

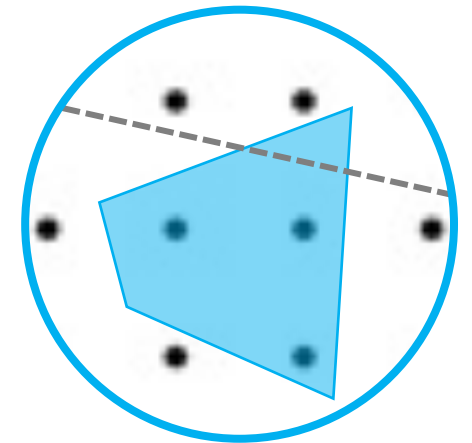
# Theoretical foundations of **machine learning** for **cutting plane** selection

**Ellen Vitercik**

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

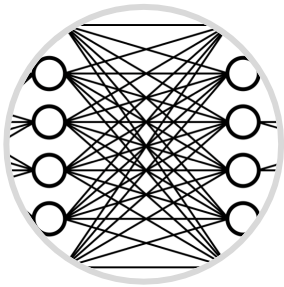
Balcan, Sandholm, Prasad, Vitercik [NeurIPS'21]

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

CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELM 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_BTTOL 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_NUMERICAL EMPHASIS 102	CPX_PARAM_TUNINGMEASURE 163	CPX_PARAM_RELAXPREIND 132	CPXPARAM_MIP_Strategy_MIQCPStrat 93	CPX_PARAM_FRACCAND 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_OBJLLIM 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_CPUMASK 48
CPX_PARAM_POLISHAFTERDETTIME 111	CPXPARAM_Benders_Strategy 30	CPX_PARAM_SCRIND 143	CPXPARAM_Preprocessing_Linear 120	CPX_PARAM_ITLIM 80	CPX_PARAM_CRAININD 50
CPX_PARAM_POLISHAFTEREPAGAP 112	CPXPARAM_Benders_Tolerances_feasibilitycut 35	CPX_PARAM_SIFTALG 143	CPXPARAM_Preprocessing_Reduce 131	CPX_PARAM_LANDPCUTS 82	CPX_PARAM_CUTLO 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_CUTSFACOR 52
CPX_PARAM_POLISHAFTERNODE 115	CPXPARAM_CPUmask 48	CPX_PARAM_SIMDISPLAY 145	CPXPARAM_Read_Scale 142	CPX_PARAM_MCFCUTS 82	CPX_PARAM_CUTUP 53
CPX_PARAM_POLISHAFTERTIME 116	CPXPARAM_DistMIP_Rampup_Duration 128	CPX_PARAM_SINGLIM 146	CPXPARAM_ScreenOutput 143	CPX_PARAM_MEMORY EMPHASIS 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_PPRIIND 118	CPXPARAM_MIP_Cuts_LocalImplied 77	CPX_PARAM_SOLNPOOLGAP 148	CPXPARAM_Sifting_Iterations 145	CPX_PARAM_MIP EMPHASIS 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
	CPXPARAM_MIP_Cuts_ZeroHalfCut 170	CPX_PARAM_SOLNPOOLREPLACE 151	CPXPARAM_Simplex_Limits_Singularity 146	CPX_PARAM_MIPKAPPASTATS 89	CPX_PARAM_DPRIIND 59

# 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, AAI '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; ...]

**Applied  
research**

**Theory  
research**

2000

2022

# Outline

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



# Integer programs

maximize  $\mathbf{c} \cdot \mathbf{z}$   
subject to  $A\mathbf{z} \leq \mathbf{b}$   
 $\mathbf{z} \in \mathbb{Z}^n$

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

# 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  $\mathcal{D}$

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

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

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

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



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

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

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

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

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

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

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



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

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

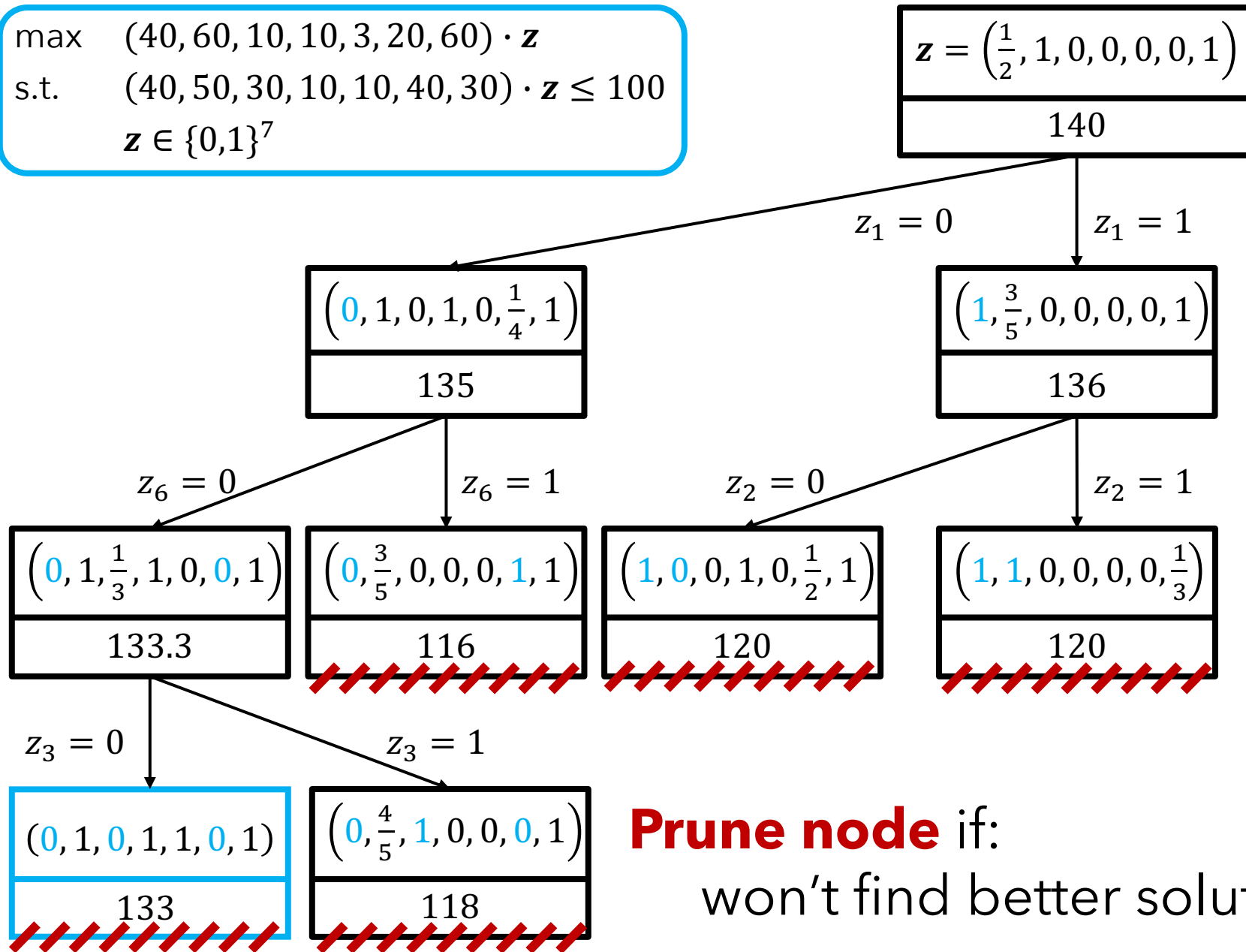
**Focus of this talk:** Will  $\hat{\mu}$  have good **future** performance?

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

# Outline

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

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



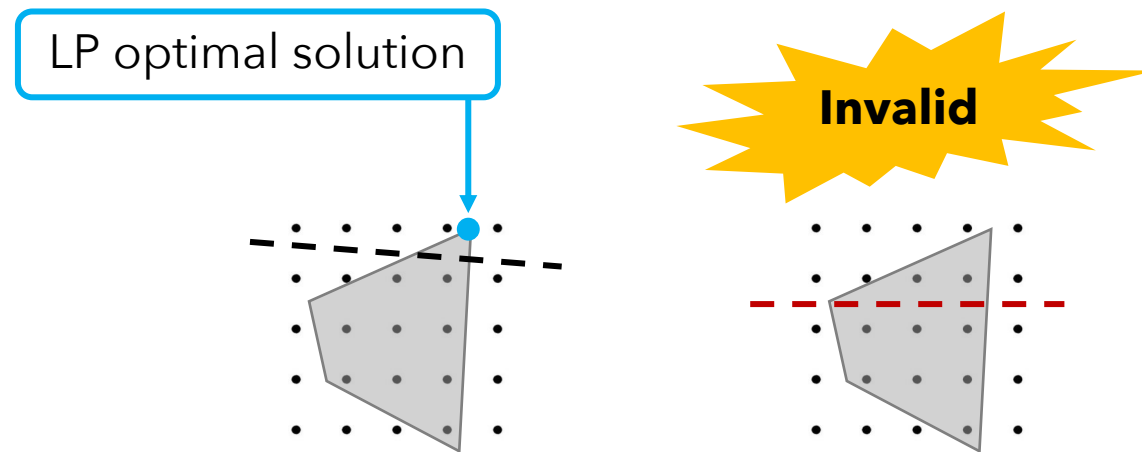
Branch  
and  
bound  
(B&B)

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

# 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?





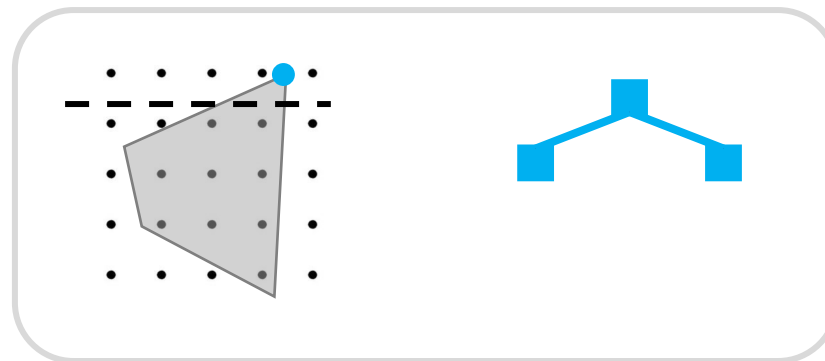
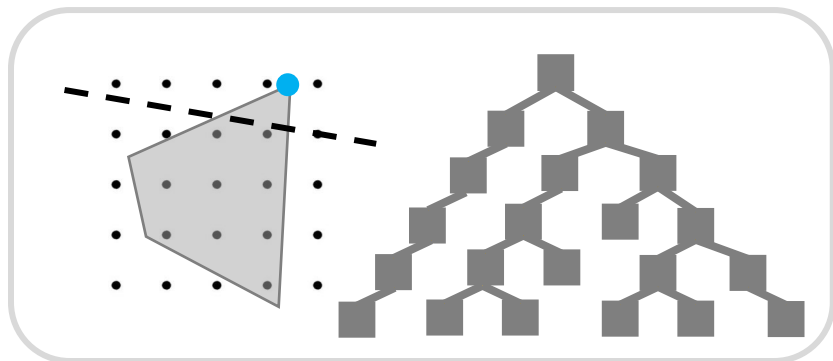
# Key challenge

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

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

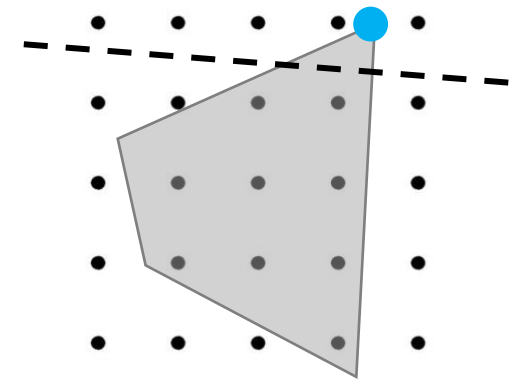
# Chvátal-Gomory cuts

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

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

## Important properties:

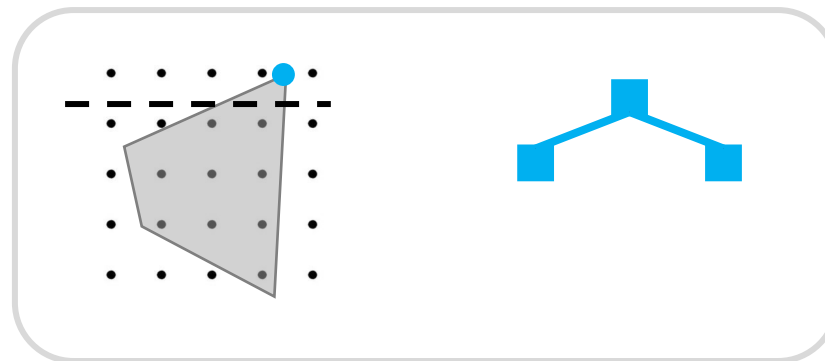
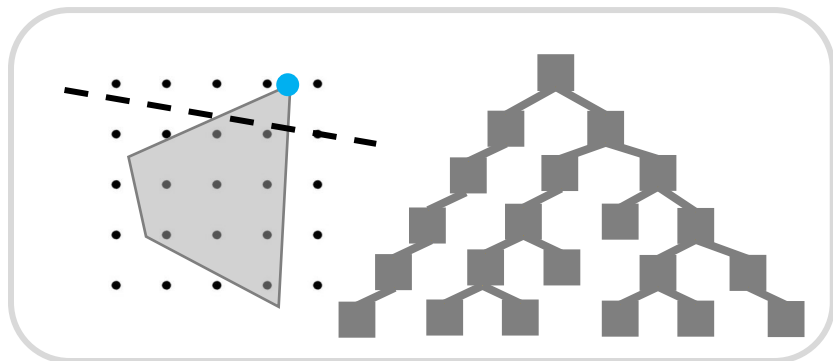
- CG cuts are valid
- Can be chosen so it separates the LP opt



# Key challenge: Sensitivity of B&C

**Theorem** [informal]:

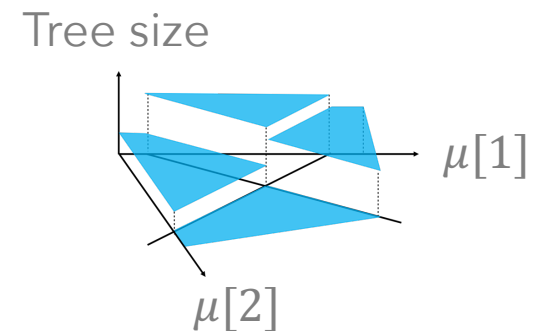
Tiny changes to  $\mu$  can lead to exponential jumps in tree size



# Key lemma

**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

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



# Key lemma

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

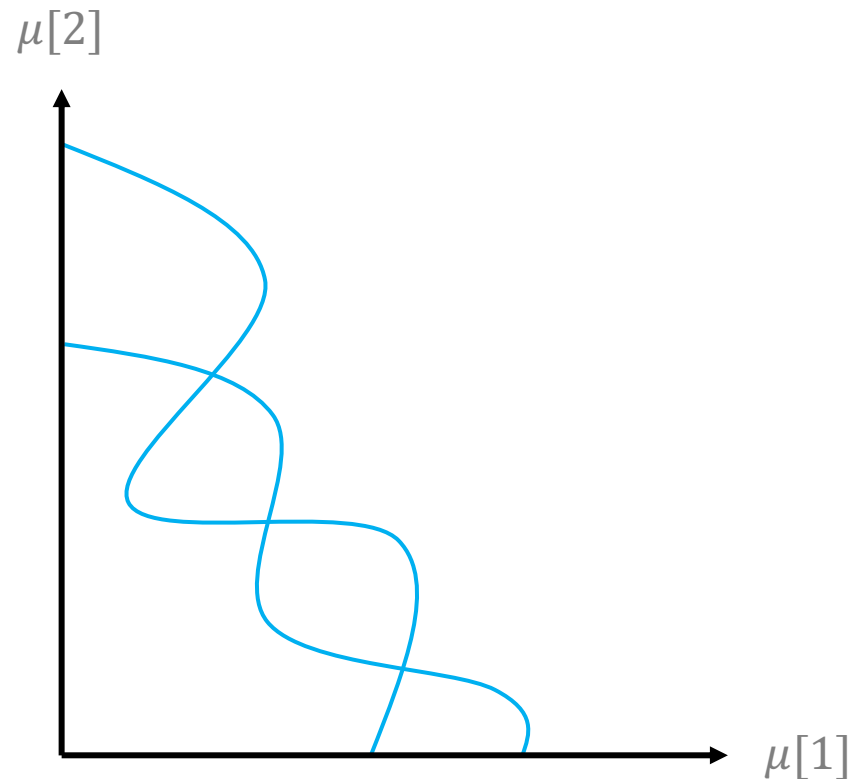
$$\lfloor \boldsymbol{\mu}^T \mathbf{a}_i \rfloor = k_i \text{ iff } k_i \leq \boldsymbol{\mu}^T \mathbf{a}_i < k_i + 1$$

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

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

# Beyond Chvátal-Gomory cuts

For more complex families, boundaries can be more complex



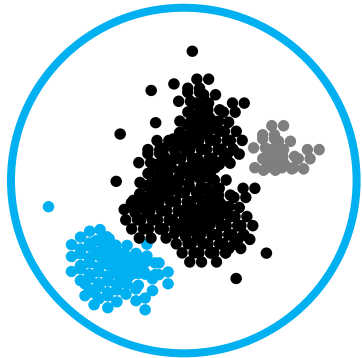
# Outline

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

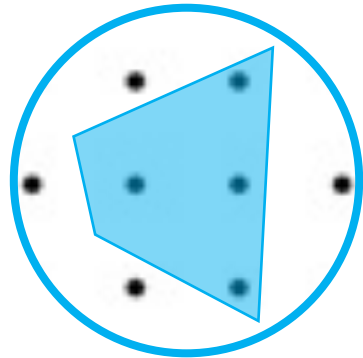


# Overview

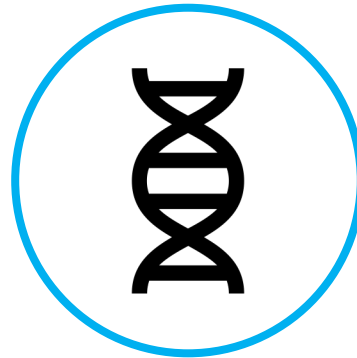
A **unifying** structure connects **seemingly disparate** problems:



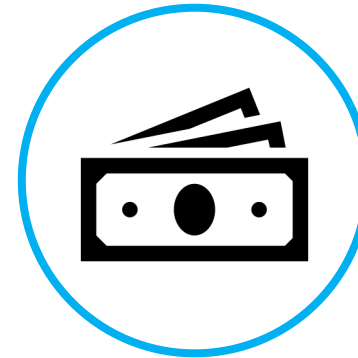
**Clustering**  
algorithm  
configuration



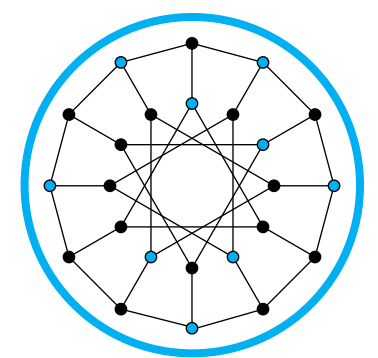
**Integer programming**  
algorithm  
configuration



**Computational biology**  
algorithm  
configuration



**Mechanism**  
configuration



**Greedy**  
algorithm  
configuration

Use to provide generalization bounds

# 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, 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  $\mathcal{U}$

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

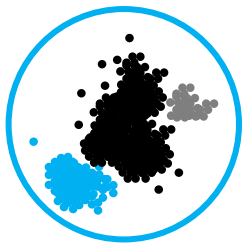
$u_x^*(\mu) = u_{\mu}(x)$

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

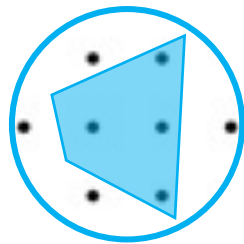
- Dual functions have simple, Euclidean domain
- Often have ample structure can use to bound complexity of  $\mathcal{U}$

# Piecewise-structured functions

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



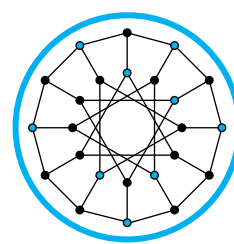
**Clustering**  
algorithm  
configuration



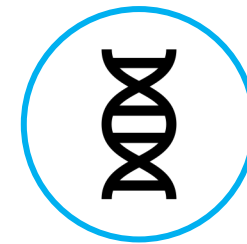
**Integer programming**  
algorithm  
configuration



**Selling mechanism**  
configuration



**Greedy**  
algorithm  
configuration



**Computational biology**  
algorithm  
configuration

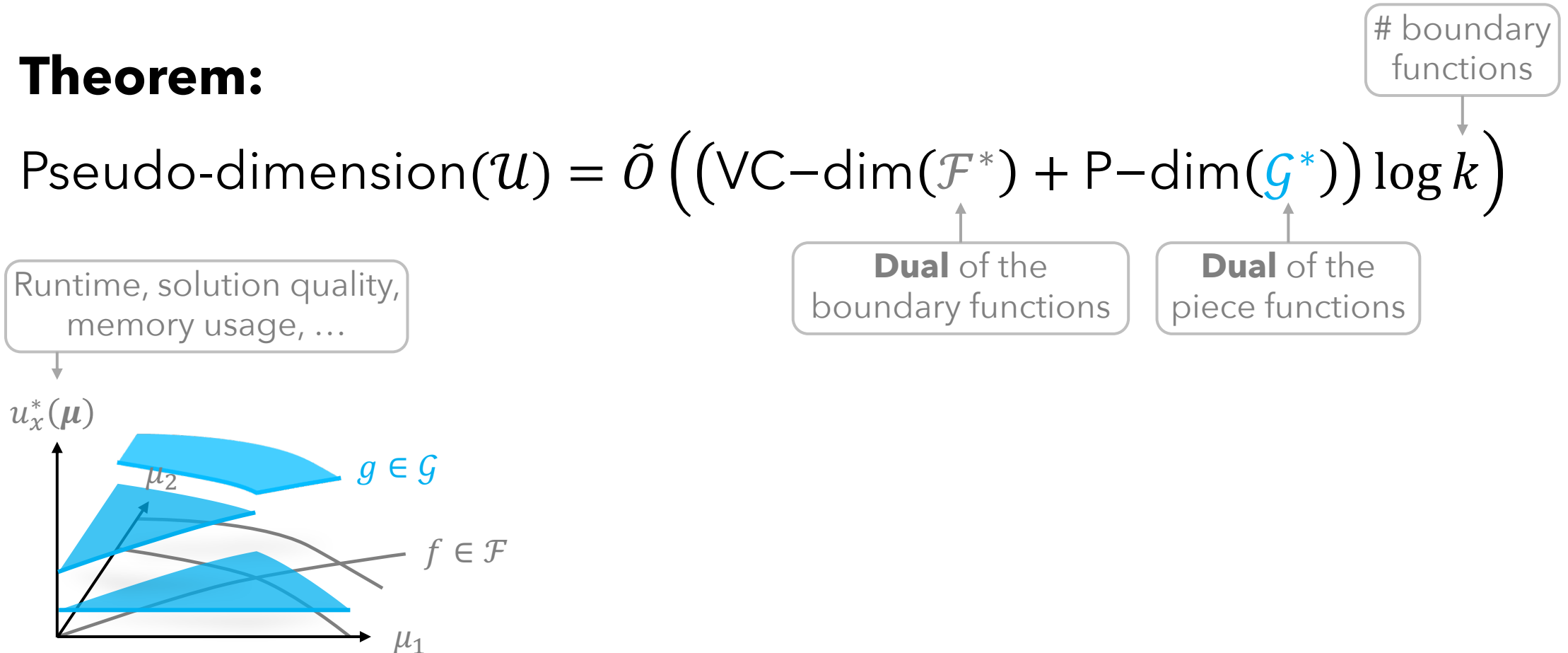


**Voting mechanism**  
configuration

# Generalization to future inputs

## Theorem:

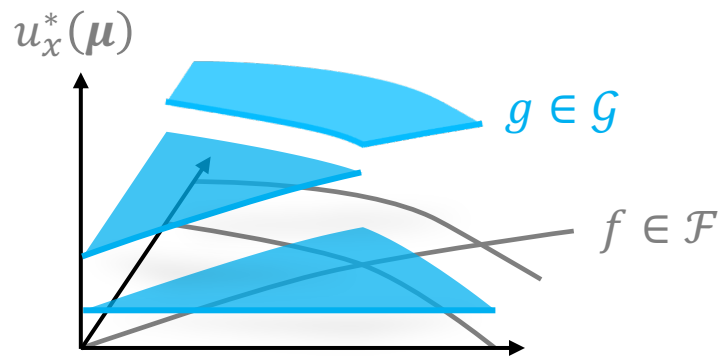
$$\text{Pseudo-dimension}(\mathcal{U}) = \tilde{O} \left( (\text{VC-dim}(\mathcal{F}^*) + \text{P-dim}(\mathcal{G}^*)) \log k \right)$$



# Generalization to future inputs

With high probability, for all  $\mu$ :

$$|\text{Avg utility on training set} - \text{expected utility}| = \tilde{O} \left( \underset{\substack{\uparrow \\ \text{Upper bound} \\ \text{on utility}}}{H} \sqrt{\frac{C_{\mathcal{F}^*} + C_{\mathcal{G}^*}}{\underset{\substack{\uparrow \\ \text{Training} \\ \text{set size}}}{N}}} \right)$$



# Application to cutting planes

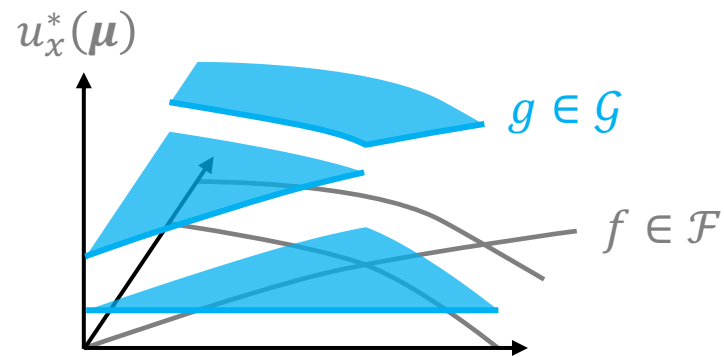
## Theorem:

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

# constraints

Max over support  
of distribution

# variables



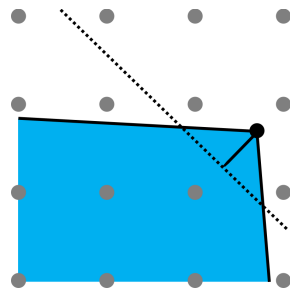


# Outline

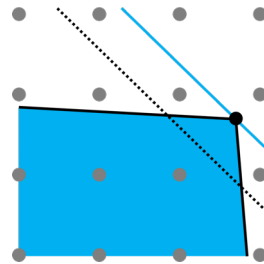
1. Introduction
2. Integer programming
3. Beyond integer programming
- 4. Conclusions**

# Future directions

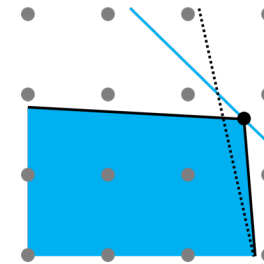
Existing solvers choose cuts from finite pool using heuristics



**Efficacy**



**Good  
parallelism**



**Worse  
parallelism**

Machine learning to design new cut selection policies

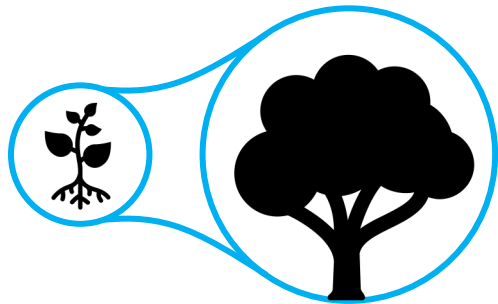
# Future directions

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

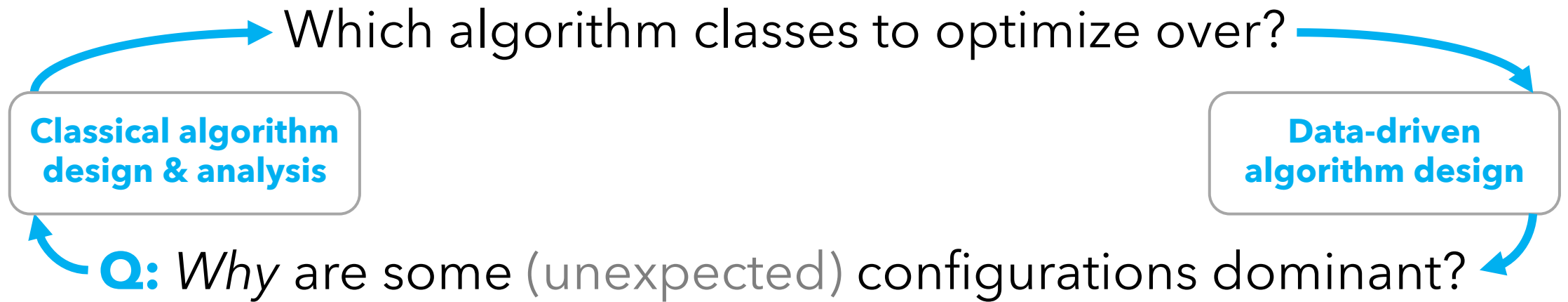
Applied research: Dai et al., NeurIPS'17; Agrawal et al., ICML'20; ...

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



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]

