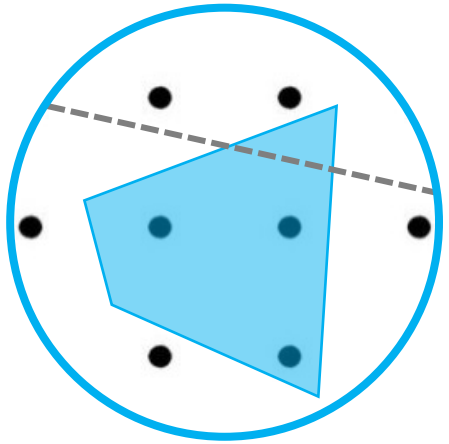
Sample Complexity of **Tree Search** Configuration: **Cutting Planes** and Beyond

Ellen Vitercik

UC Berkeley



An important property of algorithms used in practice is

broad applicability

Example: **Integer programming solvers**

Most popular tool for solving combinatorial (& nonconvex) problems

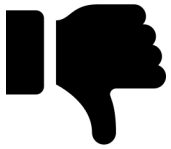
Routing

Manufacturing

Scheduling

Planning

Finance



…but they can have **unsatisfactory** default performance

Slow runtime, poor solutions, …

Integer programming (IP)

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

- CPLEX has **170-page** manual describing **172** parameters
- Tuning by hand is notoriously **slow**, **tedious**, and **error-prone**

Integer programming (IP)

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

- CPLEX has **170-page** manual describing **172** parameters
- Tuning by hand is notoriously **slow**, **tedious**, and **error-prone**

What's the best **configuration** for the application at hand?

Best configuration for **routing** problems
likely not suited for **scheduling**

In practice, we have data about the application domain

Data we could harness to **learn** the best configuration

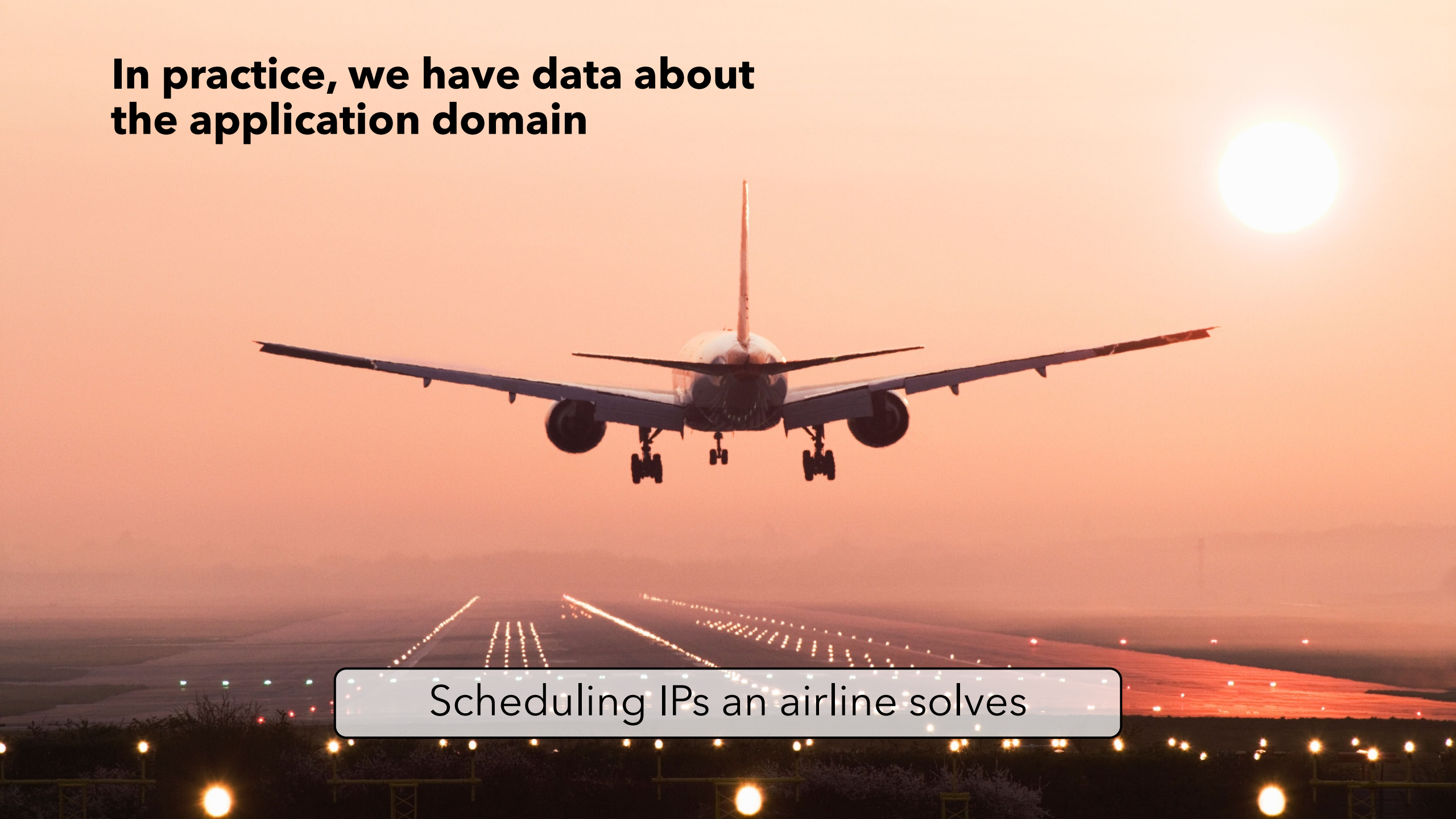
In practice, we have data about the application domain

Routing IPs a shipping company solves

In practice, we have data about the application domain

Clustering IPs a biology lab solves

In practice, we have data about the application domain



Scheduling IPs an airline solves

ML for discrete optimization

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

Horvitz, Ruan, Gomes, Krautz, Selman, Chickering	UAI'01
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)*

Outline

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

Integer program

Integer program (IP)

max $\boldsymbol{c} \cdot \boldsymbol{z}$

s.t. $A\boldsymbol{z} \leq \boldsymbol{b}$

$\boldsymbol{z} \in \mathbb{Z}^n$

Tons of applications:





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 $\hat{\boldsymbol{u}}$ with good avg performance

Search tree size, runtime, etc.

Key question: How to find $\hat{\boldsymbol{u}}$ 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], …

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 $\hat{\boldsymbol{u}}$ with good avg performance

Search tree size, runtime, etc.

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

Convergence

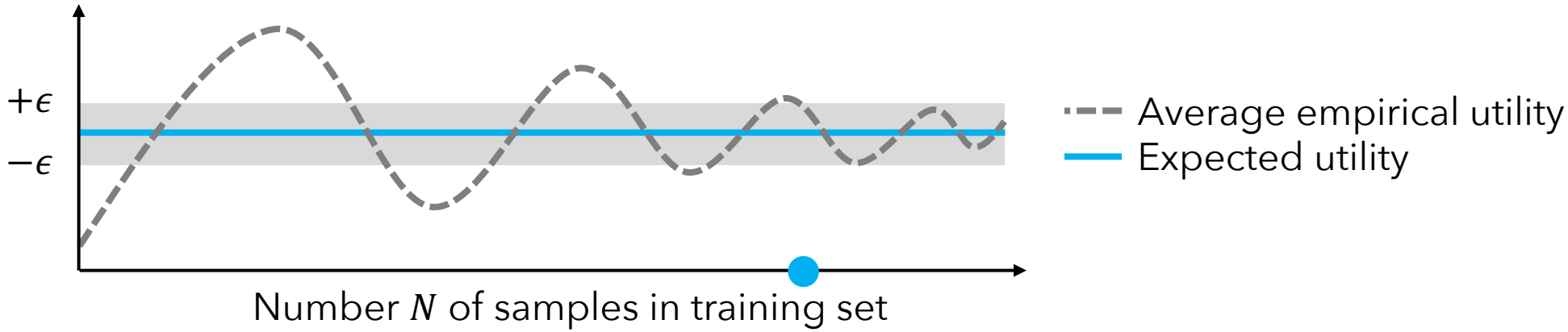
Is the expected utility of $\hat{\boldsymbol{u}}$ also high?

As #samples grows, avg utility converges to expected utility

Key question: How many samples enough to ensure for any $\boldsymbol{u}$, |empirical average utility – expected utility| $\leq \epsilon$?

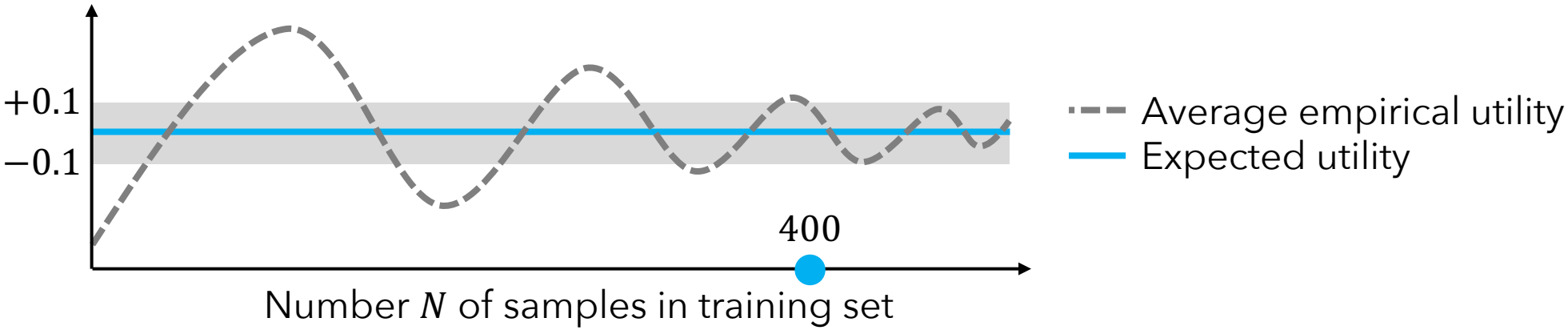
Convergence

Key question: How many samples enough to ensure for any $\boldsymbol{u}$, |empirical average utility – expected utility| $\leq \epsilon$?



Convergence

Key question: How many samples enough to ensure for any $\boldsymbol{u}$, |empirical average utility – expected utility| $\leq \epsilon$?



Convergence

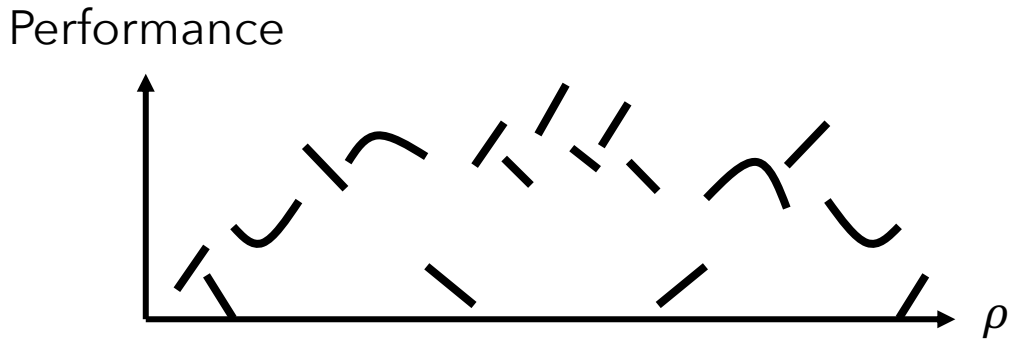
Key question: How many samples enough to ensure for any $\boldsymbol{u}$, |empirical average utility – expected utility| $\leq \epsilon$?

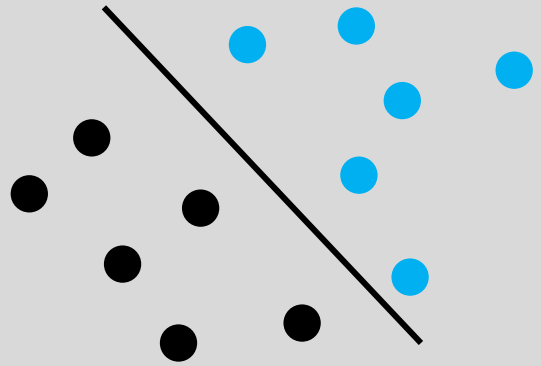
Guarantees apply **no matter how** parameters are configured

Optimally or suboptimally, manually or automatedly

Good **average empirical** utility ➡ Good **expected** utility

Primary challenge in combinatorial domains:
Algorithmic performance is a **volatile** function of parameters
Complex connection between parameters and performance



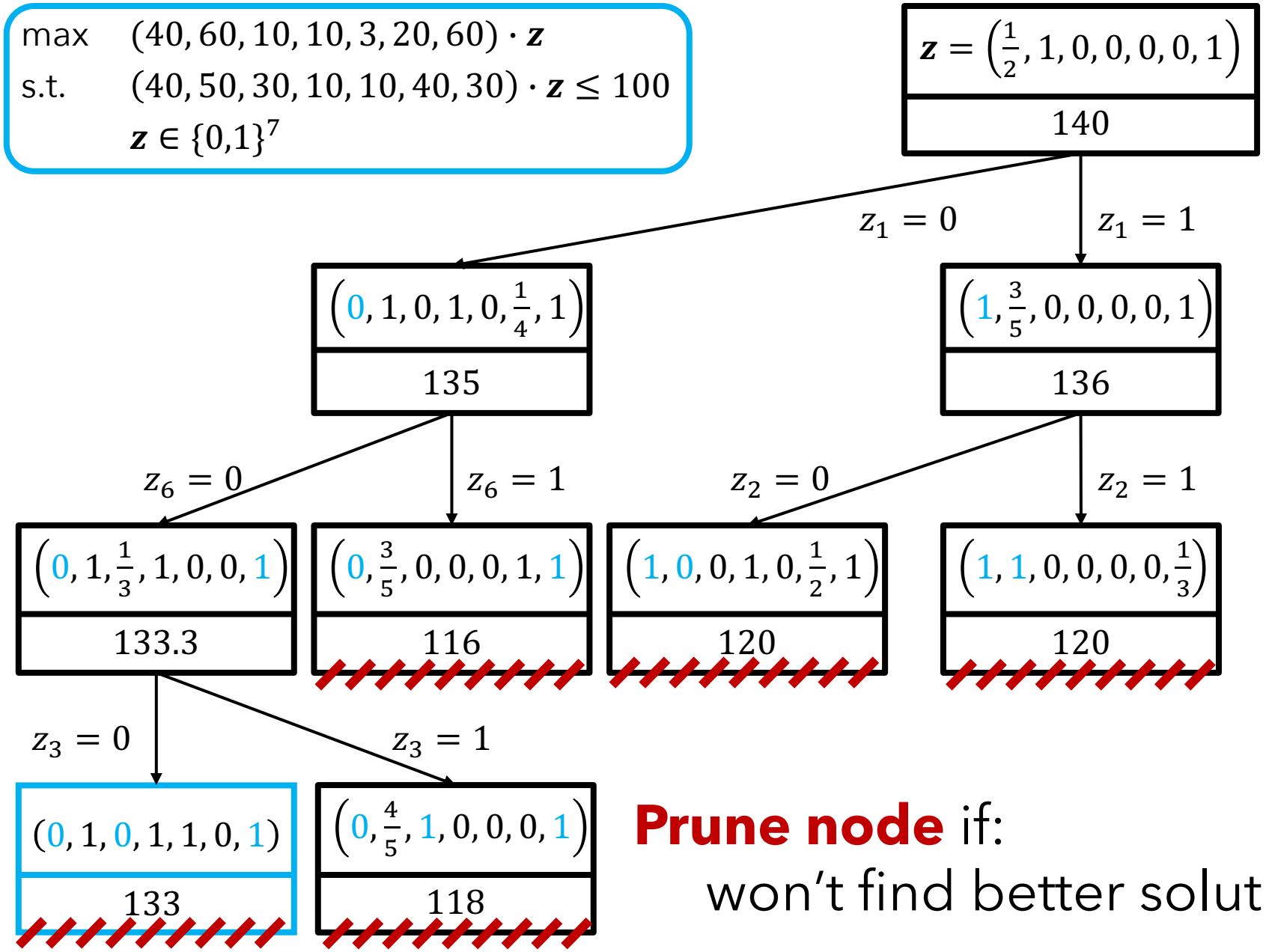


For well-understood functions in machine learning theory:
Simple connection between function parameters and value

Outline

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

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

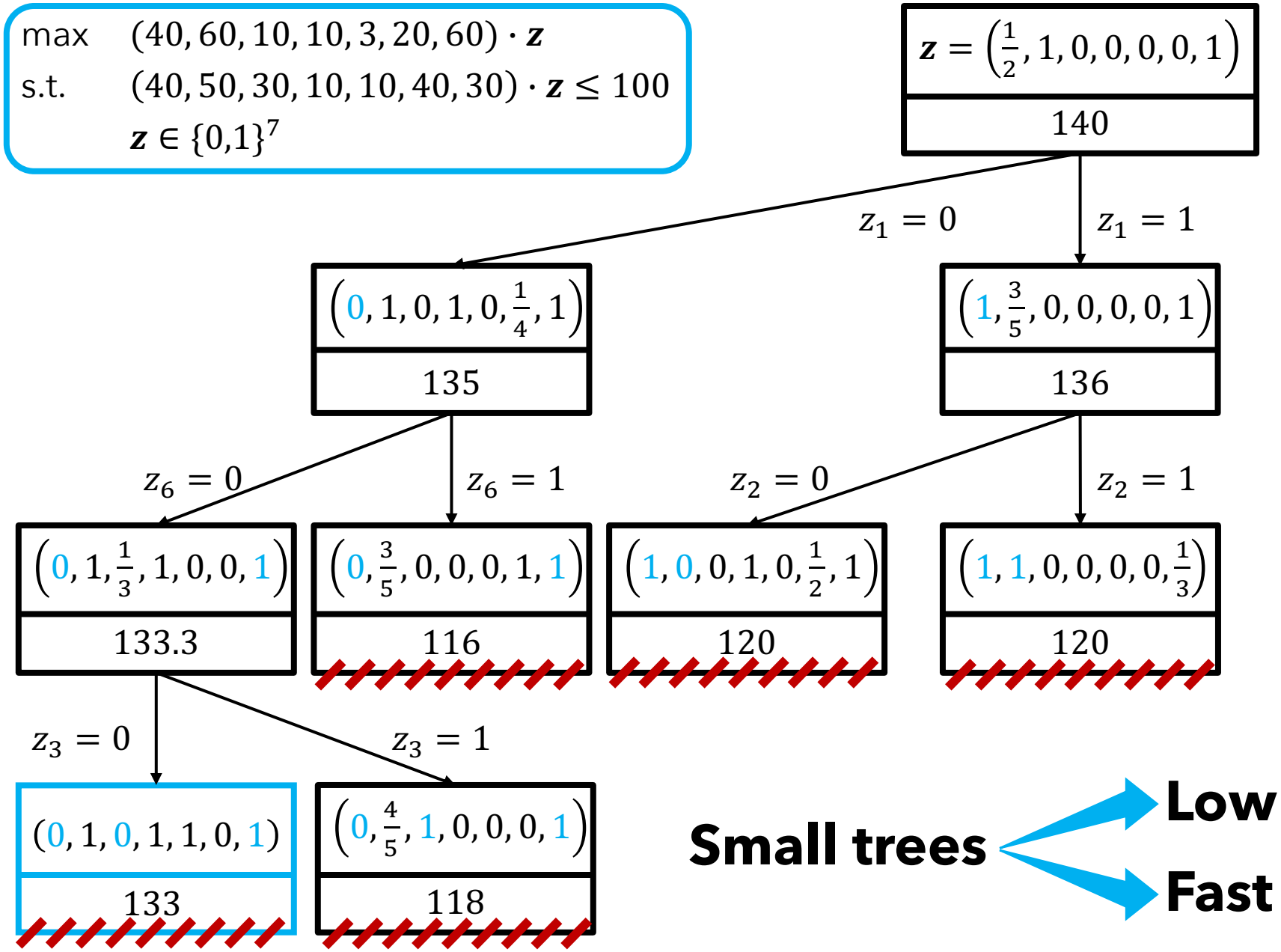


Branch and bound (B&B)

Prune node if:

won't find better solution along branch

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



Branch and bound (B&B)

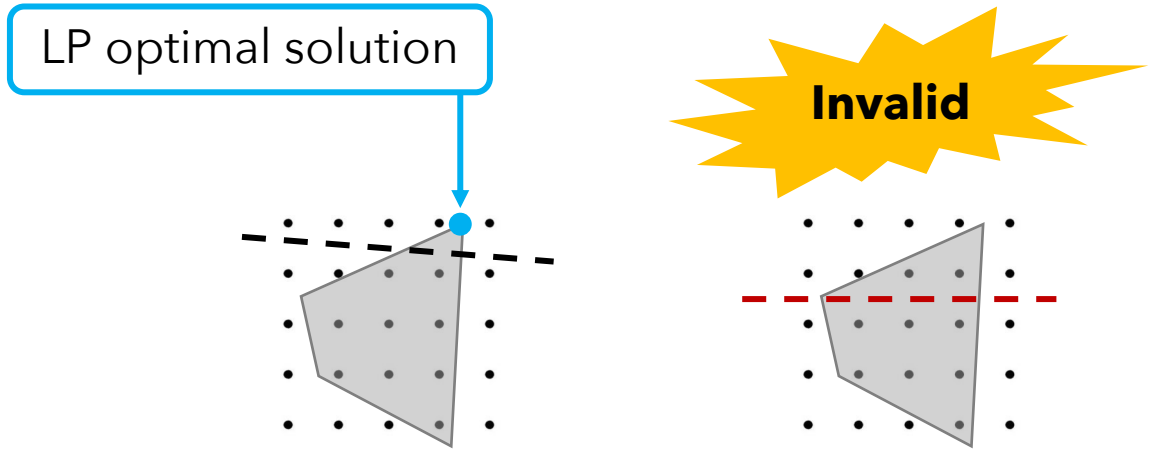
Small trees → **Low memory usage**

Small trees → **Fast runtime**

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?

Outline

1. Introduction
2. Integer programming
    i. Overview of results
    ii. Branch-and-bound
    iii. Our results
        a. Chvátal-Gomory cutting planes
        b. Cut selection policies
        c. Beyond cutting planes
3. Beyond integer programming
4. Conclusions

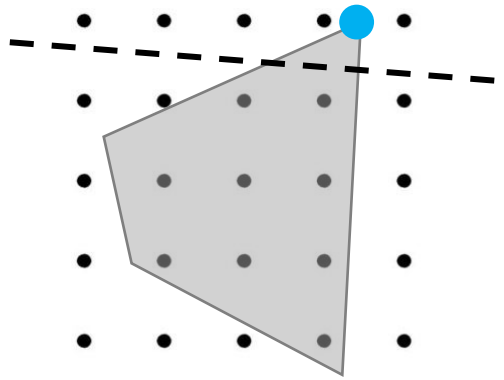
Chvátal-Gomory cuts

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

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

Important properties:

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



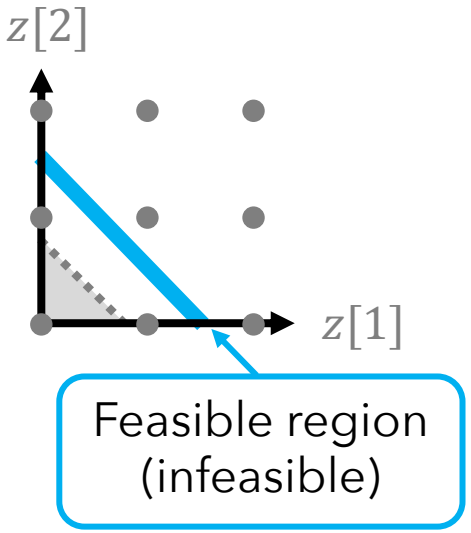
Balcan, Sandholm, Prasad, **Vitercik**, NeurIPS'21

Key challenge: Sensitivity of B&C

Theorem [informal]:

Small changes to $\boldsymbol{u}$ can lead to exponential jumps in tree size

Proof idea:



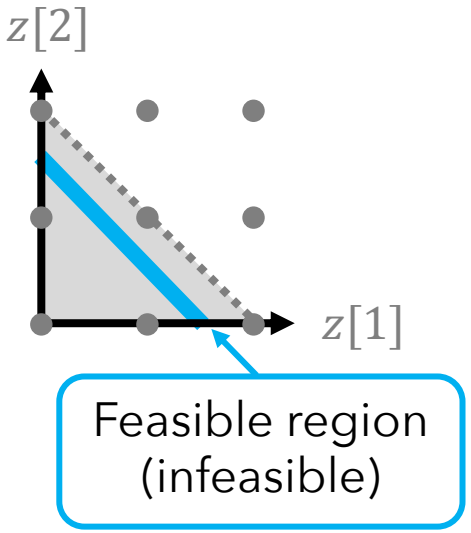
- Vanilla B&B builds a tree of size $2^{O(n)}$
- If $\frac{n+1}{2n} \leq u[1] - u[2] < 1$:
  - CG cut complete removes feasible region
  - B&C terminates immediately

Key challenge: Sensitivity of B&C

Theorem [informal]:

Small changes to $\boldsymbol{u}$ can lead to exponential jumps in tree size

Proof idea:



- Vanilla B&B builds a tree of size $2^{O(n)}$
- If $\frac{n+1}{2n} \leq u[1] - u[2] < 1$:
  - CG cut complete removes feasible region
  - B&C terminates immediately
- Else:
  - CG cut doesn't remove any of the feasible region
  - B&C builds tree of size $2^{O(n)}$

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

Generalization to future inputs

Generalization bound: For all $\boldsymbol{u}$,

|avg performance on training set – expected performance| ≤ **?**

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

Generalization to future inputs

Generalization bound: For all $\boldsymbol{u}$, ← Generalize to sequences and waves of cuts

|avg performance on training set – future performance| ≤ **?**

E.g., tree size

If this difference is small, **no matter** how we choose $\boldsymbol{u} \in \mathbb{R}^m$:

Strong **average** performance ➡ Strong **future** performance

Generalization to future inputs

Generalization bound: For all $\boldsymbol{u}$, ← Generalize to sequences and waves of cuts

|avg performance on training set – future performance| ≤ **?**

E.g., tree size

If this difference is small, **no matter** how we choose $\boldsymbol{u} \in \mathbb{R}^m$:

Strong **average** performance ➡ Strong **future** performance

Theorem: Generalization bound of $\tilde{O}\left(H\sqrt{\frac{m\log(\|A\|_{1,1}+\|\boldsymbol{b}\|_1+n)}{N}}\right)$

Upper bound on performance

Training set size

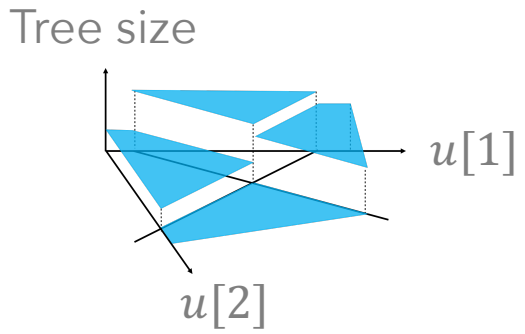
Balcan, Sandholm, Prasad, **Vitercik**, NeurIPS'21

Proof idea

Theorem: Generalization bound of $\tilde{O}\left(H\sqrt{\frac{m\log(\|A\|_{1,1}+\|\boldsymbol{b}\|_1+n)}{N}}\right)$

Proof idea:

- Tree size is a piecewise-constant function of $\boldsymbol{u} \in [0,1)^m$
  - $O(\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n)$ hyperplanes partition $[0,1)^m$ into regions s.t. in any one region, B&C tree is fixed
  - Despite **infinite** number of parameter settings… only **finite set** of with different behavior
- Intuitively, union bound over all regions
  - More formally: pseudo-dimension



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

Piecewise structure

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

Proof idea:

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

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

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

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

Outline

1. Introduction
2. Integer programming
   i. Overview of results
   ii. Branch-and-bound
   iii. Our results
      a. Chvátal-Gomory cutting planes
      b. Cut selection policies
      c. Beyond cutting planes
3. Beyond integer programming
4. Conclusions

Cut selection policies

In practice, cuts are often selected using a

score-based cut selection policy:

1. Compile large set of separating cuts
2. Branch on cut maximizing $\mathbf{score}(\text{cut}) \in \mathbb{R}$

Many options! Little known about which to use when

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

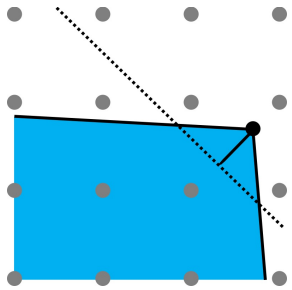
Scoring rule examples

Efficacy [Balas et al., Manage. Sci. '96]**:**

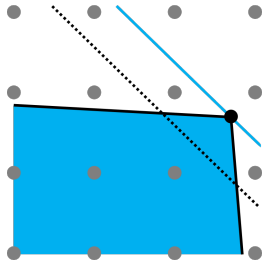
Distance between cut and LP optimum

Parallelism [Achterberg, Thesis'07]**:**

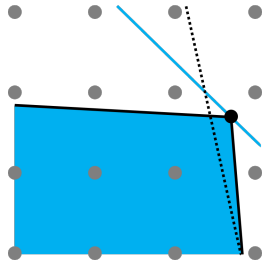
Angle between objective and cut's normal



Efficacy



Good parallelism



Worse parallelism

Learning scoring rules for CG cuts

Given d scoring rules, learn **mixture** $\mu_1 \text{score}_1 + \cdots + \mu_d \text{score}_d$

E.g., leading open-source solver SCIP uses hardcoded weights

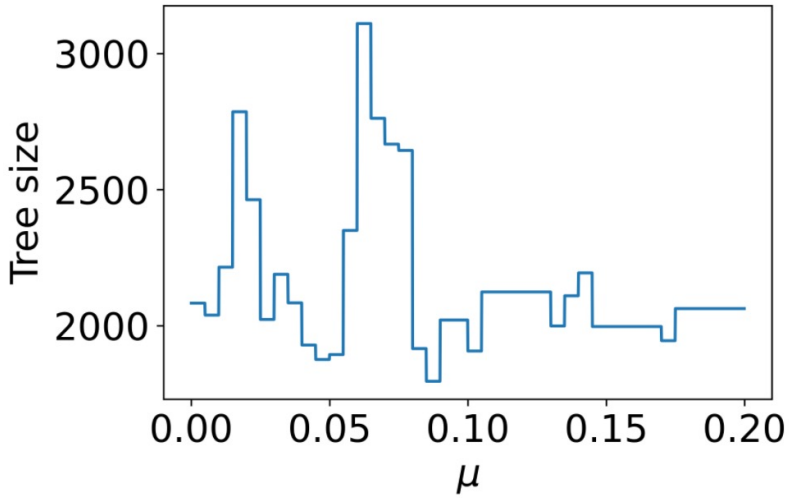
$$\frac{3}{5}\text{score}_1 + \frac{1}{10}\text{score}_2 + \frac{1}{2}\text{score}_3 + \frac{1}{10}\text{score}_4$$

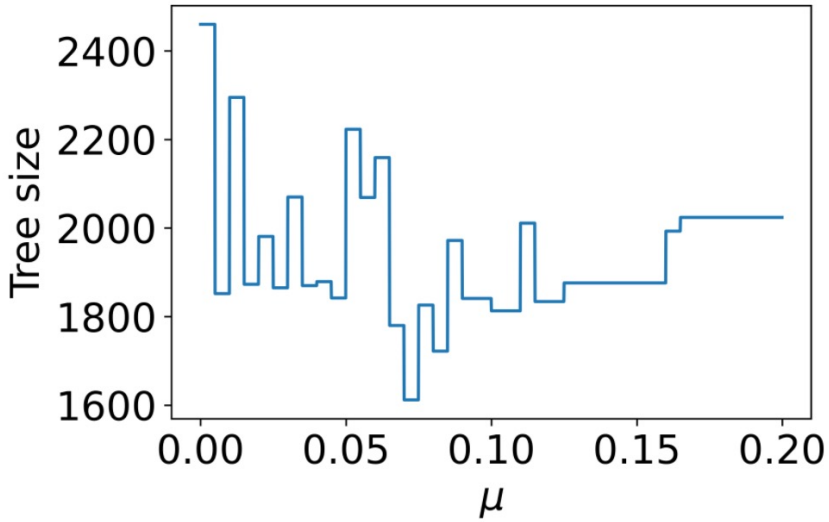
[Gamrath et al., Opt. Online '20]

Learning scoring rules for CG cuts

B&C tree size is a sensitive function of μ

E.g. mixture of $d = 2$ scores: $\mu \cdot \text{score}_1 + (1 - \mu) \cdot \text{score}_2$





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

Generalization to future inputs

Generalization bound: For all $\boldsymbol{\mu}$,

|avg performance on training set – expected performance| ≤ **?**

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

Generalization to future inputs

Generalization bound: For all $\boldsymbol{\mu}$,

|avg performance on training set – future performance| ≤ **?**

E.g., tree size

If this difference is small, **no matter** how we choose $\boldsymbol{\mu} \in \mathbb{R}^d$:

Strong **average** performance ➡ Strong **future** performance

Generalization to future inputs

Generalization bound: For all $\boldsymbol{\mu}$,

|avg performance on training set – future performance| ≤ **?**

E.g., tree size

If this difference is small, **no matter** how we choose $\boldsymbol{\mu} \in \mathbb{R}^d$:

Strong **average** performance ➡ Strong **future** performance

Theorem: Generalization bound of $\tilde{O}\left(H\sqrt{\frac{dmw^2\log(\|A\|_{1,1}+\|\boldsymbol{b}\|_1+n)}{N}}\right)$

Upper bound on performance

sequential cuts at root

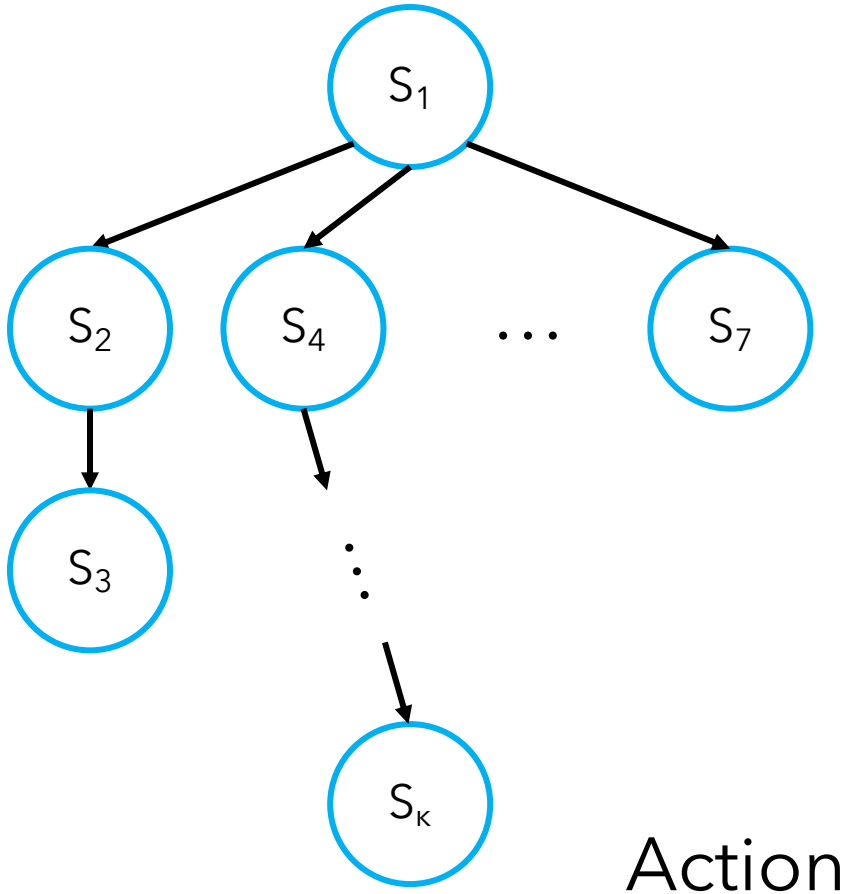
Training set size

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

Outline

1. Introduction
2. Integer programming
   i. Overview of results
   ii. Branch-and-bound
   iii. Our results
      a. Chvátal-Gomory cutting planes
      b. Cut selection policies
      c. Beyond cutting planes
3. Beyond integer programming
4. Conclusions

General tree search



At each iteration:

- Perform *t actions*
  - Node selection
  - Variable selection
  - Cutting plane selection
  - …
- T_j possible actions of type $j = 1, \ldots, t$
- Maximum of κ iterations

Actions chosen using mixtures of scoring rules

Generalization to future inputs

Upper bound on performance | Total # of weights

Theorem: Generalization bound of $\tilde{O}\left(H\sqrt{\frac{d\kappa \sum_{j=1}^{t} \log T_j}{N}}\right)$

Recovers result by Balcan, Dick, Sandholm, **V** [ICML '18]:
Studied *only* variable selection

First guarantee that handles multiple critical aspects of B&C:
Node selection, **variable** selection, and **cut** selection

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

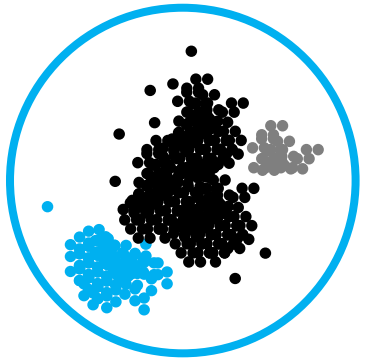
Outline

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

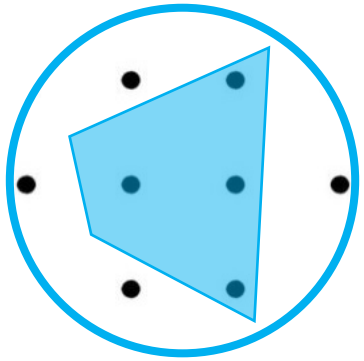
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**	STOC'21
Balcan, Nagarajan, **Vitercik**, White	COLT'17
Balcan, Dick, **Vitercik**	FOCS'18

Overview

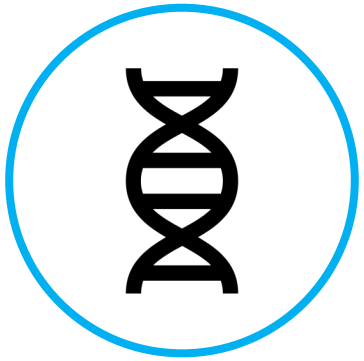
A **unifying** structure connects **seemingly disparate** problems:



Clustering
algorithm
configuration

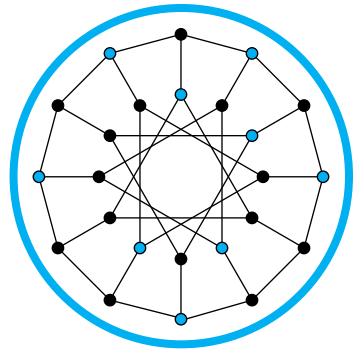


Integer programming
algorithm
configuration



Computational biology
algorithm
configuration

Mechanism
configuration



Greedy
algorithm
configuration

We use this structure to provide extremely **general** guarantees

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

Piecewise structure

Key question:

If configuration has good **avg** performance over a training set,

…will it have good **expected** performance?

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

Piecewise structure

Key question:

If configuration has good **avg** performance over a training set,

…will it have good **future** performance?

Runtime, solution quality, memory usage, revenue …

Answer this question for any parameterized algorithm where:
Performance is a **piecewise-structured** function of parameters

Piecewise constant, linear, polynomial, …

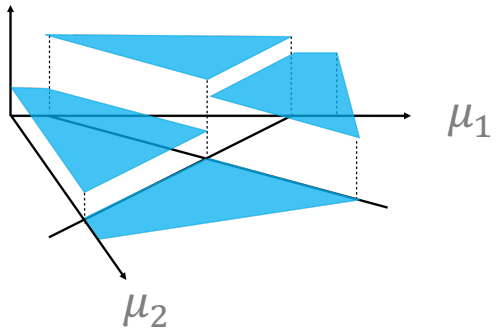
Piecewise structure

Performance is a **piecewise-structured** function of parameters

Piecewise constant, linear, polynomial, …

Integer programming

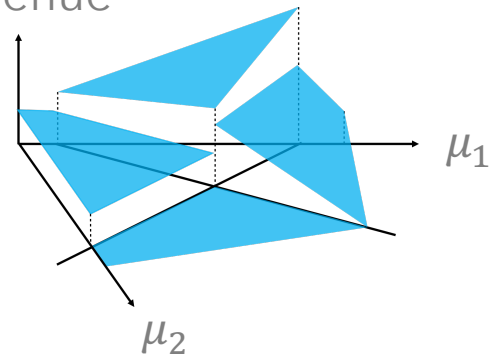
Tree size



Piecewise constant

Mechanism configuration

Revenue



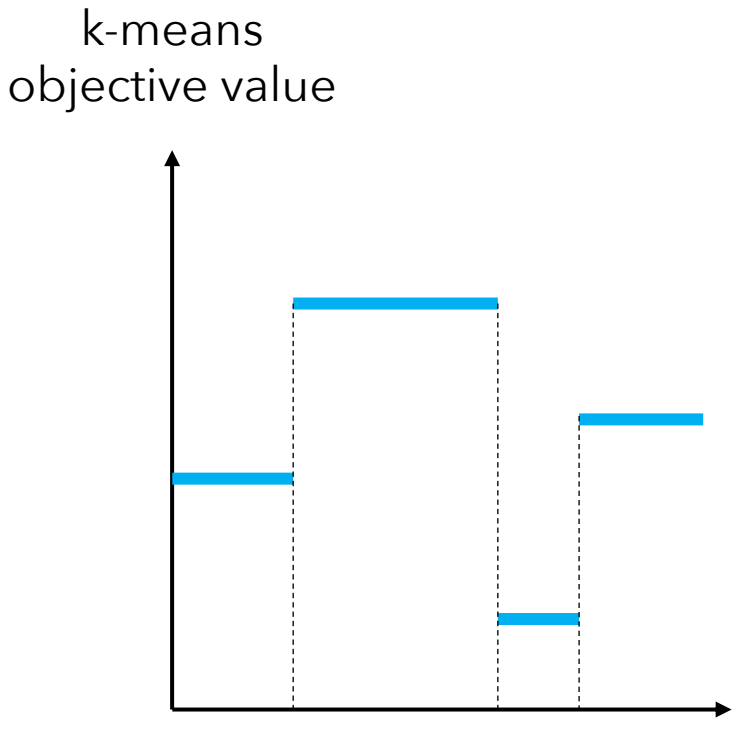
Piecewise linear

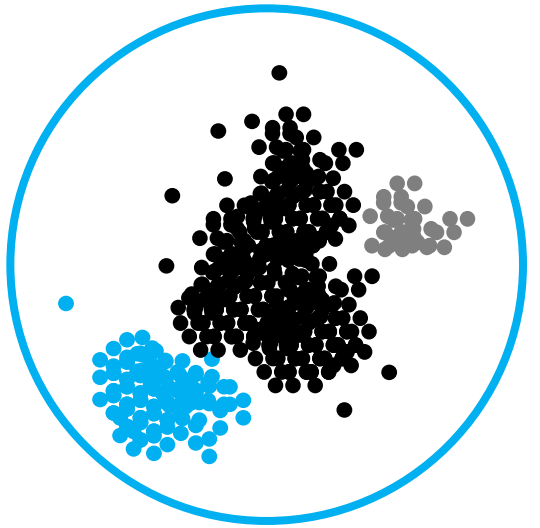
Piecewise …

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

Example: Clustering

Balcan, Nagarajan, **Vitercik**, White [COLT'17]





Interpolates between
complete-, single-, average-linkage

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, pseudo-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

$\mathcal{U} = \left\{u_{\boldsymbol{\mu}} : \mathcal{X} \to \mathbb{R} \,\middle|\, \boldsymbol{\mu} \in \mathbb{R}^d\right\}$ "Primal" function class

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

Challenge: $\mathcal{U}$ is gnarly

E.g., in integer programming:

- Each domain element is an IP
- Unclear how to plot or visualize functions $u_{\boldsymbol{\mu}}$
- No obvious notions of Lipschitz continuity or smoothness to rely on

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

$u_x^*(\boldsymbol{\mu})$ = utility as function of parameters

$u_x^*(\boldsymbol{\mu}) = u_{\boldsymbol{\mu}}(x)$

$\mathcal{U}^* = \{u_x^* : \mathbb{R}^d \to \mathbb{R} \mid x \in \mathcal{X}\}$ "Dual" function class

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

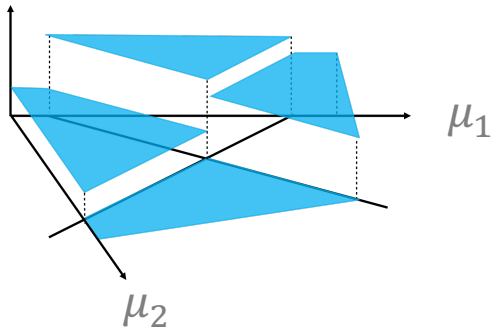
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Piecewise-structured functions

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

Integer programming

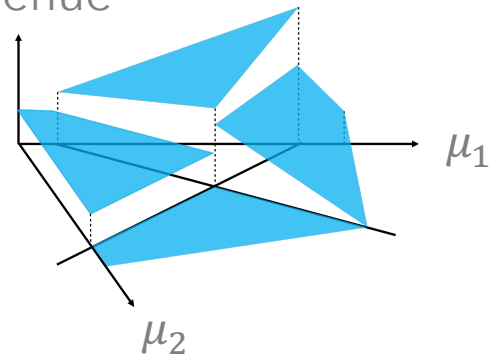
Tree size



Piecewise constant

Mechanism configuration

Revenue



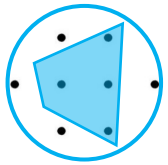
Piecewise linear

Piecewise …

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

Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:



Integer programming

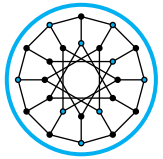
Balcan, Dick, Sandholm, **V**, ICML'18
Balcan, Nagarajan, **V**, White, COLT'17

Clustering

Balcan, Nagarajan, **V**, White, COLT'17
Balcan, Dick, White, NeurIPS'18
Balcan, Dick, Lang, ICLR'20

Computational biology

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



Greedy algorithms

Gupta, Roughgarden, ITCS'16

Mechanism configuration

Balcan, Sandholm, **V**, EC'18

Online configuration [Gupta, Roughgarden, ITCS'16, Cohen-Addad and Kanade, AISTATS'17]

Exploited piecewise-Lipschitz structure to provide regret bounds
[Balcan, Dick, **V**, FOCS'18; Balcan, Dick, Pegden, UAI'20; Balcan, Dick, Sharma, AISTATS'20]

Generalization to future inputs

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

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

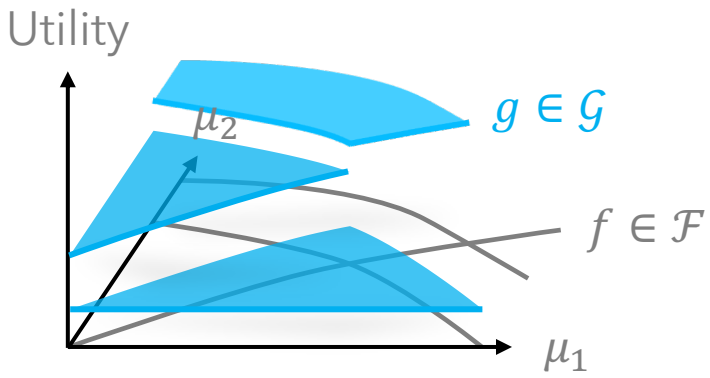
Future

Intrinsic complexities of $\mathcal{F}$ and $\mathcal{G}$

Upper bound on utility

Training set size

Runtime, solution quality, memory usage, …



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

Generalization to future inputs

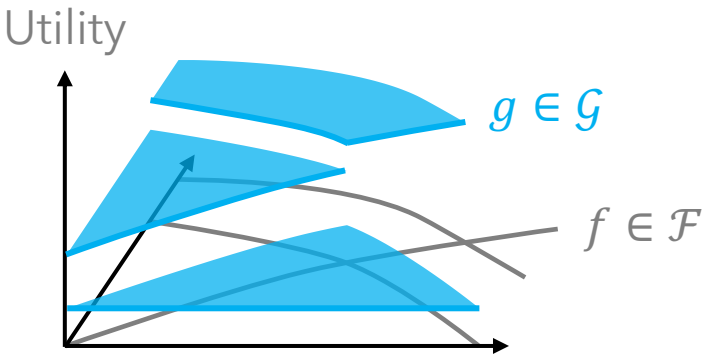
Theorem:

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

boundary functions

Dual of the boundary functions

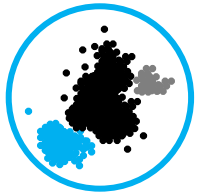
Dual of the piece functions



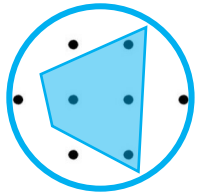
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Main message

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



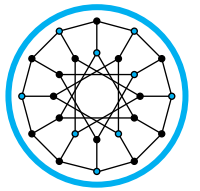
Clustering
algorithm
configuration



Integer programming
algorithm
configuration

Computational biology
algorithm
configuration

Mechanism
configuration



Greedy
algorithm
configuration

We use this structure to provide extremely **general** guarantees

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

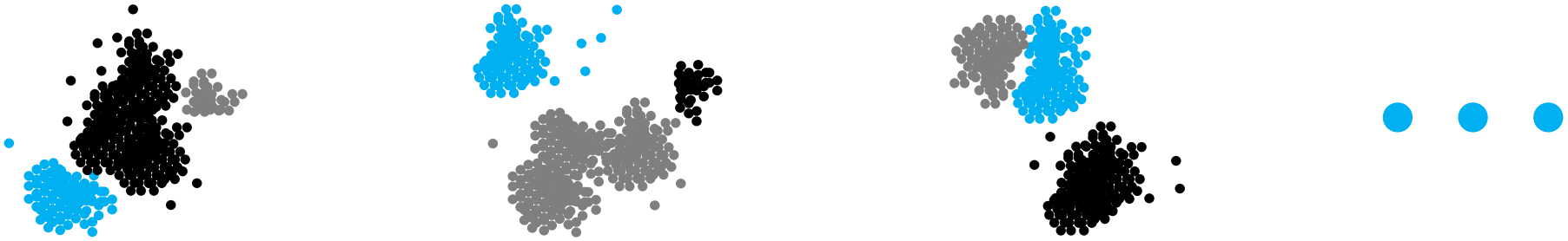
Outline

1. Introduction
2. Integer programming
3. Beyond integer programming
    1. Learning with an i.i.d. training set
    2. **Online learning**
4. Conclusions

Online algorithm configuration

What if inputs are not i.i.d., but even adversarial?

Day 1: μ_1 Day 2: μ_2 Day 3: μ_3



Goal: Compete with best parameter setting in hindsight

- Impossible in the worst case
- Under what conditions is online configuration possible?

Gupta, Roughgarden, ITCS'16; Cohen-Addad, Kanade, AISTATS'17; Balcan, Dick, **Vitercik**, FOCS'18; Balcan, Dick, Pegden, UAI'20; …

Outline

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

Conclusions

Study B&C, the most popular tool for combinatorial problems

- 1st guarantees for learning high-performing cuts & selection policies

Generalized our guarantees to a general model of tree search

- Handles multiple critical aspects of B&C:
  Node selection, **variable** selection, and **cut** selection

Results that unify a diverse array of configuration problems

- Clustering, IP solving, comp bio, greedy algorithm configuration…

Sample Complexity of **Tree Search** Configuration: **Cutting Planes** and Beyond

Ellen Vitercik

UC Berkeley