstra '83, Kannan, Tardos '87)

ILP solvable in time $n^{\mathcal{O}(n)} \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$*.* n *= dimension,* $\langle \bullet \rangle$ *= encoding length*

Fixed Dimension: Volume & Flatness

Theorem (Lenstra '83, Kannan, Tardos '87)

ILP solvable in time $n^{\mathcal{O}(n)} \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$. *$n$ = dimension, $\langle \bullet \rangle$ = encoding length*

Parameterized complexity perspective:

- Runtime $f(\alpha) \cdot \mathrm{poly}(\beta)$ with
  parameter $\alpha = n$ and
  input $\beta = \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$

$f(\alpha)\,\mathrm{poly}(\beta)$	clearly better than	$\beta^{f(\alpha)}$
FPT (fixed-parameter tractable)		XP

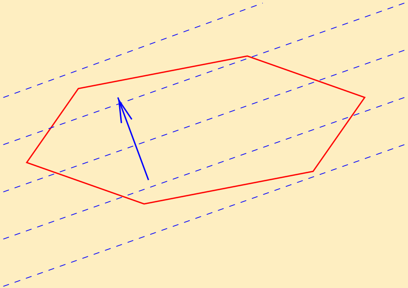
IP has many natural parameters: dimension n, #rows m, largest coefficient $\|A\|_\infty$, treewidth/treedepth of A, etc.

Fixed Dimension: Volume & Flatness

Theorem (Lenstra '83, Kannan, Tardos '87)

ILP solvable in time $n^{\mathcal{O}(n)} \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$. n = *dimension*, $\langle \bullet \rangle$ = *encoding length*

Proof idea.



Focus on feasibility. (optimization follows)

$P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$.

a) Either P has *large volume* $\Rightarrow$ must contain an integer point (*Minkowski I*)

b) Or P has *small volume* $\Rightarrow$ $\exists$ **flatness direction** $\Rightarrow$ cut into few slices & branch!

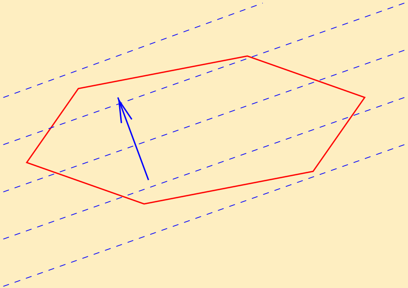
$\square$

Fixed Dimension: Volume & Flatness

Theorem (Lenstra '83, Kannan, Tardos '87)

ILP solvable in time $n^{\mathcal{O}(n)} \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$. *$n$ = dimension, $\langle \bullet \rangle$ = encoding length*

Proof idea.



Focus on feasibility. (optimization follows)
$P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$.

a) Either P has *large volume* $\Rightarrow$ must contain an integer point (*Minkowski I*)

b) Or P has *small volume* $\Rightarrow$ $\exists$ **flatness direction** $\Rightarrow$ cut into few slices & branch!

$\square$

👁 Nothing specific to *linear* IP – same idea works for any convex set P.

Further questions: indefinite objectives, adding quantifiers, etc.

Variable Dimension: Iterative Augmentation

Variable Dimension: Unifying Theory

👁 **Real world is high-dimensional!**

Brief history of variable dimension IP:

- 1960's: Total Unimodularity (paths, matchings, flows) [Hoffman, Kruskal]
- 1980's: ILPs with few rows (generalized knapsack) [Papadimitriou; Eisenbrand, Weismantel]
- 2010–: Iterative methods for block structured programs [Aschenbrenner, Chen, De Loera, Hemmecke, Köppe, Lee, Marx, Onn, Romanchuk, Schulz, Weismantel]
- 2015–: Tree-structured ILPs [Ganian, Jansen, Kratsch, Ordyniak, Ramanujan]

Variable Dimension: Unifying Theory

Real world is high-dimensional!

Brief history of variable dimension IP:

- 1980's: ILPs with few rows (generalized knapsack) [Papadimitriou; Eisenbrand, Weismantel]
- 2010–: Iterative methods for block structured programs [Aschenbrenner, Chen, De Loera, Hemmecke, Köppe, Lee, Marx, Onn, Romanchuk, Schulz, Weismantel]
- 2015–: Tree-structured ILPs [Ganian, Jansen, Kratsch, Ordyniak, Ramanujan]

No strongly polynomial algorithms for these classes (and few overall: TU, bimodular, binet).

Variable Dimension: Unifying Theory

👁 **Real world is high-dimensional!**

Brief history of variable dimension IP:

- 1980's: ILPs with few rows (generalized knapsack) [proximity, DP]
- 2010–: Iterative methods for block structured programs [augmentation, Ramsey, Algebra, DP]
- 2015–: Tree-structured ILPs [Lenstra, treewidth]

No strongly polynomial algorithms for these classes (and few overall: TU, bimodular, binet). Seemingly disconnected classes, different methods.

Variable Dimension: Unifying Theory

⇓

👁 **Real world is high-dimensional!**

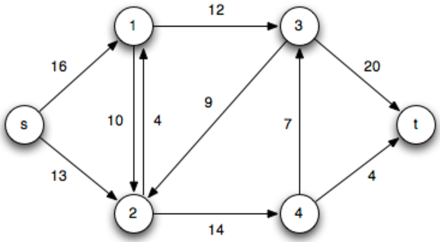
Brief history of variable dimension IP:

- 1980's: ILPs with few rows (generalized knapsack)
- 2010–: Iterative methods for block structured programs
- 2015–: Tree-structured ILPs

No strongly polynomial algorithms for these classes (and few overall: TU, bimodular, binet). Seemingly disconnected classes, different methods.

My best contribution: improve, unify, make strongly polynomial *all* of these results! [K., Levin, Onn '18] + forthcoming book [Hildebrand, Köppe, K.]

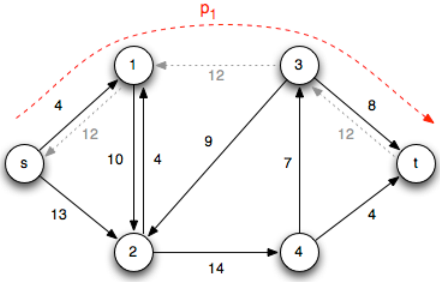
Iterative Augmentation



Max flow

augmenting step $\equiv$ augmenting path p
(because flows decompose into paths)
p *feasible* if enough capacity
p *augmenting* if adds positive (trivial)
flow *optimal* if $\nexists$ augmenting path

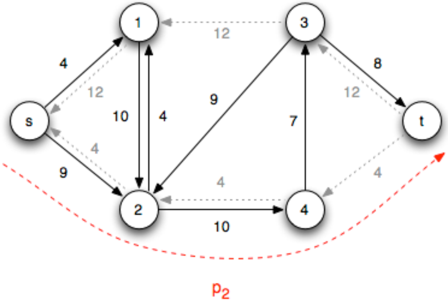
Iterative Augmentation



Max flow

augmenting step $\equiv$ augmenting path p
(because flows decompose into paths)
p *feasible* if enough capacity
p *augmenting* if adds positive (trivial)
flow *optimal* if $\nexists$ augmenting path

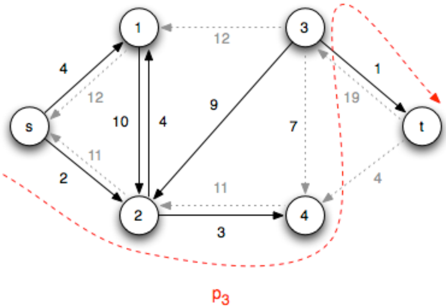
Iterative Augmentation



Max flow

augmenting step $\equiv$ augmenting path p
(because flows decompose into paths)
p *feasible* if enough capacity
p *augmenting* if adds positive (trivial)
flow *optimal* if $\nexists$ augmenting path

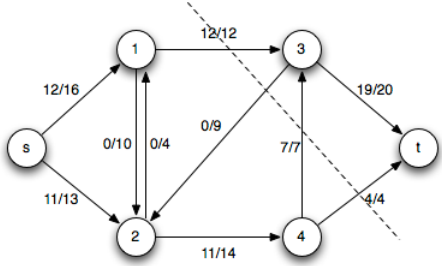
Iterative Augmentation



Max flow

augmenting step $\equiv$ augmenting path p
(because flows decompose into paths)
p *feasible* if enough capacity
p *augmenting* if adds positive (trivial)
flow *optimal* if $\nexists$ augmenting path

Iterative Augmentation



Max flow

augmenting step $\equiv$ augmenting path p
(because flows decompose into paths)
p *feasible* if enough capacity
p *augmenting* if adds positive (trivial)
flow *optimal* if $\nexists$ augmenting path

Iterative Augmentation

$$\min \mathbf{wx} \ : \ A\mathbf{x} = \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n$$

Integer Programming	**Max flow**
$\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A) = \{\mathbf{g} \in \mathbb{Z}^n \mid A\mathbf{g} = \mathbf{0}\}$	augmenting step $\equiv$ augmenting path p
$(A\mathbf{x} = \mathbf{b} \implies A(\mathbf{x}+\mathbf{g}) = \mathbf{b})$	(because flows decompose into paths)
$\mathbf{g}$ *feasible* if $\mathbf{l} \le \mathbf{x}+\mathbf{g} \le \mathbf{u}$	p *feasible* if enough capacity
$\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x}+\mathbf{g}) < \mathbf{wx}$	p *augmenting* if adds positive (trivial)
$\mathbf{x}$ *optimal* if $\nexists$ augmenting $\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A)$	flow *optimal* if $\nexists$ augmenting path

Iterative Augmentation

$$\min \mathbf{wx} \;:\; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

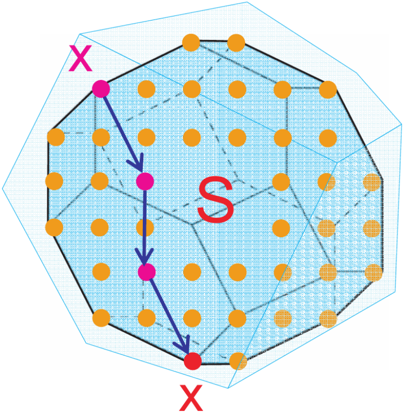
Integer Programming

$\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A) = \{\mathbf{g} \in \mathbb{Z}^n \mid A\mathbf{g} = \mathbf{0}\}$
$(A\mathbf{x} = \mathbf{b} \implies A(\mathbf{x}+\mathbf{g}) = \mathbf{b})$

$\mathbf{g}$ *feasible* if $\mathbf{l} \le \mathbf{x}+\mathbf{g} \le \mathbf{u}$

$\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x}+\mathbf{g}) < \mathbf{wx}$

$\mathbf{x}$ *optimal* if $\nexists$ augmenting $\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A)$



Iterative Augmentation

$$\min \mathbf{wx} \;:\; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Integer Programming

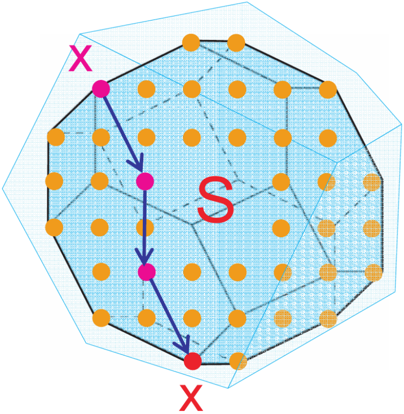
$\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A) = \{\mathbf{g} \in \mathbb{Z}^n \mid A\mathbf{g} = \mathbf{0}\}$
$(A\mathbf{x} = \mathbf{b} \implies A(\mathbf{x}+\mathbf{g}) = \mathbf{b})$

$\mathbf{g}$ *feasible* if $\mathbf{l} \le \mathbf{x}+\mathbf{g} \le \mathbf{u}$

$\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x}+\mathbf{g}) < \mathbf{wx}$

$\mathbf{x}$ *optimal* if $\nexists$ augmenting $\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A)$

BUT $\mathrm{Ker}_{\mathbb{Z}}(A)$ is too big and wild...



Iterative Augmentation

$$\min \mathbf{wx} \;:\; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Integer Programming

$\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A) = \{\mathbf{g} \in \mathbb{Z}^n \mid A\mathbf{g} = \mathbf{0}\}$
$(A\mathbf{x} = \mathbf{b} \implies A(\mathbf{x}+\mathbf{g}) = \mathbf{b})$
$\mathbf{g}$ *feasible* if $\mathbf{l} \le \mathbf{x} + \mathbf{g} \le \mathbf{u}$
$\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x}+\mathbf{g}) < \mathbf{wx}$
$\mathbf{x}$ *optimal* if $\nexists$ augmenting $\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A)$
BUT $\mathrm{Ker}_{\mathbb{Z}}(A)$ is too big and wild...

Goal: Find $\mathcal{T} \subseteq \mathrm{Ker}_{\mathbb{Z}}(A)$, s.t.

1. $\mathbf{x}$ not opt then $\exists$ aug $\mathbf{g} \in \mathcal{T}$
2. good convergence for repeatedly adding "good" $\mathbf{g} \in \mathcal{T}$,
3. algorithmically tame (big is OK)

Iterative Augmentation

$$\min \mathbf{wx} \;:\; A\mathbf{x} = \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n$$

Integer Programming

$\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A) = \{\mathbf{g} \in \mathbb{Z}^n \mid A\mathbf{g} = \mathbf{0}\}$
$(A\mathbf{x} = \mathbf{b} \implies A(\mathbf{x}+\mathbf{g}) = \mathbf{b})$
$\mathbf{g}$ *feasible* if $\mathbf{l} \le \mathbf{x}+\mathbf{g} \le \mathbf{u}$
$\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x}+\mathbf{g}) < \mathbf{wx}$
$\mathbf{x}$ *optimal* if $\nexists$ augmenting $\mathbf{g} \in \mathrm{Ker}_{\mathbb{Z}}(A)$
BUT $\mathrm{Ker}_{\mathbb{Z}}(A)$ is too big and wild...

Goal: Find $\mathcal{T} \subseteq \mathrm{Ker}_{\mathbb{Z}}(A)$, s.t.

1. $\mathbf{x}$ not opt then $\exists$ aug $\mathbf{g} \in \mathcal{T}$
2. good convergence for repeatedly adding "good" $\mathbf{g} \in \mathcal{T}$,
3. algorithmically tame (big is OK)

Answer:

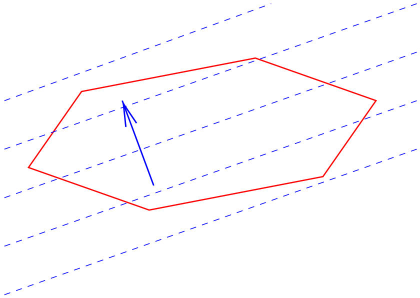
Definition (Graver basis)

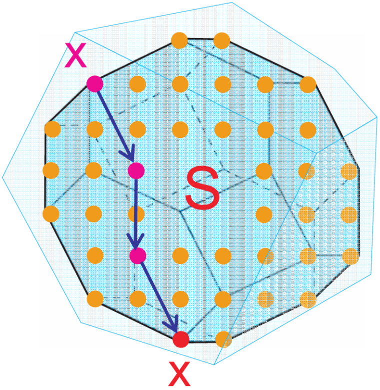
$\mathcal{G}(A) = \{\mathbf{x} \in \mathrm{Ker}_{\mathbb{Z}}(A) \mid \mathbf{x} \text{ is } \sqsubseteq\text{-minimal}\}$

$(\mathbf{x} \sqsubseteq \mathbf{y} \Leftrightarrow \mathbf{x}$ and $\mathbf{y}$ in one orthant $\wedge\ |x_i| \le |y_i|$; $\mathbf{g} \in \mathcal{G}(A) \approx$ "closest to origin")

Iterative Augmentation

$$\min \mathbf{w}\mathbf{x} \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$





Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \;:\; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Definition (Graver-best Step)

A *Graver-best step* for $\mathbf{x}$ is $\mathbf{h}$ s.t. $\mathbf{x}+\mathbf{h}$ is feasible and at least as good as any feasible $\mathbf{x}+\lambda\mathbf{g}$ with $\lambda \in \mathbb{N}$ and $\mathbf{g} \in \mathcal{G}(A)$.

Definition (Graver-best Oracle)

A *Graver-best oracle* for a matrix A is one that queried on $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$ and $\mathbf{x}$, returns a Graver-best step $\mathbf{h}$ for $\mathbf{x}$.

Lemma (Hemmecke, Onn, Weismantel '10)

ILP solvable in $\mathcal{O}(n \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle)$ calls to a Graver-best oracle.

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \;:\; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Definition (Graver-best Step)

A *Graver-best step* for $\mathbf{x}$ is $\mathbf{h}$ s.t. $\mathbf{x} + \mathbf{h}$ is feasible and at least as good as any feasible $\mathbf{x} + \lambda\mathbf{g}$ with $\lambda \in \mathbb{N}$ and $\mathbf{g} \in \mathcal{G}(A)$.

Definition (Graver-best Oracle)

A *Graver-best oracle* for a matrix A is one that queried on $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$ and $\mathbf{x}$, returns a Graver-best step $\mathbf{h}$ for $\mathbf{x}$.

Lemma (Hemmecke, Onn, Weismantel '10)

ILP solvable in $\mathcal{O}(n \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle)$ calls to a Graver-best oracle. :(

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \;:\; A\mathbf{x} = \mathbf{b},\, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Definition (Graver-best Step)

A *Graver-best step* for $\mathbf{x}$ is $\mathbf{h}$ s.t. $\mathbf{x} + \mathbf{h}$ is feasible and at least as good as any feasible $\mathbf{x} + \lambda\mathbf{g}$ with $\lambda \in \mathbb{N}$ and $\mathbf{g} \in \mathcal{G}(A)$.

Definition (Graver-best Oracle)

A *Graver-best oracle* for a matrix A is one that queried on $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$ and $\mathbf{x}$, returns a Graver-best step $\mathbf{h}$ for $\mathbf{x}$.

Lemma (Hemmecke, Onn, Weismantel '10)

ILP solvable in $\mathcal{O}(n \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle)$ calls to a Graver-best oracle. :(

Theorem (K., Levin, Onn '18)

ILP solvable in poly$(n \cdot \langle A \rangle)$ calls to a Graver-best oracle. :)

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Theorem (K., Levin, Onn '18)

ILP solvable in poly$(n \cdot \langle A \rangle)$ calls to a Graver-best oracle.

Proof.

1. Solve LP relaxation in poly$(n \cdot \langle A \rangle)$ time [Tardos '86]

□

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Theorem (K., Levin, Onn '18)

ILP solvable in poly$(n \cdot \langle A \rangle)$ calls to a Graver-best oracle.

Proof.

1. Solve LP relaxation in poly$(n \cdot \langle A \rangle)$ time [Tardos '86]
2. Proximity: integer opt not far from continuous opt $\Rightarrow$ shrink bounds $\mathbf{l}', \mathbf{u}'$, shrink rhs $\mathbf{b}'$.

□

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Theorem (K., Levin, Onn '18)

ILP solvable in poly$(n \cdot \langle A \rangle)$ calls to a Graver-best oracle.

Proof.

1. Solve LP relaxation in poly$(n \cdot \langle A \rangle)$ time [Tardos '86]
2. Proximity: integer opt not far from continuous opt $\Rightarrow$ shrink bounds $\mathbf{l}', \mathbf{u}'$, shrink rhs $\mathbf{b}'$.
3. Reduce objective: $\mathbf{l}', \mathbf{u}'$ give small box $\Rightarrow$ equiv. $\mathbf{w}'$ w/ small $\|\mathbf{w}'\|_\infty$
   [Frank, Tardos '87] + better bounds on $\|\mathbf{w}'\|_\infty$ [WIP]

□

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \ : \ A\mathbf{x} = \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n$$

Theorem (K., Levin, Onn '18)

ILP solvable in poly$(n \cdot \langle A \rangle)$ calls to a Graver-best oracle.

Proof.

1. Solve LP relaxation in poly$(n \cdot \langle A \rangle)$ time [Tardos '86]
2. Proximity: integer opt not far from continuous opt $\Rightarrow$ shrink bounds $\mathbf{l}', \mathbf{u}'$, shrink rhs $\mathbf{b}'$.
3. Reduce objective: $\mathbf{l}', \mathbf{u}'$ give small box $\Rightarrow$ equiv. $\mathbf{w}'$ w/ small $\|\mathbf{w}'\|_\infty$ [Frank, Tardos '87] + better bounds on $\|\mathbf{w}'\|_\infty$ [WIP]
4. Convergence: $(2n-2)\langle A, \mathbf{w}', \mathbf{b}', \mathbf{l}', \mathbf{u}' \rangle =$ poly$(n \cdot \langle A \rangle)$ Graver-best steps suffice to reach optimum.

$\square$

Strongly Polynomial Oracle Time Algorithm

$$\min \mathbf{w}\mathbf{x} \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{l} \le \mathbf{x} \le \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n$$

Theorem (K., Levin, Onn '18)

ILP solvable in poly$(n \cdot \langle A \rangle)$ calls to a Graver-best oracle.

Proof.

1. Solve LP relaxation in poly$(n \cdot \langle A \rangle)$ time [Tardos '86]
2. Proximity: integer opt not far from continuous opt $\Rightarrow$ shrink bounds $\mathbf{l}', \mathbf{u}'$, shrink rhs $\mathbf{b}'$.
3. Reduce objective: $\mathbf{l}', \mathbf{u}'$ give small box $\Rightarrow$ equiv. $\mathbf{w}'$ w/ small $\|\mathbf{w}'\|_\infty$
   [Frank, Tardos '87] + better bounds on $\|\mathbf{w}'\|_\infty$ [WIP]
4. Convergence: $(2n-2)\langle A, \mathbf{w}', \mathbf{b}', \mathbf{l}', \mathbf{u}' \rangle =$ poly$(n \cdot \langle A \rangle)$ Graver-best steps suffice to reach optimum.

$\square$

Q: Where do I get the oracle?

Effective Graver-best Oracles

Primal graph $G_P(A)$**:**

vertices $\sim$ columns

edges $\sim$ two columns & $\exists$row non-zero in both columns

Dual graph: $G_D(A) = G_P(A^\intercal)$ (swap columns/rows)

Primal/dual treewidth/treedepth: tw/td of $G_P(A)/G_D(A)$

Effective Graver-best Oracles

Primal graph $G_P(A)$**:**

vertices $\sim$ columns

edges $\sim$ two columns & $\exists$row non-zero in both columns

Dual graph: $G_D(A) = G_P(A^\intercal)$ (swap columns/rows)

Primal/dual treewidth/treedepth: tw/td of $G_P(A)/G_D(A)$

Lemma (Primal lemma [K., Levin, Onn '18])

Effective G-b oracle if $\mathrm{tw}_P(A)$ *small and* $g_\infty(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_\infty$ *small.*

Lemma (Dual lemma [K., Levin, Onn '18])

Effective G-b oracle if $\mathrm{tw}_D(A)$ *small and* $g_1(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_1$ *small.*

Effective Graver-best Oracles

Primal graph $G_P(A)$**:**

vertices $\sim$ columns

edges $\sim$ two columns & $\exists$row non-zero in both columns

Dual graph: $G_D(A) = G_P(A^\intercal)$ (swap columns/rows)

Primal/dual treewidth/treedepth: tw/td of $G_P(A)/G_D(A)$

Lemma (Primal lemma [K., Levin, Onn '18])

Effective G-b oracle if $\mathrm{tw}_P(A)$ *small and* $g_\infty(A) = \max_{\mathbf{g} \in \mathcal{G}(A)} \|\mathbf{g}\|_\infty$ *small.*

Lemma (Dual lemma [K., Levin, Onn '18])

Effective G-b oracle if $\mathrm{tw}_D(A)$ *small and* $g_1(A) = \max_{\mathbf{g} \in \mathcal{G}(A)} \|\mathbf{g}\|_1$ *small.*

Proof idea.

DP over tree decomposition. $\square$

Effective Graver-best Oracles

Primal graph $G_P(A)$**:**

vertices $\sim$ columns

edges $\sim$ two columns & $\exists$row non-zero in both columns

Dual graph: $G_D(A) = G_P(A^\intercal)$ (swap columns/rows)

Primal/dual treewidth/treedepth: tw/td of $G_P(A)/G_D(A)$

Lemma (Primal lemma [K., Levin, Onn '18])

Effective G-b oracle if $\mathrm{tw}_P(A)$ *small and* $g_\infty(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_\infty$ *small.*

Lemma (Dual lemma [K., Levin, Onn '18])

Effective G-b oracle if $\mathrm{tw}_D(A)$ *small and* $g_1(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_1$ *small.*

Proof idea.

DP over tree decomposition. $\square$

Q: what ILP has small $\mathrm{tw}_P(A) + g_\infty(A)$ or $\mathrm{tw}_D(A) + g_1(A)$?

A: 2/multi-stage stochastic or n/tree-fold IPs! Let's have a look...

n-fold Integer Programs

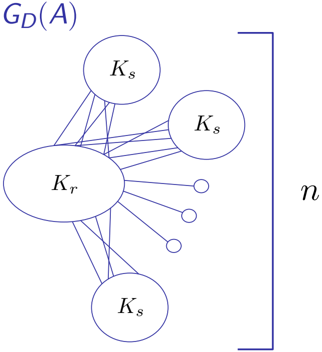
$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

$$\begin{array}{c|l} t & \\ \hline A_1 & r \\ A_2 & s \end{array}$$

n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

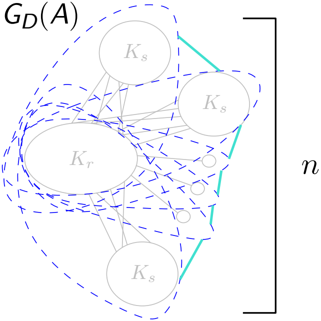
$$\begin{array}{c|c} \overset{t}{\overline{A_1}} & r \\ A_2 & s \end{array}$$



n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

$$\begin{array}{c|l} \overset{t}{\overline{A_1}} & r \\ A_2 & s \end{array}$$

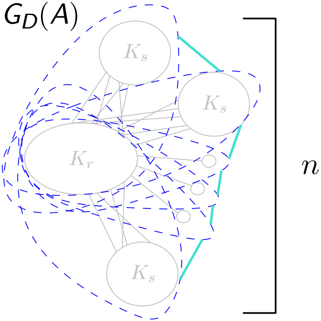


$\mathrm{tw}_D(A) \leq r + s$

n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

$$\begin{array}{c|l} \overline{\;t\;} & \\ A_1 & r \\ A_2 & s \end{array}$$



$\mathrm{tw}_D(A) \leq r + s$

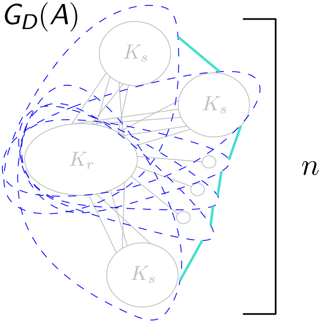
Lemma (De Loera, Hemmecke, Onn, Weismantel '08)

$g_1(A)$ is small ($f(\|A\|_\infty, r, s, t)$). :(

n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

$$\begin{array}{c|l} \overset{t}{\overline{}} & \\ A_1 & r \\ A_2 & s \end{array}$$



$\mathrm{tw}_D(A) \leq r + s$

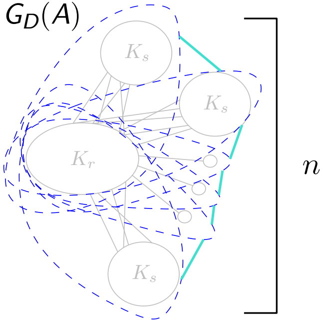
Lemma (K., Levin, Onn '18)

$g_1(A)$ is small ($f(\|A\|_\infty, r, s, t)$). :)

n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

$$\begin{array}{c|l} \overset{t}{\overline{A_1}} & r \\ A_2 & s \end{array}$$



$\mathrm{tw}_D(A) \leq r + s$

Lemma (K., Levin, Onn '18)

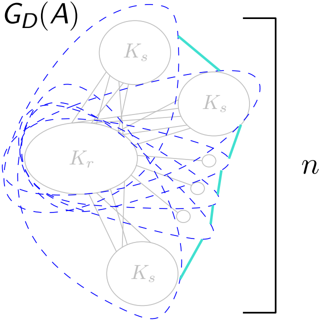
$g_1(A)$ is small ($f(\|A\|_\infty, r, s, t)$). :)

Theorem (K., Levin, Onn '18)

n-fold IP solvable in oracle time $f(\|A\|_\infty, r, s)$ poly$(nt \cdot \langle A \rangle)$

n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n} \qquad \begin{array}{c|l} \overset{t}{\overline{A_1}} & {\scriptstyle r} \\ A_2 & {\scriptstyle s} \end{array}$$



$\mathrm{tw}_D(A) \le r + s$

Lemma (K., Levin, Onn '18)

$g_1(A)$ is small ($f(\|A\|_\infty, r, s, t)$). :)

Theorem (K., Levin, Onn '18)

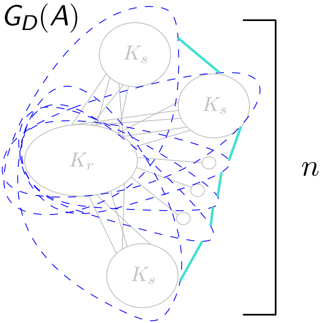
n-fold IP solvable in oracle time $f(\|A\|_\infty, r, s)\ \mathrm{poly}(nt \cdot \langle A \rangle)$

(**Previously:** $n^{f(\|A\|_\infty, r, s, t)}$ or $f(\|A\|_\infty, r, s, t) \cdot \mathrm{poly}(n \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle)$).

n-fold Integer Programs

$$A = \underbrace{\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}}_{n}$$

$$\begin{array}{c|l} \overset{t}{\overline{}} & \\ A_1 & r \\ A_2 & s \end{array}$$



$\mathrm{tw}_D(A) \leq r + s$

Lemma (K., Levin, Onn '18)

$g_1(A)$ is small ($f(\|A\|_\infty, r, s, t)$). :)

Theorem (K., Levin, Onn '18)

n-fold IP solvable in oracle time $f(\|A\|_\infty, r, s)\, \mathrm{poly}(nt \cdot \langle A \rangle)$

Generalization: Tree-fold IP – tree block structure, bounded $g_1(A)$ and $\mathrm{tw}_D(A)$.

2-stage stochastic Integer Programs

$$A = \begin{pmatrix} B_1 & B_2 & 0 & \cdots & 0 \\ B_1 & 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_1 & 0 & 0 & \cdots & B_2 \end{pmatrix}$$

transpose of n-fold IP $\Rightarrow \mathrm{tw}_P(A)$ is bounded

Lemma (Hemmecke, Schulz '01)

$g_\infty(A)$ is small.

2-stage stochastic Integer Programs

$$A = \begin{pmatrix} B_1 & B_2 & 0 & \cdots & 0 \\ B_1 & 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_1 & 0 & 0 & \cdots & B_2 \end{pmatrix}$$

👁 transpose of n-fold IP $\Rightarrow \mathrm{tw}_P(A)$ is bounded

Lemma (Hemmecke, Schulz '01)

$g_\infty(A)$ *is small.*

Generalizes to *multi-stage stochastic IP* – transpose of tree-fold IP, bounded $\mathrm{tw}_P(A)$ and $g_\infty(A)$.

2-stage stochastic Integer Programs

$$A = \begin{pmatrix} B_1 & B_2 & 0 & \cdots & 0 \\ B_1 & 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_1 & 0 & 0 & \cdots & B_2 \end{pmatrix}$$

👁 transpose of n-fold IP $\Rightarrow \mathrm{tw}_P(A)$ is bounded

Lemma (Hemmecke, Schulz '01)

$g_\infty(A)$ *is small.*

Generalizes to *multi-stage stochastic IP* – transpose of tree-fold IP, bounded $\mathrm{tw}_P(A)$ and $g_\infty(A)$.

Theorem (K., Levin, Onn '18)

2-stage stochastic IP solvable in oracle time $f(\|A\|_\infty, r, s)\,\mathit{poly}(n \cdot \langle A \rangle)$

Previously: $f(\|A\|_\infty, r, s)\,\mathrm{poly}(n \cdot \langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle)$

Treewidth & Treedepth

Surprise: Tree-fold IP is universal for all IPs with bounded $\mathrm{td}_D(A)$
(i.e., every A with small $\mathrm{td}_D(A)$ embeds into tree-fold IP matrix without blow-up)
Ditto for multi-stage stochastic IP and bounded $\mathrm{td}_P(A)$.

Treewidth & Treedepth

Surprise: Tree-fold IP is universal for all IPs with bounded $\mathrm{td}_D(A)$
(i.e., every A with small $\mathrm{td}_D(A)$ embeds into tree-fold IP matrix without blow-up)
Ditto for multi-stage stochastic IP and bounded $\mathrm{td}_P(A)$.

Treewidth & Treedepth

Surprise: Tree-fold IP is universal for all IPs with bounded $\mathrm{td}_D(A)$
(i.e., every A with small $\mathrm{td}_D(A)$ embeds into tree-fold IP matrix without blow-up)
Ditto for multi-stage stochastic IP and bounded $\mathrm{td}_P(A)$.

Theorem (K., Levin, Onn '18)

ILP solvable in time

- $f(\|A\|_\infty, \mathrm{td}_P(A)) \cdot poly(n \cdot \langle A \rangle)$
- $f(\|A\|_\infty, \mathrm{td}_D(A)) \cdot poly(n \cdot \langle A \rangle)$

Treewidth & Treedepth

Surprise: Tree-fold IP is universal for all IPs with bounded $\mathrm{td}_D(A)$
(i.e., every A with small $\mathrm{td}_D(A)$ embeds into tree-fold IP matrix without blow-up)
Ditto for multi-stage stochastic IP and bounded $\mathrm{td}_P(A)$.

Theorem (K., Levin, Onn '18)

ILP solvable in time

- $f(\|A\|_\infty, \mathrm{td}_P(A)) \cdot \mathit{poly}(n \cdot \langle A \rangle)$
- $f(\|A\|_\infty, \mathrm{td}_D(A)) \cdot \mathit{poly}(n \cdot \langle A \rangle)$

Previously only deciding feasibility in time $f(\|A, \mathbf{b}\|_\infty, \mathrm{td}_P(A)) \cdot n$; nothing known for $\mathrm{td}_D(A)$.

Treewidth & Treedepth

Surprise: Tree-fold IP is universal for all IPs with bounded $\mathrm{td}_D(A)$
(i.e., every A with small $\mathrm{td}_D(A)$ embeds into tree-fold IP matrix without blow-up)
Ditto for multi-stage stochastic IP and bounded $\mathrm{td}_P(A)$.

Theorem (K., Levin, Onn '18)

ILP solvable in time

- $f(\|A\|_\infty, \mathrm{td}_P(A)) \cdot poly(n \cdot \langle A \rangle)$
- $f(\|A\|_\infty, \mathrm{td}_D(A)) \cdot poly(n \cdot \langle A \rangle)$

Previously only deciding feasibility in time $f(\|A, \mathbf{b}\|_\infty, \mathrm{td}_P(A)) \cdot n$; nothing known for $\mathrm{td}_D(A)$.

Parameterization is tight:

- ILP not likely FPT parameterized by $\mathrm{td}_P(A)/\mathrm{td}_D(A)$ only,
- ILP is NP-hard for constant $\|A\|_\infty + \mathrm{tw}_P(A)/\mathrm{tw}_D(A)$.

Applications: Computational Social Choice

Intro: Computational Social Choice

- Ancient questions
    - Who should govern?
    - How to select them?
    - What is good for society?

Intro: Computational Social Choice

- Ancient questions
  - Who should govern?
  - How to select them?
  - What is good for society?
- Old fundamental results
  - 1743-1794: Marquis de Condorcet
  - 1733-1799: Jean-Charles de Borda
  - 1832-1898: Charles Lutwidge Dodgson (aka Lewis Carroll)

Intro: Computational Social Choice

- Ancient questions
  - Who should govern?
  - How to select them?
  - What is good for society?
- Old fundamental results
  - 1743-1794: Marquis de Condorcet
  - 1733-1799: Jean-Charles de Borda
  - 1832-1898: Charles Lutwidge Dodgson (aka Lewis Carroll)
- Recent topic
  - Brexit
  - Trump
  - Facebook

Intro: Computational Social Choice

- Ancient questions
  - Who should govern?
  - How to select them?
  - What is good for society?
- Old fundamental results
  - 1743-1794: Marquis de Condorcet
  - 1733-1799: Jean-Charles de Borda
  - 1832-1898: Charles Lutwidge Dodgson (aka Lewis Carroll)
- Recent topic
  - Brexit
  - Trump
  - Facebook

Boundaries are fruitful!

Intro: Voting

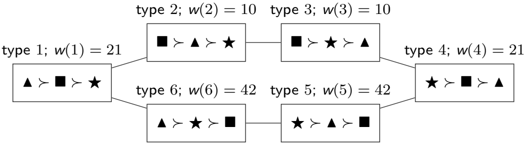
Candidates: ▲, ■, and ★.
People: preference (e.g. ■ $\succ$ ▲ $\succ$ ★), active/latent, bribery costs, etc.
(simplify: just preference)
Society: how many people of which type $\Rightarrow$ **Society graph:**

Intro: Voting

Candidates: $\blacktriangle$, $\blacksquare$, and $\bigstar$.

People: preference (e.g. $\blacksquare \succ \blacktriangle \succ \bigstar$), active/latent, bribery costs, etc.
(simplify: just preference)

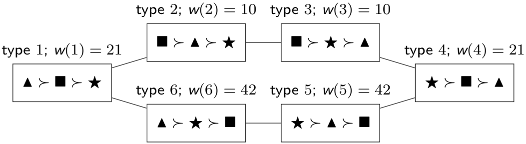
Society: how many people of which type $\Rightarrow$ **Society graph:**



Society $\mathbf{w} = (21, 10, 10, 21, 42, 42)$
edges $\equiv$ swap distance 1.

Intro: Voting

Candidates: ▲, ■, and ★.
People: preference (e.g. ■ $\succ$ ▲ $\succ$ ★), active/latent, bribery costs, etc. (simplify: just preference)
Society: how many people of which type $\Rightarrow$ **Society graph:**



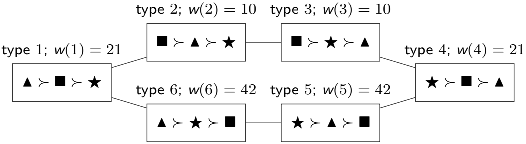
Society $\mathbf{w} = (21, 10, 10, 21, 42, 42)$
edges $\equiv$ swap distance 1.

Voting rule: given a society, *who should win?*

- Plurality = most times first
- Condorcet = beats everyone head-to-head

Intro: Voting

Candidates: ▲, ■, and ★.
People: preference (e.g. ■ $\succ$ ▲ $\succ$ ★), active/latent, bribery costs, etc.
(simplify: just preference)
Society: how many people of which type $\Rightarrow$ **Society graph:**



Society $\mathbf{w} = (21, 10, 10, 21, 42, 42)$
edges $\equiv$ swap distance 1.

Voting rule: given a society, *who should win?*

- Plurality = most times first
- Condorcet = beats everyone head-to-head
- **Dodgson** = least #swaps to Condorcet

Intro: Bribing

type 2; $w(2) = 10$

$\blacksquare \succ \blacktriangle \succ \bigstar$

type 3; $w(3) = 10$

$\blacksquare \succ \bigstar \succ \blacktriangle$

type 1; $w(1) = 21$

$\blacktriangle \succ \blacksquare \succ \bigstar$

type 4; $w(4) = 21$

$\bigstar \succ \blacksquare \succ \blacktriangle$

type 6; $w(6) = 42$

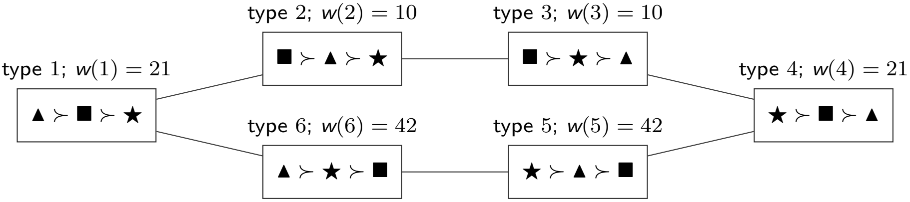
$\blacktriangle \succ \bigstar \succ \blacksquare$

type 5; $w(5) = 42$

$\bigstar \succ \blacktriangle \succ \blacksquare$

society $\mathbf{w} = (21, 10, 10, 21, 42, 42)$

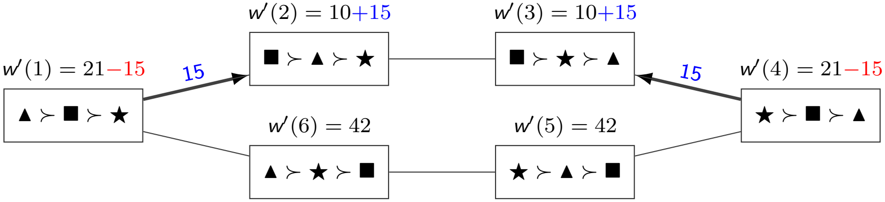
Intro: Bribing



society $\mathbf{w} = (21, 10, 10, 21, 42, 42)$

Bribery: cheapest way to move voters s.t. ■ wins Plurality?
(Assume unit cost per swap.)

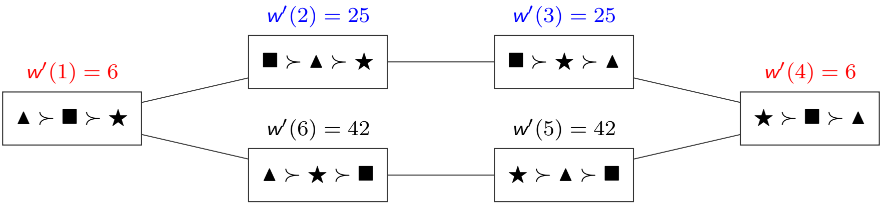
Intro: Bribing



society $\mathbf{w} = (21, 10, 10, 21, 42, 42)$
change $\boldsymbol{\Delta} = (-15, +15, +15, -15, 0, 0)$

Bribery: cheapest way to move voters s.t. ■ wins Plurality?
(Assume unit cost per swap.)

Intro: Bribing



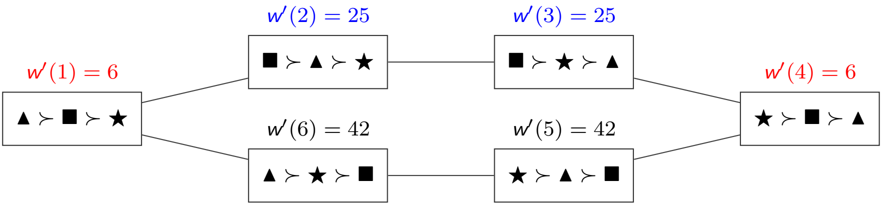
$$\mathbf{w}' = \mathbf{w} + \boldsymbol{\Delta} \text{ with } \boldsymbol{\Delta} = (-15, +15, +15, -15, 0, 0)$$

$\blacksquare$ **wins:** $48 = w(1) + w(6) = w(4) + w(5) < w(2) + w(3) = 50$

Bribery: cheapest way to move voters s.t. $\blacksquare$ wins Plurality?

(Assume unit cost per swap.)

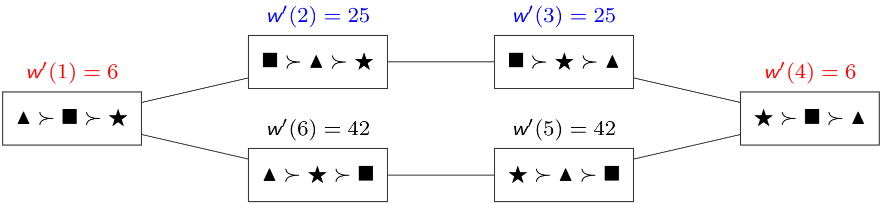
Intro: Bribing



$\mathbf{w}' = \mathbf{w} + \boldsymbol{\Delta}$ with $\boldsymbol{\Delta} = (-15, +15, +15, -15, 0, 0)$
$\blacksquare$ **wins:** $48 = w(1) + w(6) = w(4) + w(5) < w(2) + w(3) = 50$

Bribery: cheapest way to move voters s.t. $\blacksquare$ wins Plurality?
(Assume unit cost per swap.)
Robust model: captures many prior manipulation models – full bribery, only shift $\blacksquare$, pay-per-swap, add/delete voters, etc.

Intro: Bribing



$$\mathbf{w}' = \mathbf{w} + \boldsymbol{\Delta} \text{ with } \boldsymbol{\Delta} = (-15, +15, +15, -15, 0, 0)$$
$$\blacksquare \textbf{ wins: } 48 = w(1) + w(6) = w(4) + w(5) < w(2) + w(3) = 50$$

Bribery: cheapest way to move voters s.t. $\blacksquare$ wins Plurality?
(Assume unit cost per swap.)
Robust model: captures many prior manipulation models – full bribery, only shift $\blacksquare$, pay-per-swap, add/delete voters, etc.

BTW: Society graph + move + change model is "obvious" but new and very useful itself! [Faliszewski, Gonen, K., Talmon] and [AAMAS; Knop, K., Mnich]

Complexity of Bribery

Before 2017: Bribery in time $f(\#\text{types of people}) \cdot \log(\#\text{people})$ for "simple" voting rules (many ad-hoc results; all use Lenstra), BUT:

Complexity of Bribery

Before 2017: Bribery in time $f(\#\text{types of people}) \cdot \log(\#\text{people})$ for "simple" voting rules (many ad-hoc results; all use Lenstra), BUT:

- f is double-exponential :(
- cannot handle different voter costs :(
- cannot handle Dodgson's rule :(

Complexity of Bribery

Before 2017: Bribery in time $f(\#\text{types of people}) \cdot \log(\#\text{people})$ for "simple" voting rules (many ad-hoc results; all use Lenstra), BUT:

- f is double-exponential :(
- cannot handle different voter costs :(
- cannot handle Dodgson's rule :(

Challenge #1: Replace Lenstra, make single-exp!
Challenge #2: Handle different voter costs! (replace #types w/ #candidates)

[2014; Bredereck, Chen, Faliszewski, Guo, Niedermeier, Woeginger]

Complexity of Bribery

Before 2017: Bribery in time $f(\#\text{types of people}) \cdot \log(\#\text{people})$ for "simple" voting rules (many ad-hoc results; all use Lenstra), BUT:

- f is double-exponential :(
- cannot handle different voter costs :(
- cannot handle Dodgson's rule :(

Challenge #1: Replace Lenstra, make single-exp!
Challenge #2: Handle different voter costs! (replace #types w/ #candidates)

[2014; Bredereck, Chen, Faliszewski, Guo, Niedermeier, Woeginger]

Solved!

Theorem (STACS, ESA, AAMAS; Knop, K., Mnich)

Bribery in time:

1. *single-exp* $f(\#\text{candidates}) \cdot \mathit{poly}(\#\text{types}) \cdot \log(\#\text{people})$ *for "simple" rules,*
2. $f(\#\text{types}) \cdot \mathit{poly}(\#\text{people})$ *for "complex" rules, incl. Dodgson.*

Complexity of Bribery (contd.)

Theorem (STACS, ESA, AAMAS; Knop, K., Mnich)

Bribery in time:

1. *single-exp f(#candidates) $\cdot$ poly(#types) $\cdot$ $\log$(#people) for "simple" rules,*
2. *f(#types) $\cdot$ poly(#people) for "complex" rules, incl. Dodgson.*

Proof of (1).

Idea: Encode in *n*-fold IP:
Blocks $\sim$ types of people,
A block $\sim$ #ppl moving to other type,
$(A_1 \cdots A_1) \sim$ voting rule.
Apply strongly FPT *n*-fold algorithm!
need *few* constraints, *small* $\|A_1\|_\infty$.

$$\begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}$$

$\square$

Complexity of Bribery (contd.)

Theorem (STACS, ESA, AAMAS; Knop, K., Mnich)

Bribery in time:

1. *single-exp* f(*#candidates*) $\cdot$ *poly*(*#types*) $\cdot \log$(*#people*) *for "simple" rules,*
2. f(*#types*) $\cdot$ *poly*(*#people*) *for "complex" rules, incl. Dodgson.*

Proof of (2).

Want: formula $\Phi_{\text{Dodgson}} \equiv$ "$\bigstar$ is Dodgson winner" $\equiv$ least #swaps to Condorcet

$\square$

Complexity of Bribery (contd.)

Theorem (STACS, ESA, AAMAS; Knop, K., Mnich)

Bribery in time:

1. *single-exp* $f(\#\text{candidates}) \cdot \textit{poly}(\#\text{types}) \cdot \log(\#\text{people})$ *for "simple" rules,*
2. $f(\#\text{types}) \cdot \textit{poly}(\#\text{people})$ *for "complex" rules, incl. Dodgson.*

Proof of (2).

Want: formula $\Phi_{\text{Dodgson}} \equiv$ "$\bigstar$ is Dodgson winner" $\equiv$ least #swaps to Condorcet

$$\Phi_{\text{Dodgson}} \equiv \exists k \in \mathbb{N} : \begin{cases} \exists \text{ sequence of } k \text{ swaps } \rightsquigarrow \bigstar \text{ is Condorcet winner AND} \\ \forall c \neq \bigstar \text{ at least } k+1 \text{ swaps} \rightsquigarrow c \text{ is Condorcet winner.} \end{cases}$$

□

Complexity of Bribery (contd.)

Theorem (STACS, ESA, AAMAS; Knop, K., Mnich)

Bribery in time:

1. *single-exp* $f(\#\text{candidates}) \cdot poly(\#\text{types}) \cdot \log(\#\text{people})$ *for "simple" rules,*
2. $f(\#\text{types}) \cdot poly(\#\text{people})$ *for "complex" rules, incl. Dodgson.*

Proof of (2).

Want: formula $\Phi_{\text{Dodgson}} \equiv$ "$\bigstar$ is Dodgson winner" $\equiv$ least #swaps to Condorcet

$$\Phi_{\text{Dodgson}} \equiv \exists k \in \mathbb{N} : \begin{cases} \exists \text{ sequence of } k \text{ swaps } \rightsquigarrow \bigstar \text{ is Condorcet winner AND} \\ \forall c \neq \bigstar \text{ at least } k+1 \text{ swaps } \rightsquigarrow c \text{ is Condorcet winner.} \end{cases}$$

Encode Φ_{Dodgson} in terms of society / move / change vectors
$\Rightarrow$ decide $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} : \Psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ sentence $\Rightarrow$ [much modeling work]
$\Rightarrow$ decide $\forall \mathbf{x} \exists \mathbf{y} : A(\mathbf{x}, \mathbf{y}) \leq \mathbf{b}$ sentence

$\square$

Complexity of Bribery (contd.)

Theorem (STACS, ESA, AAMAS; Knop, K., Mnich)

Bribery in time:

1. *single-exp* $f(\#\text{candidates}) \cdot \text{poly}(\#\text{types}) \cdot \log(\#\text{people})$ *for "simple" rules,*
2. $f(\#\text{types}) \cdot \text{poly}(\#\text{people})$ *for "complex" rules, incl. Dodgson.*

Proof of (2).

Want: formula $\Phi_{\text{Dodgson}} \equiv$ "$\bigstar$ is Dodgson winner" $\equiv$ least #swaps to Condorcet

$$\Phi_{\text{Dodgson}} \equiv \exists k \in \mathbb{N} : \begin{cases} \exists \text{ sequence of } k \text{ swaps } \rightsquigarrow \bigstar \text{ is Condorcet winner AND} \\ \forall c \neq \bigstar \text{ at least } k+1 \text{ swaps } \rightsquigarrow c \text{ is Condorcet winner.} \end{cases}$$

Encode Φ_{Dodgson} in terms of society / move / change vectors
$\Rightarrow$ decide $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} : \Psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ sentence $\Rightarrow$ [much modeling work]
$\Rightarrow$ decide $\forall \mathbf{x} \exists \mathbf{y} : A(\mathbf{x}, \mathbf{y}) \leq \mathbf{b}$ sentence

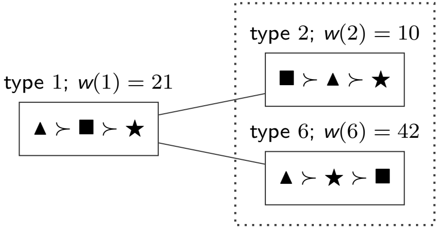
👁 **Thm [Eisenbrand, Shmonin '08]:** Can decide
$\forall \mathbf{b} \in Q \cap \mathbb{Z}^m \, \exists \mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} \leq \mathbf{b}$ in time $f(n, m) \cdot \text{poly}(\|A, \mathbf{b}\|_\infty)$ $\square$

Diffusion Process in Society Graphs

Diffusion $\approx$ how opinions spread; previous models highly intractable :(

Diffusion Process in Society Graphs

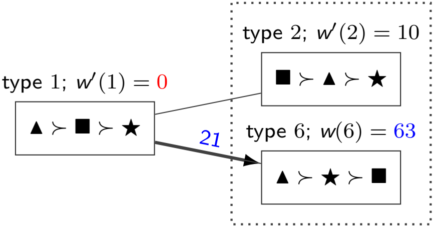
Diffusion $\approx$ how opinions spread; previous models highly intractable :(



perspective of ▲ $\succ$ ■ $\succ$ ★:
"majority of neighbors thinks ▲ $\succ$ ★ $\succ$ ■" $\Rightarrow$
peer pressure: "prob should change my mind"

Diffusion Process in Society Graphs

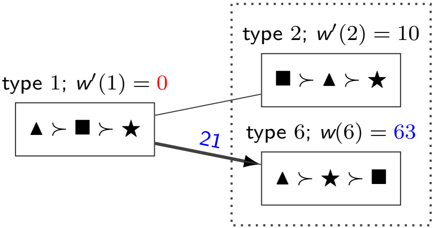
Diffusion $\approx$ how opinions spread; previous models highly intractable :(



perspective of ▲ ≻ ■ ≻ ★:
"majority of neighbors thinks ▲ ≻ ★ ≻ ■" $\Rightarrow$
peer pressure: "prob should change my mind"
(synchronous / asynchronous)

Diffusion Process in Society Graphs

Diffusion $\approx$ how opinions spread; previous models highly intractable :(

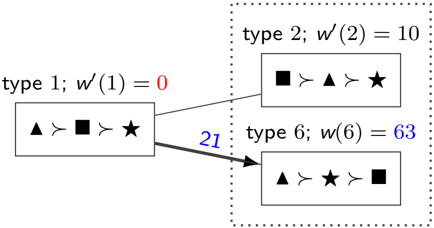


perspective of $▲ \succ ■ \succ ★$:
"majority of neighbors thinks $▲ \succ ★ \succ ■$" $\Rightarrow$
peer pressure: "prob should change my mind"
(synchronous / asynchronous)

- Diffusion stabilizes quickly

Diffusion Process in Society Graphs

Diffusion $\approx$ how opinions spread; previous models highly intractable :(



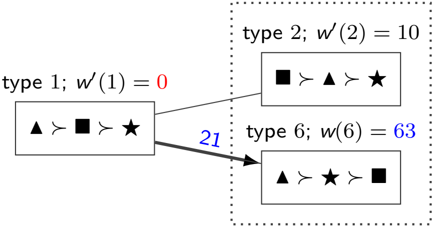
perspective of ▲ ≻ ■ ≻ ★:
"majority of neighbors thinks ▲ ≻ ★ ≻ ■" $\Rightarrow$
peer pressure: "prob should change my mind"
(synchronous / asynchronous)

- Diffusion stabilizes quickly
- few types of people $\Rightarrow$ can model process as fixed dim ILP

  Idea: ILP modeling tricks $\Rightarrow$ can express conditionals "*if* neighborhood majority of type *i* *then* move to type *i* in next step"

Diffusion Process in Society Graphs

Diffusion ≈ how opinions spread; previous models highly intractable :(



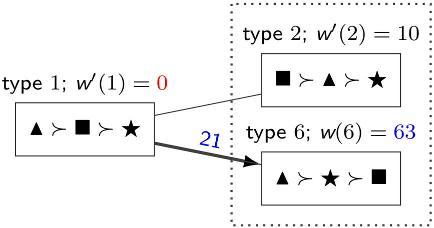
perspective of $\blacktriangle \succ \blacksquare \succ \bigstar$:
"majority of neighbors thinks $\blacktriangle \succ \bigstar \succ \blacksquare$" $\Rightarrow$
peer pressure: "prob should change my mind"
(synchronous / asynchronous)

- Diffusion stabilizes quickly
- few types of people $\Rightarrow$ can model process as fixed dim ILP

  Idea: ILP modeling tricks $\Rightarrow$ can express conditionals "*if* neighborhood majority of type *i* *then* move to type *i* in next step"
- Bribery in Society Graphs: minimum move s.t. $\bigstar$ wins after stabilization?

Diffusion Process in Society Graphs

Diffusion ≈ how opinions spread; previous models highly intractable :(



perspective of $\blacktriangle \succ \blacksquare \succ \bigstar$:
"majority of neighbors thinks $\blacktriangle \succ \bigstar \succ \blacksquare$" $\Rightarrow$
peer pressure: "prob should change my mind"
(synchronous / asynchronous)

- Diffusion stabilizes quickly
- few types of people $\Rightarrow$ can model process as fixed dim ILP
  Idea: ILP modeling tricks $\Rightarrow$ can express conditionals "*if* neighborhood majority of type *i* *then* move to type *i* in next step"
- BRIBERY IN SOCIETY GRAPHS: minimum move s.t. $\bigstar$ wins after stabilization?

Theorem (Faliszewski, Gonen, K., Talmon '18)

BRIBERY IN SOCIETY GRAPHS *solvable in time*
$f(\#\textit{types of people}) \cdot \log(\#\textit{people})$, *for most voting rules.*

Other Applications

n-**fold IP:** no applications in parameterized complexity before 2016. Now:

- **Scheduling** with short jobs and many machine types; many different objectives ($C_{\max}$, $\sum w_j C_j$, tardiness, ℓ_p-norm, weighted flow time, ...)
  [JoSh '17; Knop, K.] & [WIP]
  Efficient PTASes [Jansen, Klein, Maack, Rau '18]
- **Stringology:** double-exp $\Rightarrow$ single-exp, many problems
  [ESA; Knop, K., Mnich]
- **Graph algorithms:** graph layout problems, simple dense graphs [ditto]
- Computational Social Choice [STACS, ESA; Knop, K., Mnich]

Engineering & Research Directions

Engineering: Experiments & Outlook

Summary: "small" ℓ_1/ℓ_∞-norm augmenting steps might be good enough.

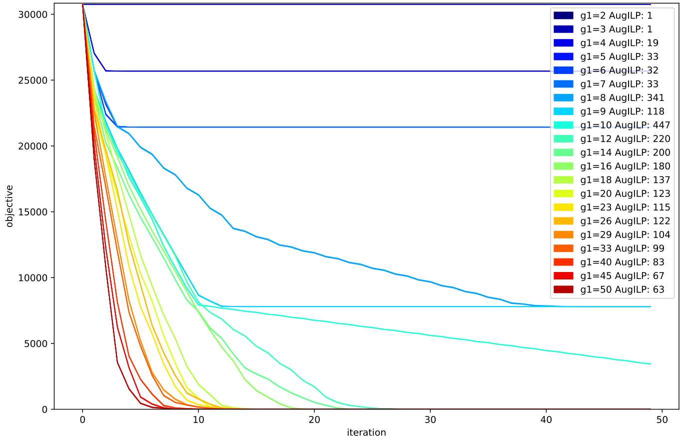
Q: How *small*? True guarantee: $g_1(A)$ – might be large in practice :(

A: Choose *some* $\mathrm{g}_1 \in \mathbb{N}$, $1 < \mathrm{g}_1 \leq g_1(A)$, and see what happens!

(What could go wrong: local optima or slow convergence) ... We tested it:

Engineering: Experiments & Outlook

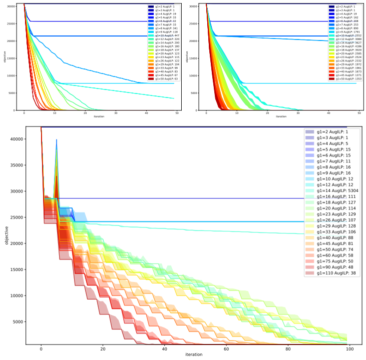
Summary: "small" ℓ_1/ℓ_∞-norm augmenting steps might be good enough.
Q: How *small*? True guarantee: $g_1(A)$ – might be large in practice :(
A: Choose *some* $\mathrm{g}_1 \in \mathbb{N}$, $1 < \mathrm{g}_1 \leq g_1(A)$, and see what happens!
(What could go wrong: local optima or slow convergence) ... We tested it:



Engineering: Experiments & Outlook

Summary: "small" ℓ_1/ℓ_∞-norm augmenting steps might be good enough.
Q: How *small*? True guarantee: $g_1(A)$ – might be large in practice :(
A: Choose *some* $\mathrm{g}_1 \in \mathbb{N}$, $1 < \mathrm{g}_1 \leq g_1(A)$, and see what happens!

(What could go wrong: local optima or slow convergence) ... We tested it:



much better than predicted worst case!

Idea: use tree decomposition to divide & conquer ILP; previously impossible due to inefficient tw computations.

> *Introducing automatic decomposition methods in primal heuristics is very interesting.*
>
> —Matthias Köppe (UC Davis)

(Student project [Altmanová, Knop, K.] & [WIP])

Research Outlook

- Big picture view of Integer Programming
  - beyond convexity? (so far just IQP par by $n + \|A\|_\infty + \|Q\|_\infty$)
  - unified theory for fixed-dim optimization
  - fixed $\Leftrightarrow$ variable dimension? (completely different techniques so far)

Research Outlook

- Big picture view of Integer Programming
  - beyond convexity? (so far just IQP par by $n + \|A\|_\infty + \|Q\|_\infty$)
  - unified theory for fixed-dim optimization
  - fixed $\Leftrightarrow$ variable dimension? (completely different techniques so far)
  - Engineering: automatic decompositional methods

Research Outlook

- Big picture view of Integer Programming
  - beyond convexity? (so far just IQP par by $n + \|A\|_\infty + \|Q\|_\infty$)
  - unified theory for fixed-dim optimization
  - fixed $\Leftrightarrow$ variable dimension? (completely different techniques so far)
  - Engineering: automatic decompositional methods
- Computational Social Choice
  - descriptive complexity of voting rules?
  - back-and-forth campaigning (polytope games)?
  - stochastic diffusion models?

Research Outlook

- Big picture view of Integer Programming
  - beyond convexity? (so far just IQP par by $n + \|A\|_\infty + \|Q\|_\infty$)
  - unified theory for fixed-dim optimization
  - fixed $\Leftrightarrow$ variable dimension? (completely different techniques so far)
  - Engineering: automatic decompositional methods
- Computational Social Choice
  - descriptive complexity of voting rules?
  - back-and-forth campaigning (polytope games)?
  - stochastic diffusion models?
- Theory of Practical Algorithms
  - SAT/ILP oracles (esp. for problems beyond NP)
  - Parameter-oblivious algorithms
  - Tunable algorithms (vs. all-or-nothing algorithms)
  - Turbocharging heuristics (it works! let's build the theory)

Research Outlook

- Big picture view of Integer Programming
  - beyond convexity? (so far just IQP par by $n + \|A\|_\infty + \|Q\|_\infty$)
  - unified theory for fixed-dim optimization
  - fixed $\Leftrightarrow$ variable dimension? (completely different techniques so far)
  - Engineering: automatic decompositional methods
- Computational Social Choice
  - descriptive complexity of voting rules?
  - back-and-forth campaigning (polytope games)?
  - stochastic diffusion models?
- Theory of Practical Algorithms
  - SAT/ILP oracles (esp. for problems beyond NP)
  - Parameter-oblivious algorithms
  - Tunable algorithms (vs. all-or-nothing algorithms)
  - Turbocharging heuristics (it works! let's build the theory)

Thank you!

Cheers to: K. Altmanová, J. Crampton, F. Eisenbrand, G. Gutin, R. Hildebrand, Ch. Hunkenschröder, K.-M. Klein, D. Knop, M. Köppe, J. Lee, A. Levin, M. Mnich, S. Onn, and R. Wattrigant.