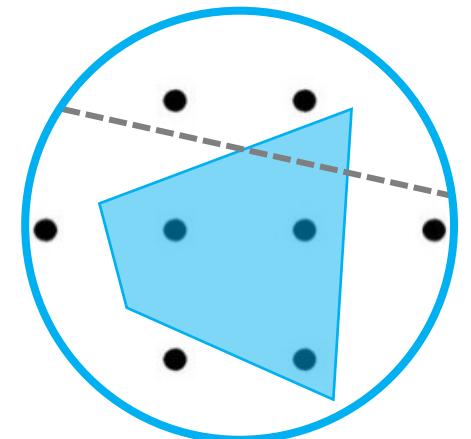


# Sample Complexity of **Tree Search** Configuration: **Cutting Planes** and Beyond

**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



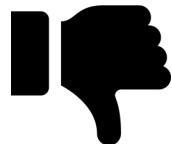
Scheduling



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**

CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELIM 160	CPX_PARAM_RANDOMSEED 130	CPXPARAM_MIP_Pool_RelGap 148	CPX_PARAM_FLOWCOVERS 70	CPX_PARAM_BRDIR 39
CPX_PARAM_NODELIM 101	CPX_PARAM_TUNINGDETTILIM 160	CPX_PARAM_REDUCE 131	CPXPARAM_MIP_Pool_Replace 151	CPX_PARAM_FLOWPATHS 71	CPX_PARAM_BTTLIM 40
CPX_PARAM_NODESEL 102	CPX_PARAM_TUNINGDISPLAY 162	CPX_PARAM_REINV 131	CPXPARAM_MIP_Strategy_Branch 39	CPX_PARAM_FPHEUR 72	CPX_PARAM_CALCQCPDUALS 41
CPX_PARAM_NUMERICALEMPHASIS 102	CPX_PARAM_TUNINGMEASURE 163	CPX_PARAM_RELAXPREIND 132	CPXPARAM_MIP_Strategy_MIQCPStrat 93	CPX_PARAM_FRACAND 73	CPX_PARAM_CLIQUES 42
CPX_PARAM_NZREADLIM 103	CPX_PARAM_TUNINGREPEAT 164	CPX_PARAM_RELOBJDIF 133	CPXPARAM_MIP_Strategy_StartAlgorithm 139	CPX_PARAM_FRACCUTS 73	CPX_PARAM_CLOCKTYPE 43
CPX_PARAM_OBJDIF 104	CPX_PARAM_TUNINGTILIM 165	CPX_PARAM_REPAIRTRIES 133	CPXPARAM_MIP_Strategy_VariableSelect 166	CPX_PARAM_FRACPASS 74	CPX_PARAM_CLONELOG 43
CPX_PARAM_OBLLIM 105	CPX_PARAM_VARSEL 166	CPX_PARAM_REPEATPRESOLVE 134	CPXPARAM_MIP_SubMIP_NodeLimit 155	CPX_PARAM_GUBCOVERS 75	CPX_PARAM_COEREDIND 44
CPX_PARAM_OBJULIM 105	CPX_PARAM_WORKDIR 167	CPX_PARAM_RINSHEUR 135	CPXPARAM_OptimalityTarget 106	CPX_PARAM_HEURFREQ 76	CPX_PARAM_COLREADLIM 45
CPX_PARAM_PARALLELMODE 108	CPX_PARAM_WORKMEM 168	CPX_PARAM_RLT 136	CPXPARAM_Output_WriteLevel 169	CPX_PARAM_IMPLBD 76	CPX_PARAM_CONFLICTDISPLAY 46
CPX_PARAM_PERIND 110	CPX_PARAM_WRITELEVEL 169	CPX_PARAM_ROWREADLIM 141	CPXPARAM_Preprocessing_Aggregator 19	CPX_PARAM_INTSOLFILEPREFIX 78	CPX_PARAM_COVERS 47
CPX_PARAM_PERLIM 111	CPX_PARAM_ZEROHALFCUTS 170	CPX_PARAM_SCAIND 142	CPXPARAM_Preprocessing_Fill 19	CPX_PARAM_INTSOLLIM 79	CPX_PARAM_CPMASK 48
CPX_PARAM_POLISHAFTERDETTIME 111	CPXPARAM_Benders_Strategy 30	CPX_PARAM_SCRIND 143	CPXPARAM_Preprocessing_Linear 120	CPX_PARAM_ITLIM 80	CPX_PARAM_CRAIND 50
CPX_PARAM_POLISHAFTEREPAGAP 112	CPXPARAM_Benders_Tolerances_feasibilitycut 35	CPX_PARAM_SIFTALG 143	CPXPARAM_Preprocessing_Reduce 131	CPX_PARAM_LANDPCUTS 82	CPX_PARAM_CUTLIM 51
CPX_PARAM_POLISHAFTEREPGAP 113	CPXPARAM_Benders_Tolerances_optimalitycut 36	CPX_PARAM_SIFTDISPLAY 144	CPXPARAM_Preprocessing_Symmetry 156	CPX_PARAM_LBHEUR 81	CPX_PARAM_CUTPASS 52
CPX_PARAM_POLISHAFTERINTSOL 114	CPXPARAM_Conflict_Algorithm 46	CPX_PARAM_SIFTITLIM 145	CPXPARAM_Read_DataCheck 54	CPX_PARAM_LPMETHOD 136	CPX_PARAM_CUTSFATOR 52
CPX_PARAM_POLISHAFTERNODE 115	CPXPARAM_CPUmask 48	CPX_PARAM_SIMDISPLAY 145	CPXPARAM_Read_Scale 142	CPX_PARAM_MCPFCUTS 82	CPX_PARAM_CUTUP 53
CPX_PARAM_POLISHAFTERTIME 116	CPXPARAM_DistMIP_Rampup_Duration 128	CPX_PARAM_SINGLIM 146	CPXPARAM_ScreenOutput 143	CPX_PARAM_MEMORYEMPHASIS 83	CPX_PARAM_DATACHECK 54
CPX_PARAM_POLISHTIME (deprecated) 116	CPXPARAM_LPMETHOD 136	CPX_PARAM_SOLNPOOLAGAP 146	CPXPARAM_Sifting_Algorithm 143	CPX_PARAM_MIPCBREDLP 84	CPX_PARAM_DEPIND 55
CPX_PARAM_POPULATELIM 117	CPXPARAM_MIP_Cuts_BQP 38	CPX_PARAM_SOLNPOOLCAPACITY 147	CPXPARAM_Sifting_Display 144	CPX_PARAM_MIPDISPLAY 85	CPX_PARAM_DETTILIM 56
CPX_PARAM_PPRIND 118	CPXPARAM_MIP_Cuts_LocalImpplied 77	CPX_PARAM_SOLNPOOLGAP 148	CPXPARAM_Sifting_Iterations 145	CPX_PARAM_MIPMEMPHASIS 87	CPX_PARAM_DISJCUTS 57
CPX_PARAM_PREDUAL 119	CPXPARAM_MIP_Cuts_RLT 136	CPX_PARAM_SOLNPOOLINTENSITY 149	CPXPARAM_Simplex_Display 145	CPX_PARAM_MIPINTERVAL 88	CPX_PARAM_DIVETYPE 58
CPX_PARAM_PREIND 120	CPXPARAM_MIP_Cuts_ZeroHalfCut 170	CPX_PARAM_SOLNPOOLREPLACE 151	CPXPARAM_Simplex_Limits_Singularity 146	CPX_PARAM_MIPKAPPASTATS 89	CPX_PARAM_DPRIIND 59
CPX_PARAM_PRELINEAR 120	CPXPARAM_MIP_Limits_CutsFactor 52	CPX_PARAM_SOLUTIONTARGET	CPXPARAM_SolutionType 152	CPX_PARAM_MIPORDIND 90	CPX_PARAM_EACHCUTLIM 60
CPX_PARAM_PREPASS 121	CPXPARAM_MIP_Limits_RampupDefTimeLimit 127	deprecated: see CPXPARAM_OptimalityTarget 106	CPXPARAM_Threads 157	CPX_PARAM_MIPORDTYPE 91	CPX_PARAM_EPAGAP 61
CPX_PARAM_PRESLVND 122	CPXPARAM_MIP_Limits_Solutions 79	CPXPARAM_TUNETYPE 152	CPXPARAM_TimeLimit 159	CPX_PARAM_MIPSEARCH 92	CPX_PARAM_EPGAP 61
CPX_PARAM_PRICELIM 123	CPXPARAM_MIP_Limits_StrongCand 154	CPX_PARAM_STARTALG 139	CPXPARAM_Tune_DefTimeLimit 160	CPX_PARAM_MIQCPSTRAT 93	CPX_PARAM_EPINT 62
CPX_PARAM_PROBE 123	CPXPARAM_MIP_Limits_StrongIt 154	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Display 162	CPX_PARAM_MIRCUTS 94	CPX_PARAM_EPMRK 64
CPX_PARAM_PROBEDETTIME 124	CPXPARAM_MIP_Limits_TreeMemory 160	CPX_PARAM_STRONGITLIM 154	CPXPARAM_Tune_Measure 163	CPX_PARAM_MPSSLONGNUM 94	CPX_PARAM_EPOPT 65
CPX_PARAM_PROBETIME 124	CPXPARAM_MIP_OrderType 91	CPX_PARAM_SUBALG 99	CPXPARAM_Tune_Repeat 164	CPX_PARAM_NETDISPLAY 95	CPX_PARAM_EPPER 65
CPX_PARAM_QPMAKEPSDIND 125	CPXPARAM_MIP_Pool_AbsGap 146	CPX_PARAM_SUBMIPNODELIMIT 155	CPXPARAM_Tune_TimeLimit 165	CPX_PARAM_NETEPOPT 96	CPX_PARAM_EPRELAX 66
CPX_PARAM_QPMETHOD 138	CPXPARAM_MIP_Pool_Capacity 147	CPX_PARAM_SYMMETRY 156	CPXPARAM_WorkDir 167	CPX_PARAM_NETEPRS 96	CPX_PARAM_EPRIHS 67
CPX_PARAM_QPNZREADLIM 126	CPXPARAM_MIP_Pool_Intensity 149	CPX_PARAM_THREADS 157	CPXPARAM_WorkMem 168	CPX_PARAM_NETFIND 97	CPX_PARAM_FEASOPTMODE 68
		CPX_PARAM_TILIM 159	CraInd 50	CPX_PARAM_NETITLIM 98	CPX_PARAM_FILEENCODING 69

# 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**



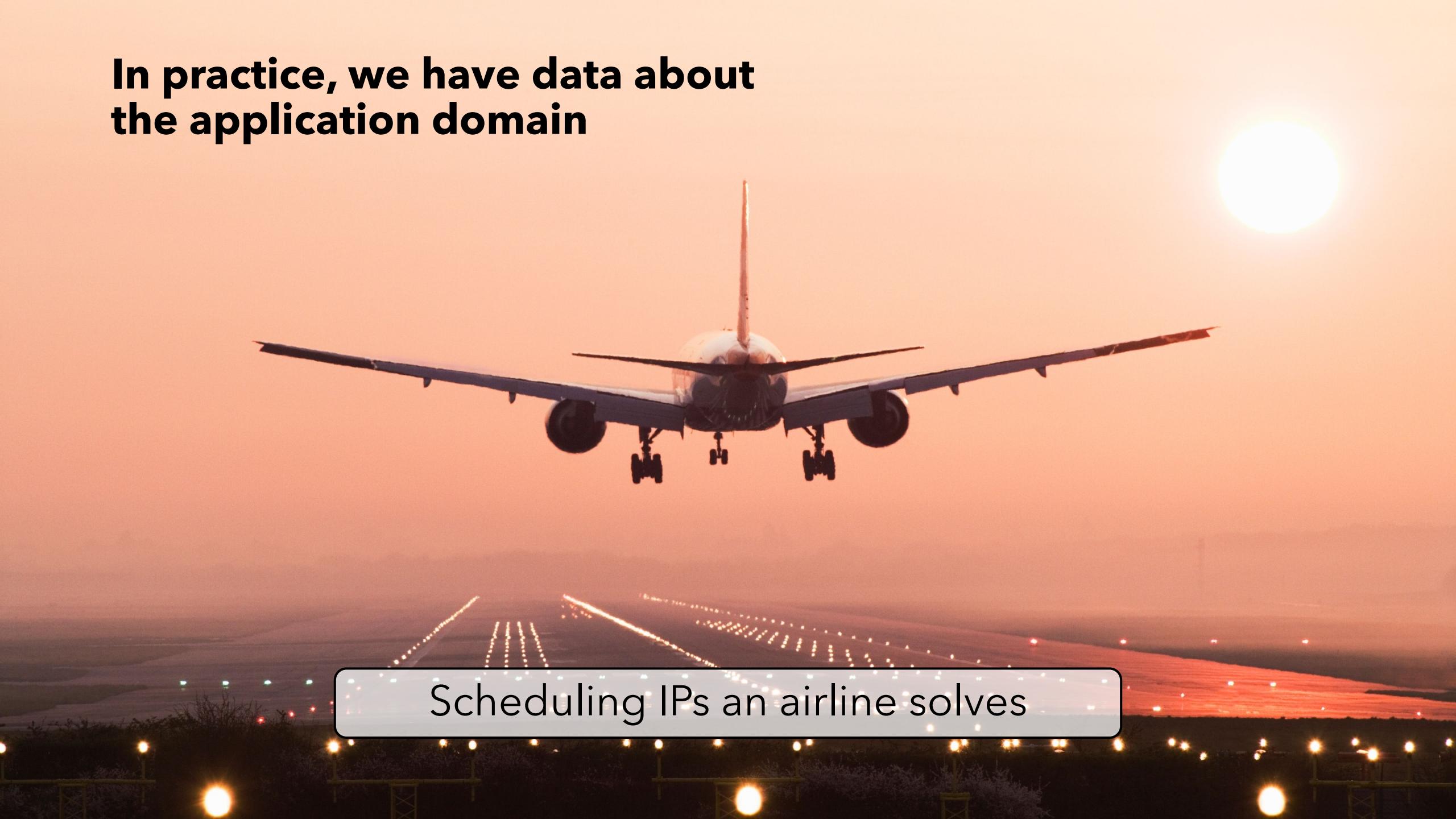
Routing IPs a shipping company solves

**In practice, we have data about  
the application domain**



Clustering IPs a biology lab solves

**In practice, we have data about  
the application domain**

A photograph of a large commercial airplane, likely a Boeing 777, captured from a low angle looking up. The plane is silhouetted against a bright, orange-hued sky at sunset or sunrise. Its landing gear is deployed, and it is positioned directly above a runway. The runway is marked with glowing white and yellow lights. In the foreground, there's some dark ground with scattered lights from airport infrastructure. A white rectangular callout box is overlaid on the bottom left of the image, containing the text.

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 Sandholm	JAIR'09 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)**

$$\max \quad \mathbf{c} \cdot \mathbf{z}$$

$$\text{s.t.} \quad A\mathbf{z} \leq \mathbf{b}$$

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

**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  $\mathcal{D}$

$$\{A^{(1)}, \mathbf{b}^{(1)}, \mathbf{c}^{(1)}\}$$

$$\{A^{(2)}, \mathbf{b}^{(2)}, \mathbf{c}^{(2)}\}$$

$$\{A^{(3)}, \mathbf{b}^{(3)}, \mathbf{c}^{(3)}\}$$

$$\{A^{(4)}, \mathbf{b}^{(4)}, \mathbf{c}^{(4)}\}$$



3. Return parameter settings  $\hat{\mathbf{u}}$  with good avg performance

**Search tree size, runtime, etc.**

**Key question:** How to find  $\hat{\mathbf{u}}$  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  $\mathcal{D}$

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


3. Return parameter settings  $\hat{\mathbf{u}}$  with good avg performance

**Search tree size, runtime, etc.**

**Focus of this talk:** Will  $\hat{\mathbf{u}}$  have good **future** performance?

**More formally:** Is the expected utility of  $\hat{\mathbf{u}}$  also high?



# Convergence

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

As #samples grows, avg utility converges to expected utility

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

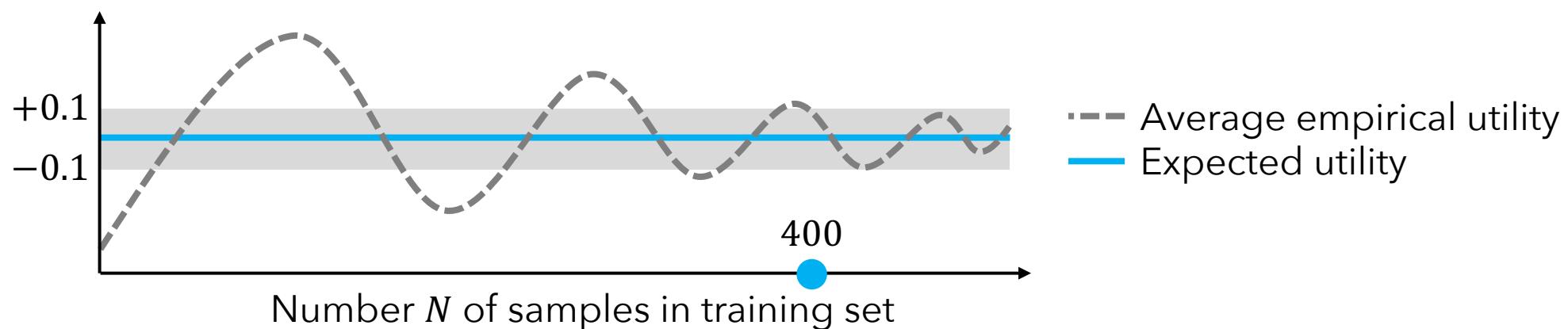
# Convergence

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



# Convergence

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



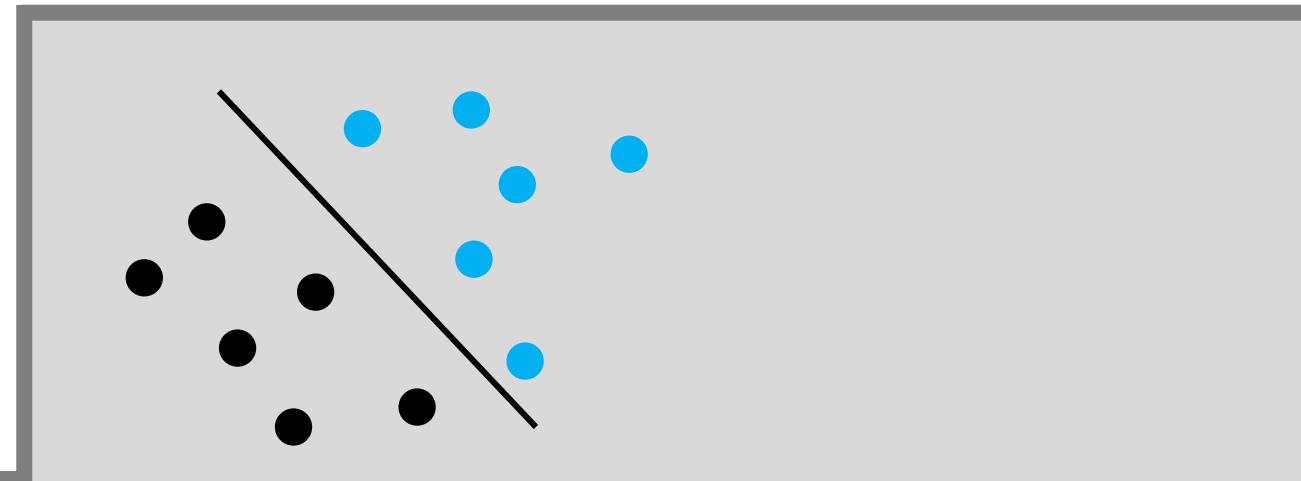
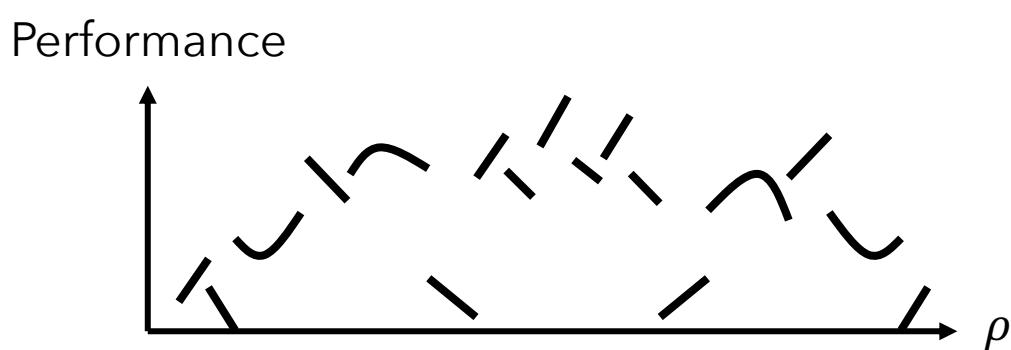
# Convergence

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

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

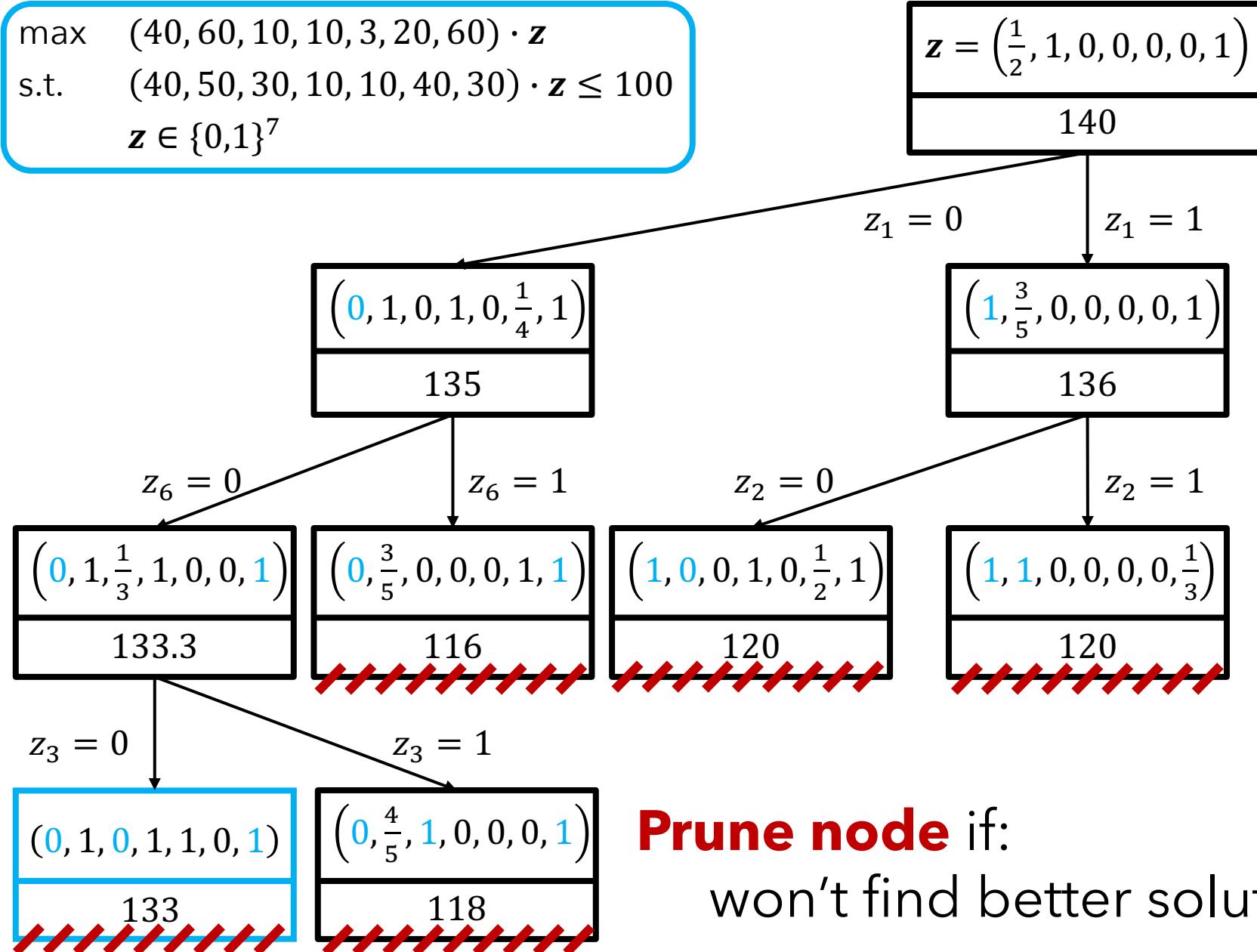


For well-understood functions in machine learning theory:  
**Simple** connection between function parameters and value

# Outline

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

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

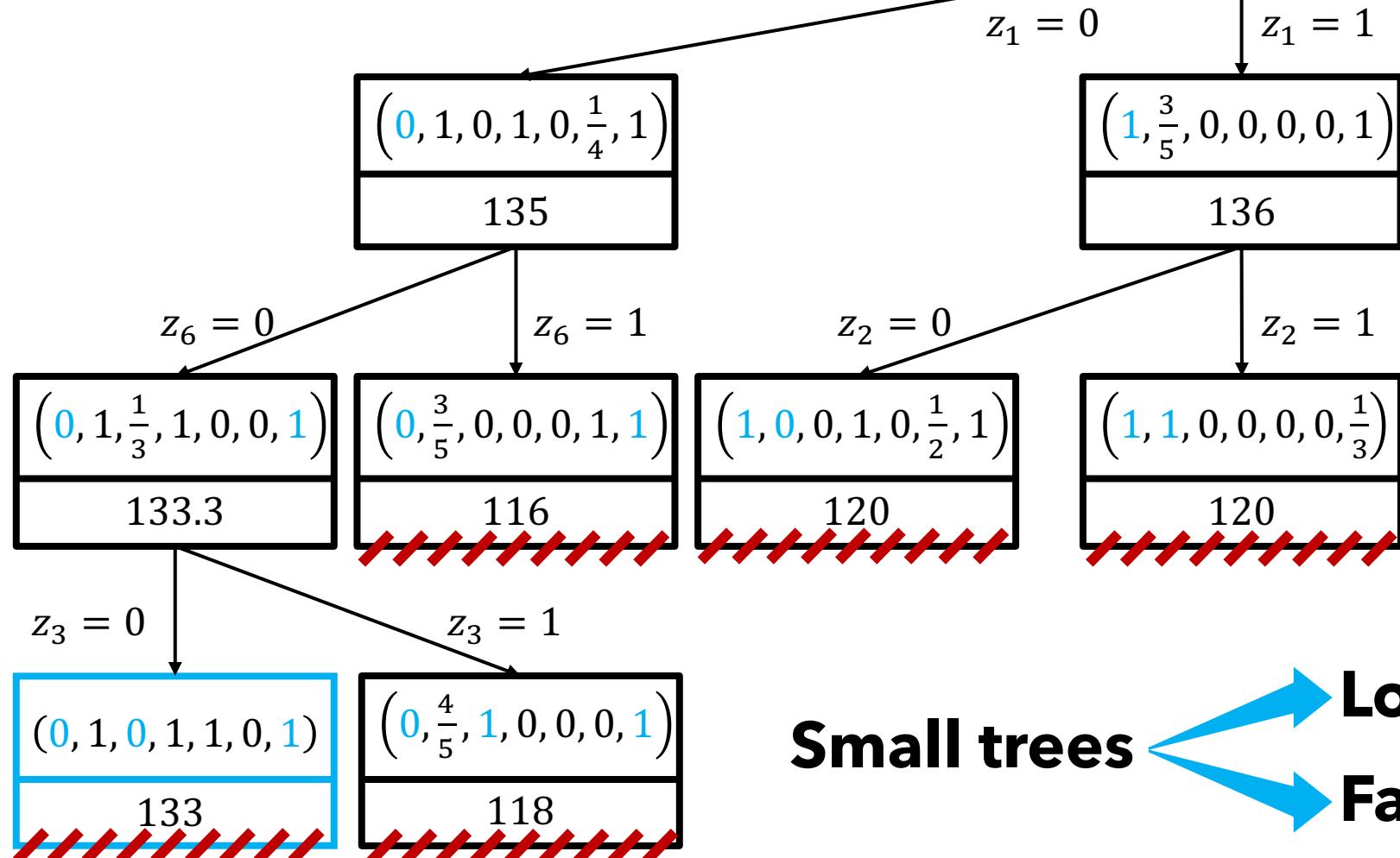


# Branch and bound (B&B)

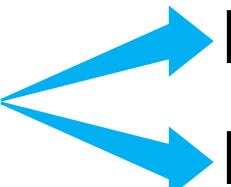
**Prune node** if:  
won't find better solution along branch

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

$$\begin{array}{c}
 \mathbf{z} = \left( \frac{1}{2}, 1, 0, 0, 0, 0, 1 \right) \\
 \hline
 140
 \end{array}$$



Branch  
and  
bound  
(B&B)

**Small trees** 

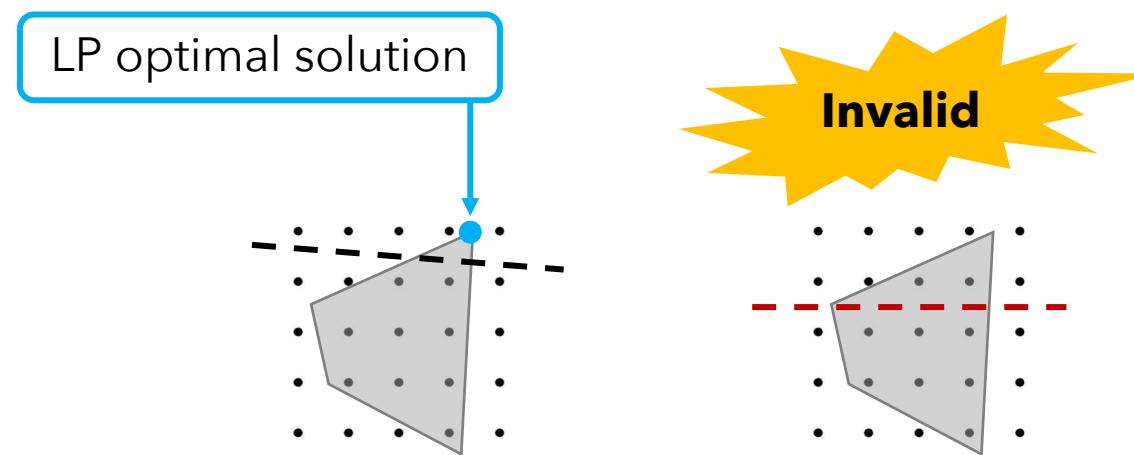
**Low memory usage**

**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



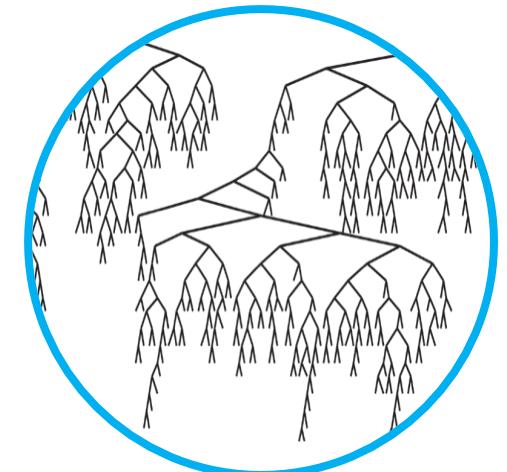
# Cutting planes

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

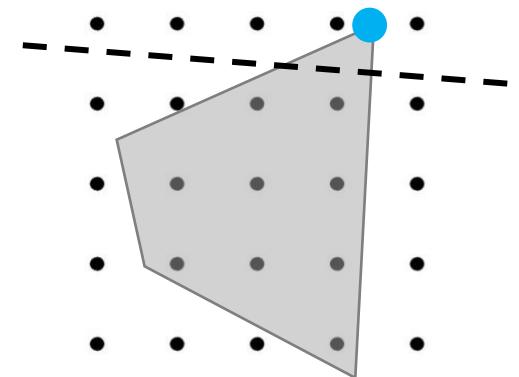
# Chvátal-Gomory cuts

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

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

## Important properties:

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

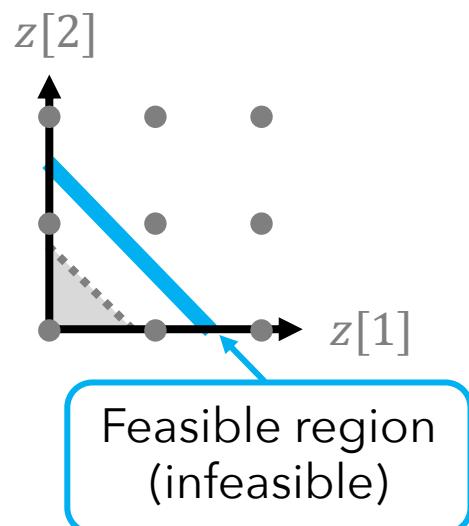


# Key challenge: Sensitivity of B&C

**Theorem** [informal]:

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

*Proof idea:*



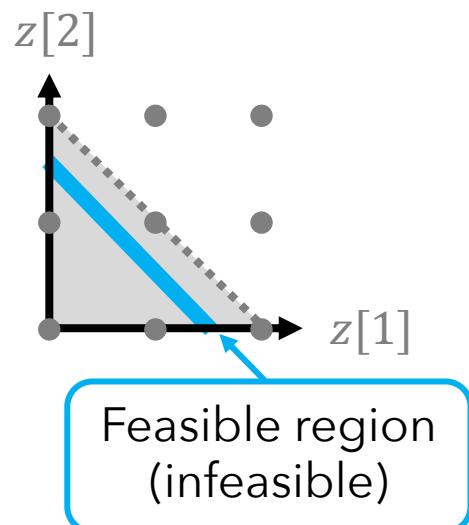
- Vanilla B&B builds a tree of size  $2^{O(n)}$
- If  $\frac{n+1}{2n} \leq u[1] - u[2] < 1$ :
  - CG cut completely removes feasible region
  - B&C terminates immediately

# Key challenge: Sensitivity of B&C

**Theorem** [informal]:

Small changes to  $\mathbf{u}$  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 completely removes feasible region
  - B&C terminates immediately
- Else:
  - CG cut doesn't remove any of the feasible region
  - B&C builds tree of size  $2^{O(n)}$

# Generalization to future inputs

**Generalization bound:** For all  $u$ ,

|avg performance on training set - expected performance|  $\leq ?$

# Generalization to future inputs

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

$$|\text{avg performance on training set} - \underbrace{\text{future performance}}_{\text{E.g., tree size}}| \leq ?$$

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

Strong **average** performance  $\rightarrow$  Strong **future** performance



# Generalization to future inputs

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

$$|\text{avg performance on training set} - \text{future performance}| \leq ?$$

E.g., tree size

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

Strong **average** performance  $\Rightarrow$  Strong **future** performance

$$\text{Upper bound on performance}$$
$$\tilde{\mathcal{O}}\left(H \sqrt{\frac{m \log(\|A\|_{1,1} + \|b\|_1 + n)}{N}}\right)$$

↑  
Training set size

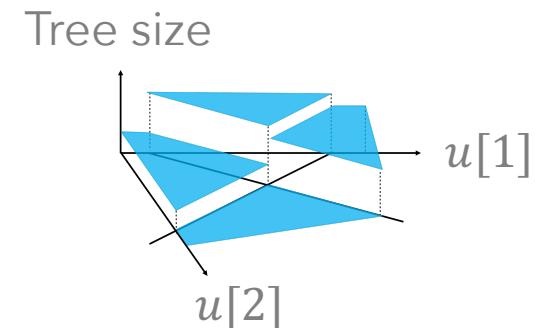
**Theorem:** Generalization bound of

# Proof idea

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

*Proof idea:*

- Tree size is a piecewise-constant function of  $\mathbf{u} \in [0,1]^m$ 
  - $O(\|A\|_{1,1} + \|b\|_1 + n)$  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



# Piecewise structure

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

*Proof idea:*

- CG cut parameterized by  $\mathbf{u} \in [0,1)^m$  is  $[\mathbf{u}^T A] \mathbf{z} \leq [\mathbf{u}^T \mathbf{b}]$
- For any  $\mathbf{u}$  and column  $\mathbf{a}_i$ ,  $[\mathbf{u}^T \mathbf{a}_i] \in [-\|\mathbf{a}_i\|_1, \|\mathbf{a}_i\|_1]$
- For each integer  $k_i \in [-\|\mathbf{a}_i\|_1, \|\mathbf{a}_i\|_1]$ :  
$$[\mathbf{u}^T \mathbf{a}_i] = k_i \text{ iff } k_i \leq \mathbf{u}^T \mathbf{a}_i < k_i + 1$$
- In any region defined by intersection of halfspaces:  
 $([\mathbf{u}^T \mathbf{a}_1], \dots, [\mathbf{u}^T \mathbf{a}_m])$  is constant

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

# 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  $\text{score}(\text{cut}) \in \mathbb{R}$

**Many** options! Little known about which to use when

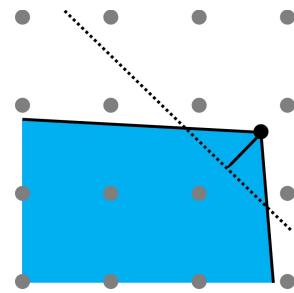
# Scoring rule examples

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

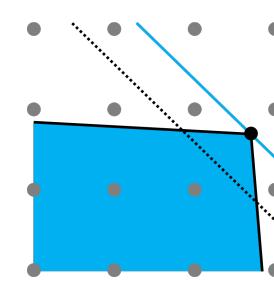
Distance between cut and LP optimum

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

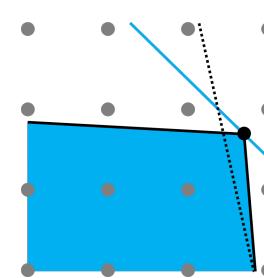
Angle between objective and cut's normal



**Efficacy**



**Good parallelism**



**Worse parallelism**

# Learning scoring rules for CG cuts

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

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

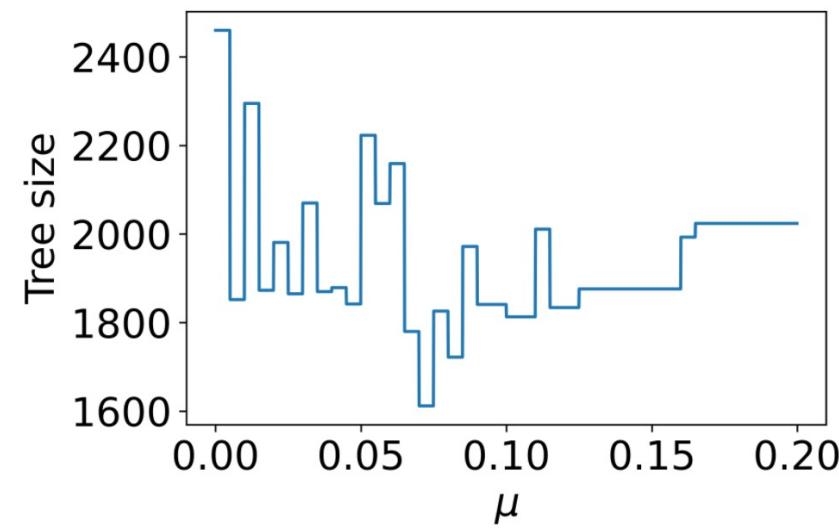
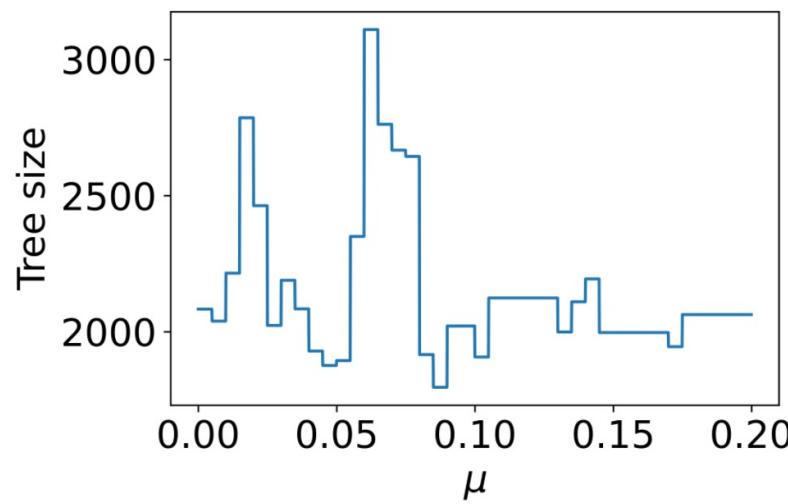
$$\frac{3}{5} \text{score}_1 + \frac{1}{10} \text{score}_2 + \frac{1}{2} \text{score}_3 + \frac{1}{10} \text{score}_4$$

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

# Learning scoring rules for CG cuts

B&C tree size is a sensitive function of  $\mu$

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



# Generalization to future inputs

**Generalization bound:** For all  $\mu$ ,

|avg performance on training set - expected performance|  $\leq ?$

# Generalization to future inputs

**Generalization bound:** For all  $\mu$ ,

$$|\text{avg performance on training set} - \underbrace{\text{future performance}}_{\text{E.g., tree size}}| \leq ?$$

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

Strong **average** performance  $\rightarrow$  Strong **future** performance



# Generalization to future inputs

**Generalization bound:** For all  $\mu$ ,

$$|\text{avg performance on training set} - \text{future performance}| \leq ?$$

E.g., tree size

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

Strong **average** performance  $\Rightarrow$  Strong **future** performance

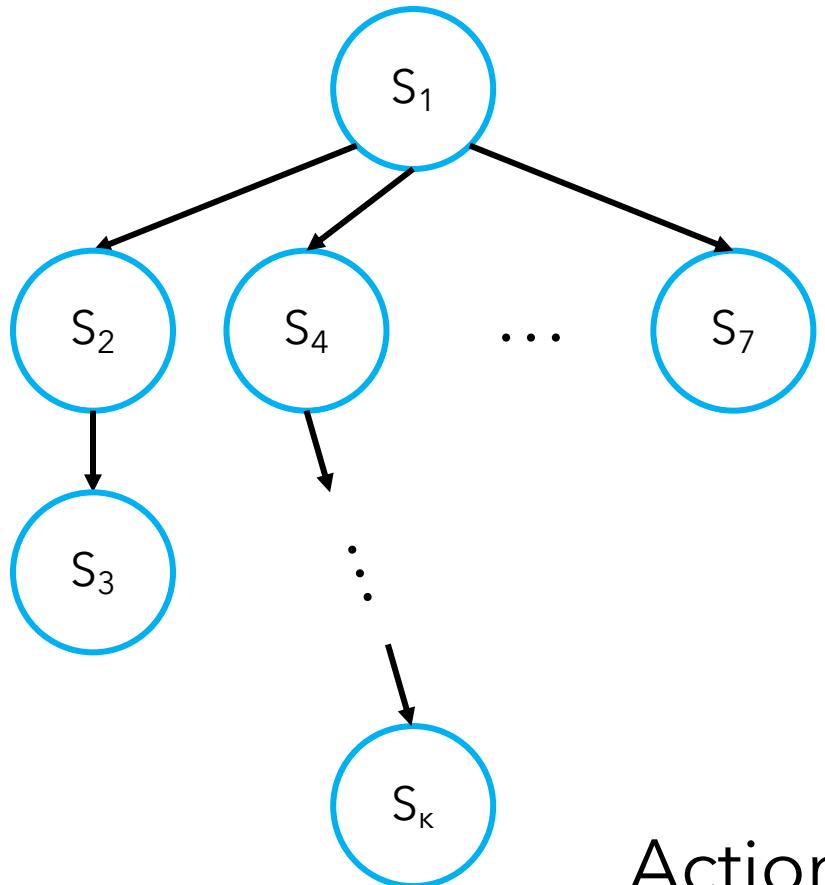
**Theorem:** Generalization bound of  $\tilde{O}\left(H \sqrt{\frac{dmw^2 \log(\|A\|_{1,1} + \|b\|_1 + n)}{N}}\right)$

The diagram illustrates the components of the generalization bound formula. At the top, two boxes are shown: 'Upper bound on performance' and '# sequential cuts at root'. Arrows point from these boxes down to a large square root term in the formula. Below the square root term, an arrow points up to the variable 'N', which is labeled 'Training set size' in a box at the bottom right.

# 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



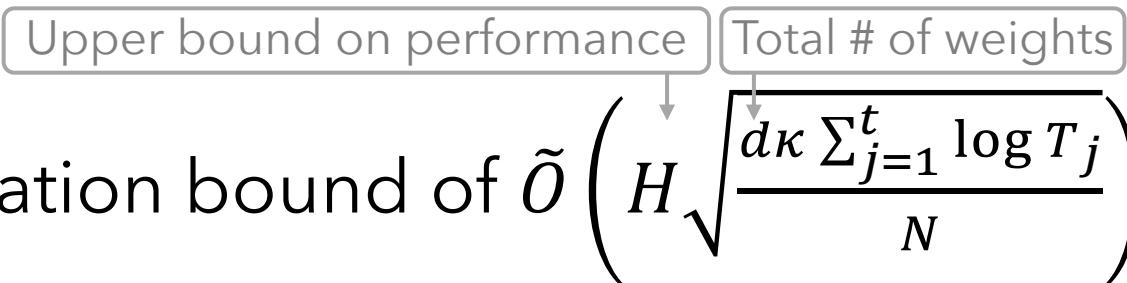
At each iteration:

- Perform  $t$  actions
  - Node selection
  - Variable selection
  - Cutting plane selection
  - ...
- $T_j$  possible actions of type  $j = 1, \dots, t$
- Maximum of  $\kappa$  iterations

Actions chosen using mixtures of scoring rules

# Generalization to future inputs

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



Upper bound on performance      Total # of weights

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

# Outline

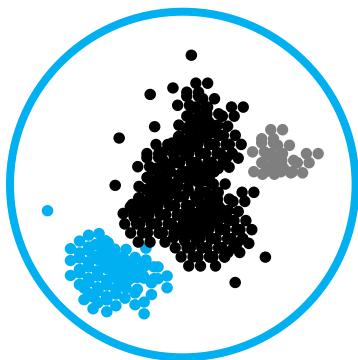
1. Introduction
2. Integer programming
- 3. Beyond integer programming**
4. Conclusions

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**  
Balcan, Nagarajan, **Vitercik**, White  
Balcan, Dick, **Vitercik**

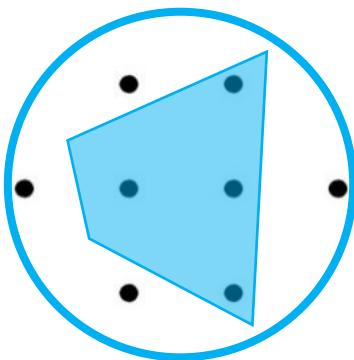
STOC'21  
COLT'17  
FOCS'18

# Overview

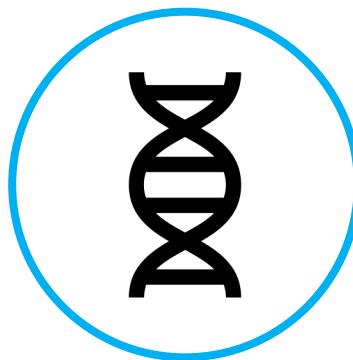
A **unifying** structure connects **seemingly disparate** problems:



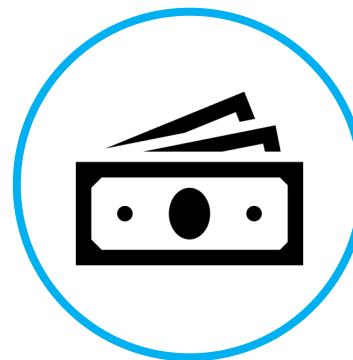
**Clustering**  
algorithm  
configuration



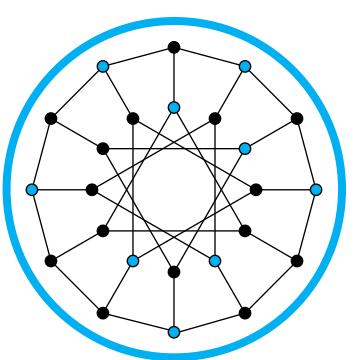
**Integer  
programming**  
algorithm  
configuration



**Computational  
biology**  
algorithm  
configuration



**Mechanism**  
configuration



**Greedy**  
algorithm  
configuration

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

# Piecewise structure

## Key question:

If configuration has good **avg** performance over a training set,  
...will it have good **expected** performance?

# 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

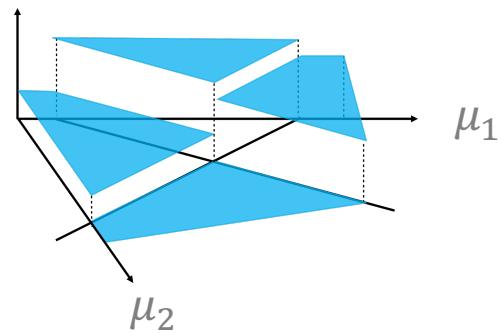
Performance is a **piecewise-structured** function of parameters

Piecewise constant, linear, polynomial, ...



**Integer programming**

Tree size

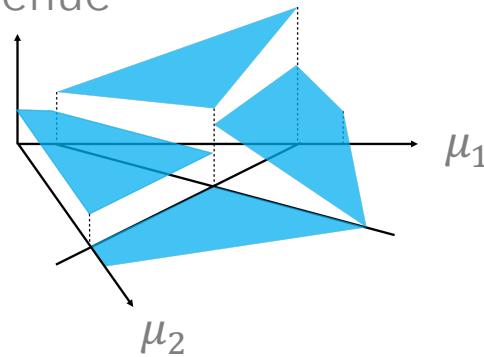


**Piecewise constant**



**Mechanism configuration**

Revenue



**Piecewise linear**

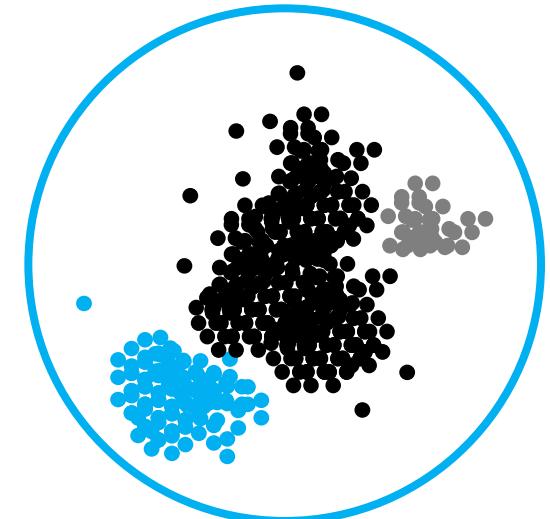
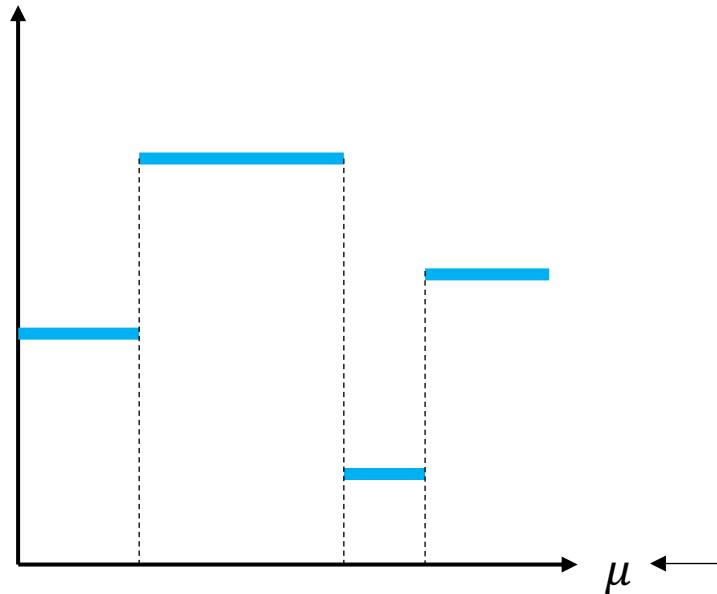


**Piecewise ...**

# Example: Clustering

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

k-means  
objective value



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:

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

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

# Primal & dual classes

$u_{\mu}(x)$  = utility of algorithm parameterized by  $\mu$  on input  $x$   
 $\mathcal{U} = \{u_{\mu}: \mathcal{X} \rightarrow \mathbb{R} \mid \mu \in \mathbb{R}^d\}$  “Primal” function class

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

VC dimension, pseudo-dimension, Rademacher complexity, ...

# Primal & dual classes

$u_{\mu}(x)$  = utility of algorithm parameterized by  $\mu$  on input  $x$   
 $\mathcal{U} = \{u_{\mu}: \mathcal{X} \rightarrow \mathbb{R} \mid \mu \in \mathbb{R}^d\}$  “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  $u_{\mu}$
- No obvious notions of Lipschitz continuity or smoothness to rely on

# Primal & dual classes

$u_{\mu}(x)$  = utility of algorithm parameterized by  $\mu$  on input  $x$   
 $\mathcal{U} = \{u_{\mu}: \mathcal{X} \rightarrow \mathbb{R} \mid \mu \in \mathbb{R}^d\}$  “Primal” function class

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

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

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

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

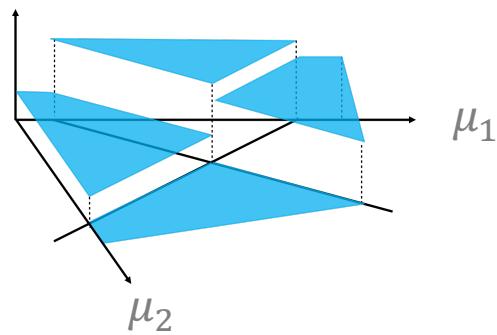
# Piecewise-structured functions

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



**Integer programming**

Tree size

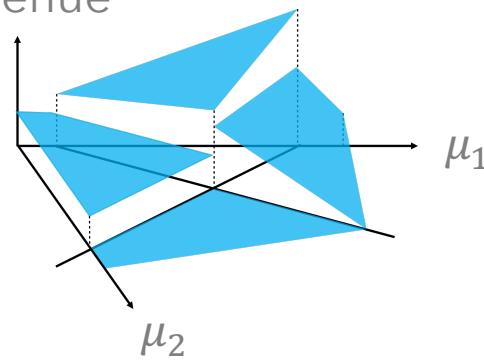


**Piecewise constant**



**Mechanism configuration**

Revenue



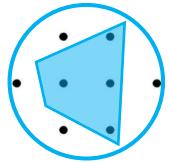
**Piecewise linear**



**Piecewise ...**

# Piecewise structure

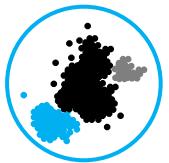
Piecewise structure unifies **seemingly disparate** problems:



## **Integer programming**

Balcan, Dick, Sandholm, , ICML'18

Balcan, Nagarajan, , White, COLT'17

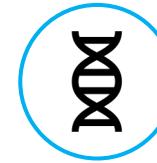


## **Clustering**

Balcan, Nagarajan, , White, COLT'17

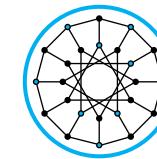
Balcan, Dick, White, NeurIPS'18

Balcan, Dick, Lang, ICLR'20



## **Computational biology**

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



## **Greedy algorithms**

Gupta, Roughgarden, ITCS'16



## **Mechanism configuration**

Balcan, Sandholm, , EC'18

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

Exploited piecewise-Lipschitz structure to provide regret bounds

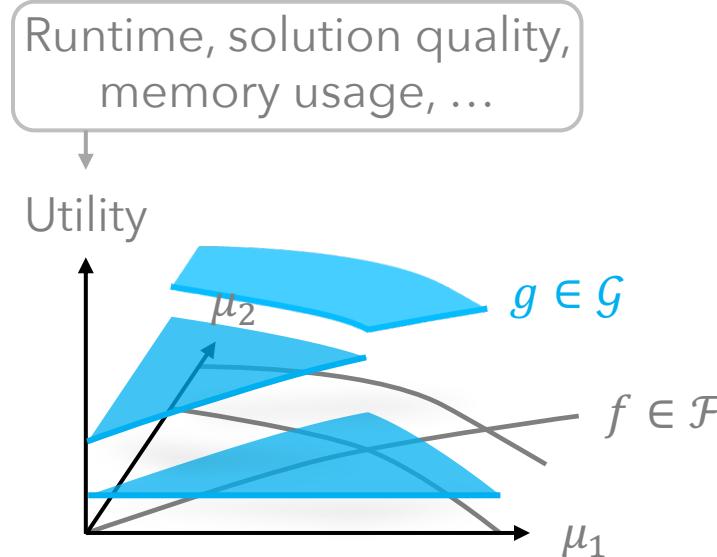
[Balcan, Dick, , FOCS'18; Balcan, Dick, Pegden, UAI'20; Balcan, Dick, Sharma, AISTATS'20]

# Generalization to future inputs

With high probability, for all  $\mu$ :

Avg utility on training set - expected utility

Future



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

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

Upper bound on utility

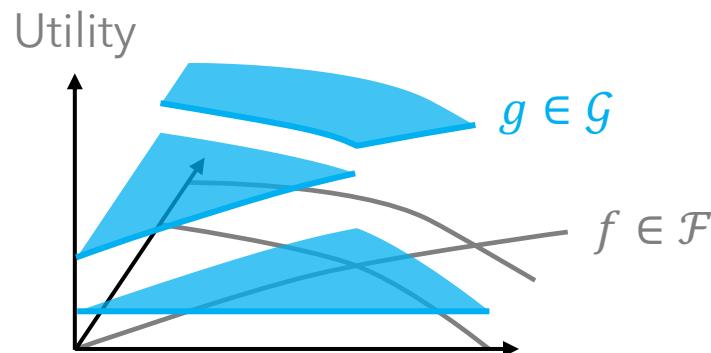
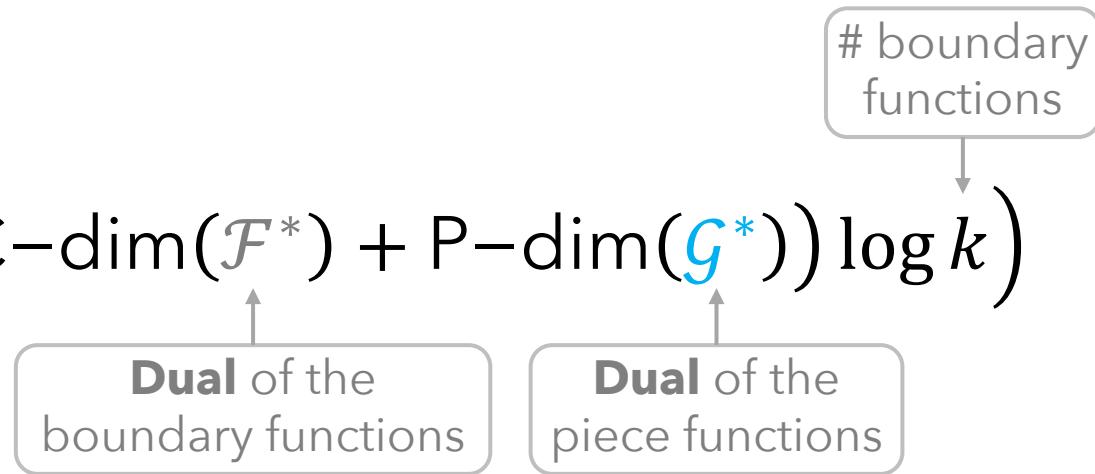
Training set size



# Generalization to future inputs

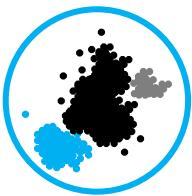
**Theorem:**

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

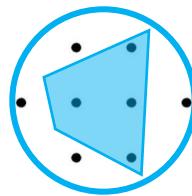


# Main message

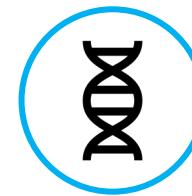
A **unifying** structure connects **seemingly disparate** problems:



**Clustering**  
algorithm  
configuration



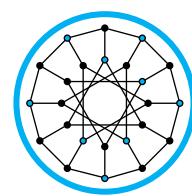
**Integer  
programming**  
algorithm  
configuration



**Computational  
biology**  
algorithm  
configuration



**Mechanism**  
configuration



**Greedy**  
algorithm  
configuration

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

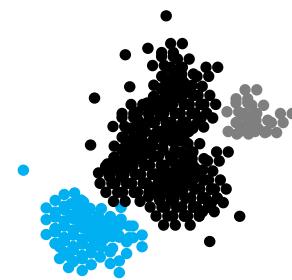
# Outline

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

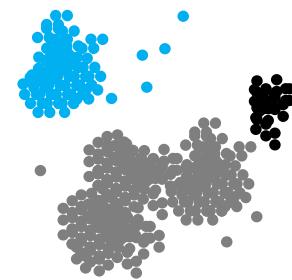
# Online algorithm configuration

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

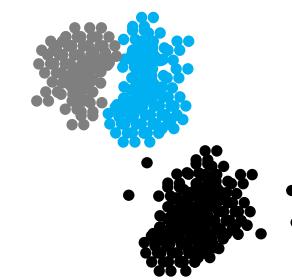
Day 1:  $\mu_1$



Day 2:  $\mu_2$



Day 3:  $\mu_3$



⋮

**Goal:** Compete with best parameter setting in hindsight

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

# Outline

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

# Conclusions

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

- 1<sup>st</sup> 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

