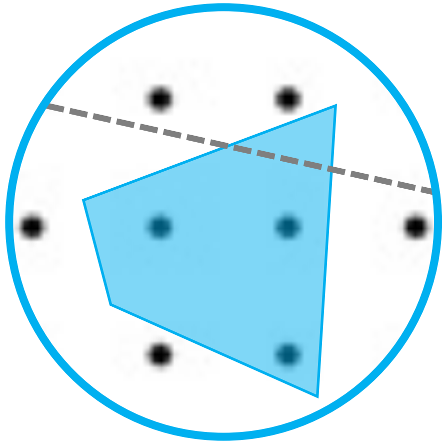
Theoretical foundations of **machine learning** for **cutting plane** selection

Ellen Vitercik

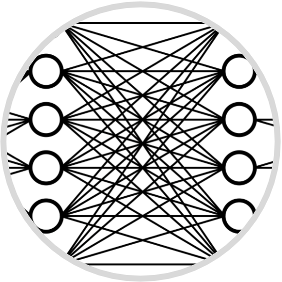
Berkeley (EECS) → **Stanford** (MS&E + CS)

Balcan, Sandholm, Prasad, Vitercik [NeurIPS'21]
Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik [STOC'21]



Integer programming solvers

Most popular tool for solving combinatorial problems



Robust ML

Routing

Manufacturing

Scheduling

Planning

ML for integer programming

Used heavily throughout industry and science

Many different ways to incorporate **learning** into solving

E.g., IP solvers (CPLEX, Gurobi) have a **ton** of parameters

CPLEX has **170-page** manual describing **172** parameters

ML for integer programming

Used heavily throughout industry and science

Many different ways to incorporate **learning** into solving

- E.g., IP solvers (CPLEX, Gurobi) have a **ton** of parameters
  - CPLEX has **170-page** manual describing **172** parameters

Solving is extremely difficult, so ML can make a huge difference

Companies often have lots of **data** about their applications

- E.g., all the scheduling IPs an airline solves day after day

ML for integer programming

Lots of interest from an **empirical** perspective, e.g.:

Leyton-Brown, Nudelman, Andrew, McFadden, Shoham	IJCAI'03, CP'03
Hutter, Hoos, Leyton-Brown, Stützle	JAIR'09
Sandholm	Handbook of Market Design'13
He, Daume, Eisner	NeurIPS'14
Khalil, Le Bodic, Song, Nemhauser, Dilkina	AAAI'16
Song, Lanka, Yue, Dilkina	NeurIPS'20
Tang, Agrawal, Faenza	ICML'20
Huang, Wang, Liu, Zhen, Zhang, Yuan, Hao, Yu, Wang	Pattern Recognition '22

This talk:

Guarantees for IP parameter optimization *(cut selection)*

ML for algorithm design

Integer & linear programming
[Leyton-Brown, Nudelman, Andrew, McFadden, Shoham, CP ’03; …]

Constraint satisfaction
[Horvitz, Ruan, Gomes, Krautz, Selman, Chickering, UAI’01; …]

Economics (mechanism design)
[Likhodedov, Sandholm, AAAI ’04, ’05; …]

Computational biology
[Majoros, Salzberg, Bioinformatics’04; …]

Applied research

2000

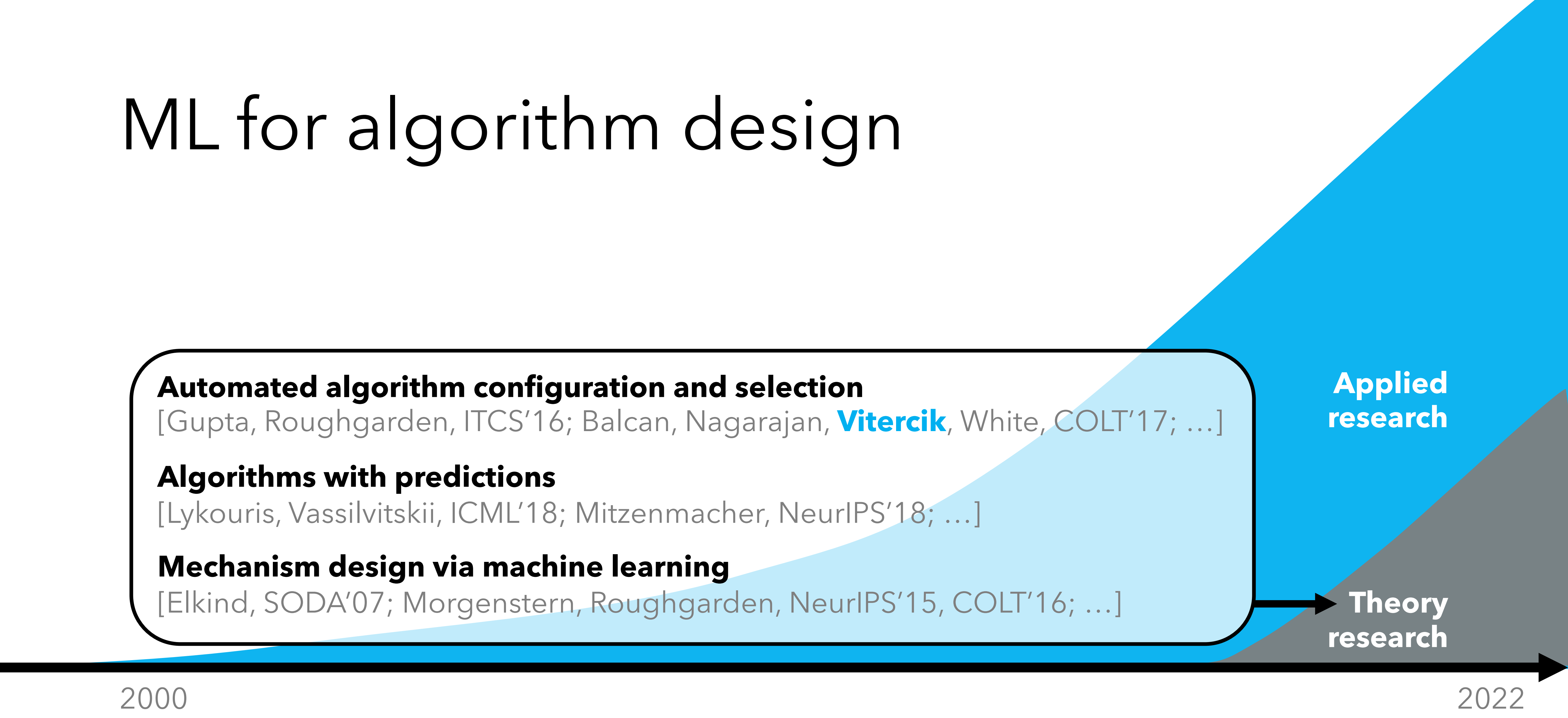
2022

ML for algorithm design

Automated algorithm configuration and selection
[Gupta, Roughgarden, ITCS'16; Balcan, Nagarajan, **Vitercik**, White, COLT'17; …]

Algorithms with predictions
[Lykouris, Vassilvitskii, ICML'18; Mitzenmacher, NeurIPS'18; …]

Mechanism design via machine learning
[Elkind, SODA'07; Morgenstern, Roughgarden, NeurIPS'15, COLT'16; …]



Outline

1. Introduction
2. Integer programming
    i. **Overview**
    ii. Branch-and-bound
    iii. Our results
3. Beyond integer programming
4. Conclusions

Integer programs

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{subject to} & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \boldsymbol{z} \in \mathbb{Z}^n \end{array}$$

$A \in \mathbb{Z}^{m \times n}, \boldsymbol{b} \in \mathbb{Z}^m$

Modeling the application domain

IPs drawn from unknown application-specific distribution $\mathcal{D}$

E.g., **distribution over routing problems**

Widely assumed in applied research, e.g.:

- Horvitz, Ruan, Gomez, Kautz, Selman, Chickering — UAI'01
- Xu, Hutter, Hoos, Leyton-Brown — JAIR'08
- He, Daumé, Eisner — NeurIPS'14

And theoretical research on algorithm configuration, e.g.:

- Gupta, Roughgarden — ITCS'16
- Balcan — Book Chapter'20

Automated configuration procedure

1. Fix parameterized IP solver
2. Receive training set of "typical" IPs sampled from $\mathcal{D}$

$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}$ $\{A^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}\}$ $\{A^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}\}$ $\{A^{(4)}, \boldsymbol{b}^{(4)}, \boldsymbol{c}^{(4)}\}$ ● ● ●

3. Return parameter settings $\hat{\boldsymbol{\mu}}$ with good avg performance

Search tree size, runtime, etc.

Key question: How to find $\hat{\boldsymbol{\mu}}$ with good avg performance?

Hutter et al. [JAIR'09, LION'11], Ansótegui et al. [CP'09], Sandholm [Handbook of Market Design '13], Khalil et al. [AAAI'16], Balcan, Sandholm, **Vitercik** [AAAI'20], …

Automated configuration procedure

1. Fix parameterized IP solver
2. Receive training set of "typical" IPs sampled from $\mathcal{D}$

$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}$ $\{A^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}\}$ $\{A^{(3)}, \boldsymbol{b}^{(3)}, \boldsymbol{c}^{(3)}\}$ $\{A^{(4)}, \boldsymbol{b}^{(4)}, \boldsymbol{c}^{(4)}\}$ • • •

3. Return parameter settings $\hat{\boldsymbol{\mu}}$ with good avg performance

Search tree size, runtime, etc.

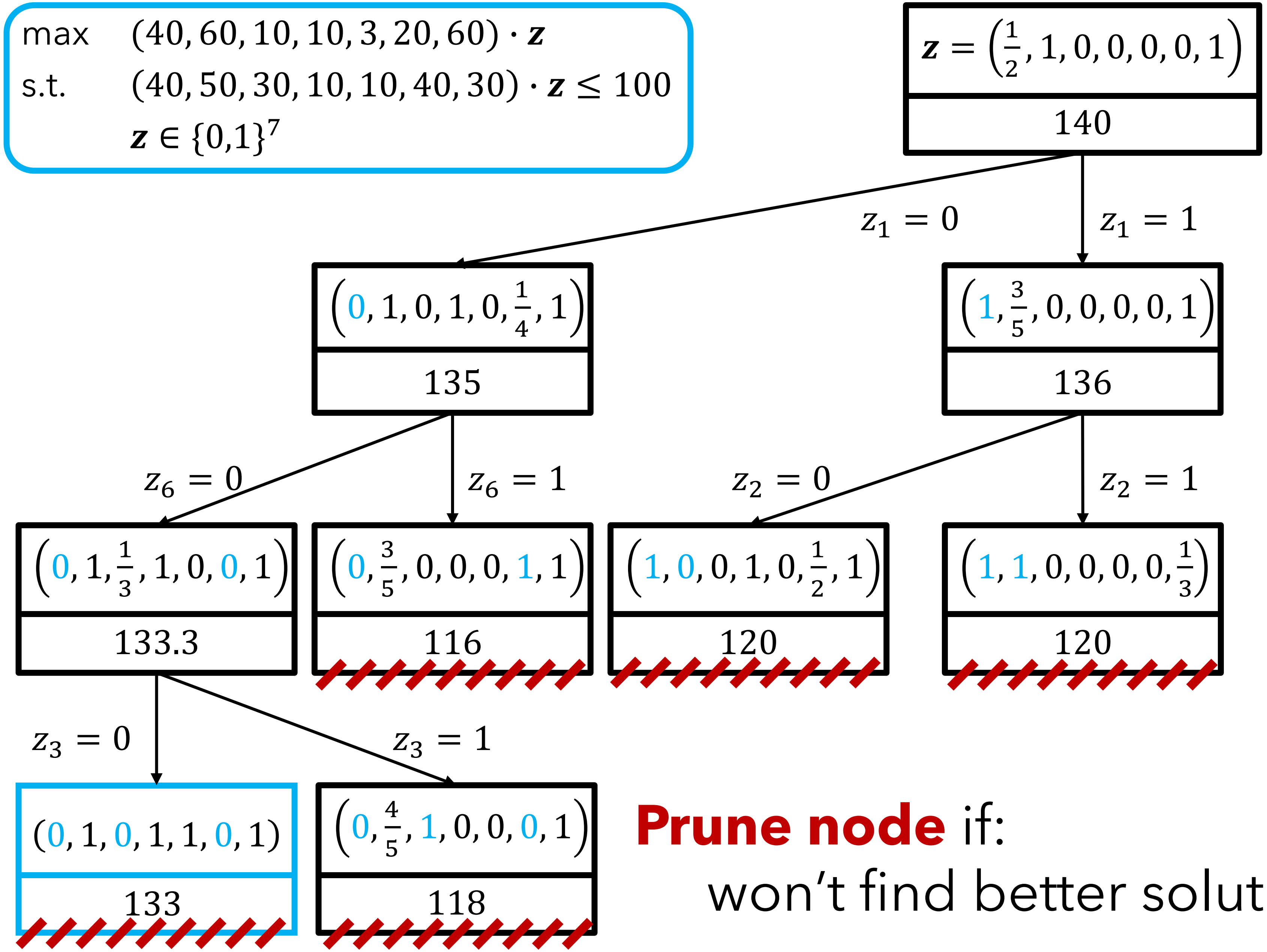
Focus of this talk: Will $\hat{\boldsymbol{\mu}}$ have good **future** performance?
More formally: Is the expected utility of $\hat{\boldsymbol{\mu}}$ also high?

Outline

1. Introduction
2. Integer programming
   i. Overview
   ii. Branch-and-bound
   iii. Our results
3. Beyond integer programming
4. Conclusions

$$\begin{array}{ll} \max & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{z} \\ \text{s.t.} & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{z} \leq 100 \\ & \boldsymbol{z} \in \{0,1\}^7 \end{array}$$

Branch and bound (B&B)



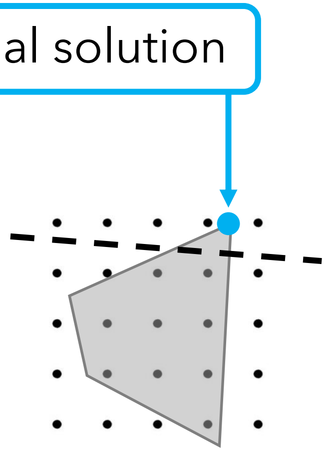
Prune node if:

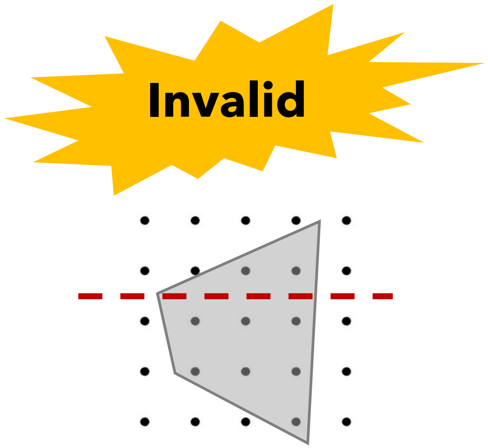
won't find better solution along branch

Cutting planes

Additional constraints that:

- Separate the LP optimal solution
    - Tightens LP relaxation to prune nodes sooner
- Don't separate any integer point





Cutting planes

Modern IP solvers add cutting planes through the B&B tree

"Branch-and-cut"

Responsible for breakthrough speedups of IP solvers

Cornuéjols, Annals of OR '07

Challenges:

- Many different types of cutting planes
  - Chvátal-Gomory cuts, cover cuts, clique cuts, …
- How to choose which cuts to apply?

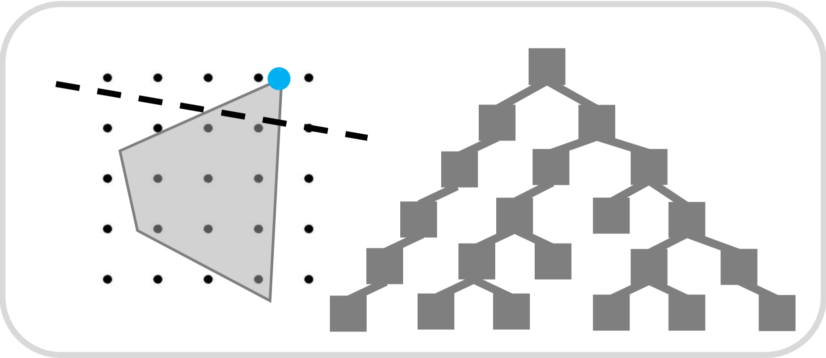
Key challenge

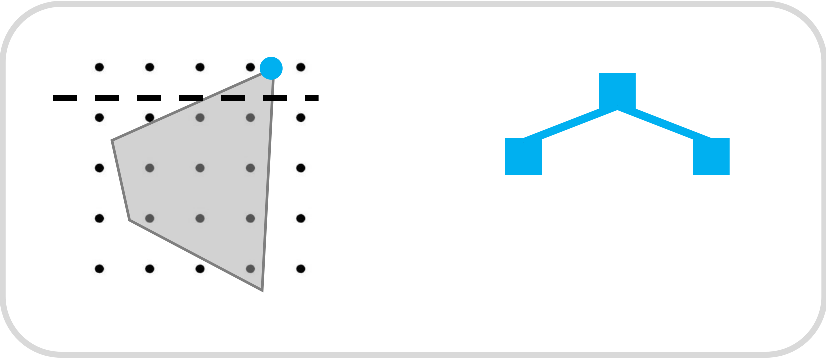
Cut (typically) remains in LPs throughout **entire** tree search

Every aspect of tree search depends on LP guidance

Node selection, variable selection, pruning, …

Tiny change in cut can cause **major changes to tree**





Outline

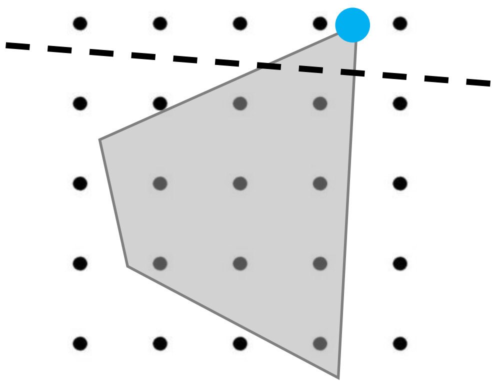
Chvátal-Gomory cuts

We study the canonical family of *Chvátal-Gomory (CG)* cuts

CG cut parameterized by $\boldsymbol{\mu} \in [0,1)^m$ is $\lfloor \boldsymbol{\mu}^T A \rfloor \boldsymbol{z} \leq \lfloor \boldsymbol{\mu}^T \boldsymbol{b} \rfloor$

Important properties:

- CG cuts are valid
- Can be chosen so it separates the LP opt

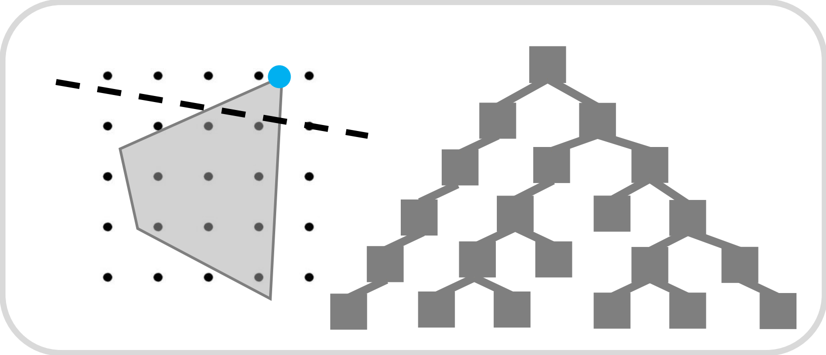


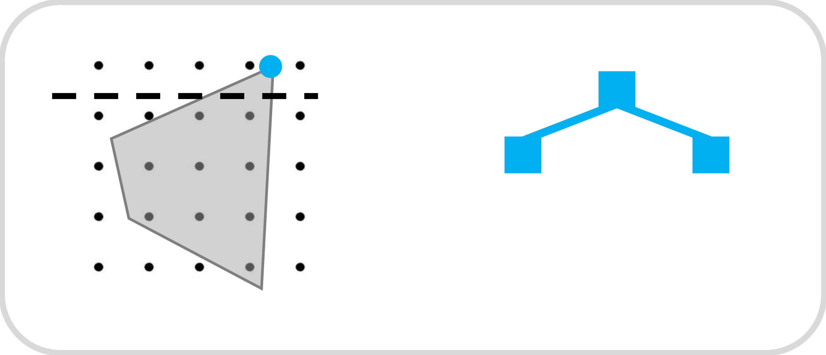
Balcan, Sandholm, Prasad, **Vitercik**, NeurIPS'21

Key challenge: Sensitivity of B&C

Theorem [informal]:

Tiny changes to $\boldsymbol{\mu}$ can lead to exponential jumps in tree size



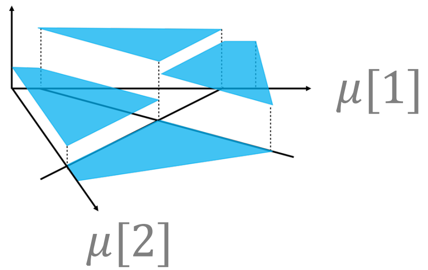


Balcan, Sandholm, Prasad, **Vitercik**, NeurIPS'21

Key lemma

Lemma: $O\left(\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n\right)$ hyperplanes partition $[0,1)^m$ into regions s.t. in any one region, B&C tree is fixed

Tree size is a piecewise-constant function of $\boldsymbol{\mu} \in [0,1)^m$



Balcan, Sandholm, Prasad, **Vitercik**, NeurIPS'21

Key lemma

Lemma: $O(\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n)$ hyperplanes partition $[0,1)^m$ into regions s.t. in any one region, B&C tree is fixed

Proof idea:

- CG cut parameterized by $\boldsymbol{\mu} \in [0,1)^m$ is $\lfloor \boldsymbol{\mu}^T A \rfloor \boldsymbol{z} \leq \lfloor \boldsymbol{\mu}^T \boldsymbol{b} \rfloor$
- For any $\boldsymbol{u}$ and column $\boldsymbol{a}_i$, $\lfloor \boldsymbol{\mu}^T \boldsymbol{a}_i \rfloor \in [-\|\boldsymbol{a}_i\|_1, \|\boldsymbol{a}_i\|_1]$
- For each integer $k_i \in [-\|\boldsymbol{a}_i\|_1, \|\boldsymbol{a}_i\|_1]$:

$$\lfloor \boldsymbol{\mu}^T \boldsymbol{a}_i \rfloor = k_i \text{ iff } k_i \leq \boldsymbol{\mu}^T \boldsymbol{a}_i < k_i + 1$$

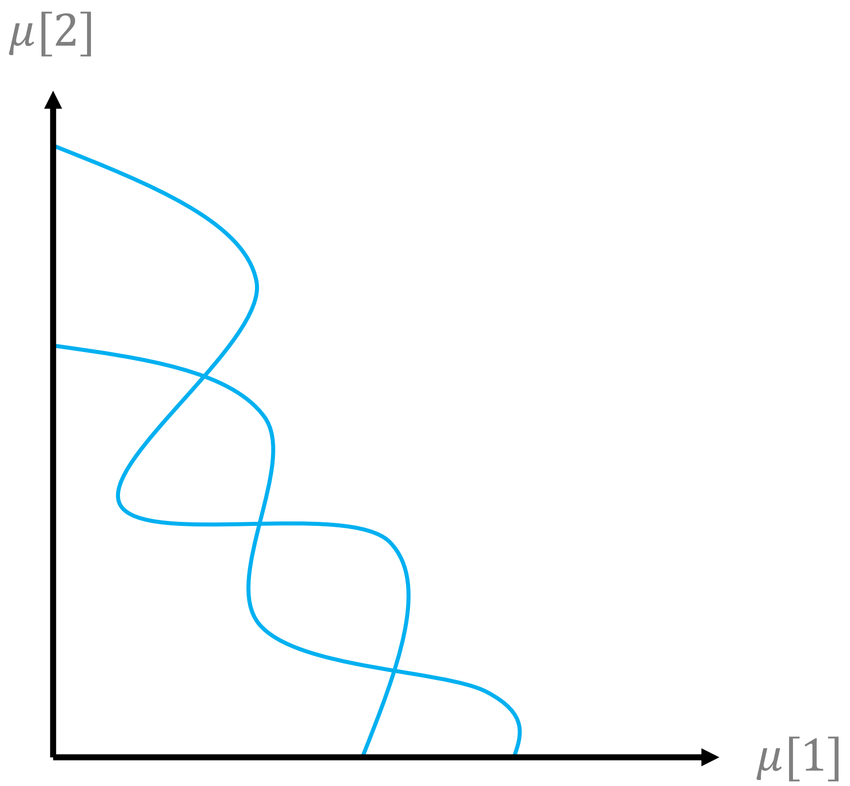
$O(\|A\|_{1,1} + n)$ halfspaces

- In any region defined by intersection of halfspaces: $(\lfloor \boldsymbol{\mu}^T \boldsymbol{a}_1 \rfloor, \ldots, \lfloor \boldsymbol{\mu}^T \boldsymbol{a}_m \rfloor)$ is constant

Balcan, Sandholm, Prasad, **Vitercik**, NeurIPS'21

Beyond Chvátal-Gomory cuts

For more complex families, boundaries can be more complex



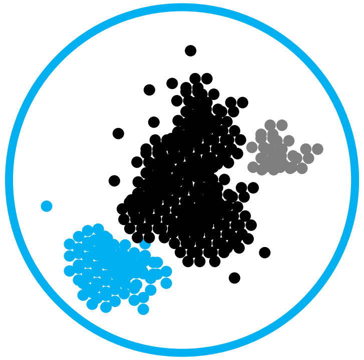
Balcan, Sandholm, Prasad, **Vitercik**, arXiv'22

Outline

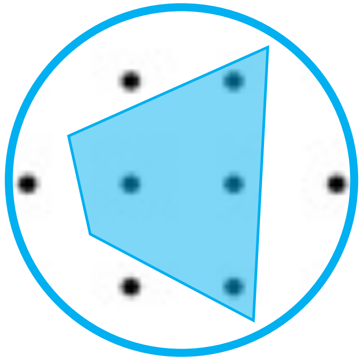
1. Introduction
2. Integer programming
3. **Beyond integer programming**
4. Conclusions

Overview

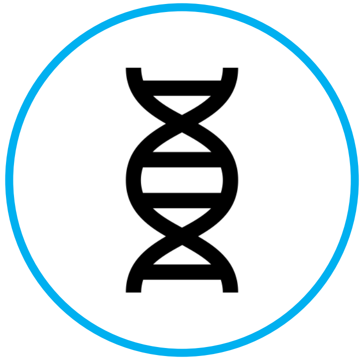
A **unifying** structure connects **seemingly disparate** problems:



Clustering algorithm configuration

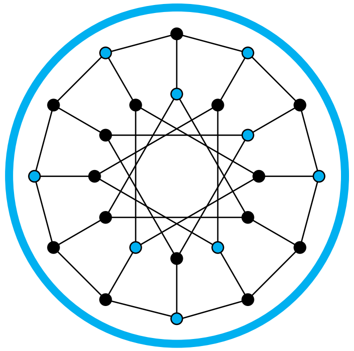


Integer programming algorithm configuration



Computational biology algorithm configuration

Mechanism configuration



Greedy algorithm configuration

Use to provide generalization bounds

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

General algorithm configuration model

$\mathbb{R}^d$: Set of all algorithm parameter settings

$\mathcal{X}$: Set of all algorithm inputs

E.g., integer programs

Algorithmic performance:

$u_{\boldsymbol{\mu}}(x)$ = utility of algorithm parameterized by $\boldsymbol{\mu}$ on input x

E.g., runtime, solution quality, revenue, memory usage …

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Primal & dual classes

$u_{\boldsymbol{\mu}}(x)$ = utility of algorithm parameterized by $\boldsymbol{\mu}$ on input x

$\mathcal{U} = \{u_{\boldsymbol{\mu}}: \mathcal{X} \to \mathbb{R} \mid \boldsymbol{\mu} \in \mathbb{R}^d\}$ "Primal" function class

Typically, prove guarantees by bounding ***complexity*** of $\mathcal{U}$

VC dimension, Rademacher complexity, …

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Primal & dual classes

$u_{\boldsymbol{\mu}}(x)$ = utility of algorithm parameterized by $\boldsymbol{\mu}$ on input x
$\mathcal{U} \;\; = \{u_{\boldsymbol{\mu}}: \mathcal{X} \to \mathbb{R} \,|\, \boldsymbol{\mu} \in \mathbb{R}^d\}$ "Primal" function class

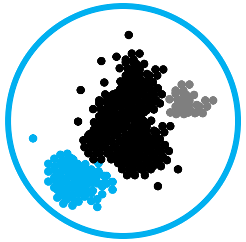
Typically, prove guarantees by bounding ***complexity*** of $\mathcal{U}$

$u_x^*(\boldsymbol{\mu})$ = utility as function of parameters
$u_x^*(\boldsymbol{\mu}) = u_{\boldsymbol{\mu}}(x)$
$\mathcal{U}^* = \{u_x^*: \mathbb{R}^d \to \mathbb{R} \,|\, x \in \mathcal{X}\}$ "Dual" function class

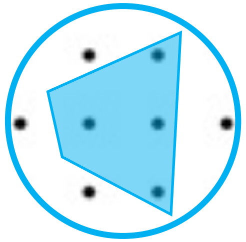
- Dual functions have simple, Euclidean domain
- Often have ample structure can use to bound complexity of $\mathcal{U}$

Piecewise-structured functions

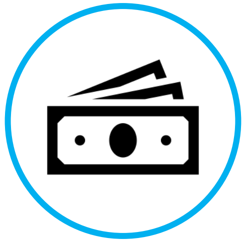
Dual functions $u_x^*: \mathbb{R}^d \to \mathbb{R}$ are **piecewise-structured**



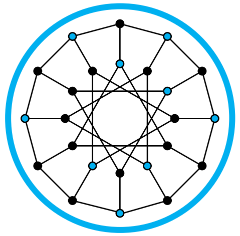
Clustering algorithm configuration



Integer programming algorithm configuration



Selling mechanism configuration



Greedy algorithm configuration

Computational biology algorithm configuration

Voting mechanism configuration

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Generalization to future inputs

Theorem:

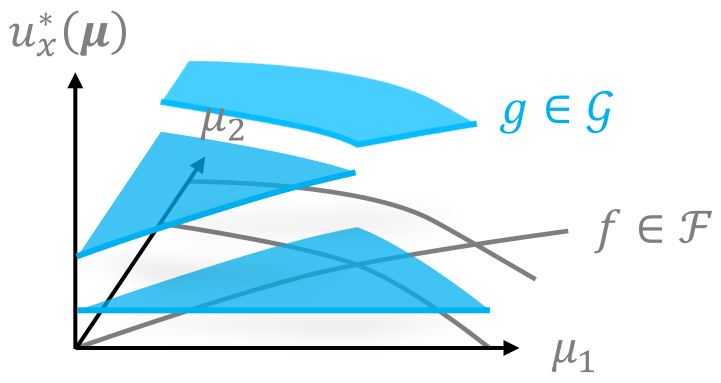
$$\text{Pseudo-dimension}(\mathcal{U}) = \tilde{O}\left(\left(\text{VC}-\text{dim}(\mathcal{F}^*) + \text{P}-\text{dim}(\mathcal{G}^*)\right)\log k\right)$$

boundary functions

Dual of the boundary functions

Dual of the piece functions

Runtime, solution quality, memory usage, …



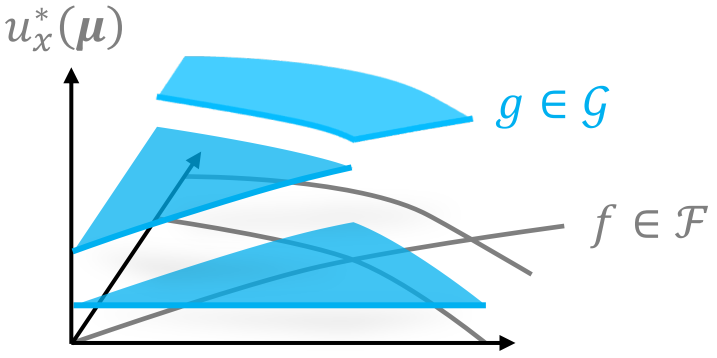
Generalization to future inputs

With high probability, for all $\boldsymbol{\mu}$:

|**Avg** utility on training set – **expected** utility| $= \tilde{O}\left(H\sqrt{\frac{C_{\mathcal{F}^*}+C_{\mathcal{G}^*}}{N}}\right)$

Upper bound on utility

Training set size



Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Application to cutting planes

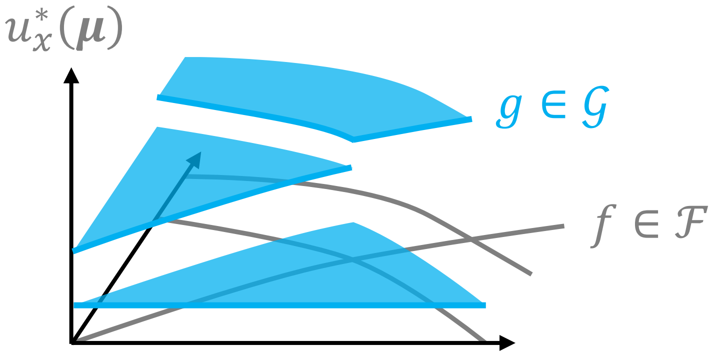
Theorem:

$$\text{Pseudo-dimension}(\mathcal{U}) = \tilde{O}\left(m \log\left(\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n\right)\right)$$

constraints

Max over support of distribution

variables



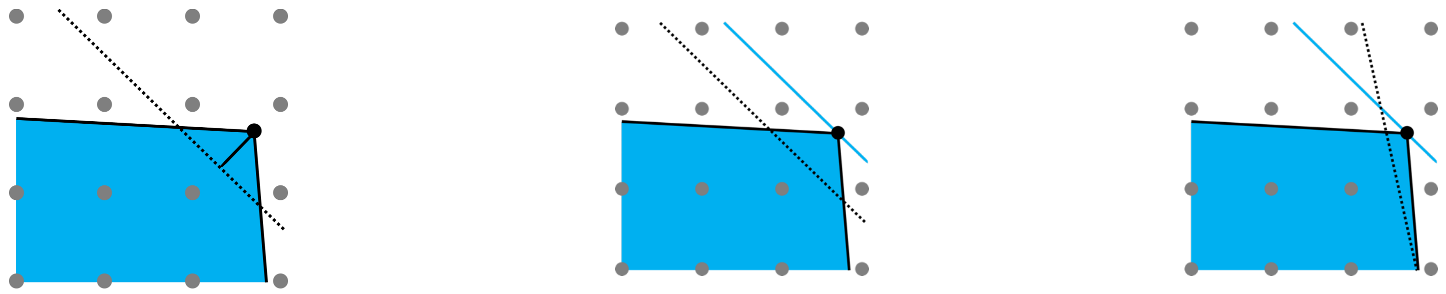
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Outline

1. Introduction
2. Integer programming
3. Beyond integer programming
4. **Conclusions**

Future directions

Existing solvers choose cuts from finite pool using heuristics



Efficacy **Good parallelism** **Worse parallelism**

Machine learning to design new cut selection policies

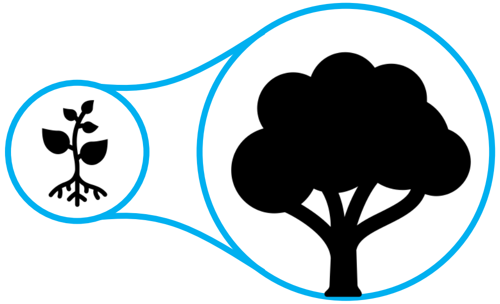
Future directions

Machine-learned algorithms can **scale to larger instances**

Applied research: Dai et al., NeurIPS’17; Agrawal et al., ICML’20; …

Eventually, solve IPs **no one’s ever been able to solve**

More generally, given a single huge IP, how to use ML to solve?



Future directions

Which algorithm classes to optimize over?

Classical algorithm design & analysis

Data-driven algorithm design

Q: *Why* are some (unexpected) configurations dominant?

E.g., Dai et al. [NeurIPS'17] write that their RL alg discovered:

"New and interesting" greedy strategies for MAXCUT and MVC

"which **intuitively make sense** but have **not been analyzed** before,"

thus could be a "good **assistive tool** for discovering new algorithms."

Theoretical foundations of **machine learning** for **cutting plane** selection

Ellen Vitercik

Berkeley (EECS) → **Stanford** (MS&E + CS)

Balcan, Sandholm, Prasad, Vitercik [NeurIPS'21]
Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik [STOC'21]