# **Machine learning for algorithm design:** Theoretical guarantees and applied frontiers

Ellen Vitercik

Stanford University

# How to integrate **machine learning** into **algorithm design**?

- **Algorithm configuration**
  *How to tune an algorithm's parameters?*
- **Algorithm selection**
  *Given a variety of algorithms, which to use?*
- **Algorithm design**
  *Can machine learning guide algorithm discovery?*

# How to integrate **machine learning** into **algorithm design**?

- **Algorithm configuration**
  *How to tune an algorithm's parameters?*
- **Algorithm selection**
  *Given a variety of algorithms, which to use?*
- **Algorithm design**
  *Can machine learning guide algorithm discovery?*

# Algorithm configuration

Example: **Integer programming solvers**

Most popular tool for solving combinatorial (& nonconvex) problems


Routing

Manufacturing

Scheduling

Planning

Finance

# Algorithm configuration

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

# Algorithm configuration

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

# How to integrate **machine learning** into **algorithm design**?

- **Algorithm configuration**
  *How to tune an algorithm's parameters?*
- **Algorithm selection**
  *Given a variety of algorithms, which to use?*
- **Algorithm design**
  *Can machine learning guide algorithm discovery?*

# Algorithm selection in theory

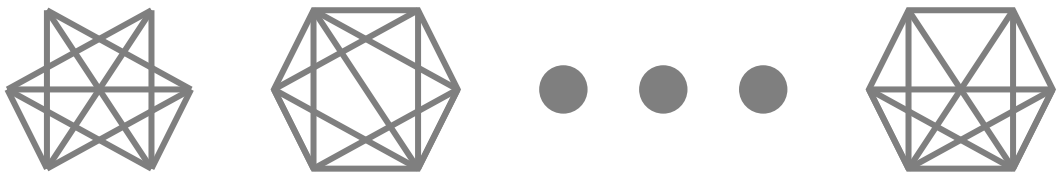
**Worst-case analysis** has been the main framework for decades

*Has led to beautiful, practical algorithms*

Worst-case instances **rarely occur in practice**

**In practice:**

Instances solved in **past** are similar to **future** instances…



# In practice, we have data about the application domain


Routing problems a shipping company solves

# In practice, we have data about the application domain

Clustering problems a biology lab solves

# In practice, we have data about the application domain

Scheduling problems an airline solves

# Existing research

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

**Integer programming**
[Hutter, Hoos, Leyton-Brown, CPAIOR '10; …]

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

**Computational biology**
[Majoros, Salzberg, Bioinformatics'04; …]

**Applied research**

2000

2023

# Existing research

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

**Learning-augmented algorithms**
[Lykouris, Vassilvitskii, ICML'18; Mitzenmacher, NeurIPS'18; …]

**Sample complexity of revenue maximization**
[Balcan, Blum, Hartline, Mansour, FOCS'05; Elkind, SODA'07; …]

**Applied research**

**Theory research**

2000

2023

# ML + algorithm design: Potential impact

**Example: integer programming**

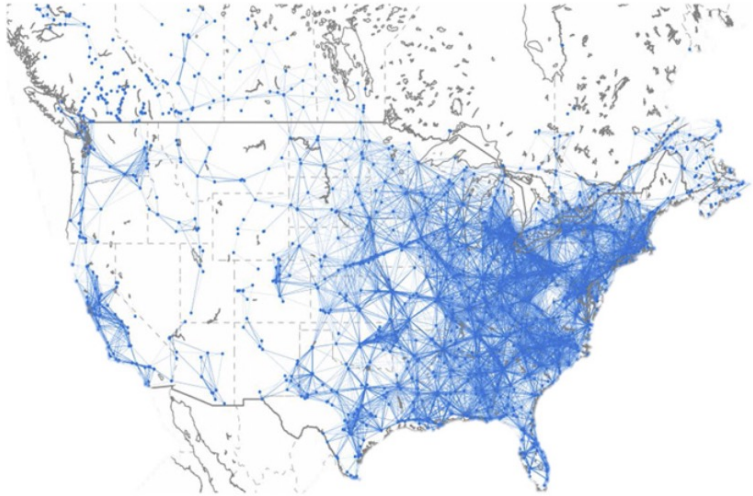
- Used heavily throughout industry and science
- **Many** different ways to incorporate **learning** into solving
- Solving is very difficult, so ML can make a huge difference

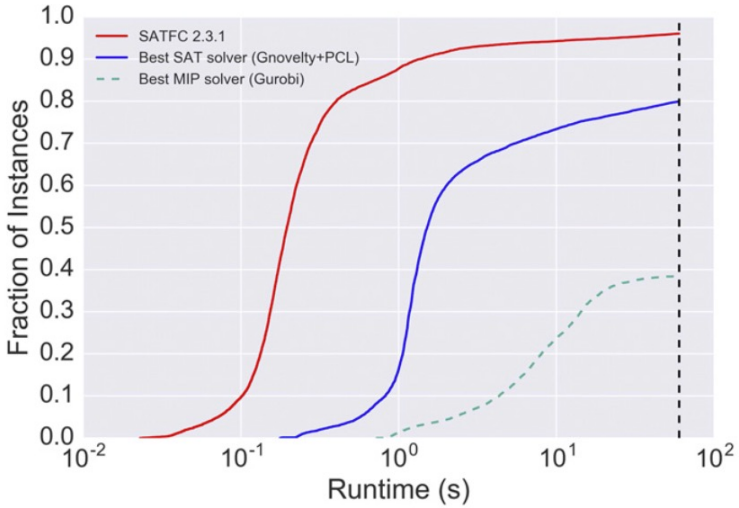




# Example: Spectrum auctions

- In ‘16–’17, FCC held a $19.8 billion radio spectrum auction
  - Involves solving huge graph-coloring problems






- SATFC uses algorithm configuration + selection
- Simulations indicate SATFC saved the government billions

Leyton-Brown et al., PNAS’17; Leyton-Brown and Hutter, ICML’19 tutorial

# Plan for tutorial

1. **Theoretical guarantees**
   a. Statistical guarantees for algorithm configuration
   b. Online algorithm configuration

2. **Applied techniques**
   a. Graph neural networks

# Plan for tutorial

1. **Theoretical guarantees**
   a. **Statistical guarantees for algorithm configuration**
   b. Online algorithm configuration

2. **Applied techniques**
   a. Graph neural networks

Gupta, Roughgarden, ITCS'16
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21
Balcan, Prasad, Sandholm, **Vitercik**, NeurIPS'21
Balcan, Prasad, Sandholm, **Vitercik**, NeurIPS'22

# Running example: Sequence alignment

**Goal:** Line up pairs of strings

**Applications:** Biology, natural language processing, etc.

vitterchik

Did you mean: **vitercik**

# Sequence alignment algorithms

Input: Two sequences $S$ and $S'$

$$S = \text{A C T G}$$
$$S' = \text{G T C A}$$

Output: Alignment of $S$ and $S'$

Gap

A – – C T G

– G T C A –

Insertion/deletion (*indel*)

Match

Mismatch

# Sequence alignment algorithms

Standard algorithm with parameters $\rho_1, \rho_2, \rho_3 \geq 0$:

Return alignment maximizing:

$$(\#\text{ matches}) - \rho_1 \cdot (\#\text{ mismatches}) - \rho_2 \cdot (\#\text{ indels}) - \rho_3 \cdot (\#\text{ gaps})$$

$$S = \text{A C T G}$$
$$S' = \text{G T C A}$$

Gap

A – – C T G

– G T C A –

Insertion/deletion (*indel*)

Match

Mismatch

# Sequence alignment algorithms

Can sometimes access **ground-truth, reference** alignment

E.g., in computational biology: Bahr et al., Nucleic Acids Res.'01; Raghava et al., BMC Bioinformatics '03; Edgar, Nucleic Acids Res.'04; Walle et al., Bioinformatics'04

Requires extensive manual alignments

…rather just run parameterized algorithm

How to tune algorithm's parameters?

*"There is **considerable disagreement** among molecular biologists about the **correct choice**"* [Gusfield et al. '94]

```
A - - C T G
- G T C A -
```

# Sequence alignment algorithms

```
-GRTCPKPDDLPFSTVVP-LKTFYEPGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
E-VKCPFPSRPDNGFVNYPAKPTLYYKDKATFGCHDGYSLDGP-EEIECTKLGNWSAMPSC-KA
```

Ground-truth alignment of protein sequences

# Sequence alignment algorithms

```
-GRTCPKPDDLPFSTVVP-LKTFYEPGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
E-VKCPFPSRPDNGFVNYPAKPTLYYKDKATFGCHDGYSLDGP-EEIECTKLGNWSAMPSC-KA
```

Ground-truth alignment of protein sequences

```
GRTCP---KPDDLPFSTVVPLKTFYEPGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
EVKCPFPSRPDN-GFVNYPAKPTLYYK-DKATFGCHDGY-SLDGPEEIECTKLGNWS-AMPSCKA
```

Alignment by algorithm with **poorly-tuned** parameters

# Sequence alignment algorithms

```
-GRTCPKPDDLPFSTVVP-LKTFYEPGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
E-VKCPFPSRPDNGFVNYPAKPTLYYKDKATFGCHDGYSLDGP-EEIECTKLGNWSAMPSC-KA
```

Ground-truth alignment of protein sequences

```
GRTCP---KPDDLPFSTVVPLKTFYEPGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
EVKCPFPSRPDN-GFVNYPAKPTLYYK-DKATFGCHDGY-SLDGPEEIECTKLGNWS-AMPSCKA
```

Alignment by algorithm with **poorly-tuned** parameters

```
GRTCPKPDDLPFSTV-VPLKTFYEPGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
EVKCPFPSRPDNGFVNYPAKPTLYYKDKATFGCHDGY-SLDGPEEIECTKLGNWSA-MPSCKA
```

Alignment by algorithm with **well-tuned** parameters

# Automated parameter tuning procedure

1. Fix parameterized algorithm
2. Receive training set $T$ of "typical" inputs

Sequence $S_1$
Sequence $S_1'$
Reference alignment $A_1$

Sequence $S_2$
Sequence $S_2'$
Reference alignment $A_2$

• • •

3. Find parameter setting w/ good avg performance over $T$

Runtime, solution quality, etc.

# Automated parameter tuning procedure

1. Fix parameterized algorithm
2. Receive training set $T$ of "typical" inputs

Sequence $S_1$
Sequence $S_1'$
Reference alignment $A_1$

Sequence $S_2$
Sequence $S_2'$
Reference alignment $A_2$

• • •

3. Find parameter setting w/ <u>good avg performance</u> over $T$

On average, output alignment is close to reference alignment

# Automated parameter tuning procedure

1. Fix parameterized algorithm
2. Receive training set $T$ of "typical" inputs

Sequence $S_1$
Sequence $S_1'$
Reference alignment $A_1$

Sequence $S_2$
Sequence $S_2'$
Reference alignment $A_2$

• • •

3. Find parameter setting w/ good avg performance over $T$

**Key question:**

How to find parameter setting with good avg performance?

# Automated parameter tuning procedure

**Key question:**

How to find parameter setting with good avg performance?

E.g., for sequence alignment:
algorithm by Gusfield et al. [’94]

Many other generic search strategies

E.g., Hutter et al. [JAIR’09, LION’11], Ansótegui et al. [CP’09], …

# Automated parameter tuning procedure

1. Fix parameterized algorithm
2. Receive training set $T$ of "typical" inputs

Sequence $S_1$
Sequence $S_1'$
Reference alignment $A_1$

Sequence $S_2$
Sequence $S_2'$
Reference alignment $A_2$

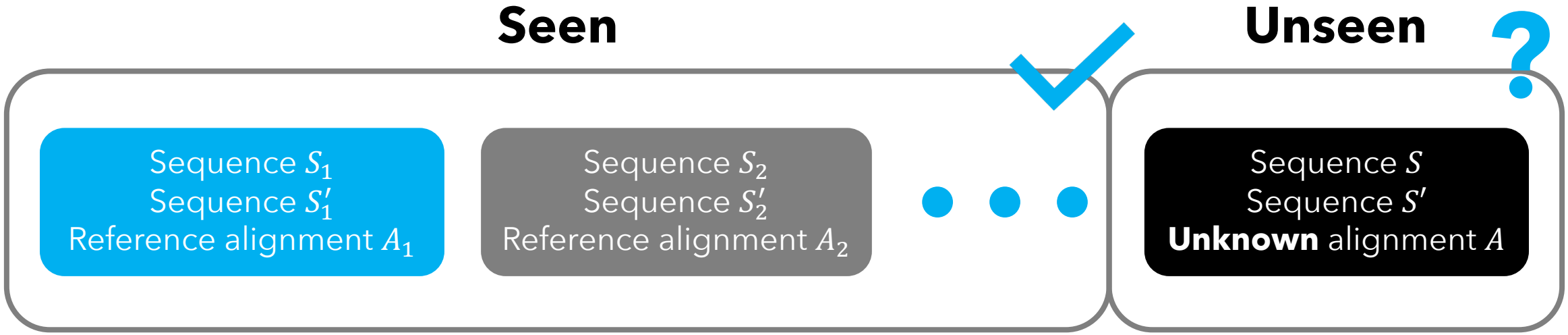
• • •

3. Find parameter setting w/ good avg performance over $T$

**Key question (focus of this section):**

Will that parameter setting have good **future** performance?

# Automated parameter tuning procedure



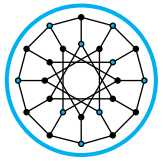

**Key question (focus of this section):**

Will that parameter setting have good **future** performance?

# Generalization

**Key question (focus of this section):**

Good performance on **average** over **training set** implies good **future** performance?



**Greedy algorithms**

Gupta, Roughgarden, ITCS'16

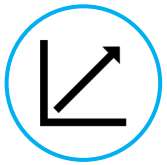
First to ask question for algorithm configuration

**Clustering**

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

**Search**

Sakaue, Oki, NeurIPS'22



**Numerical linear algebra**

Bartlett et al., COLT'22

**And many other areas…**

# This section: Main result

**Key question (focus of this section):**

Good performance on **average** over **training set** implies good **future** performance?

Answer this question for any parameterized algorithm where:

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

Piecewise constant, linear, quadratic, …

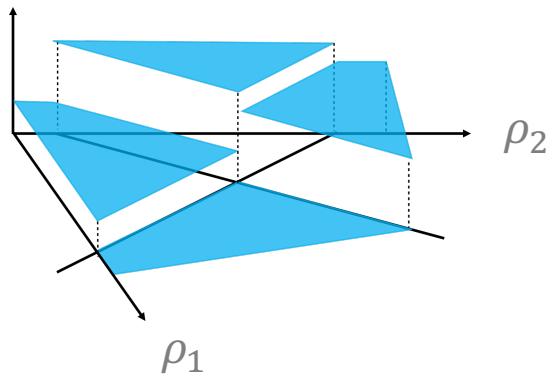
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

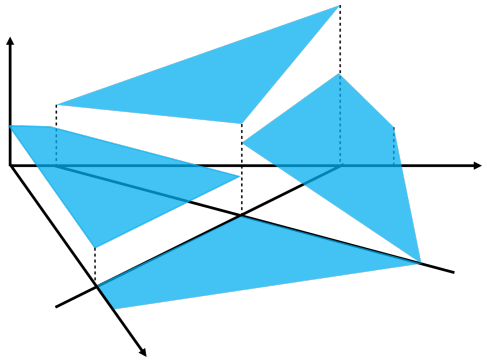
# This section: Main result

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

Piecewise constant, linear, quadratic, …

Algorithmic performance on fixed input



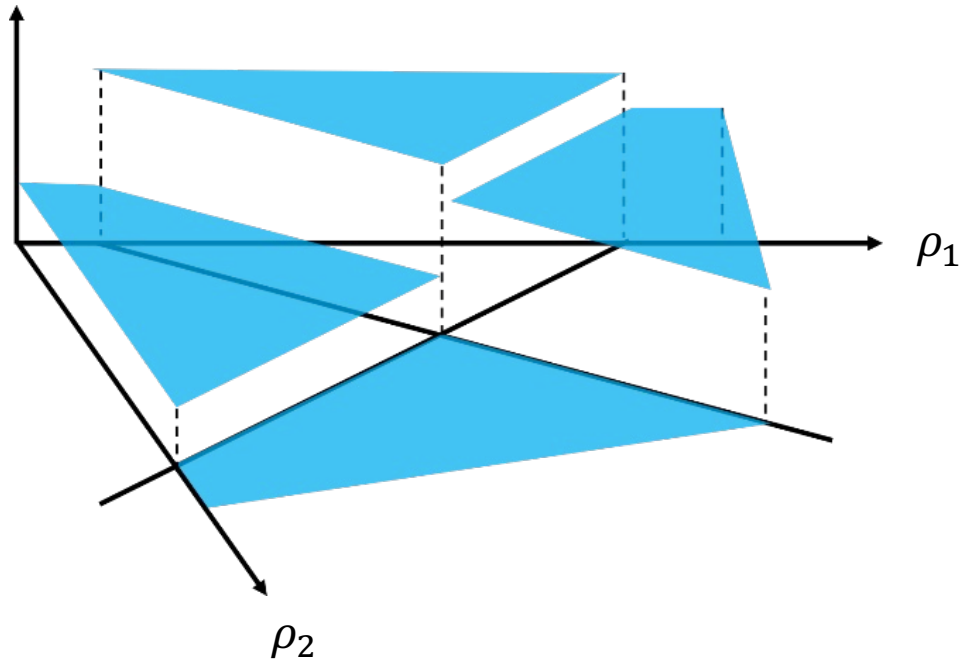
• • •

**Piecewise constant** **Piecewise linear** **Piecewise …**

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

# Example: Sequence alignment

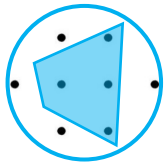
Distance between **algorithm's output** given $S, S'$
and **ground-truth** alignment is p-wise constant




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

# Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:



**Integer programming**

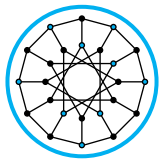
Balcan, Dick, Sandholm, **V**, ICML'18
Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22

**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

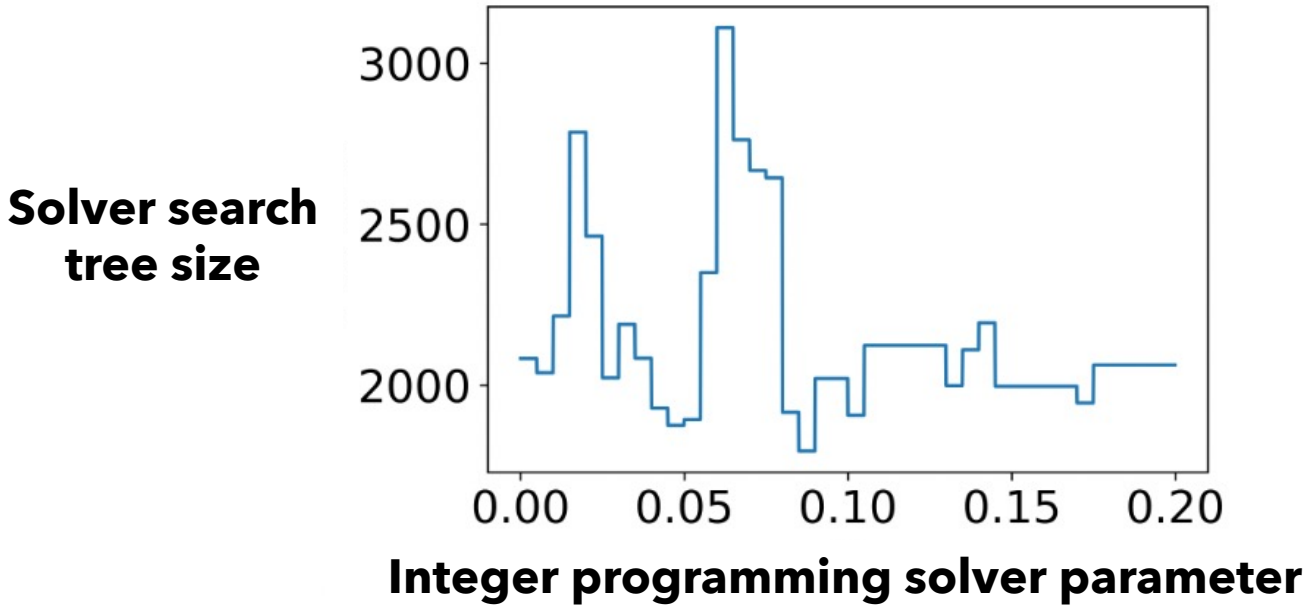
Ties to a long line of research on machine learning for **revenue maximization**
Likhodedov, Sandholm, AAAI'04, '05; Balcan, Blum, Hartline, Mansour, FOCS'05; Elkind, SODA'07; Cole, Roughgarden, STOC'14; Mohri, Medina, ICML'14; Devanur, Huang, Psomas, STOC'16; …

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

# Primary challenge

Algorithmic performance is a **volatile** function of parameters

**Complex** connection between parameters and performance




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

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. **Model**
   ii. Piecewise-structured algorithmic performance
   iii. Main result
   iv. Applications
2. Online algorithm configuration

# Model

$\mathbb{R}^d$: Set of all parameters

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

# Example: Sequence alignment

$\mathbb{R}^3$: Set of alignment algorithm parameters

$\mathcal{X}$: Set of sequence pairs

$$\begin{aligned} S &= \mathrm{A\ C\ T\ G} \\ S' &= \mathrm{G\ T\ C\ A} \end{aligned}$$

One sequence pair $x = (S, S') \in \mathcal{X}$

# Algorithmic performance

$u_{\boldsymbol{\rho}}(x)$ = utility of algorithm parameterized by $\boldsymbol{\rho} \in \mathbb{R}^d$ on input $x$

*E.g., runtime, solution quality, distance to ground truth, …*

Assume $u_{\boldsymbol{\rho}}(x) \in [-1,1]$

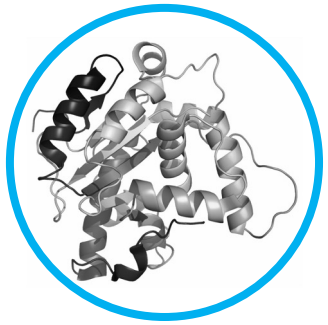
Can be generalized to $u_{\boldsymbol{\rho}}(x) \in [-H, H]$

# Model

Standard assumption: Unknown distribution $\mathcal{D}$ over inputs

*Distribution models specific application domain at hand*

E.g., distribution over pairs of DNA strands



E.g., distribution over pairs of protein sequences

# Generalization bounds

**Key question:** For any parameter setting $\boldsymbol{\rho}$,
is **average** utility on training set close to **expected** utility?

**Formally:** Given samples $x_1, \ldots, x_N \sim \mathcal{D}$, for any $\boldsymbol{\rho}$,

$$\left| \underbrace{\frac{1}{N}\sum_{i=1}^{N} u_{\boldsymbol{\rho}}(x_i)}_{\text{Empirical average utility}} - \underbrace{\mathbb{E}_{x\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(x)\right]}_{\text{Expected utility}} \right| \leq ?$$

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

# Outline (theoretical guarantees)

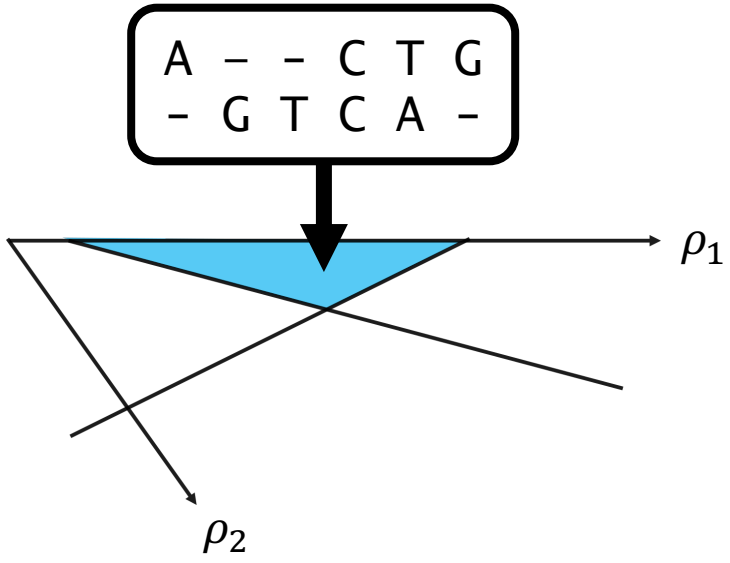
1. Statistical guarantees for algorithm configuration
   i. Model
   **ii. Piecewise-structured algorithmic performance**
      **a. Example: Sequence alignment**
      b. Dual function definition
   iii. Main result
   iv. Applications
2. Online algorithm configuration

# Sequence alignment algorithms

**Lemma:**

For any pair $S, S'$, there's a partition of $\mathbb{R}^3$ s.t. in any region, algorithm's output is fixed across all parameters in region

$S$ = A C T G
$S'$ = G T C A




Gusfield et al., Algorithmica '94; Fernández-Baca et al., J. of Discrete Alg. '04

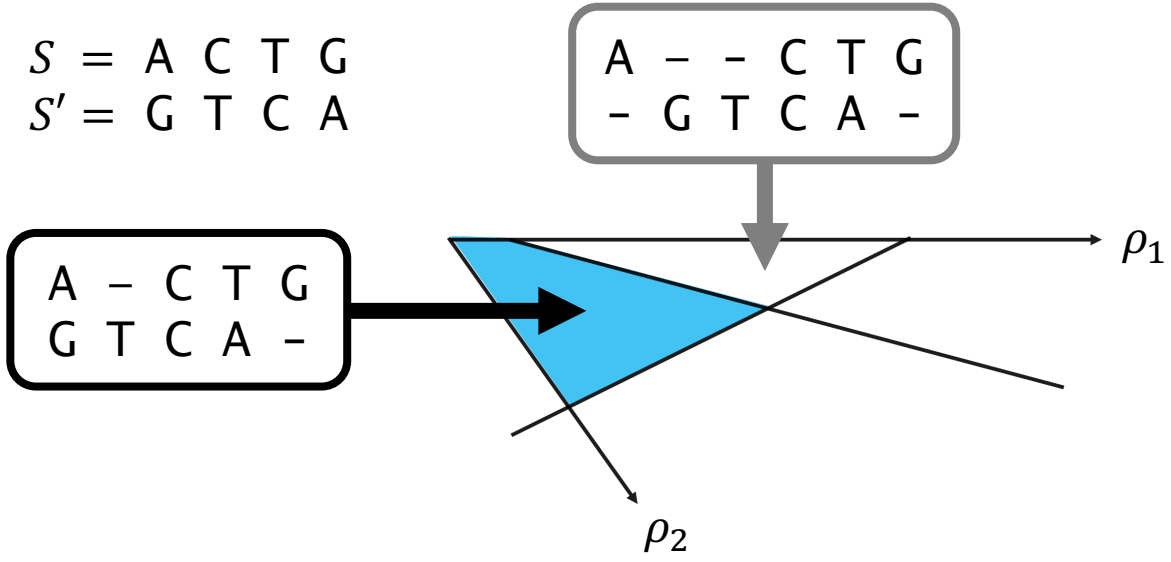
# Sequence alignment algorithms

**Lemma:**

Defined by $(\max\{|S|, |S'|\})^3$ hyperplanes

For any pair $S, S'$, there's a partition of $\mathbb{R}^3$ s.t. in any region, algorithm's output is fixed across all parameters in region

$S$ = A C T G
$S'$ = G T C A



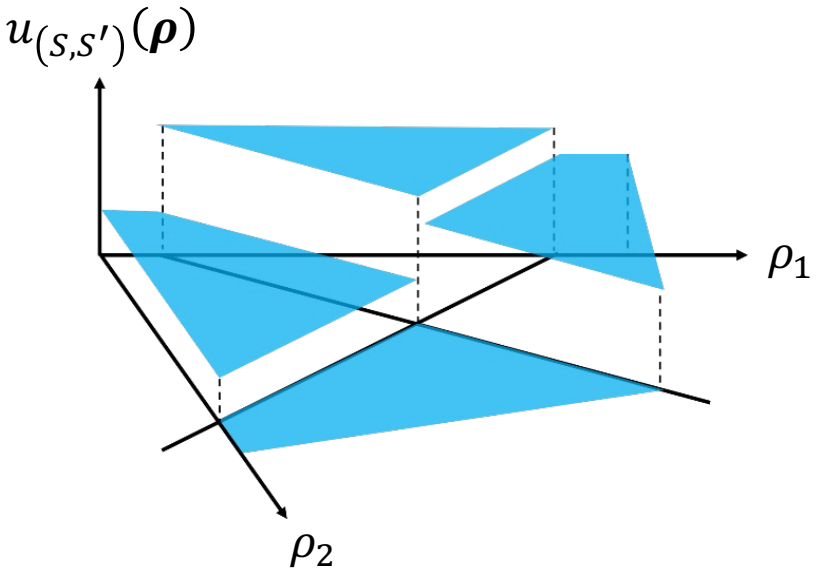

Gusfield et al., Algorithmica '94; Fernández-Baca et al., J. of Discrete Alg. '04

# Piecewise-constant utility function

**Corollary:**

Utility is piecewise constant function of parameters

Distance between algorithm's output and ground-truth alignment




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

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. Model
   ii. Piecewise-structured algorithmic performance
      a. Example: Sequence alignment
      **b. Dual function definition**
   iii. Main result
   iv. Applications
2. Online algorithm configuration

# Primal & dual classes

$u_{\boldsymbol{\rho}}(x)$ = utility of algorithm parameterized by $\boldsymbol{\rho} \in \mathbb{R}^d$ on input $x$

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

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

**Challenge:** $\mathcal{U}$ is gnarly

E.g., in sequence alignment:

- Each domain element is a pair of sequences
- Unclear how to plot or visualize functions $u_{\boldsymbol{\rho}}$
- No obvious notions of Lipschitz continuity or smoothness to rely on

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

# Primal & dual classes

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

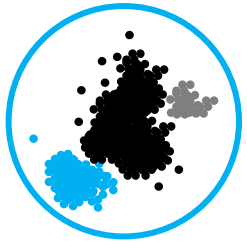
$u_x^*(\boldsymbol{\rho})$ = utility as function of parameters
$u_x^*(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(x)$
$\mathcal{U}^* = \{u_x^*: \mathbb{R}^d \to \mathbb{R} \mid x \in \mathcal{X}\}$ "Dual" function class

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

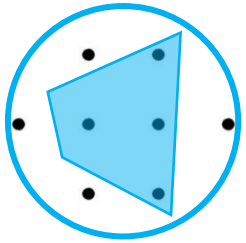
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Piecewise-structured functions

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

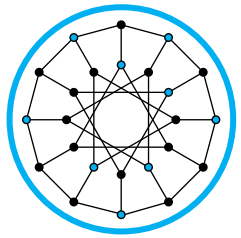


**Clustering**
algorithm configuration

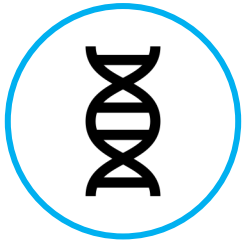


**Integer programming**
algorithm configuration

**Selling mechanism**
configuration



**Greedy**
algorithm configuration



**Computational biology**
algorithm configuration

**Voting mechanism**
configuration

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

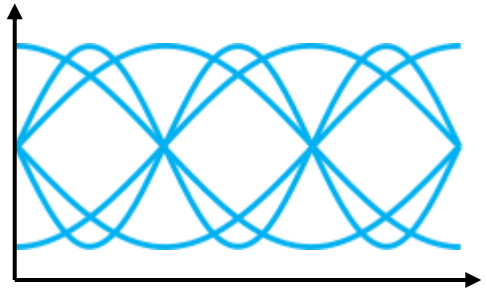
# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. Model
   ii. Piecewise-structured algorithmic performance
   **iii. Main result**
   iv. Applications
2. Online algorithm configuration

# Intrinsic complexity

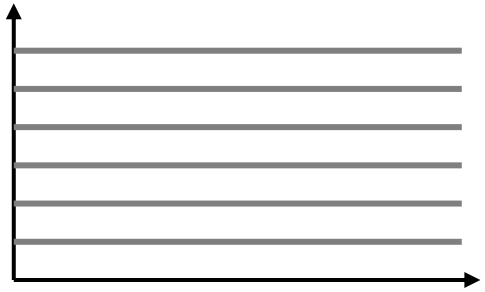
"Intrinsic complexity" of function class $\mathcal{G}$

- Measures how well functions in $\mathcal{G}$ fit complex patterns
- Specific ways to quantify "intrinsic complexity":
  - VC dimension
  - Pseudo-dimension



More complex

Less complex

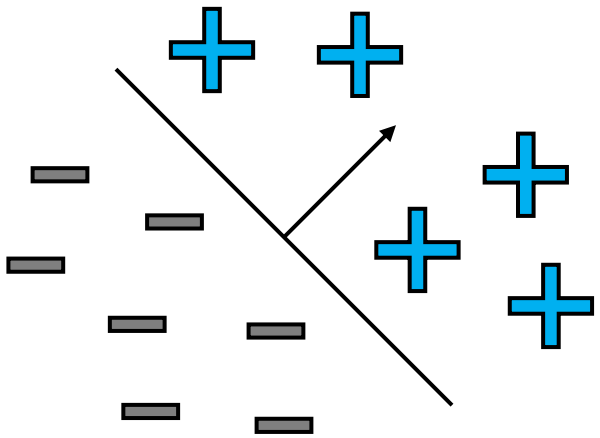


# VC dimension

Complexity measure for binary-valued function classes $\mathcal{F}$

(Classes of functions $f: \mathcal{Y} \to \{-1,1\}$)

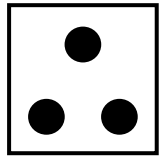
E.g., linear separators



# VC dimension

Size of the largest set $\mathcal{S} \subseteq \mathcal{Y}$
that can be labeled in all $2^{|\mathcal{S}|}$ ways by functions in $\mathcal{F}$

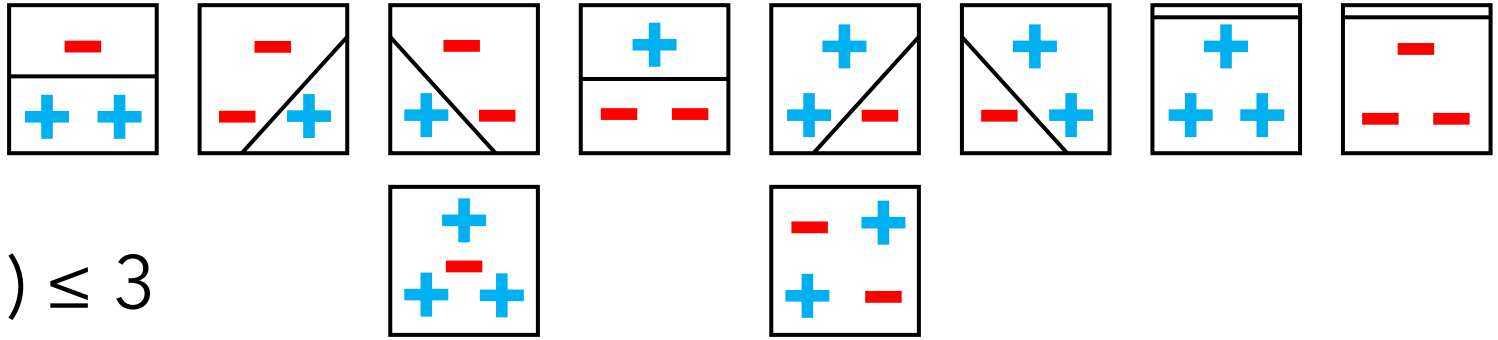
Example: $\mathcal{F}$ = Linear separators in $\mathbb{R}^2$ VCdim($\mathcal{F}$) ≥ 3



# VC dimension

Size of the largest set $\mathcal{S} \subseteq \mathcal{Y}$
that can be labeled in all $2^{|\mathcal{S}|}$ ways by functions in $\mathcal{F}$

Example: $\mathcal{F}$ = Linear separators in $\mathbb{R}^2$ VCdim($\mathcal{F}$) ≥ 3



VCdim($\mathcal{F}$) ≤ 3

VCdim({Linear separators in $\mathbb{R}^d$}) $= d + 1$

# VC dimension

Size of the largest set $\mathcal{S} \subseteq \mathcal{Y}$
that can be labeled in all $2^{|\mathcal{S}|}$ ways by functions in $\mathcal{F}$

Mathematically, for $\mathcal{S} = \{y_1, \dots, y_N\}$,

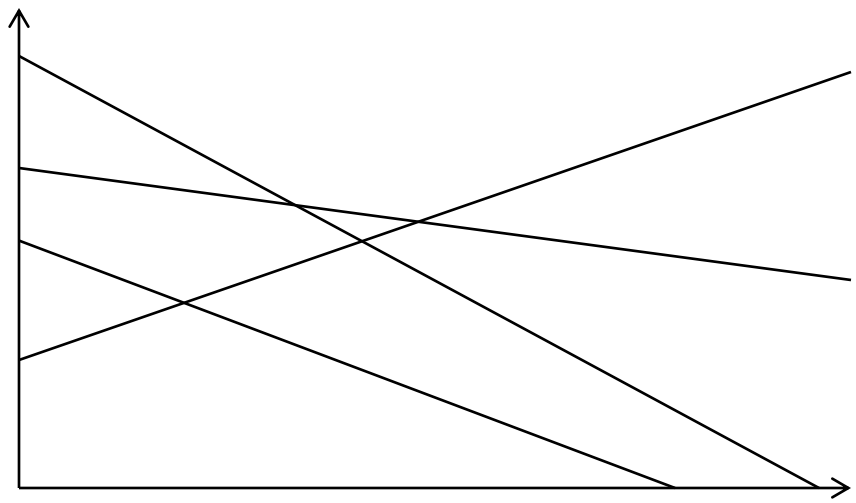
$$\left| \left\{ \begin{pmatrix} f(y_1) \\ \vdots \\ f(y_N) \end{pmatrix} : f \in \mathcal{F} \right\} \right| = 2^N$$

# Pseudo-dimension

Complexity measure for real-valued function classes $\mathcal{G}$

(Classes of functions $g: \mathcal{Y} \to [-1,1]$)

E.g., affine functions



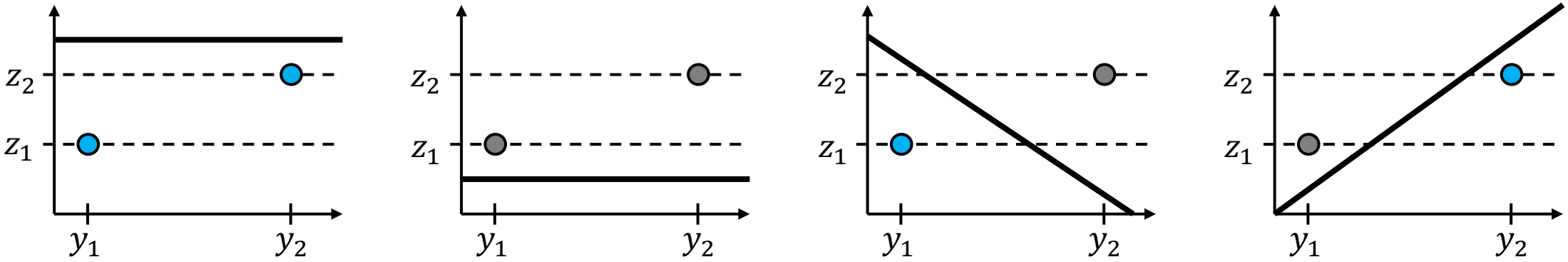
# Pseudo-dimension of $\mathcal{G}$

Size of the largest set $\{y_1, \dots, y_N\} \subseteq \mathcal{Y}$ s.t.:

for some *targets* $z_1, \dots, z_N \in \mathbb{R}$,

all $2^N$ above/below patterns achieved by functions in $\mathcal{G}$

**Example:** $\mathcal{G}$ = Affine functions in $\mathbb{R}$ $\qquad$ $\text{Pdim}(\mathcal{G}) \geq 2$




Can also show that $\text{Pdim}(\mathcal{G}) \leq 2$

# Pseudo-dimension of $\mathcal{G}$

Size of the largest set $\{y_1, \dots, y_N\} \subseteq \mathcal{Y}$ s.t.:

for some *targets* $z_1, \dots, z_N \in \mathbb{R}$,

all $2^N$ above/below patterns achieved by functions in $\mathcal{G}$

Mathematically,

$$\left| \left\{ \begin{pmatrix} \mathbf{1}_{\{g(y_1) \geq z_1\}} \\ \vdots \\ \mathbf{1}_{\{g(y_N) \geq z_N\}} \end{pmatrix} : g \in \mathcal{G} \right\} \right| = 2^N$$

# Sample complexity using pseudo-dim

In the context of **algorithm configuration**:

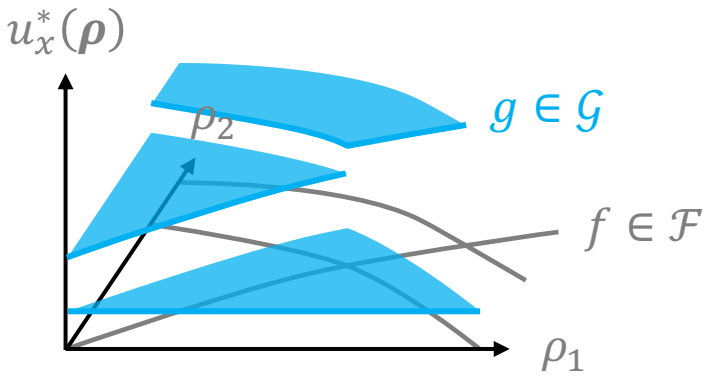
- $\mathcal{U} = \{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\}$ measure algorithm **performance**
- For $\epsilon, \delta \in (0,1)$, let $N = O\left(\frac{\mathrm{Pdim}(\mathcal{U})}{\epsilon^2} \log \frac{1}{\delta}\right)$
- With probability at least $1 - \delta$ over $x_1, \ldots, x_N \sim \mathcal{D}, \forall \boldsymbol{\rho} \in \mathbb{R}^d$,

$$\left| \underbrace{\frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(x_i)}_{\text{Empirical average utility}} - \underbrace{\mathbb{E}_{x \sim \mathcal{D}}\left[u_{\boldsymbol{\rho}}(x)\right]}_{\text{Expected utility}} \right| \leq \epsilon$$

# Main result (informal)

Boundary functions $f_1, \ldots, f_k \in \mathcal{F}$ partition $\mathbb{R}^d$ s.t. in each region,
$u_x^*(\boldsymbol{\rho}) = g(\boldsymbol{\rho})$ for some $g \in \mathcal{G}$.

Training set of size $\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*)+\mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right)$ implies
WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$




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

# Main result (informal)

Boundary functions $f_1, \ldots, f_k \in \mathcal{F}$ partition $\mathbb{R}^d$ s.t. in each region,
$u_x^*(\boldsymbol{\rho}) = g(\boldsymbol{\rho})$ for some $g \in \mathcal{G}$.

**Theorem:**

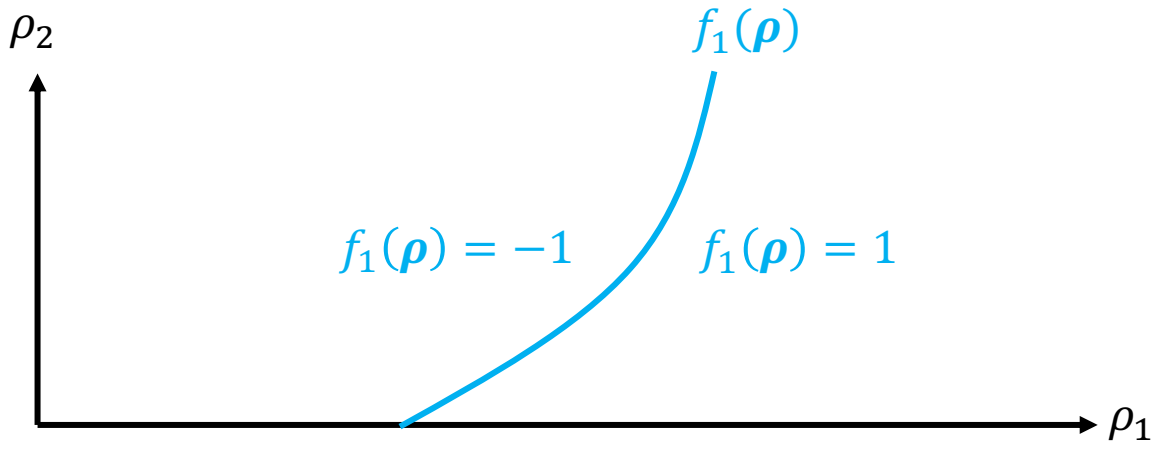
$$\mathrm{Pdim}(\mathcal{U}) = \tilde{O}\big((\mathrm{VCdim}(\mathcal{F}^*) + \mathrm{Pdim}(\mathcal{G}^*)) \log k\big)$$

**Primal** function class $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$

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

# Key lemma

Each boundary function $f: \mathbb{R}^d \to \{-1,1\}$ splits $\mathbb{R}^d$ into 2 regions

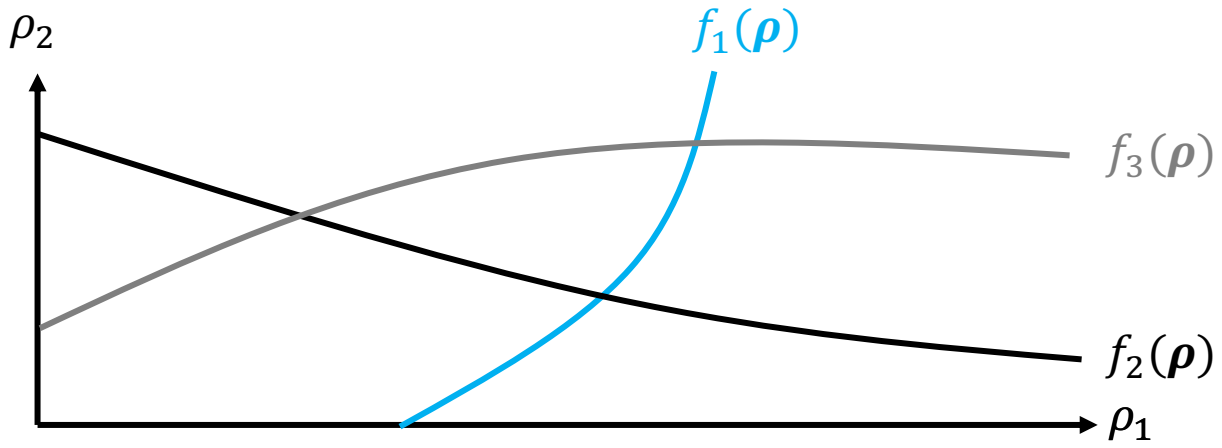



Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Key lemma

Given $D$ boundaries, how many sign patterns do they make?

$$\left|\left\{\begin{pmatrix} f_1(\boldsymbol{\rho}) \\ \vdots \\ f_D(\boldsymbol{\rho}) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \leq ?$$




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

# Key lemma

Given $D$ boundaries, how many sign patterns do they make?

$$\left|\left\{\begin{pmatrix} f_1(\boldsymbol{\rho}) \\ \vdots \\ f_D(\boldsymbol{\rho}) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \leq ?$$

**Note:** Sauer's lemma tells us that for any $D$ points $\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_D \in \mathbb{R}^d$

$$\left|\left\{\begin{pmatrix} f(\boldsymbol{\rho}_1) \\ \vdots \\ f(\boldsymbol{\rho}_D) \end{pmatrix} : f \in \mathcal{F}\right\}\right| \leq (eD)^{\mathrm{VCdim}(\mathcal{F})}$$

This is where transitioning to the dual comes in handy!

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

# Key lemma

Given $D$ boundaries, how many sign patterns do they make?

$$\left|\left\{\begin{pmatrix} f_1(\boldsymbol{\rho}) \\ \vdots \\ f_D(\boldsymbol{\rho}) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \leq (eD)^{\mathrm{VCdim}(\mathcal{F}^*)}$$

**Note:** Sauer's lemma tells us that for any $D$ points $\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_D \in \mathbb{R}^d$

$$\left|\left\{\begin{pmatrix} f(\boldsymbol{\rho}_1) \\ \vdots \\ f(\boldsymbol{\rho}_D) \end{pmatrix} : f \in \mathcal{F}\right\}\right| \leq (eD)^{\mathrm{VCdim}(\mathcal{F})}$$

This is where transitioning to the dual comes in handy!

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

# Proof ideas

For any problem instances $x_1, \ldots, x_N$ and *targets* $z_1, \ldots, z_N \in \mathbb{R}$,

$$\left|\left\{\begin{pmatrix} \operatorname{sgn}\left(u_{\boldsymbol{\rho}}(x_1) - z_1\right) \\ \vdots \\ \operatorname{sgn}\left(u_{\boldsymbol{\rho}}(x_N) - z_N\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d \right\}\right| \leq ?$$

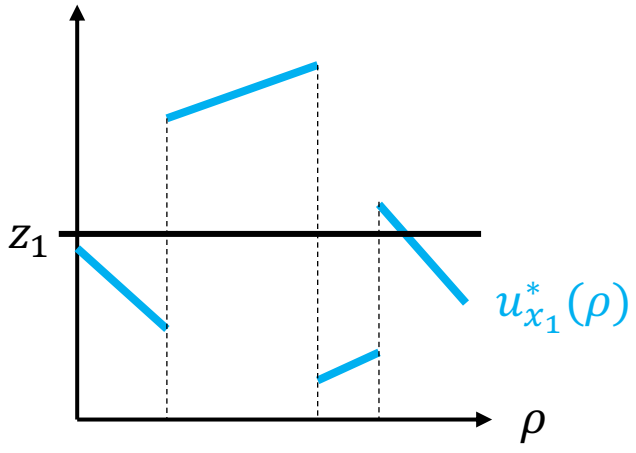
Switching to the dual functions,

$$\left|\left\{\begin{pmatrix} \operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right) \\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d \right\}\right| \leq ?$$

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

# Proof ideas

$$\left|\left\{\begin{pmatrix} \operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right) \\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d \right\}\right| \leq \ ?$$



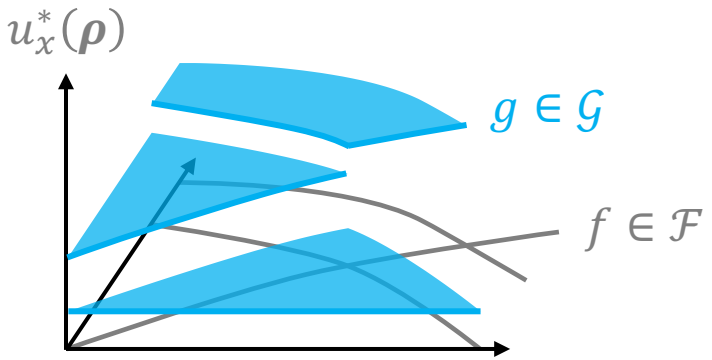

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Proof ideas

$$\left|\left\{\begin{pmatrix} \operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right) \\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d \right\}\right| \leq \ ?$$

The duals $u^*_{x_1}, \ldots, u^*_{x_N}$ correspond to $Nk$ boundary functions in $\mathcal{F}$

How many regions $R_1, \ldots, R_M$ in $\mathbb{R}^d$? $M \leq (eNk)^{\mathrm{VCdim}(\mathcal{F}^*)}$

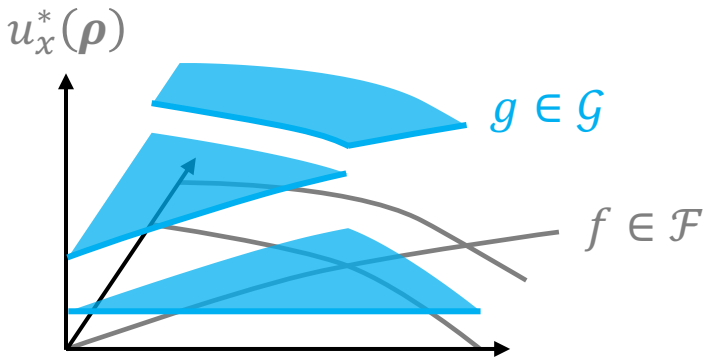



Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Proof ideas

$$\left|\left\{\begin{pmatrix} \operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right) \\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right) \end{pmatrix} : \boldsymbol{\rho} \in \boldsymbol{R_j}\right\}\right| \leq \textbf{?}$$

$\forall \boldsymbol{\rho} \in R_j$, duals are simultaneously structured: $u^*_{x_i}(\boldsymbol{\rho}) = g_i(\boldsymbol{\rho}), \forall i$




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

# Proof ideas

$$\left|\left\{\begin{pmatrix}\operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right)\\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right)\end{pmatrix} : \boldsymbol{\rho} \in \boldsymbol{R_j}\right\}\right| \leq \ ?$$

$\forall \boldsymbol{\rho} \in R_j$, duals are simultaneously structured: $u^*_{x_i}(\boldsymbol{\rho}) = g_i(\boldsymbol{\rho}), \forall i$

$$\left|\left\{\begin{pmatrix}\operatorname{sgn}(g_1(\boldsymbol{\rho}) - z_1)\\ \vdots \\ \operatorname{sgn}(g_N(\boldsymbol{\rho}) - z_N)\end{pmatrix} : \boldsymbol{\rho} \in R_j\right\}\right| \leq \ ?$$

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

# Proof ideas

$$\left|\left\{\begin{pmatrix}\operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right)\\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right)\end{pmatrix} : \boldsymbol{\rho} \in \boldsymbol{R_j}\right\}\right| \leq \ ?$$

$\forall \boldsymbol{\rho} \in R_j$, duals are simultaneously structured: $u^*_{x_i}(\boldsymbol{\rho}) = g_i(\boldsymbol{\rho}), \forall i$

$$\left|\left\{\begin{pmatrix}\operatorname{sgn}(g_1(\boldsymbol{\rho}) - z_1)\\ \vdots \\ \operatorname{sgn}(g_N(\boldsymbol{\rho}) - z_N)\end{pmatrix} : \boldsymbol{\rho} \in R_j\right\}\right| \leq \underline{(eN)^{\mathrm{Pdim}(\mathcal{G}^*)}}$$

**Follows from key lemma**

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

# Proof ideas

$$\left|\left\{\begin{pmatrix} \operatorname{sgn}\left(u^*_{x_1}(\boldsymbol{\rho}) - z_1\right) \\ \vdots \\ \operatorname{sgn}\left(u^*_{x_N}(\boldsymbol{\rho}) - z_N\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d \right\}\right|$$

$$\leq \underbrace{(eNk)^{\mathrm{VCdim}(\mathcal{F}^*)}}_{\text{Number of regions}} \underbrace{(eN)^{\mathrm{Pdim}(\mathcal{G}^*)}}_{\text{Number of sign patterns within each region}}$$

$\mathrm{Pdim}(\mathcal{U})$ equals largest $N$ s.t. $2^{\mathrm{N}} \leq (eNk)^{\mathrm{VCdim}(\mathcal{F}^*)}(eN)^{\mathrm{Pdim}(\mathcal{G}^*)}$,

so $\mathrm{Pdim}(\mathcal{U}) = \tilde{O}\left((\mathrm{VCdim}(\mathcal{F}^*) + \mathrm{Pdim}(\mathcal{G}^*)) \log k\right)$

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

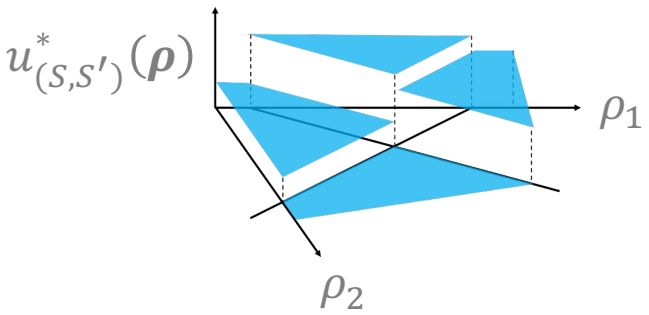
# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. Model
   ii. Piecewise-structured algorithmic performance
   iii. Main result
   **iv. Applications**
      **a. Sequence alignment**
      b. Greedy algorithms
      c. Cutting planes
2. Online algorithm configuration

# Piecewise constant dual functions

**Lemma:**

Utility is piecewise constant function of parameters



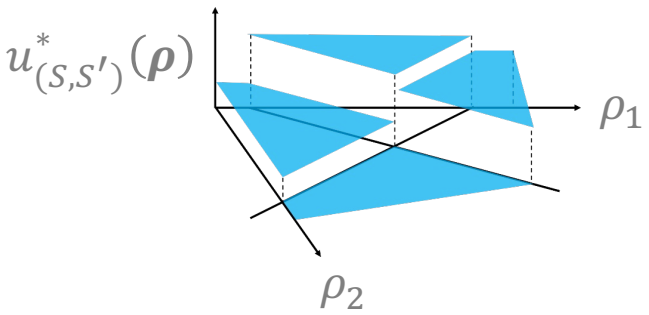

Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Sequence alignment guarantees

**Theorem:** Training set of size

$$\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right) = \tilde{O}\left(\frac{\log(\text{max seq. length})}{\epsilon^2}\right)$$

implies WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$




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

# Sequence alignment guarantees

**Theorem:** Training set of size

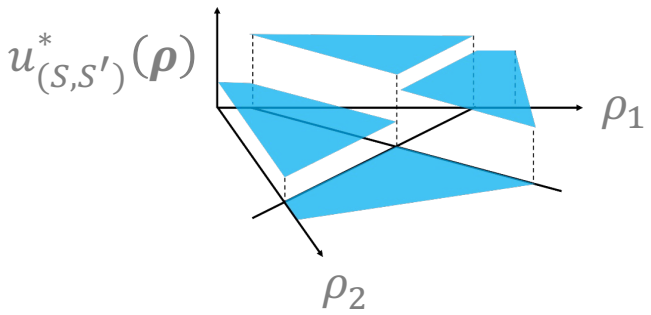
$$\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right) = \tilde{O}\left(\frac{\log(\text{max seq. length})}{\epsilon^2}\right)$$

$\mathcal{G}$ = constant functions in $\mathbb{R}^3$
$\mathrm{Pdim}(\mathcal{G}^*) = O(1)$

$\mathcal{F}$ = hyperplanes in $\mathbb{R}^3$
$\mathrm{VCdim}(\mathcal{F}^*) = O(1)$

$(\text{max sequence length})^3$

implies WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$




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

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. Model
   ii. Piecewise-structured algorithmic performance
   iii. Main result
   iv. Applications
      a. Sequence alignment
      **b. Greedy algorithms**
      c. Cutting planes
2. Online algorithm configuration

# Example: MWIS

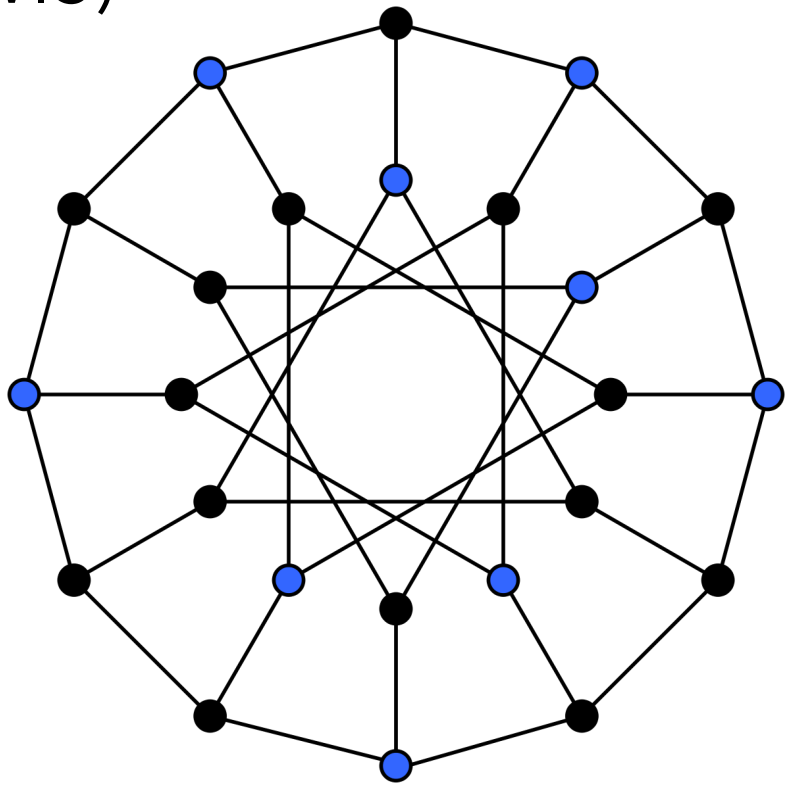
**Maximum weight independent set** (MWIS)

**Problem instance:**

- Graph $G = (V, E)$
- $n$ vertices with weights $w_1, \ldots, w_n \geq 0$

**Goal:** find subset $S \subseteq [n]$

- Maximizing $\sum_{i \in S} w_i$
- No nodes $i, j \in S$ are connected: $(i, j) \notin E$



# Example: MWIS

**Greedy heuristic:**

Greedily add vertices $v$ in decreasing order of $\frac{w_v}{(1+\deg(v))}$

*Maintaining independence*

**Parameterized heuristic** [Gupta, Roughgarden, ITCS'16]**:**

Greedily add nodes in decreasing order of $\frac{w_v}{(1+\deg(v))^{\rho}}$, $\rho \geq 0$

[Inspired by knapsack heuristic by Lehmann et al., JACM'02]

# Example: MWIS

Given a MWIS instance $x$, $u_x^*(\rho)$ = weight of IS algorithm returns

**Theorem** [Gupta, Roughgarden, ITCS'16]**:**

$u_x^*(\rho)$ is piecewise-constant with at most $n^2$ pieces

Gupta, Roughgarden, ITCS'16

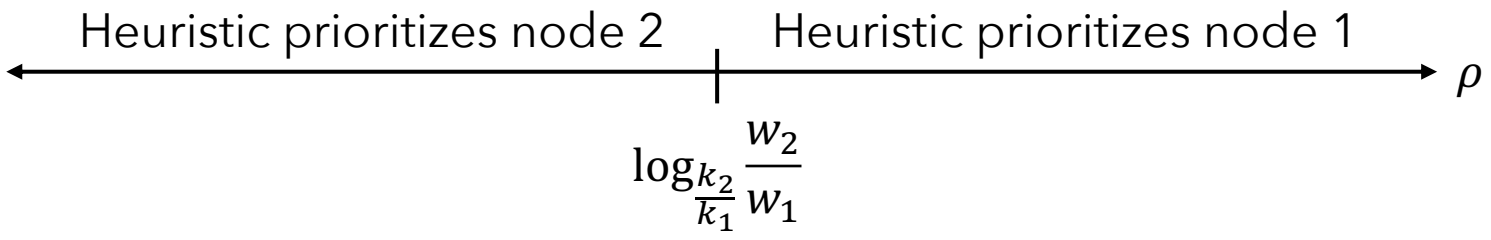
# Example: MWIS

Given a MWIS instance $x$, $u_x^*(\rho)$ = weight of IS algorithm returns

- Weights $w_1, \ldots, w_n \geq 0$
- $\deg(i) + 1 = k_i$

Algorithm parameterized by $\rho$ would add **node 1** before **2** if:

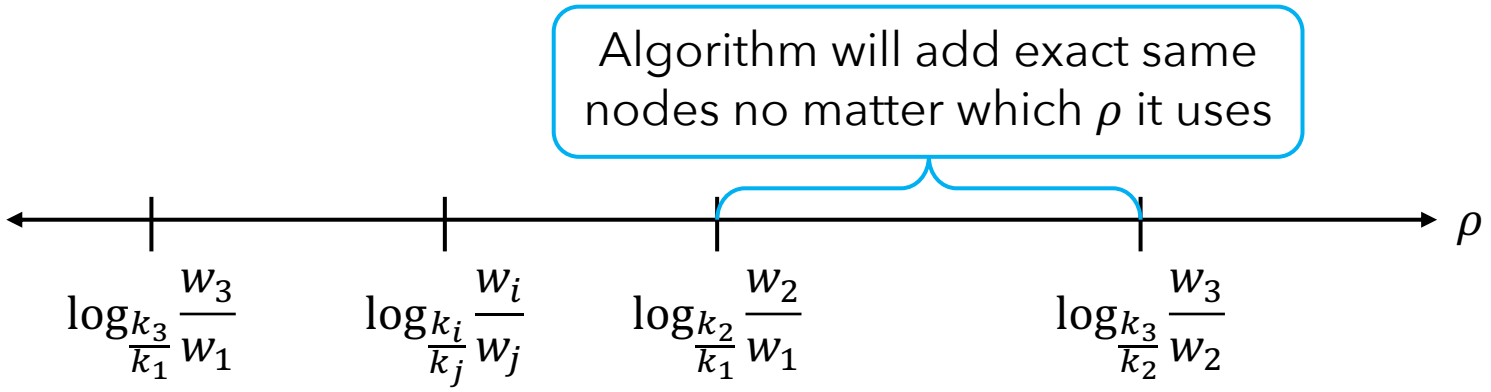
$$\frac{w_1}{k_1^\rho} \geq \frac{w_2}{k_2^\rho} \quad \Leftrightarrow \quad \rho \geq \log_{\frac{k_2}{k_1}} \frac{w_2}{w_1}$$




Gupta, Roughgarden, ITCS'16

# Example: MWIS

- $\binom{n}{2}$ thresholds per instance
- Partition $\mathbb{R}$ into regions where algorithm's output is fixed

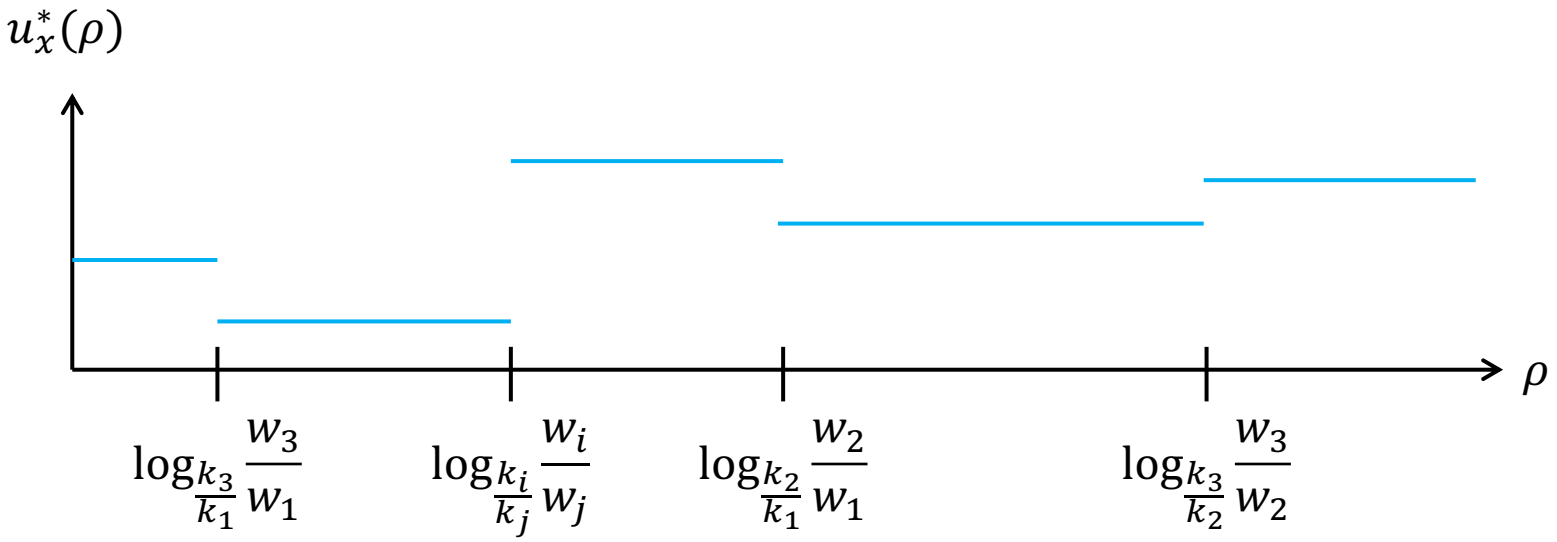



Gupta, Roughgarden, ITCS'16

# Example: MWIS

- $\binom{n}{2}$ thresholds per instance
- Partition $\mathbb{R}$ into regions where algorithm's output is fixed

$\Rightarrow u_x^*(\rho)$ is constant




Gupta, Roughgarden, ITCS'16

# MWIS guarantees

**Theorem:** Training set of size

$$\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right) = \tilde{O}\left(\frac{\log n}{\epsilon^2}\right)$$

implies WHP $\forall \rho$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$

Gupta, Roughgarden, ITCS'16; Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# MWIS guarantees

**Theorem:** Training set of size

$$\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right) = \tilde{O}\left(\frac{\log n}{\epsilon^2}\right)$$

$\mathcal{G}$ = constant functions
$\mathrm{Pdim}(\mathcal{G}^*) = O(1)$

$\mathcal{F}$ = thresholds
$\mathrm{VCdim}(\mathcal{F}^*) = O(1)$

$n^2$

implies WHP $\forall \rho$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$

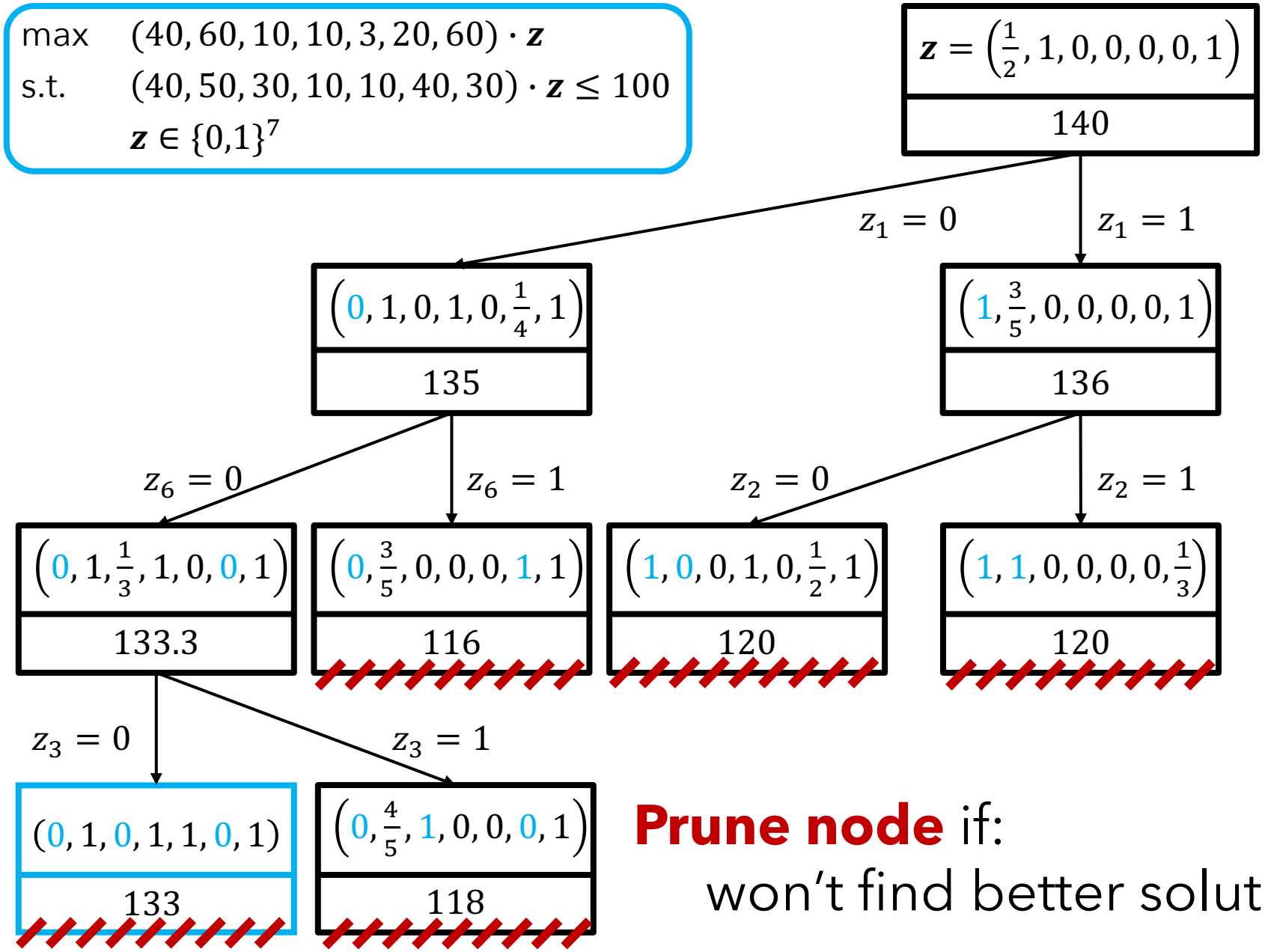
Gupta, Roughgarden, ITCS'16; Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. Model
   ii. Piecewise-structured algorithmic performance
   iii. Main result
   iv. Applications
      a. Sequence alignment
      b. Greedy algorithms
      **c. Cutting planes**
2. Online algorithm configuration

# Branch and bound (B&B)

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



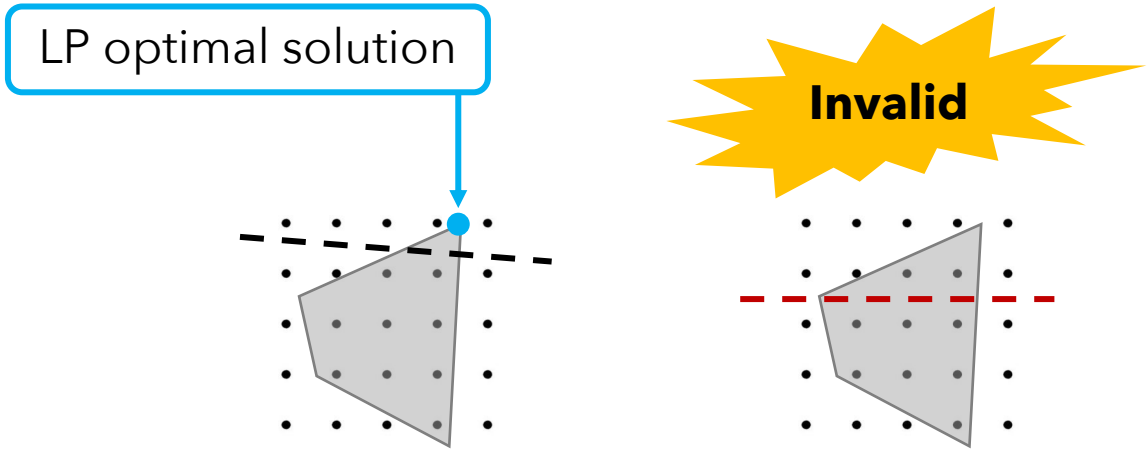

**Prune node** if:

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?

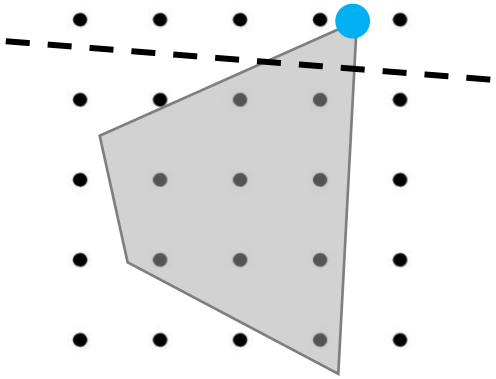
# Chvátal-Gomory cuts

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

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

**Important properties:**

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



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

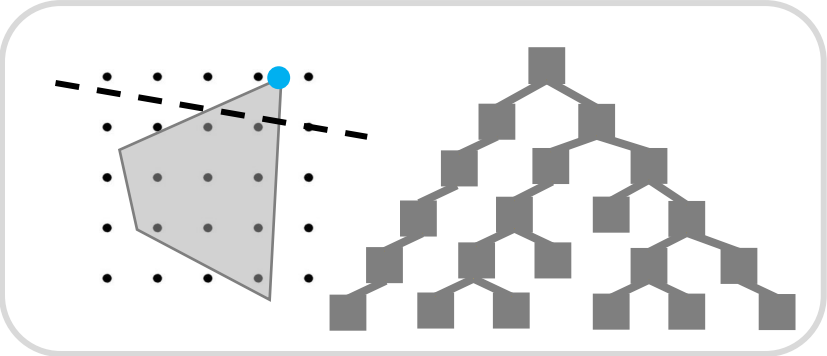
# Key challenge

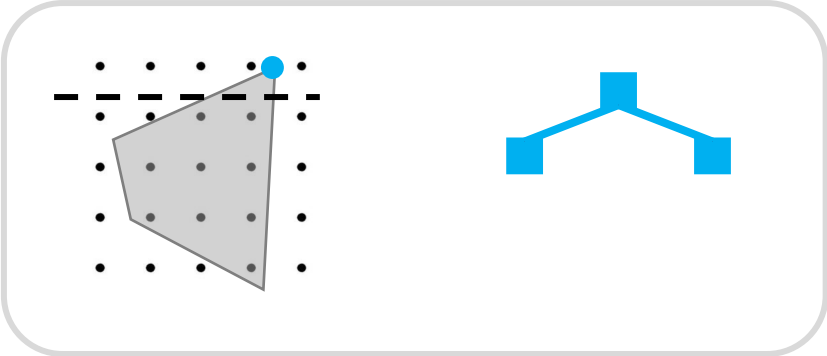
Cut (typically) remains in LPs throughout **entire** tree search

**Every aspect** of tree search depends on LP guidance

*Node selection, variable selection, pruning, …*

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



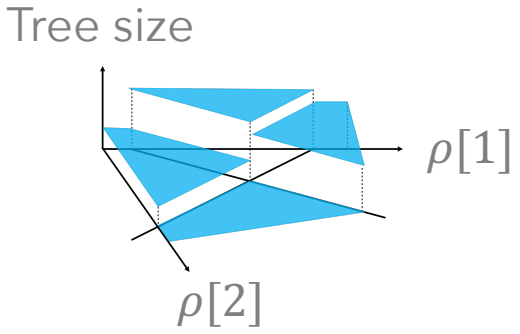


Balcan, Prasad, Sandholm, **Vitercik**, NeurIPS'21

# Key lemma

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

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




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

# Key lemma

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

*Proof idea:*

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

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

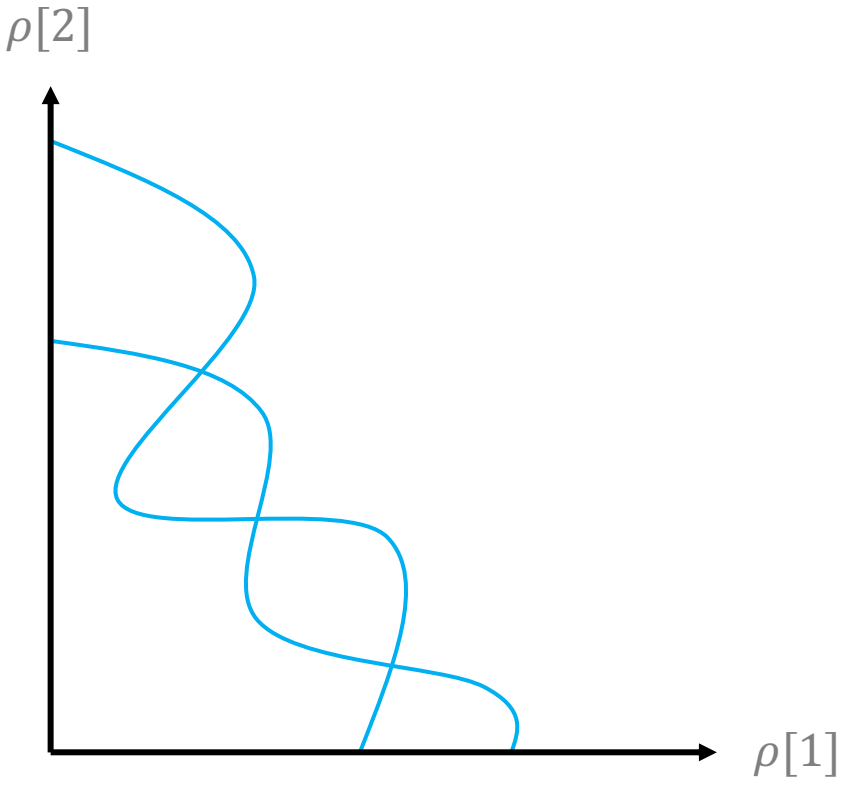
$O(\|A\|_{1,1} + n)$ halfspaces

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

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

# Beyond Chvátal-Gomory cuts

For more complex families, boundaries can be more complex




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

# Cutting plane guarantees

**Theorem:** Training set of size

$$\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right) = \tilde{O}\left(\frac{m\log\left(\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n\right)}{\epsilon^2}\right)$$

implies WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$

Balcan, Prasad, Sandholm, **Vitercik**, NeurIPS'21; Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Cutting plane guarantees

**Theorem:** Training set of size

$$\tilde{O}\left(\frac{\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right) = \tilde{O}\left(\frac{m\log\left(\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n\right)}{\epsilon^2}\right)$$

$\|A\|_{1,1} + \|\boldsymbol{b}\|_1 + n$

$\mathcal{F}$ = hyperplanes in $\mathbb{R}^m$
$\mathrm{VCdim}(\mathcal{F}^*) = O(m)$

$\mathcal{G}$ = constant functions in $\mathbb{R}^m$
$\mathrm{Pdim}(\mathcal{G}^*) = O(m)$

implies WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$

Balcan, Prasad, Sandholm, **Vitercik**, NeurIPS'21; Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. **Online algorithm configuration**

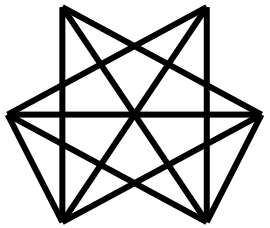
Gupta, Roughgarden, ITCS'16
Balcan, Dick, **Vitercik**, FOCS'18
Balcan, Dick, Pegden, UAI'20

# Online algorithm configuration

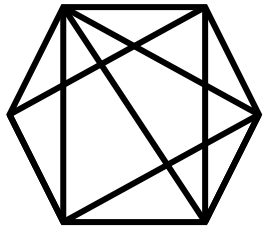
What if inputs are not i.i.d., but even adversarial?

E.g., MWIS:

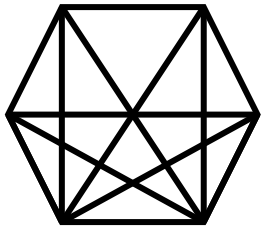
Day 1: $\rho_1$



Day 2: $\rho_2$



Day 3: $\rho_3$



• • •

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

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

# Online model

Over $T$ timesteps $t = 1, \ldots, T$:

1. Learner chooses **parameter setting** $\boldsymbol{\rho}_t$
2. Nature (or adversary 😈) chooses **problem instance** $x_t$
3. Learner obtains **reward** $u_{\boldsymbol{\rho}_t}(x_t) = u^*_{x_t}(\boldsymbol{\rho}_t)$
4. Learner **observes function** $u^*_{x_t}$ (full information feedback)
    - Simplest setting so we'll start here
    - Will look at other feedback models later (e.g., bandit)

# Online model

Over $T$ timesteps $t = 1, \ldots, T$:

1. Learner chooses **parameter setting** $\boldsymbol{\rho}_t$
2. Nature (or adversary 😈) chooses **problem instance** $x_t$
3. Learner obtains **reward** $u_{\boldsymbol{\rho}_t}(x_t) = u^*_{x_t}(\boldsymbol{\rho}_t)$
4. Learner **observes function** $u^*_{x_t}$ (full information feedback)

**Goal:** Minimize ***regret*** $\max_{\boldsymbol{\rho}} \sum_{t=1}^{T} u_{\boldsymbol{\rho}}(x_t) - \sum_{t=1}^{T} u_{\boldsymbol{\rho}_t}(x_t)$

Ideally, $\frac{1}{T} \cdot (\text{Regret}) \to 0$ as $T \to \infty$

*On average, competing with best algorithm in hindsight*

# Outline (theoretical guarantees)

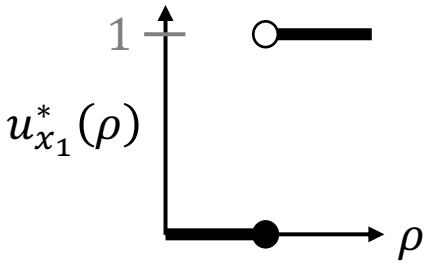
1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. **Worst-case instance**
   ii. Dispersion
   iii. Semi-bandit model

# Worst-case MWIS instance

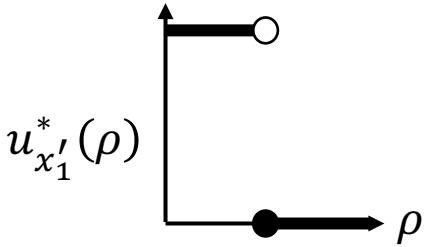
Exists adversary choosing MWIS instances s.t.:

**Every** full information online algorithm has **linear regret**

Round 1:




Dual function: Utility on instance $x_1$ as function of $\rho$




Dual function: Utility on instance $x_1'$ as function of $\rho$

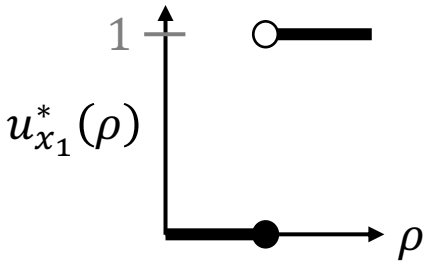
Gupta and Roughgarden, ITCS'16

# Worst-case MWIS instance

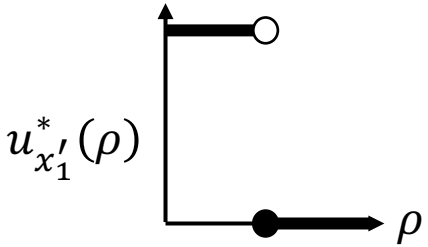
Exists adversary choosing MWIS instances s.t.:

**Every** full information online algorithm has **linear regret**

Round 1:




Adversary chooses $x_1$ or $x_1'$ with equal probability



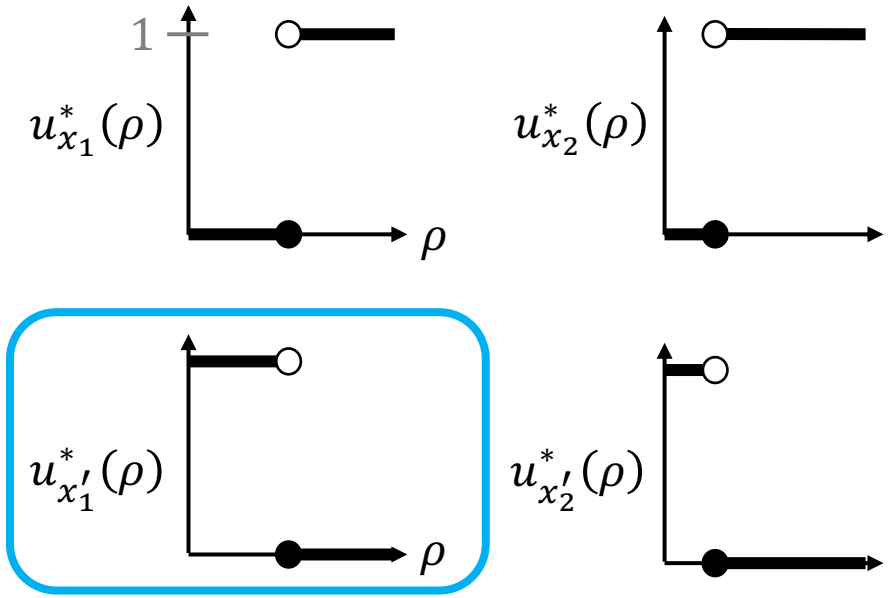

Gupta and Roughgarden, ITCS'16

# Worst-case MWIS instance

Exists adversary choosing MWIS instances s.t.:

**Every** full information online algorithm has **linear regret**

Round 1: Round 2:



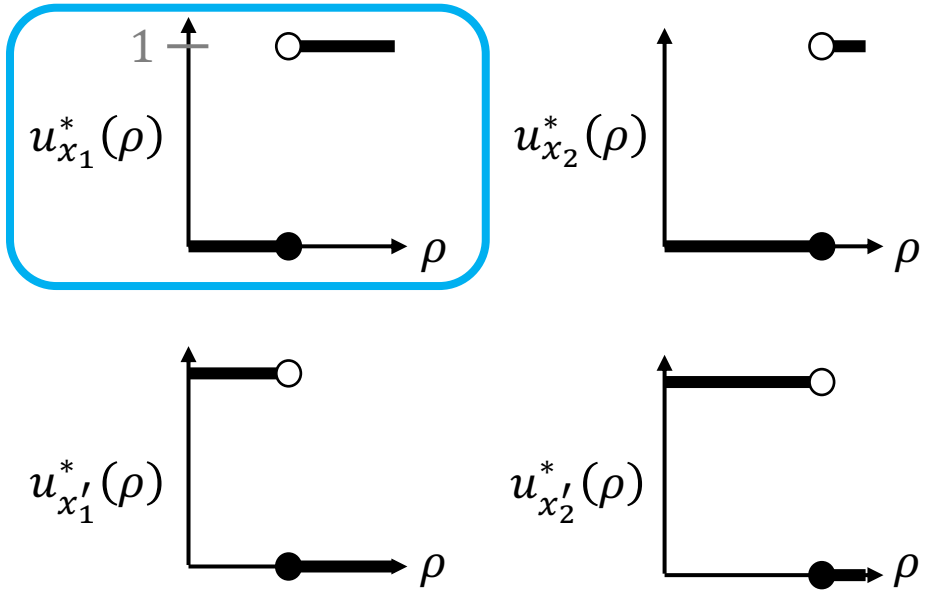

Gupta and Roughgarden, ITCS'16

# Worst-case MWIS instance

Exists adversary choosing MWIS instances s.t.:

**Every** full information online algorithm has **linear regret**

Round 1: Round 2:




Repeatedly halves optimal region

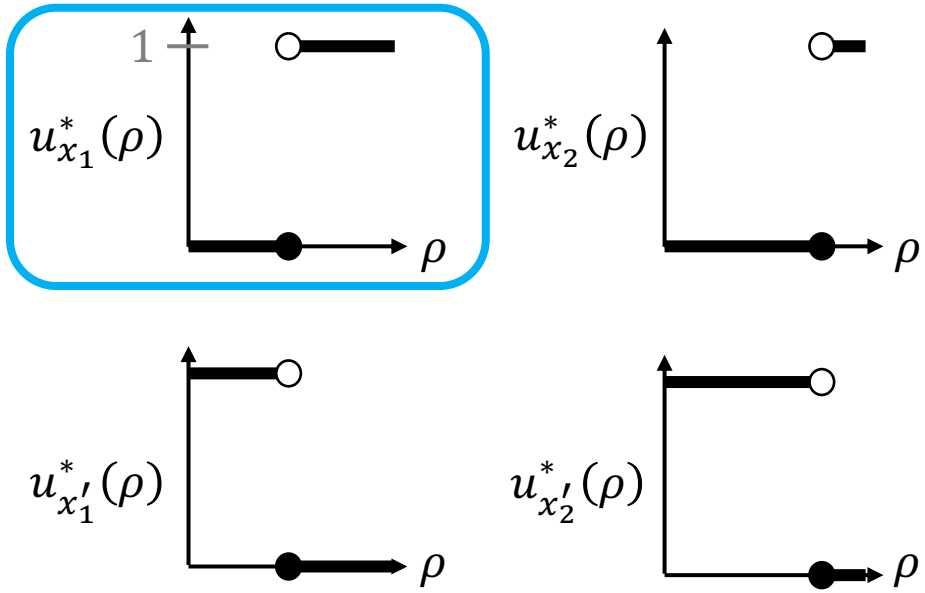
Gupta and Roughgarden, ITCS'16

# Worst-case MWIS instance

Exists adversary choosing MWIS instances s.t.:

**Every** full information online algorithm has **linear regret**

Round 1: Round 2:




Repeatedly halves optimal region

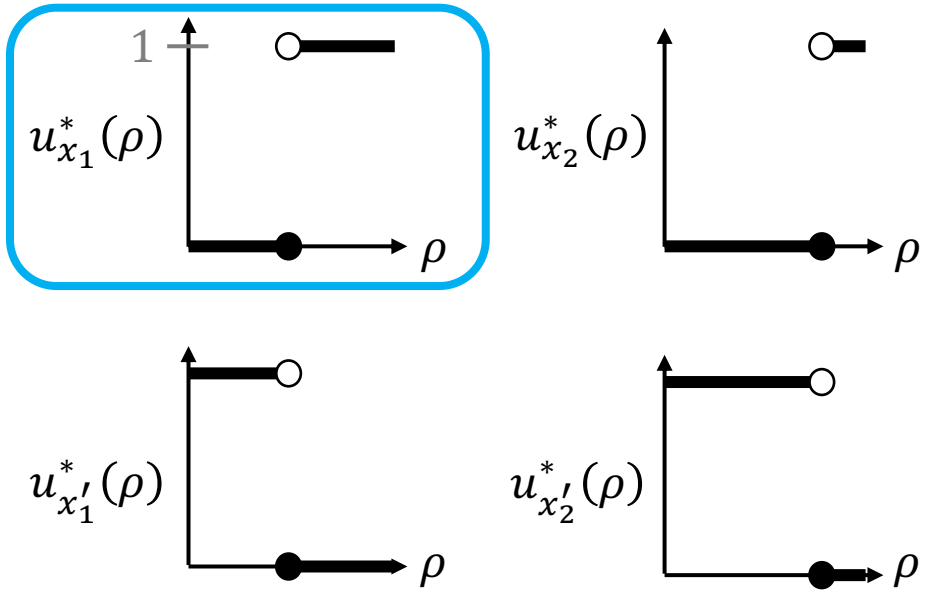
Gupta and Roughgarden, ITCS'16

# Worst-case MWIS instance

Exists adversary choosing MWIS instances s.t.:

**Every** full information online algorithm has **linear regret**

Round 1: Round 2:




Repeatedly halves optimal region

Learner's expected reward: $\frac{T}{2}$

Reward of best $\rho$ in hindsight: $T$

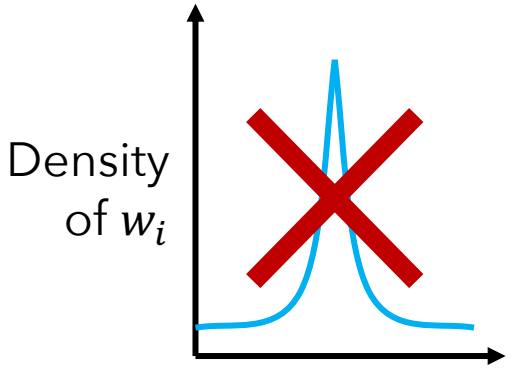
Expected regret = $\frac{T}{2}$

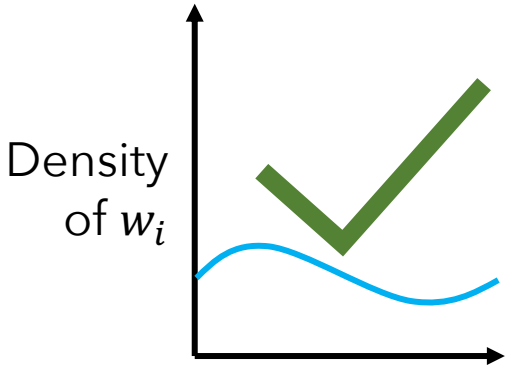
Gupta and Roughgarden, ITCS'16

# Smoothed adversary: MWIS

Sub-linear regret is possible if adversary has a "shaky hand":

- Node weights $w_1, \ldots, w_n$ and degrees $k_1, \ldots, k_n$ are stochastic
- Joint density of $(w_i, w_j, k_i, k_j)$ is bounded




Later generalized by Cohen-Addad, Kanade [AISTATS, '17]; Balcan, Dick, Vitercik [FOCS'18]; Balcan et al. [UAI'20]; …

Gupta and Roughgarden, ITCS'16

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. Worst-case instance
   **ii. Dispersion**
   iii. Semi-bandit model

# Dispersion

Mean adversary concentrates discontinuities near maximizer $\rho^*$

Even points very close to $\rho^*$ have low utility!

$u^*_{x_1}, \ldots, u^*_{x_T}: \underline{B(\mathbf{0}, 1)} \to [-1,1]$ are $(w, k)$***-dispersed at point*** $\boldsymbol{\rho}$ if:

*Can be generalized to any bounded subset*

Balcan, Dick, **Vitercik**, FOCS'18

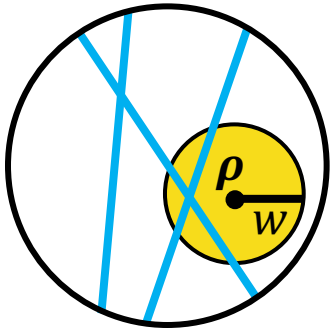
# Dispersion

Mean adversary concentrates discontinuities near maximizer $\rho^*$

Even points very close to $\rho^*$ have low utility!

$u^*_{x_1}, \ldots, u^*_{x_T} : B(\mathbf{0}, 1) \to [-1,1]$ are ***$(w, k)$-dispersed at point $\rho$*** if:

$\ell_2$-ball $B(\boldsymbol{\rho}, w)$ contains discontinuities for $\leq k$ of $u^*_{x_1}, \ldots, u^*_{x_T}$




Ball of radius $w$ about $\boldsymbol{\rho}$ contains 2 discontinuities

$\Rightarrow (w, 2)$-dispersed at $\boldsymbol{\rho}$

Balcan, Dick, **Vitercik**, FOCS'18

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. Worst-case instance
   ii. Dispersion
      a. **Algorithm**
      b. Regret bound
      c. Bandit feedback
      d. Proving dispersion holds
   iii. Semi-bandit model

# Exponentially weighted forecaster

[Freund, Schapire, JCSS'97, Cesa-Bianchi & Lugosi '06, …]

**input:** Learning rate $\eta > 0$

**initialization:** $U_0(\boldsymbol{\rho}) = 0$ is the constant function

for $t = 1, \dots, T$:

choose distribution $\boldsymbol{q}_t$ over $\mathbb{R}^d$ such that $\underline{\boldsymbol{q}_t(\boldsymbol{\rho}) \propto \exp\big(\eta U_{t-1}(\boldsymbol{\rho})\big)}$

Exponentially upweight high-performance parameter settings

choose parameter setting $\boldsymbol{\rho}_t \sim \boldsymbol{q}_t$, receive reward $u^*_{x_t}(\boldsymbol{\rho}_t)$

observe utility function $u^*_{x_t} : B(\mathbf{0}, 1) \to [0,1]$

update $U_t = U_{t-1} + u^*_{x_t}$

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. Worst-case instance
   ii. Dispersion
      a. Algorithm
      **b. Regret bound**
      c. Bandit feedback
      d. Proving dispersion holds
   iii. Semi-bandit model

# Regret

Regret $= \sum_{t=1}^{T} u_{x_t}^*(\boldsymbol{\rho}^*) - \sum_{t=1}^{T} u_{x_t}^*(\boldsymbol{\rho}_t)$

**Theorem:** Suppse $u_{x_1}^*, \ldots, u_{x_T}^* : B(\mathbf{0}, 1) \to [0,1]$ are:

1. Piecewise $L$-Lipschitz
2. $(w, k)$-dispersed at $\boldsymbol{\rho}^*$

EWF has regret $O\left(\sqrt{Td \log \frac{1}{w}} + TLw + k\right)$

**When is this a good bound?**

For $w = \frac{1}{L\sqrt{T}}$ and $k = \tilde{O}(\sqrt{T})$, regret is $\tilde{O}(\sqrt{Td})$

Balcan, Dick, **Vitercik**, FOCS'18

# Regret upper bound: Proof sketch

$$W_t = \int_{B(\mathbf{0},1)} \exp\big(\eta U_t(\boldsymbol{\rho})\big) d\boldsymbol{\rho} \qquad \left( U_t(\boldsymbol{\rho}) = \sum_{\tau=1}^{t} u_\tau^*(\boldsymbol{\rho}) \right)$$

**Goal:** Something in terms of $\mathrm{OPT} = \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho}^*)$ $\leq \frac{W_T}{W_0} \leq$ Something in terms of $\mathrm{ALG} = \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho}_t)$

Learner's performance (ALG) is sufficiently large compared to OPT

Balcan, Dick, **Vitercik**, FOCS'18

# Regret upper bound: Proof sketch

$$W_t = \int_{B(\mathbf{0},1)} \exp\big(\eta U_t(\boldsymbol{\rho})\big) d\boldsymbol{\rho} \qquad \left( U_t(\boldsymbol{\rho}) = \sum_{\tau=1}^{t} u_\tau^*(\boldsymbol{\rho}) \right)$$

**Goal:** Something in terms of $\mathrm{OPT} = \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho}^*)$ $\leq \dfrac{W_T}{W_0} \leq \exp\big(\mathrm{ALG}(e^\eta - 1)\big)$

Standard EWF analysis

Balcan, Dick, **Vitercik**, FOCS'18

# Regret upper bound: Proof sketch

$$W_t = \int_{B(\mathbf{0},1)} \exp\big(\eta U_t(\boldsymbol{\rho})\big) d\boldsymbol{\rho} \qquad \left( U_t(\boldsymbol{\rho}) = \sum_{\tau=1}^{t} u_\tau^*(\boldsymbol{\rho}) \right)$$

**Goal:** Something in terms of $\mathrm{OPT} = \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho}^*)$ $\leq \dfrac{W_T}{W_0} \leq \exp\big(\mathrm{ALG}(e^\eta - 1)\big)$

$$W_T = \int_{B(\mathbf{0},1)} \exp\left(\eta \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho})\right) d\boldsymbol{\rho} \geq \int_{B(\boldsymbol{\rho}^*,w)} \exp\left(\eta \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho})\right) d\boldsymbol{\rho}$$

Balcan, Dick, **Vitercik**, FOCS'18

# Regret upper bound: Proof sketch

**Goal:** Something in terms of $\mathrm{OPT} = \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho}^*)$ $\leq \frac{W_T}{W_0} \leq \exp\big(\mathrm{ALG}(e^{\eta} - 1)\big)$

$$W_T = \int_{B(\mathbf{0},1)} \exp\left(\eta \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho})\right) d\boldsymbol{\rho} \geq \int_{B(\boldsymbol{\rho}^*,w)} \exp\left(\eta \sum_{t=1}^{T} u_t^*(\boldsymbol{\rho})\right) d\boldsymbol{\rho}$$

$$\geq \int_{B(\boldsymbol{\rho}^*,w)} \exp\big(\eta(\mathrm{OPT} - k - TLw)\big) d\boldsymbol{\rho}$$

$$= \mathrm{Vol}\big(B(\boldsymbol{\rho}^*, w)\big) \exp\big(\eta(\mathrm{OPT} - k - TLw)\big)$$

Balcan, Dick, **Vitercik**, FOCS'18

# Regret upper bound: Proof sketch

$$\frac{\mathrm{Vol}\big(B(\boldsymbol{\rho}^*, w)\big)\exp\big(\eta(\mathrm{OPT} - k - TLw)\big)}{\mathrm{Vol}\big(B(\mathbf{0}, 1)\big)} \leq \frac{W_T}{W_0} \leq \exp\big(\mathrm{ALG}(e^\eta - 1)\big)$$

Rearranging and setting $\eta = \sqrt{\frac{d}{T}\log\frac{1}{w}}$:

$$\text{Regret} = \mathrm{OPT} - \mathrm{ALG} = O\left(\sqrt{Td\log\frac{1}{w}} + TLw + k\right)$$

Balcan, Dick, **Vitercik**, FOCS'18

# Matching lower bound

**Theorem:** For any algorithm, exist PW-constant $u_1^*, \dots, u_T^*$ s.t.:

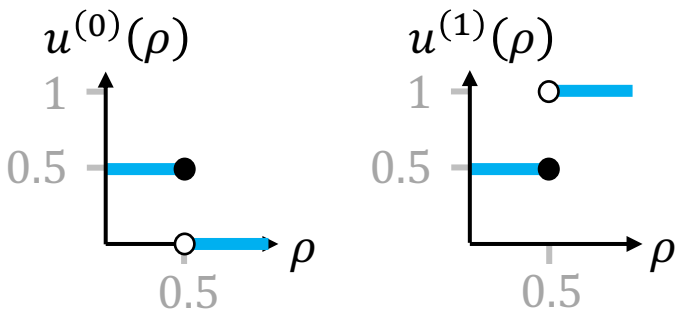
Algorithm's regret is $\Omega\left(\inf_{(w,k)} \sqrt{Td \log \frac{1}{w}} + k\right)$

Inf over all $(w, k)$-dispersion parameters that $u_1^*, \dots, u_T^*$ satisfy at $\boldsymbol{\rho}^*$

Upper bound = $O\left(\inf_{(w,k)} \sqrt{Td \log \frac{1}{w}} + k\right)$

Balcan, Dick, **Vitercik**, FOCS'18

# Regret lower bound: Proof sketch



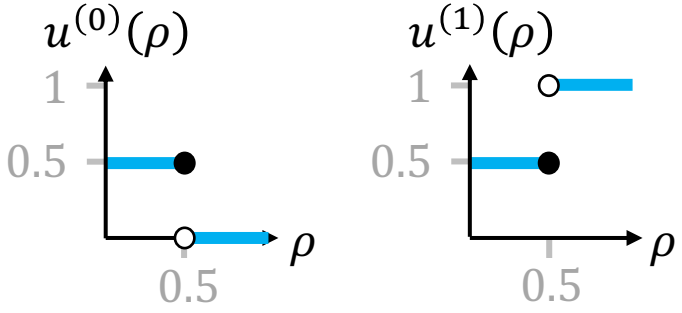

**Lemma** [Weed et al., COLT'16]**:**

Exist distributions $\mu_U, \mu_L$ over $\{u^{(0)}, u^{(1)}\}$ s.t. for any algorithm,

$$\max_{\mu_U,\mu_L} \max_{\rho\in[0,1]} \mathbb{E}\left[\sum_{t=1}^{T} u_t^*(\rho) - \sum_{t=1}^{T} u_t^*(\rho_t)\right] \geq \frac{\sqrt{T}}{32}$$

$u_1^*, \ldots, u_T^*$ drawn from worse of $\mu_U, \mu_L$

# Regret lower bound: Proof sketch




**Lemma** [Weed et al., COLT'16]**:**

Exist distributions $\mu_U, \mu_L$ over $\{u^{(0)}, u^{(1)}\}$ s.t. for any algorithm,

$$\max_{\mu_U,\mu_L} \max_{\rho\in[0,1]} \mathbb{E}\left[\sum_{t=1}^{T} u_t^*(\rho) - \sum_{t=1}^{T} u_t^*(\rho_t)\right] \geq \frac{\sqrt{T}}{32}$$

Any $\rho > 0.5$ is optimal under $\mu_U$, any $\rho \leq 0.5$ is optimal under $\mu_L$

# Regret lower bound: Proof sketch

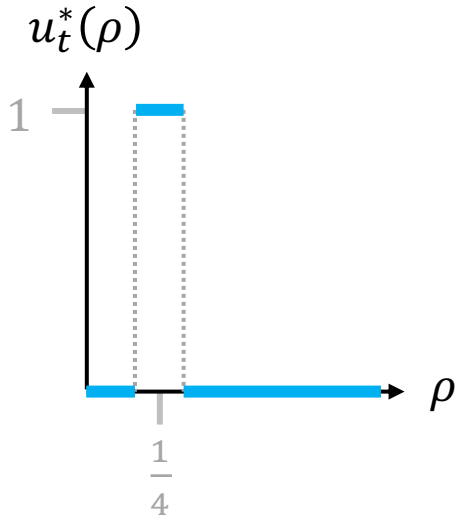
**Worst case instance:**

1. Draw $u_1^*, \ldots, u_{T-\sqrt{T}}^*$ from worse of $\mu_U, \mu_L$ and define:

$$\rho^* = \operatorname*{argmax}_{\rho \in \left\{\frac{1}{4}, \frac{3}{4}\right\}} \sum_{t=1}^{T-\sqrt{T}} u_t^*(\rho)$$

2. Define $u_t^*(\rho) = \mathbf{1}_{\left\{|\rho - \rho^*| \leq \frac{1}{10}\right\}}$ for $t > T - \sqrt{T}$

*Note:* $\rho^* \in \operatorname{argmax} \sum_{t=1}^{T} u_t^*(\rho)$




Balcan, Dick, **Vitercik**, FOCS'18

# Regret lower bound: Proof sketch

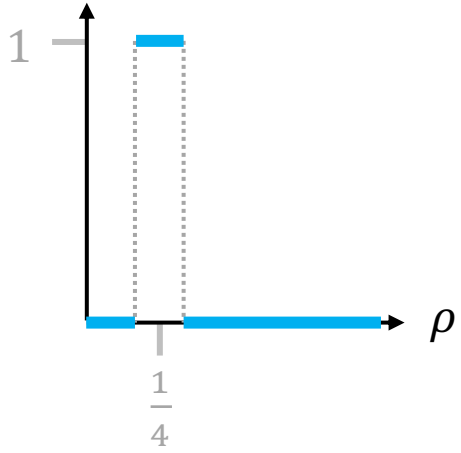
**Analysis:**

- Regret $\geq \frac{\sqrt{T}}{64}$ (follows from lemma by Weed et al., [COLT'16])
- Lower bound follows from fact that $\frac{\sqrt{T}}{64} = \Omega\left(\inf_{(w,k)} \sqrt{T \log \frac{1}{w} + k}\right)$

  Only last $k = \sqrt{T}$ functions have discontinuities in

$$\left[\rho^* - \frac{1}{8}, \rho^* + \frac{1}{8}\right]$$

  $\Rightarrow u_1^*, \ldots, u_T^*$ are $\left(w = \frac{1}{8}, k = \sqrt{T}\right)$-dispersed around $\rho^*$




Balcan, Dick, **Vitercik**, FOCS'18

# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. Worst-case instance
   ii. Dispersion
      a. Algorithm
      b. Regret bound
      **c. Bandit feedback**
      d. Proving dispersion holds
   iii. Semi-bandit model

# Bandit feedback

Over $T$ timesteps $t = 1, \ldots, T$:

1. Learner chooses **parameter setting** $\boldsymbol{\rho}_t$
2. Nature (or adversary 😈) chooses **problem instance** $x_t$
3. Learner obtains **reward** $u_{\boldsymbol{\rho}_t}(x_t) = u^*_{x_t}(\boldsymbol{\rho}_t)$
4. Learner **only** observes $u^*_{x_t}(\boldsymbol{\rho}_t)$ (not entire function)

# Bandit feedback

**Theorem:** If $u_1^*, \ldots, u_T^* : B(\mathbf{0}, 1) \to [0,1]$ are:

1. Piecewise $L$-Lipschitz
2. $(w, k)$-dispersed at $\boldsymbol{\rho}^*$

The UCB algorithm has regret $\tilde{O}\left(\sqrt{Td\left(\frac{1}{w}\right)^d} + TLw + k\right)$

- If $d = 1$, $w = \frac{1}{\sqrt[3]{T}}$, and $k = \tilde{O}\left(T^{2/3}\right)$, regret is $\tilde{O}\left(LT^{2/3}\right)$
- If $w = T^{\frac{d+1}{d+2}-1}$, $k = \tilde{O}\left(T^{\frac{d+1}{d+2}}\right)$, then regret is $\tilde{O}\left(T^{\frac{d+1}{d+2}}\left(\sqrt{d3^d} + L\right)\right)$

Balcan, Dick, **Vitercik**, FOCS'18

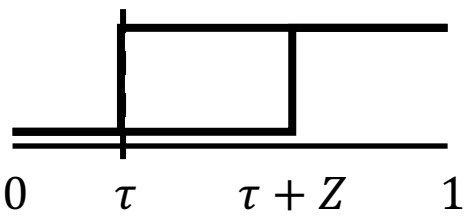
# Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. Worst-case instance
   ii. Dispersion
      a. Algorithm
      b. Regret bound
      c. Bandit feedback
      **d. Proving dispersion holds**
   iii. Semi-bandit model

# Smooth adversaries and dispersion

Adversary chooses thresholds $u_t^*: [0,1] \to \{0,1\}$

Discontinuity $\tau$ "smoothed" by adding $Z \sim N(0, \sigma^2)$




**Lemma**: WHP, $\forall w, u_1^*, \ldots, u_T^*$ are $\left(w, \tilde{O}\left(\frac{Tw}{\sigma} + \sqrt{T}\right)\right)$-dispersed

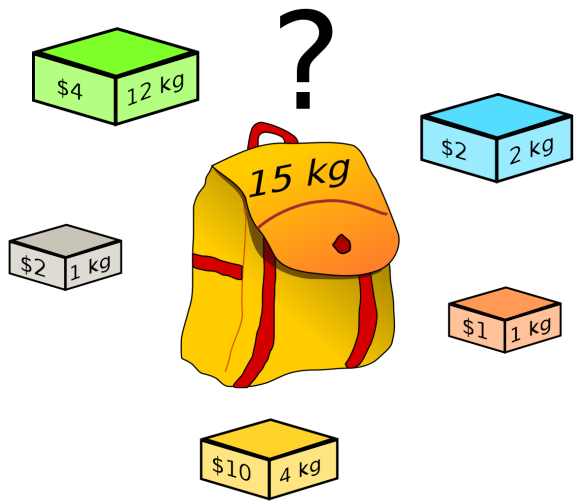
**Corollary:** $w = \frac{\sigma}{\sqrt{T}} \Rightarrow$ **Full information regret =** $\boldsymbol{O\left(\sqrt{T \log \frac{T}{\sigma}}\right)}$

Balcan, Dick, **Vitercik**, FOCS'18

# Simple example: knapsack

**Problem instance**:

- $n$ items, item $i$ has value $v_i$ and size $s_i$
- Knapsack with capacity $K$




**Goal**: find most valuable items that fit

**Algorithm** (parameterized by $\rho \geq 0$)**:**

Add items in decreasing order of $\frac{v_i}{s_i^{\rho}}$

[Gupta and Roughgarden, ITCS'16]

# Dispersion for knapsack

**Theorem**: If instances randomly distributed s.t. on each round:

1. Each $v_i$ independent from $s_i$
2. All $(v_i, v_j)$ have joint density functions with range $\subseteq [0, \kappa]$,

W.h.p., for any $\alpha \geq \frac{1}{2}$, $u_1^*, \ldots, u_T^*$ are

$$\left(\tilde{O}\left(\frac{T^{1-\alpha}}{\kappa}\right), \tilde{O}\left((\#\text{ items})^2 T^{\alpha}\right)\right)\text{-dispersed}$$

**Corollary**: Full information regret = $\tilde{O}\left((\#\text{ items})^2\sqrt{T}\right)$

Balcan, Dick, **Vitercik**, FOCS'18

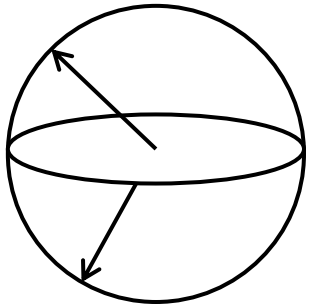
# More results for algorithm configuration

Under **no assumptions**, we show dispersion for
Integer quadratic programming approximation algs

Based on semi-definite programming relaxations

- $s$-linear rounding [Feige & Langberg '06]
- Outward rotations [Zwick '99]
  - Both generalizations of Goemans-Williamson max-cut alg ['95]

Leverage algorithm's randomness to prove dispersion



Balcan, Dick, **Vitercik**, FOCS'18

# Outline (theoretical guarantees)

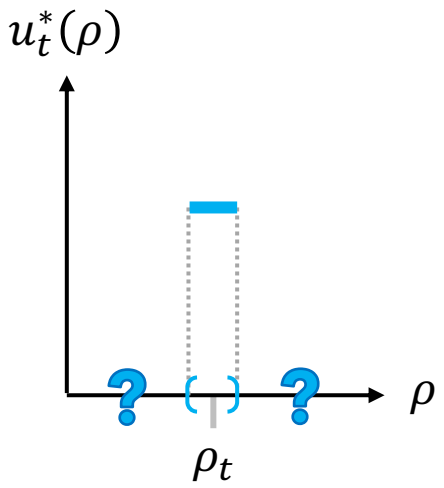
1. Statistical guarantees for algorithm configuration
2. Online algorithm configuration
   i. Worst-case instance
   ii. Dispersion
   **iii. Semi-bandit model**

# Semi-bandit model

- Computing the entire function $u_t^*(\rho)$ can be challenging
- Often, it's easy to compute interval in which $u_t^*(\rho_t)$ is constant
  - E.g., in IP, simple bookkeeping with CPLEX callbacks
- **Semi-bandit model:** learner learns $u_t^*(\rho_t)$ *and* interval

**Balcan, Dick, Pegden** [UAI'20]**:**

- Regret bounds that are nearly as good as full info
- Introduce a more general definition of dispersion



# Outline (applied techniques)

1. **GNNs overview**
2. Neural algorithmic alignment
3. Integer programming with GNNs

# Different types of tasks

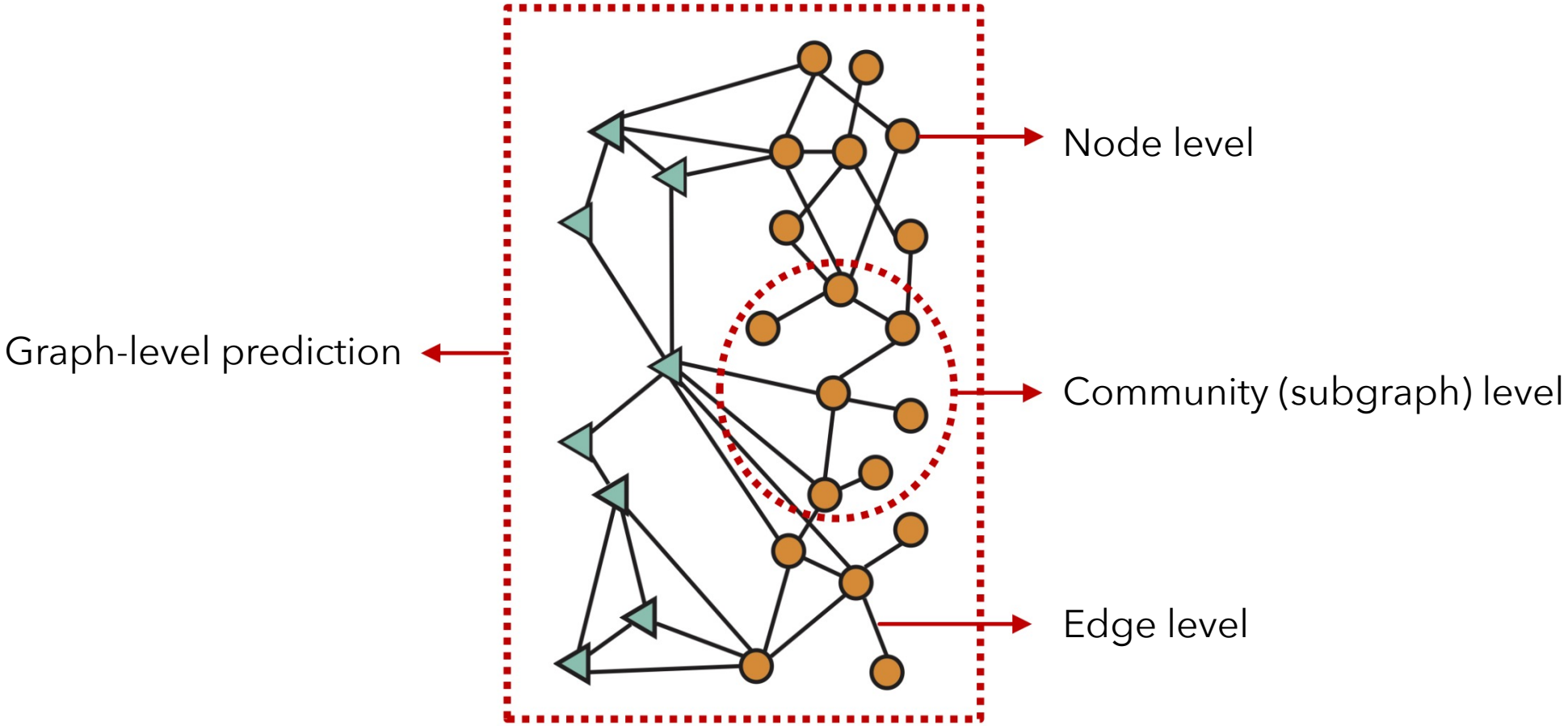
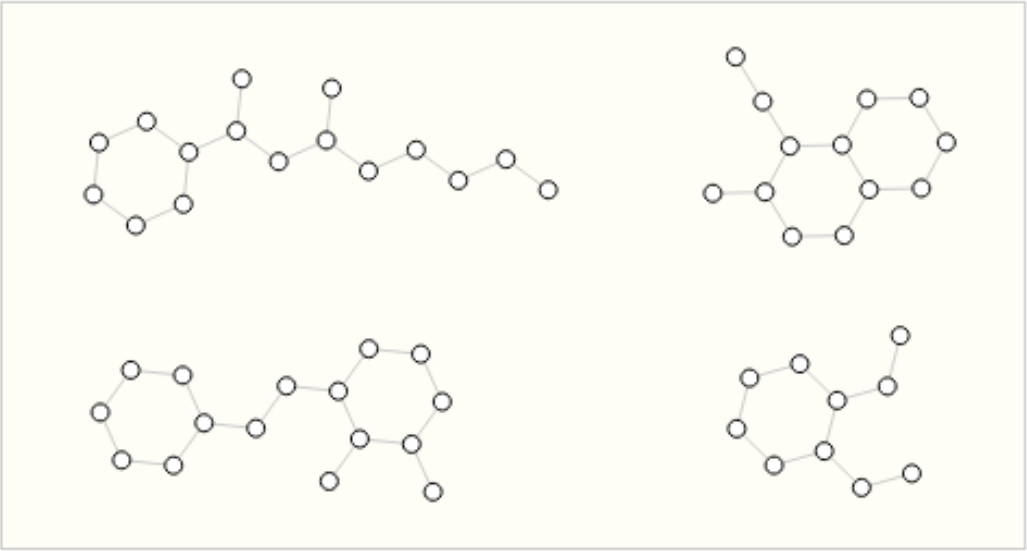
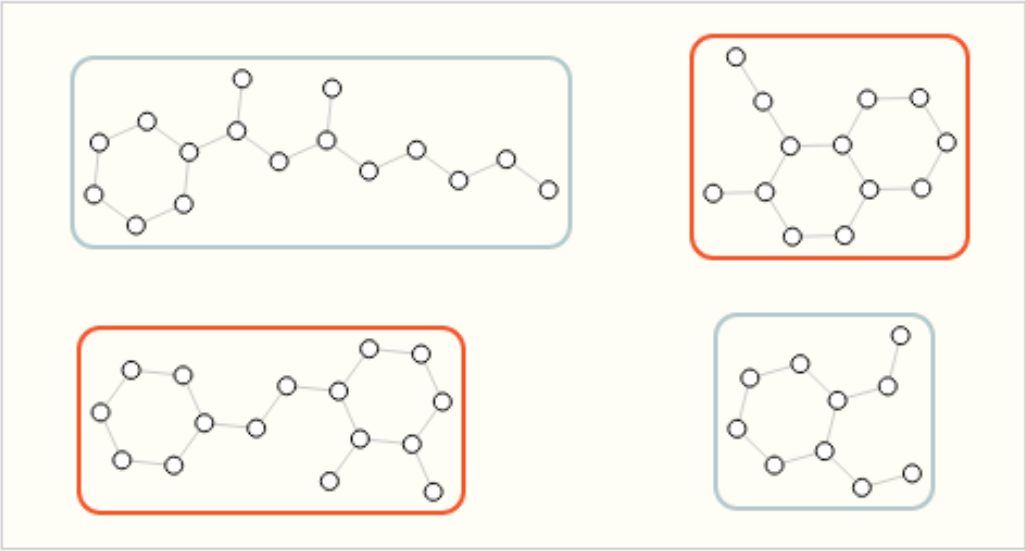



Figure by Leskovec

# Prediction with graphs: Examples





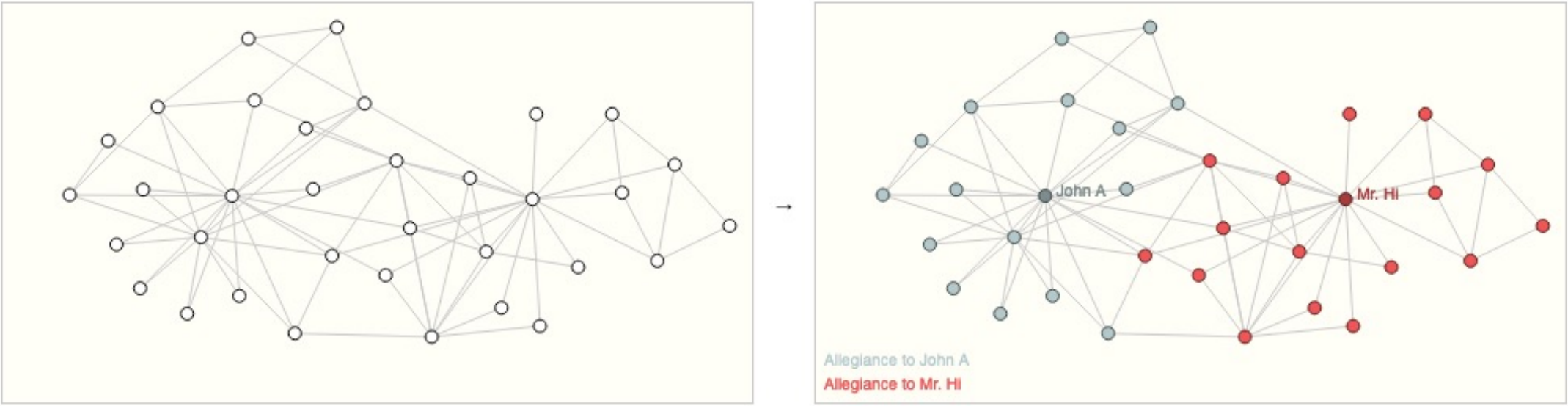
**Graph-level tasks:**

E.g., for a molecule represented as a graph, could predict:

- What the molecule smells like
- Whether it will bind to a receptor implicated in a disease

Figure by Sanchez-Lengeling et al. [‘21]

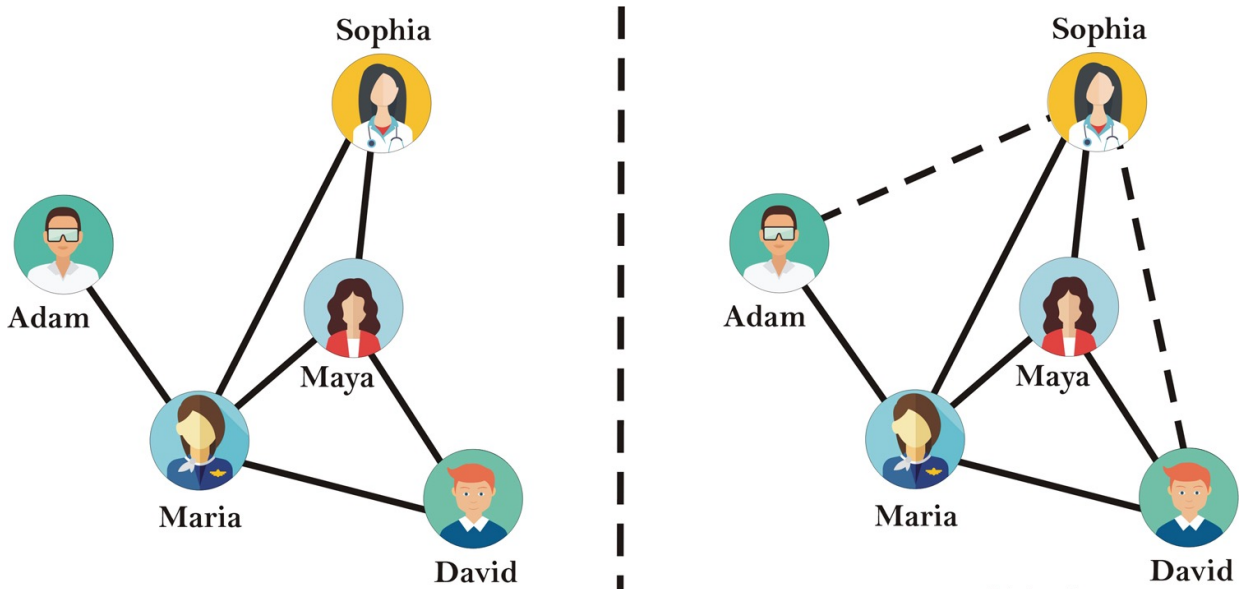
# Prediction with graphs: Examples




**Node-level tasks:**
E.g., political affiliations of users in a social network

Figure by Sanchez-Lengeling et al. [‘21]

# Prediction with graphs: Examples




**Edge-level tasks:** E.g.:

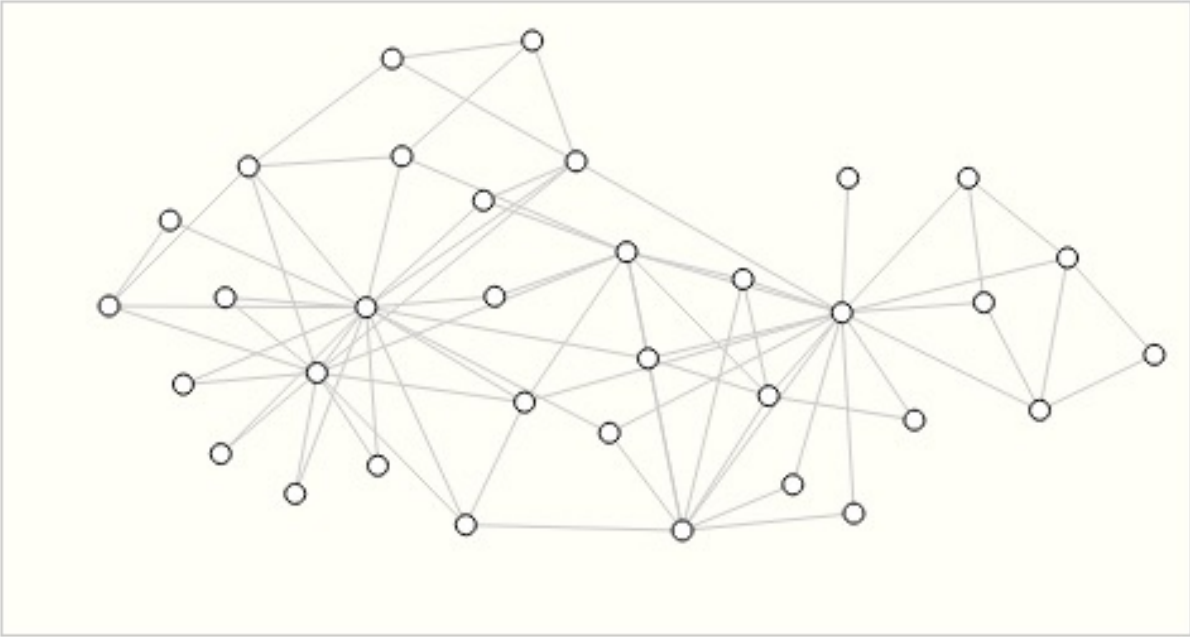
- Suggesting new friends
- Recommendations on Amazon, Netflix, …

Figure by Ahmad et al. ['20]

# GNN motivation

**Main question:**

How to utilize relational structure for better prediction?



# Graph neural networks: First step

- Design features for nodes/links/graphs
- Obtain features for all training data

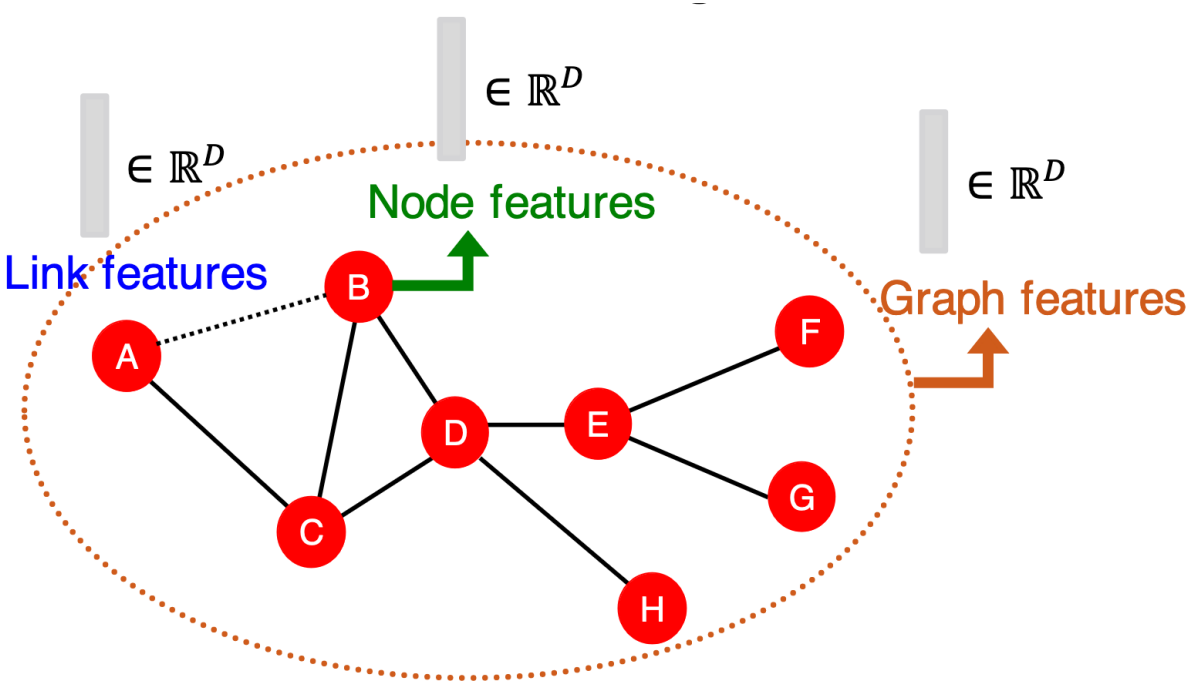
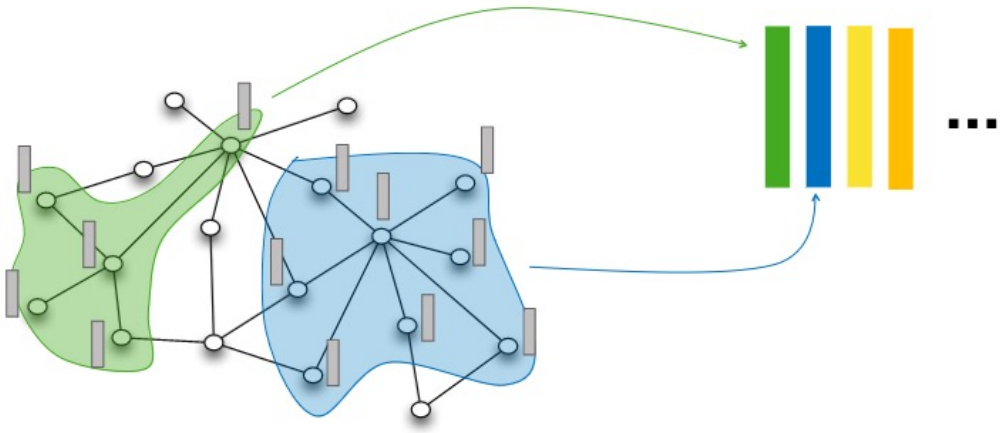



Figure by Leskovec

# Graph neural networks: Objective

**Idea:**

1. Encode each node and its neighborhood with embedding
2. Aggregate set of node embeddings into graph embedding
3. Use embeddings to make predictions



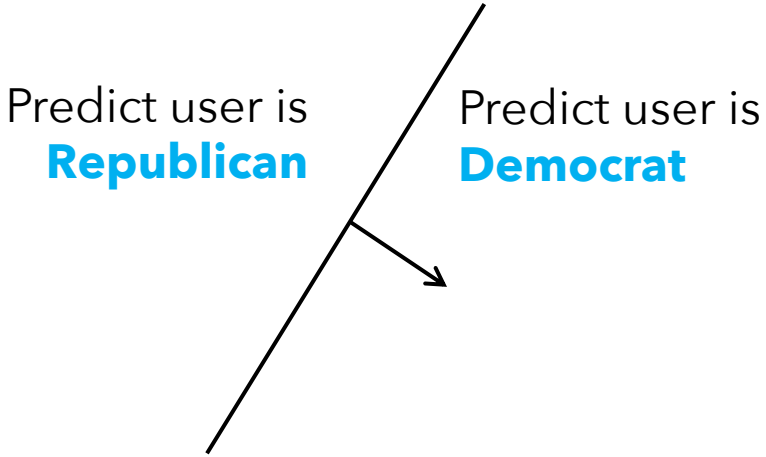
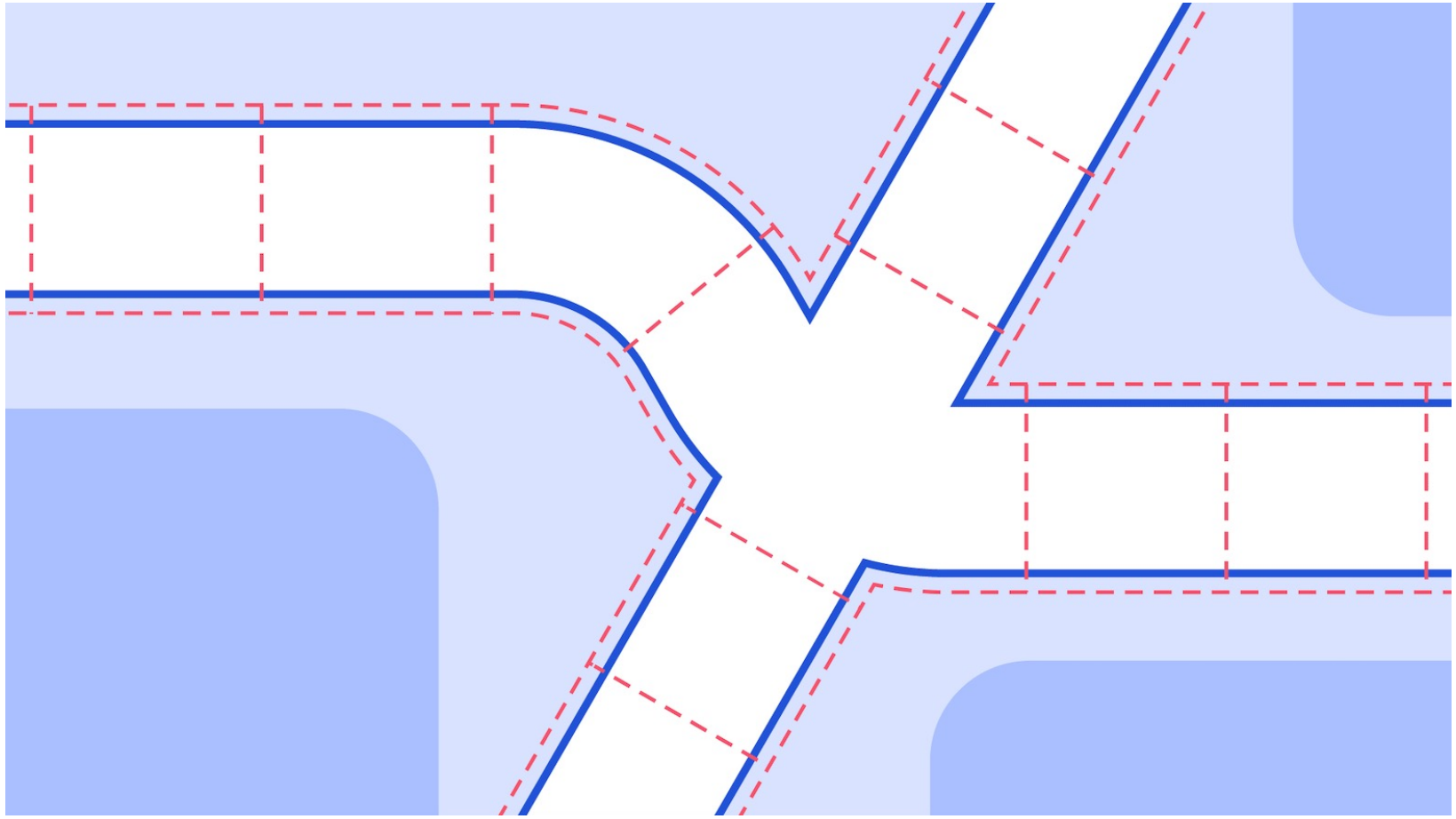
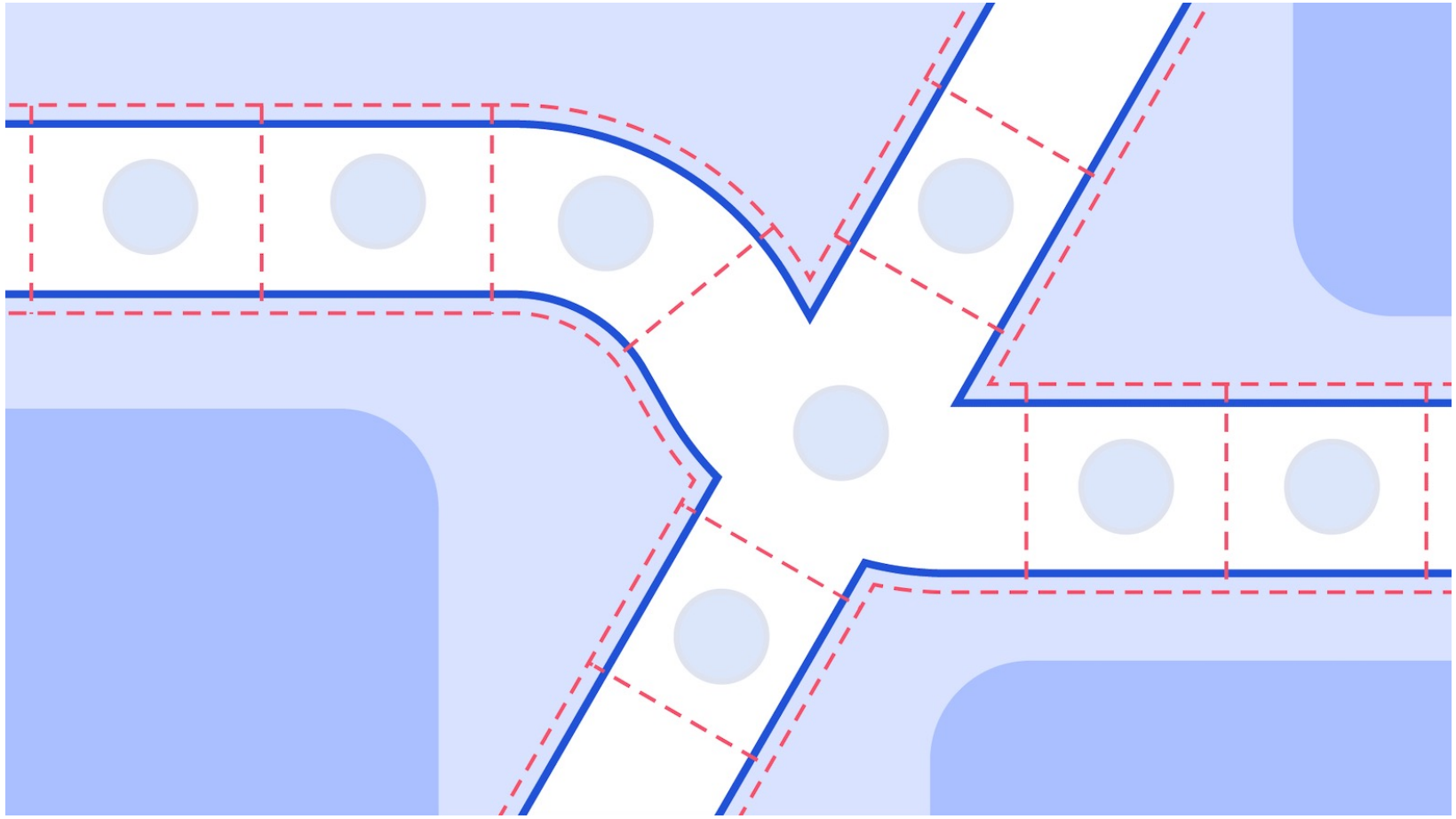




Figure by Jegelka

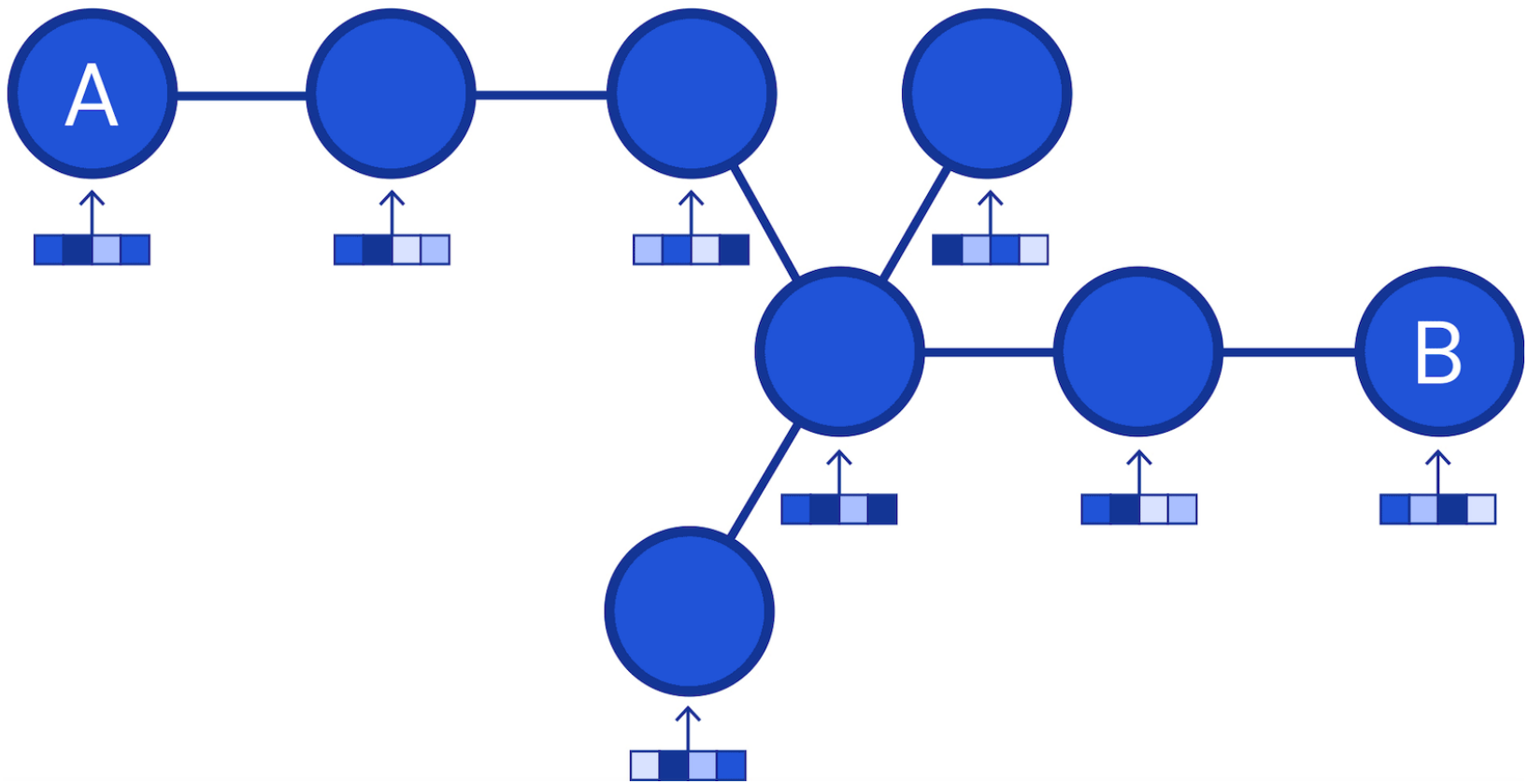
https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks



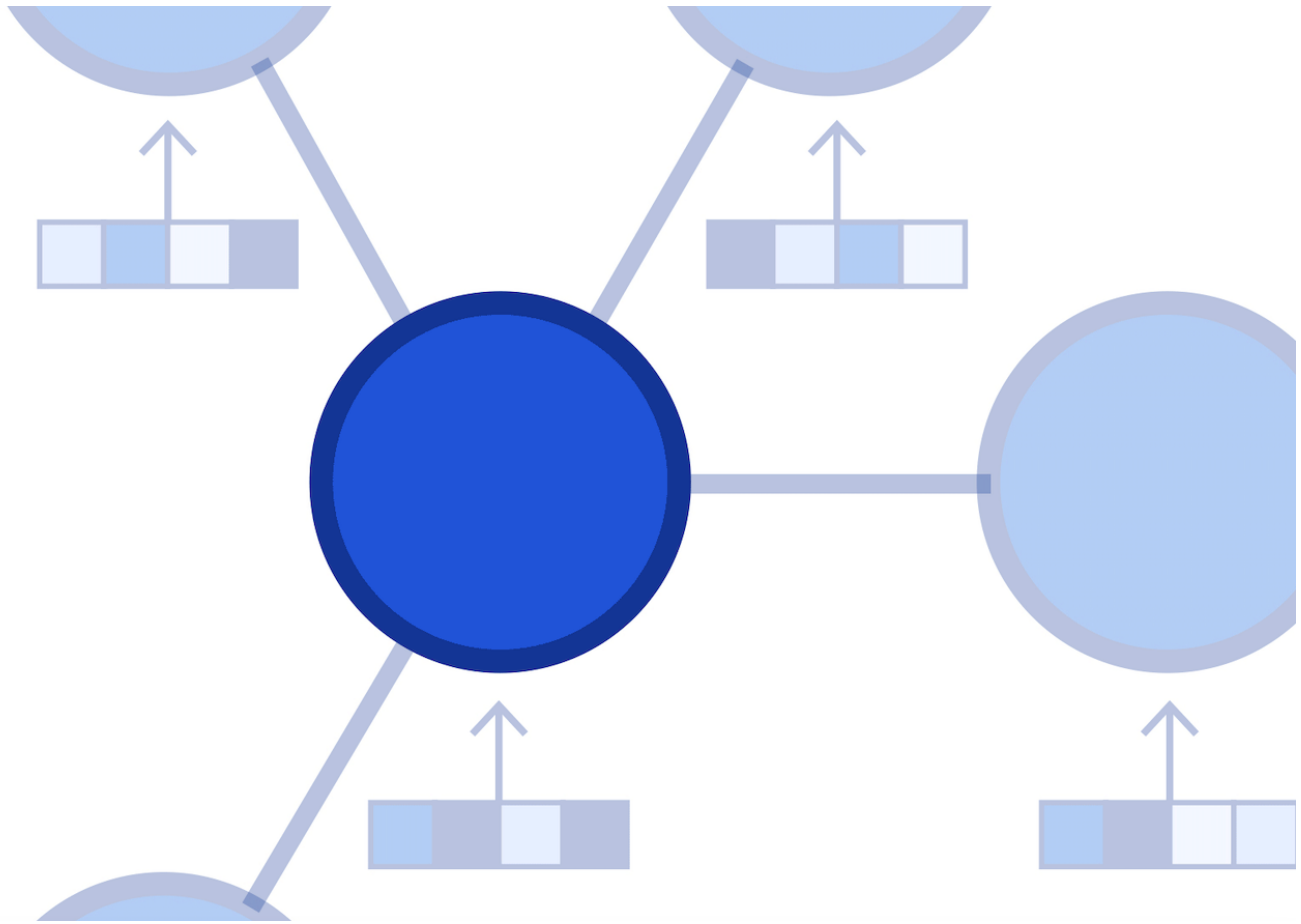
https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks



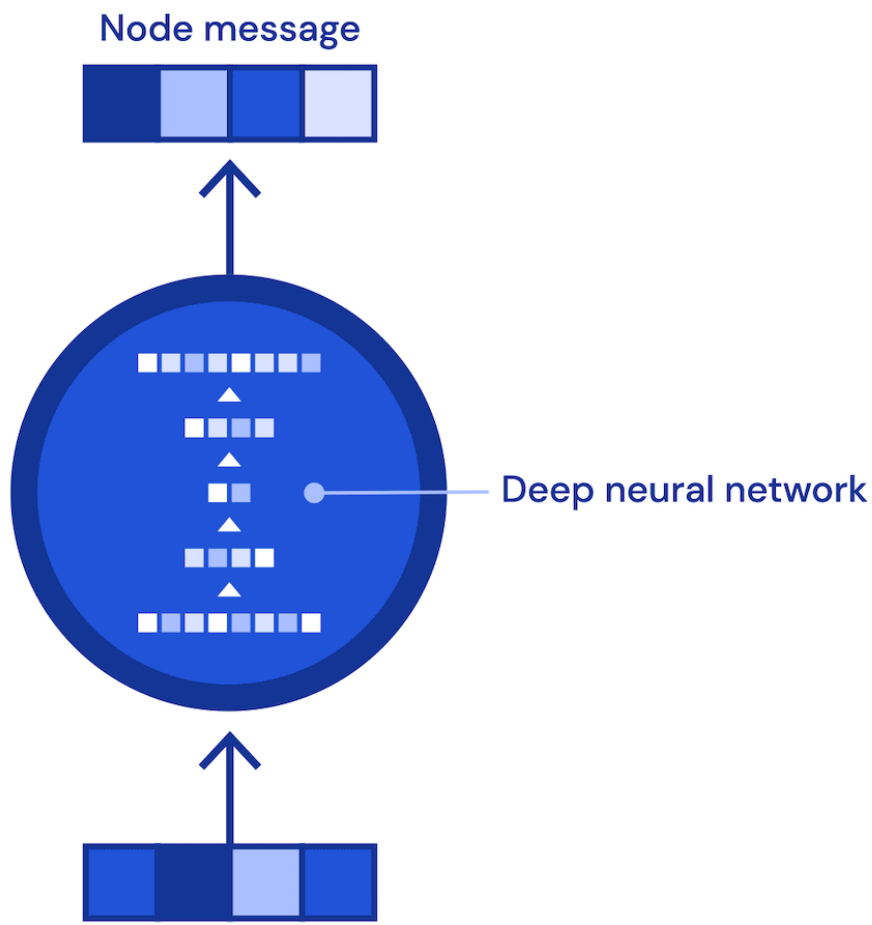
https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks



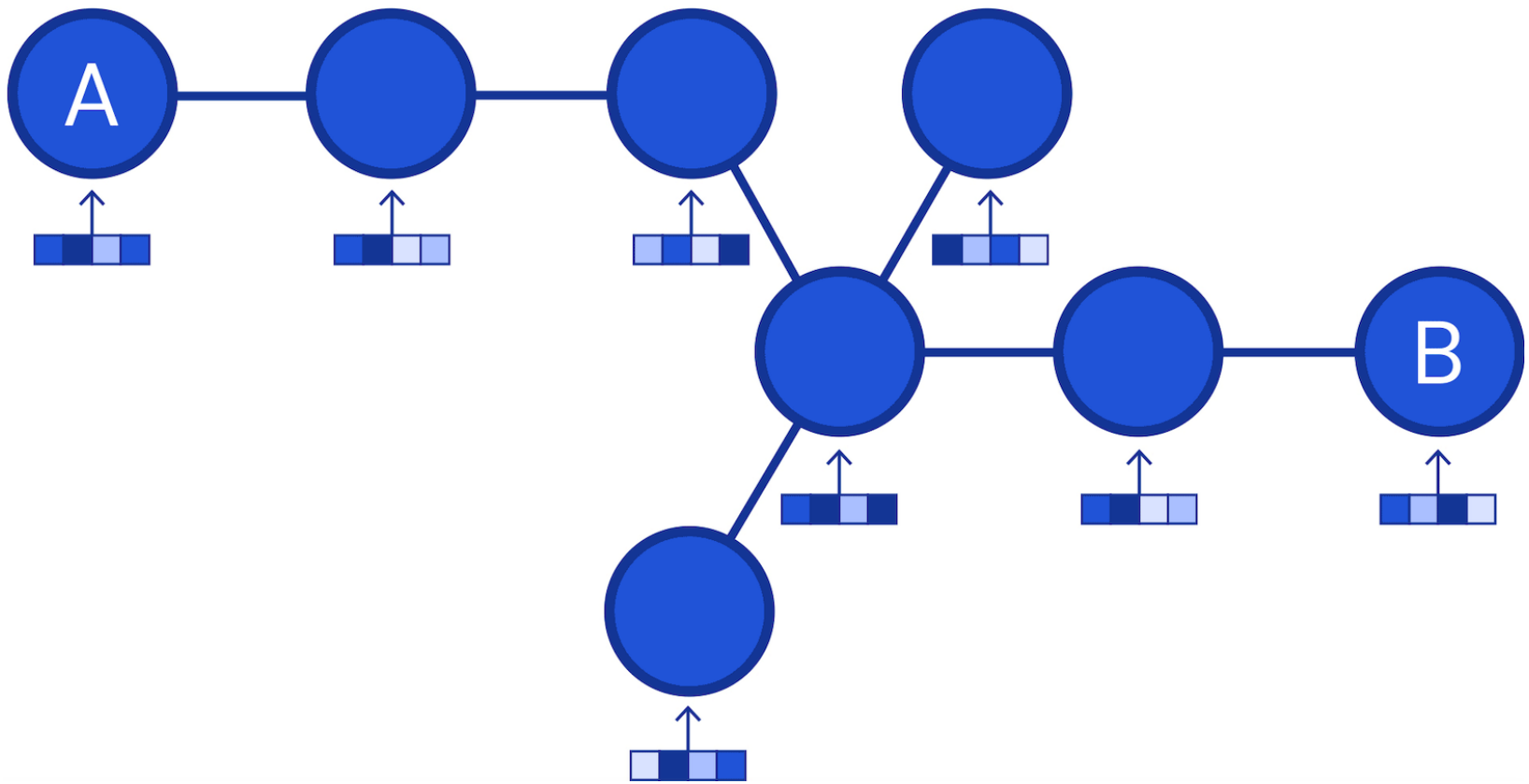

https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks



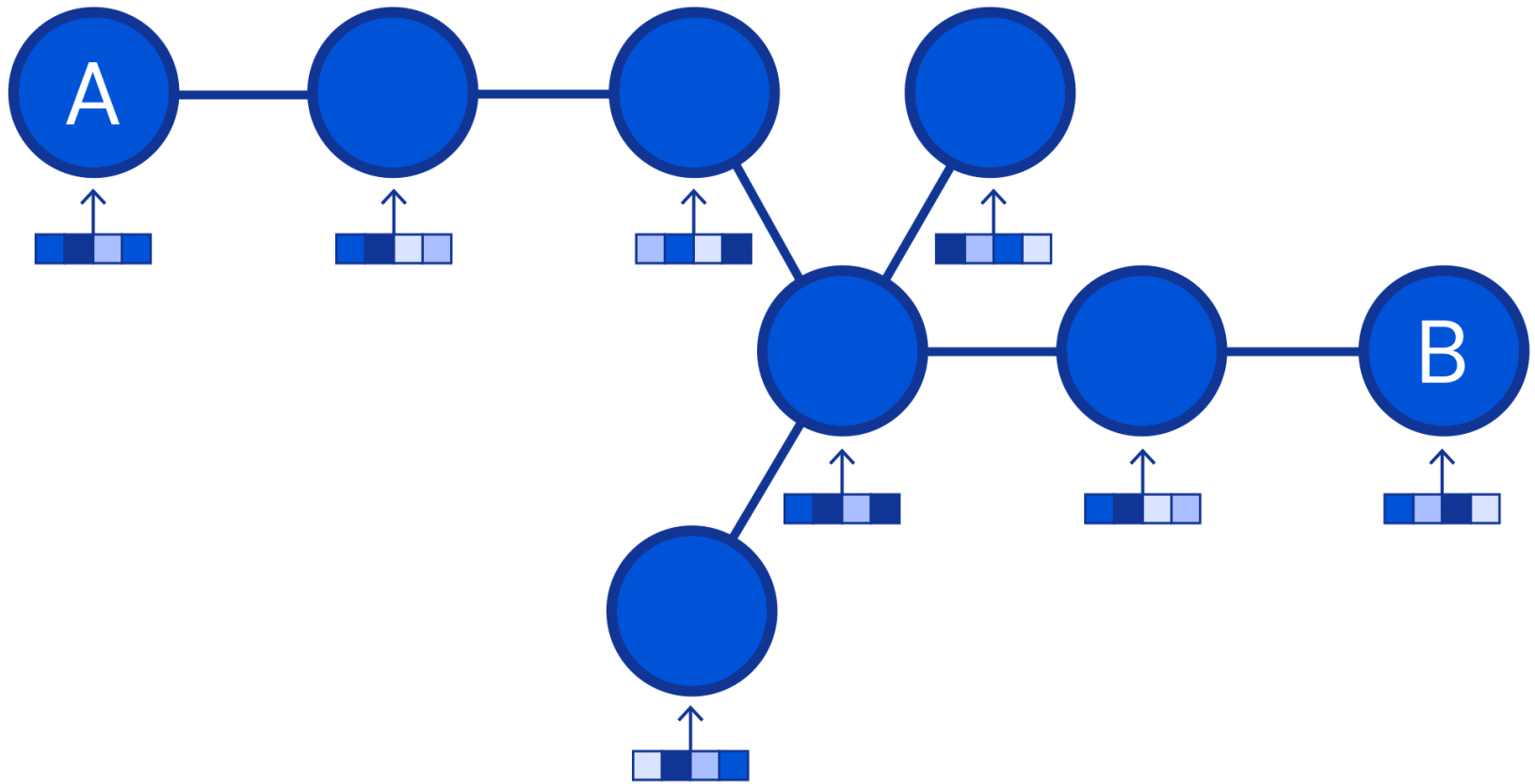
https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks




https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks




https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks




https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks

# Encoding neighborhoods: General form

$\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$ (feature representation for node $u$)

In each round $k \in [K]$, for each node $v$:

1. **Aggregate** over neighbors

$$\boldsymbol{m}_{\underbrace{N(v)}_{\text{Neighborhood of } v}}^{(k)} = \mathrm{AGGREGATE}^{(k)}\left(\left\{\boldsymbol{h}_u^{(k-1)} : u \in N(v)\right\}\right)$$

# Encoding neighborhoods: General form

$\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$ (feature representation for node $u$)

In each round $k \in [K]$, for each node $v$:

1. **Aggregate** over neighbors
$$\boldsymbol{m}_{N(v)}^{(k)} = \mathrm{AGGREGATE}^{(k)}\left(\left\{\boldsymbol{h}_u^{(k-1)} : u \in N(v)\right\}\right)$$
2. **Update** current node representation
$$\boldsymbol{h}_v^{(k)} = \mathrm{COMBINE}^{(k)}\left(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{m}_{N(v)}^{(k)}\right)$$

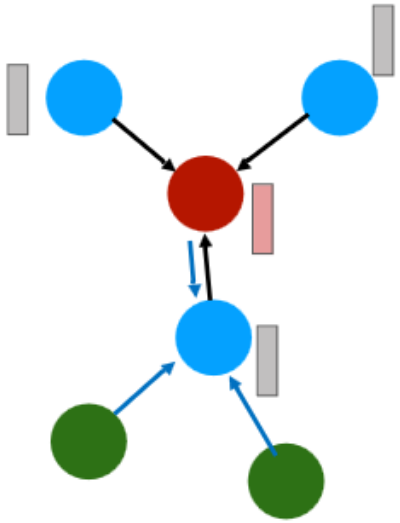
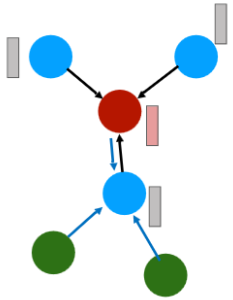


Figure by Jegelka

# The basic GNN

[Merkwirth and Lengauer '05; Scarselli et al. '09]



$$\boldsymbol{m}_{N(v)} = \text{AGGREGATE}(\{\boldsymbol{h}_u : u \in N(v)\}) = \sum_{u \in N(v)} \boldsymbol{h}_u$$

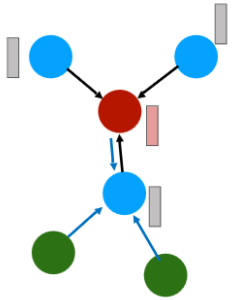
$$\text{COMBINE}\left(\boldsymbol{h}_v, \boldsymbol{m}_{N(v)}\right) = \sigma\left(W_{\text{self}}\boldsymbol{h}_v + W_{\text{neigh}}\boldsymbol{m}_{N(v)} + \boldsymbol{b}\right)$$

Trainable parameters

Non-linearity (e.g., tanh or ReLU)

Figure by Jegelka

# Aggregation functions

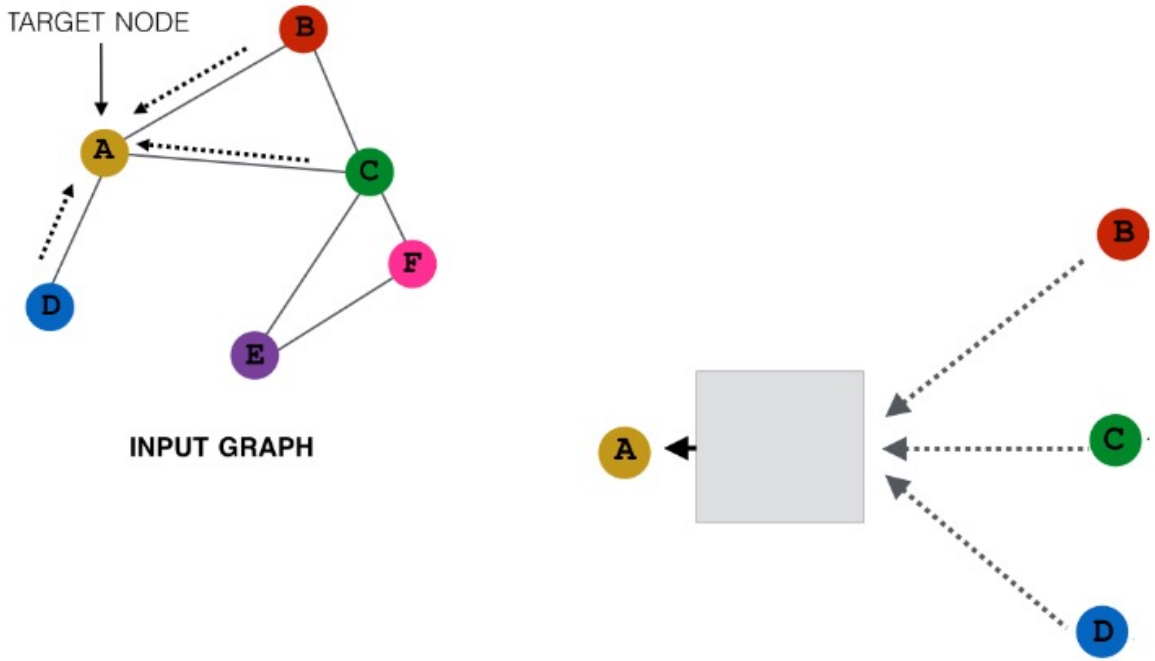


$$\boldsymbol{m}_{N(v)} = \text{AGGREGATE}(\{\boldsymbol{h}_u : u \in N(v)\}) = \bigoplus_{u \in N(v)} \boldsymbol{h}_u$$

Other element-wise aggregators, e.g.:
Maximization, averaging

Figure by Jegelka

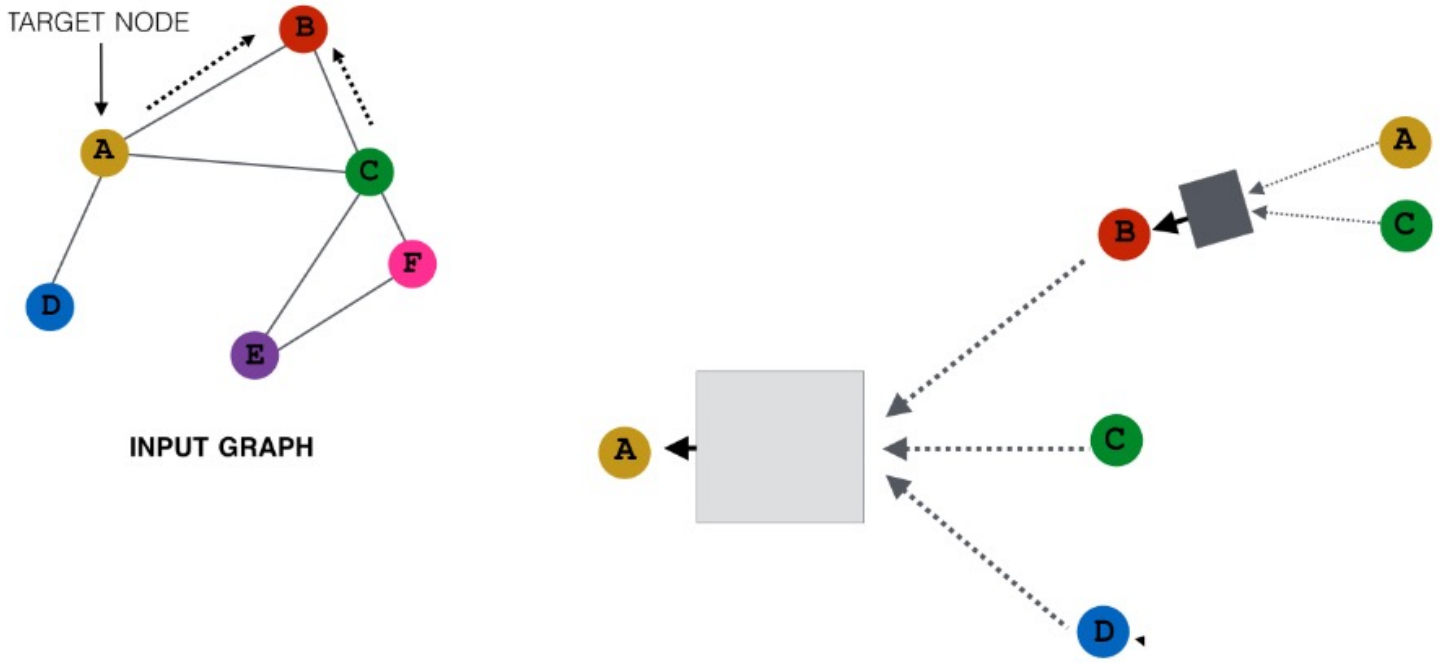
# Node embeddings unrolled




Grey boxes: aggregation functions that we learn

Figures by Leskovec

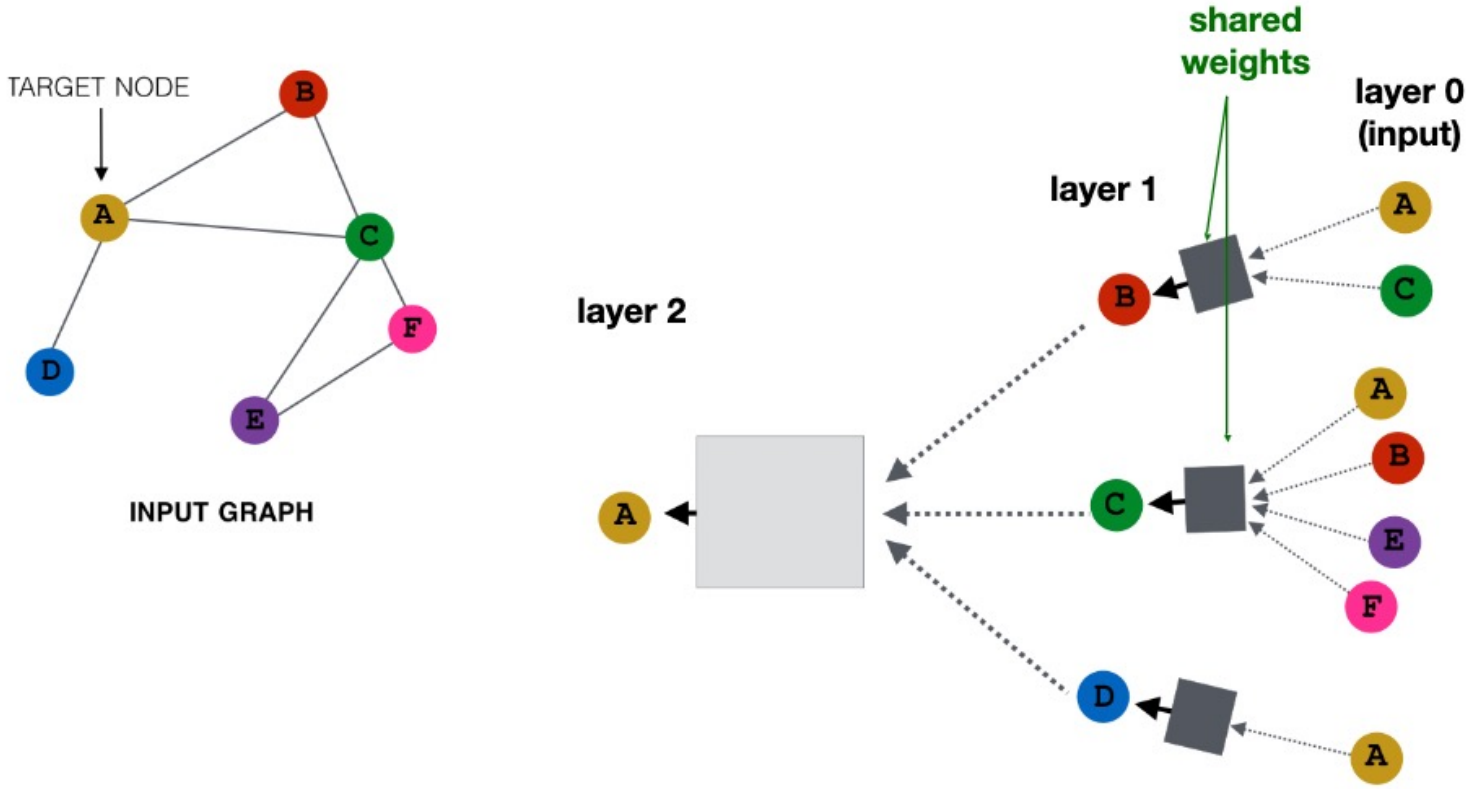
# Node embeddings unrolled




Grey boxes: aggregation functions that we learn

Figures by Leskovec

# Node embeddings unrolled

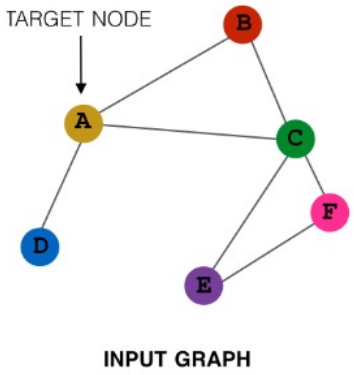


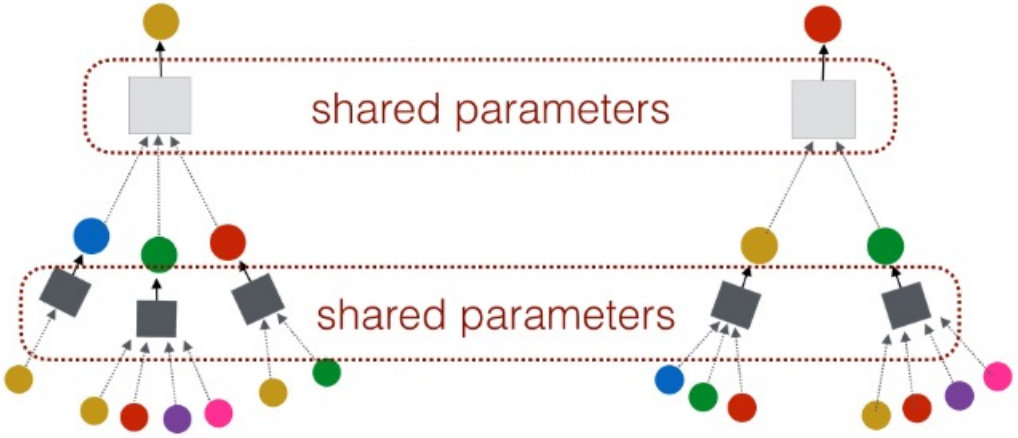

Grey boxes: aggregation functions that we learn

Figures by Leskovec

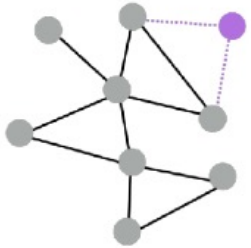
# Weight sharing

Use the same aggregation functions for all nodes



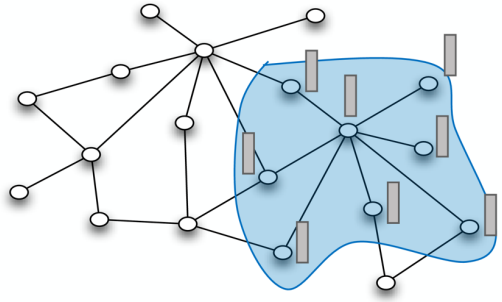

Can generate encodings for
previously unseen nodes & graphs!



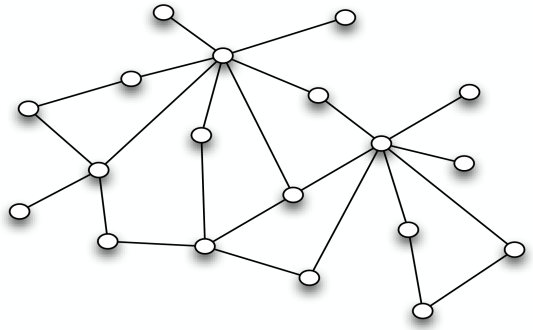
Figures by Leskovec

# Training a GNN

- What is a data point?



Node and its neighborhood



Entire graph

- What to specify?
  - Aggregate and combine functions
  - Readout function: combines node embeddings → graph embedding
  - Loss function on prediction
- Train with SGD

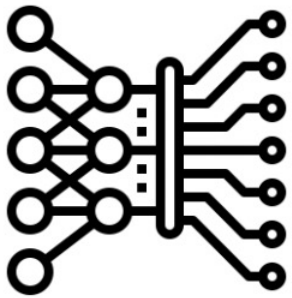
Slide by Jegelka

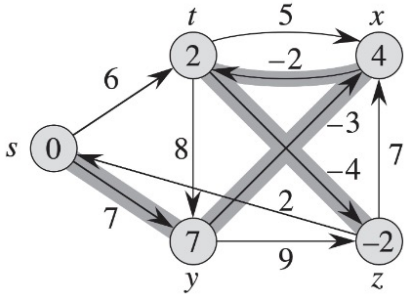
# Outline (applied techniques)

1. GNNs overview
2. **Neural algorithmic alignment**
3. Integer programming with GNNs

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20
Cappart, Chételat, Khalil, Lodi, Morris, Veličković, arXiv'21

# Problem-solving approaches






+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

+ Trivially strong generalization
+ Compositional (subroutines)
+ Guaranteed correctness
+ Interpretable operations
- Input must match spec
- Not robust to task variations

Is it possible to get the best of both worlds?

Slide by Veličković

# Previous work

Previous work:

- Shortest path [Graves et al. '16; Xu et al., '19]
- Traveling salesman [Reed and De Freitas '15]
- Boolean satisfiability [Vinyals et al. '15; Bello et al., '16; …]
- Probabilistic inference [Yoon et al., '18]

Ground-truth solutions used to drive learning

Model has **complete freedom** mapping raw inputs to solutions

# Neural graph algorithm execution

**Key observation:** Many algorithms share related **subroutines**

E.g. Bellman-Ford, BFS enumerate sets of edges adjacent to a node

**Neural graph algorithm execution**

- Learn several algorithms **simultaneously**
- Provide intermediate supervision signals

  Driven by how a known classical algorithm would process the input

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
   i. **Example algorithms**
   ii. Experiments
   iii. Additional motivation
   iv. Additional research
3. Integer programming with GNNs

# Breadth-first search

- Source node $s$
- Initial input $x_i^{(1)} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s \end{cases}$
- Node is reachable from $s$ if any of its neighbors are reachable:

$$x_i^{(t+1)} = \begin{cases} 1 & \text{if } x_i^{(t)} = 1 \\ 1 & \text{if } \exists j \text{ s.t. } (j,i) \in E \text{ and } x_j^{(t)} = 1 \\ 0 & \text{else} \end{cases}$$

- Algorithm output at round $t$: $y_i^{(t)} = x_i^{(t+1)}$

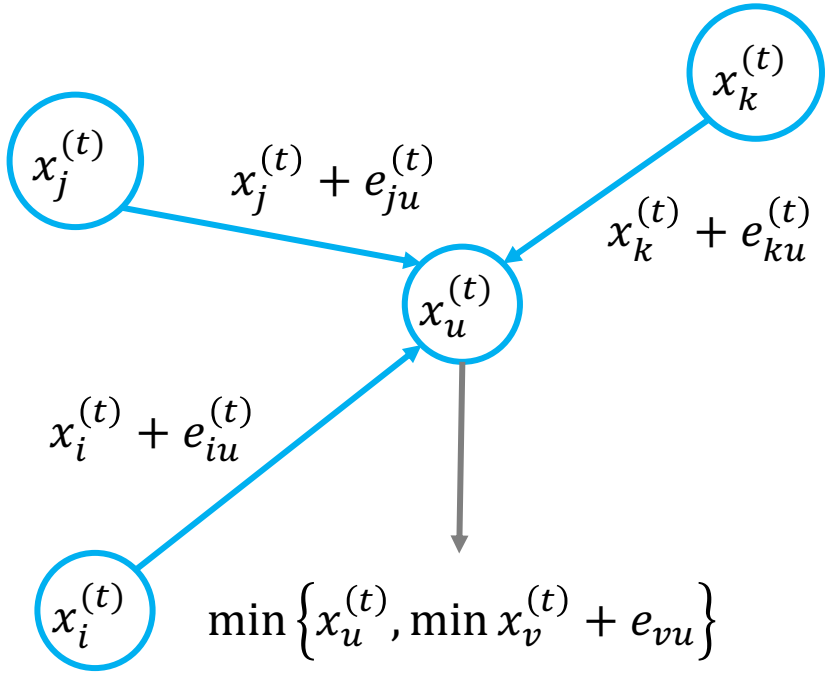
# Bellman-Ford (shortest path)

- Source node $s$
- Initial input $x_i^{(1)} = \begin{cases} 0 & \text{if } i = s \\ \infty & \text{if } i \neq s \end{cases}$
- Node is reachable from $s$ if any of its neighbors are reachable

  Update distance to node as minimal way to reach neighbors

$$x_i^{(t+1)} = \min\left\{x_i^{(t)}, \min_{(j,i)\in E} x_j^{(t)} + e_{ji}^{(t)}\right\}$$

# Bellman-Ford: Message passing




**Key idea (roughly speaking):** Train GNN so that $\boldsymbol{h}_u^{(t)} \approx x_u^{(t)}, \forall t$

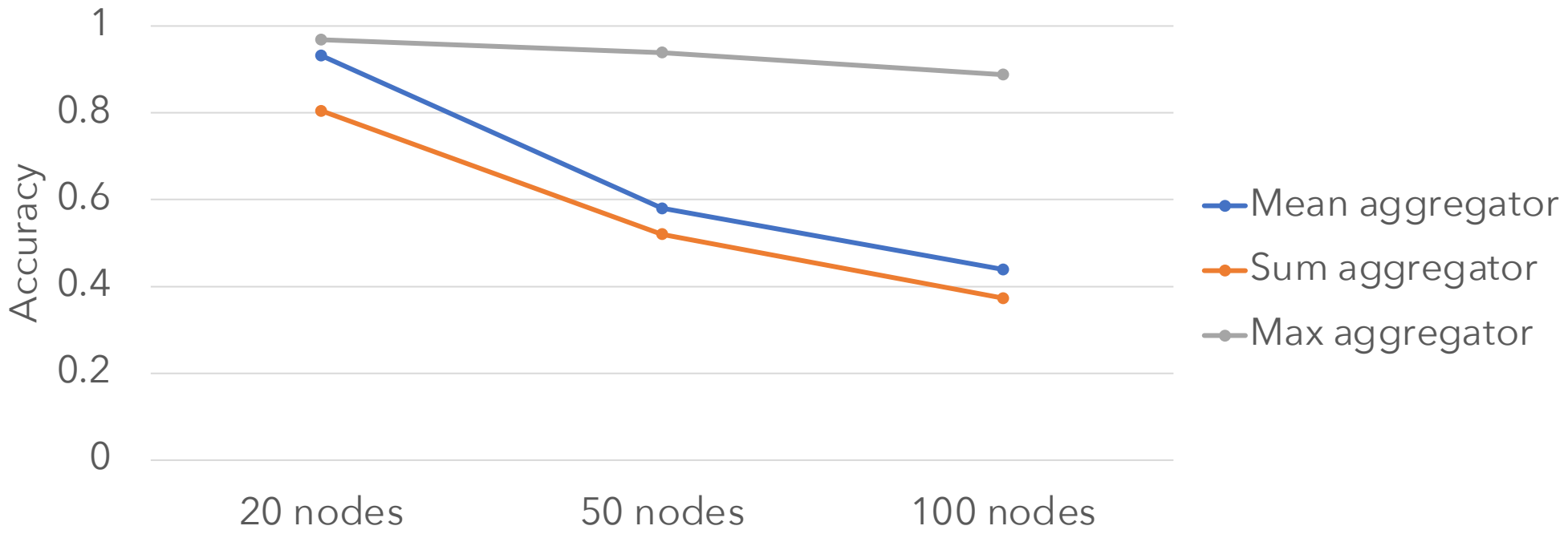
(Really, so that a function of $\boldsymbol{h}_u^{(t)} \approx x_u^{(t)}$)

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
   i. Example algorithms
   **ii. Experiments**
   iii. Additional motivation
   iv. Additional research
3. Integer programming with GNNs

# Shortest-path predecessor prediction




Improvement of max-aggregator increases with size

It **aligns** better with underlying algorithm [Xu et al., ICLR'20]

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

# Learning multiple algorithms

Learn to execute both BFS and Bellman-Ford **simultaneously**

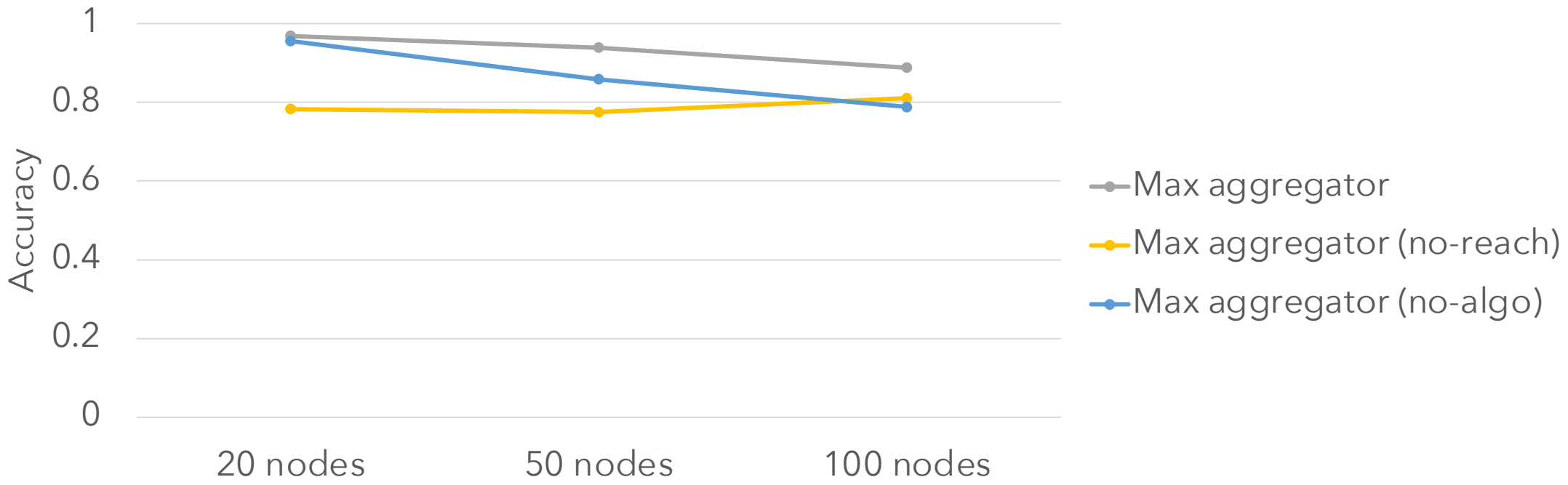
- At each step $t$, concatenate relevant $x_i^{(t)}$ and $\boldsymbol{y}_i^{(t)}$ values

**Comparisons**

- *(no-reach):* Learn Bellman-Ford alone
  - Doesn't simultaneously learn reachability
- *(no-algo):*
  - Don't supervise intermediate steps
  - Learn predecessors directly from input $x_i^{(1)}$

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

# Shortest-path predecessor prediction




- **(no-reach) results**: positive knowledge transfer
- **(no-algo) results**: benefit of supervising intermediate steps

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
   i. Example algorithms
   ii. Experiments
   **iii. Additional motivation**
   iv. Additional research
3. Integer programming with GNNs

# Key question

Key question in neural algorithmic alignment:

If we're just teaching a NN to **imitate** a classical algorithm…

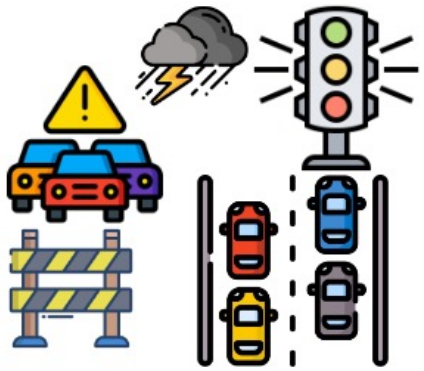
**Why not just run that algorithm?**

# Why use GNNs for algorithm design?

Classical algorithms are designed with **abstraction** in mind

Enforce their inputs to conform to stringent preconditions

However, we design algorithms to solve **real-world** problems!

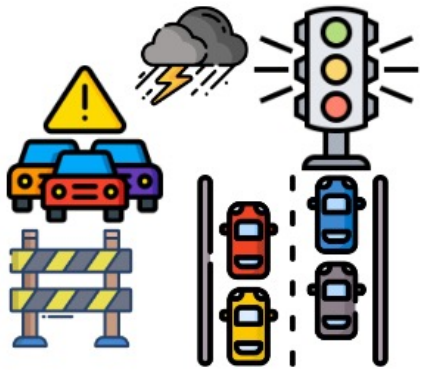


**Natural inputs**

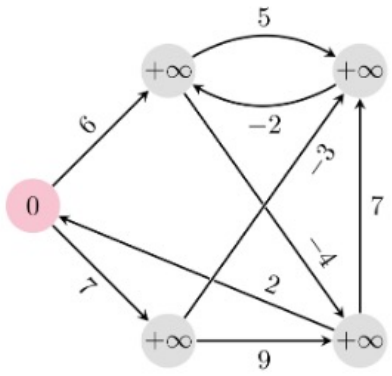
Slide by Veličković

# Abstractifying the core problem

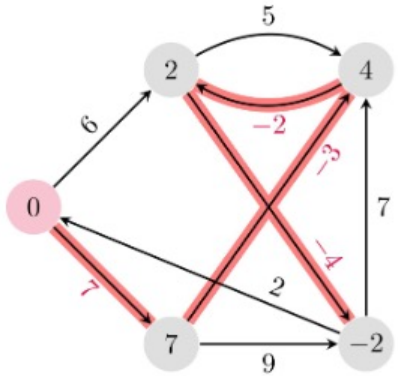
- Assume we have real-world inputs
  …but algorithm only admits abstract inputs
- Could try **manually** converting from one input to another



**Natural inputs**




**Abstract inputs**

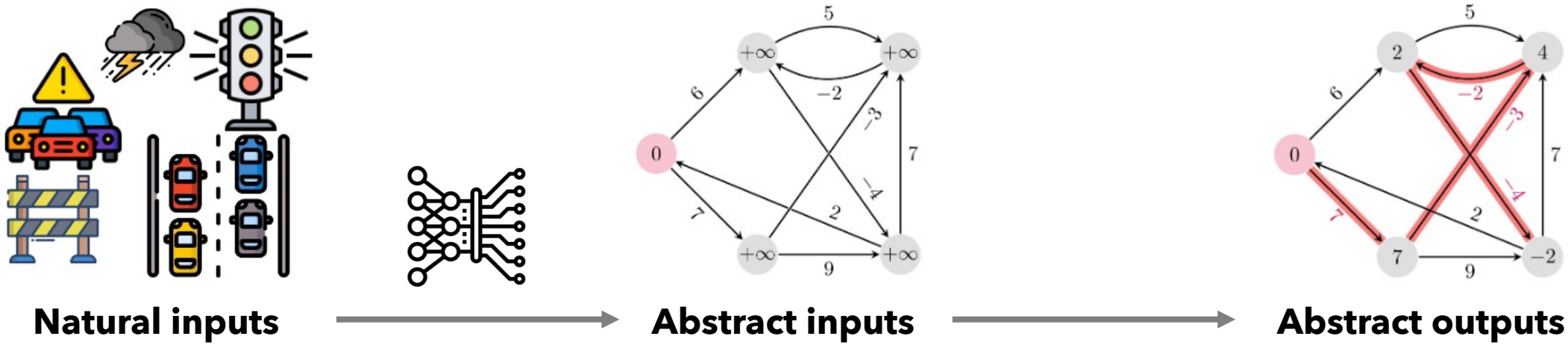



**Abstract outputs**

Slide by Veličković

# Attacking the core problem

- Alternatively, **replace** human feature extractor with NN
  - Still apply same combinatorial algorithm
- Issue: algorithms typically perform **discrete optimization**
  - Doesn't play nicely with **gradient-based** optimization of NNs




Slide by Veličković

# Algorithmic bottleneck

Second (more fundamental) issue: **data efficiency**

- Real-world data is often incredibly rich
- We still have to compress it down to scalar values

The algorithmic solver commits to using this scalar

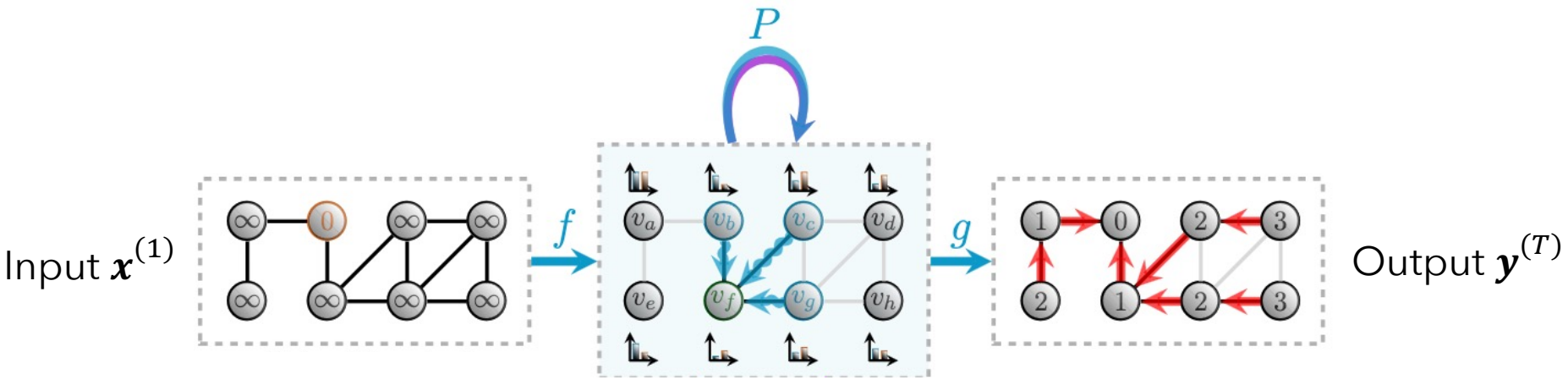
*Assumes it is perfect!*

If there's insufficient training data to estimate the scalars:

- Alg will give a **perfect solution**
- …but in a **suboptimal environment**

Slide by Veličković

# Neural algorithmic pipeline

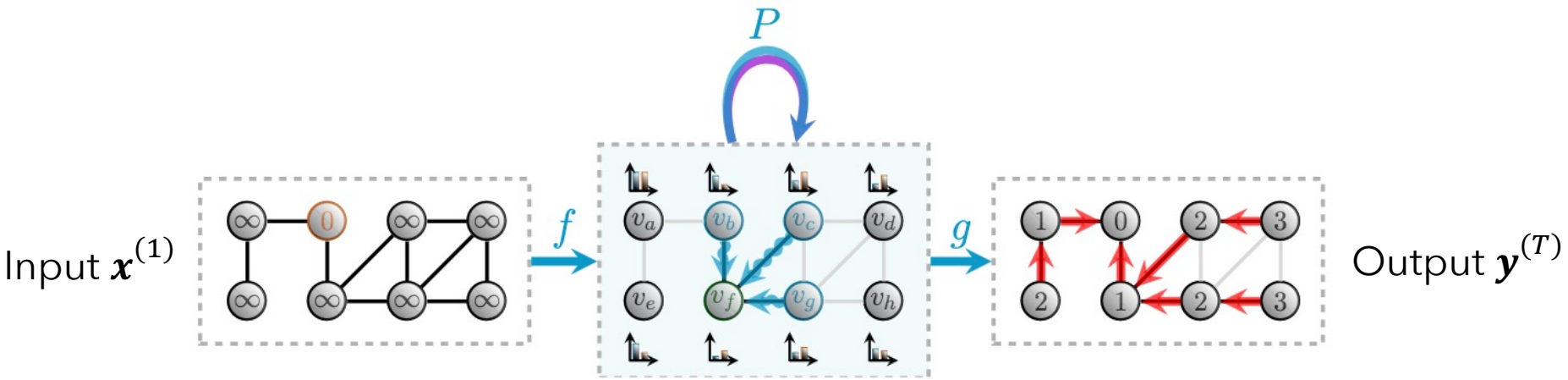



**Encoder network $f$**

- E.g., makes sure input is in correct dimension for next step

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20; Figure by Cappart et al.

# Neural algorithmic pipeline

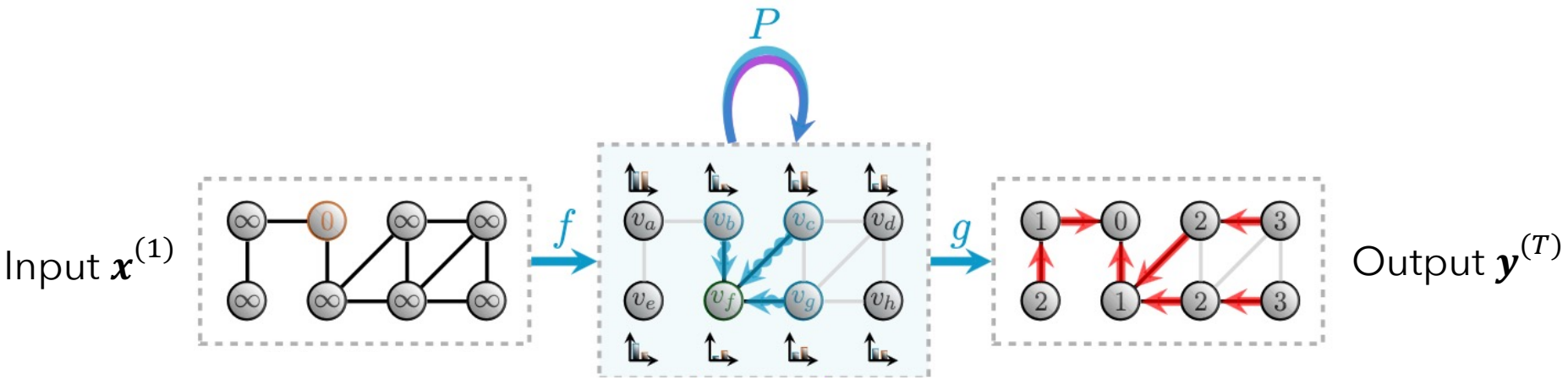



**Processor network $P$**

- Graph neural network
- Run multiple times (termination determined by a NN)

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20; Figure by Cappart et al.

# Neural algorithmic pipeline



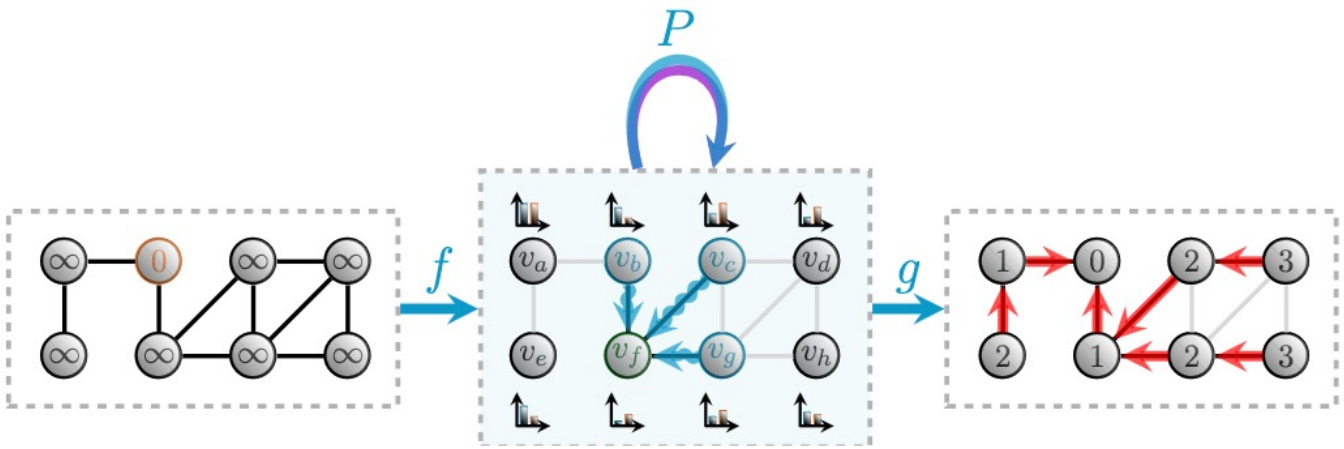

**Decoder network $g$**

- Transform's GNNs output into algorithmic output

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20; Figure by Cappart et al.

# Neural algorithmic pipeline

Abstract inputs $\bar{x}$




Abstract outputs

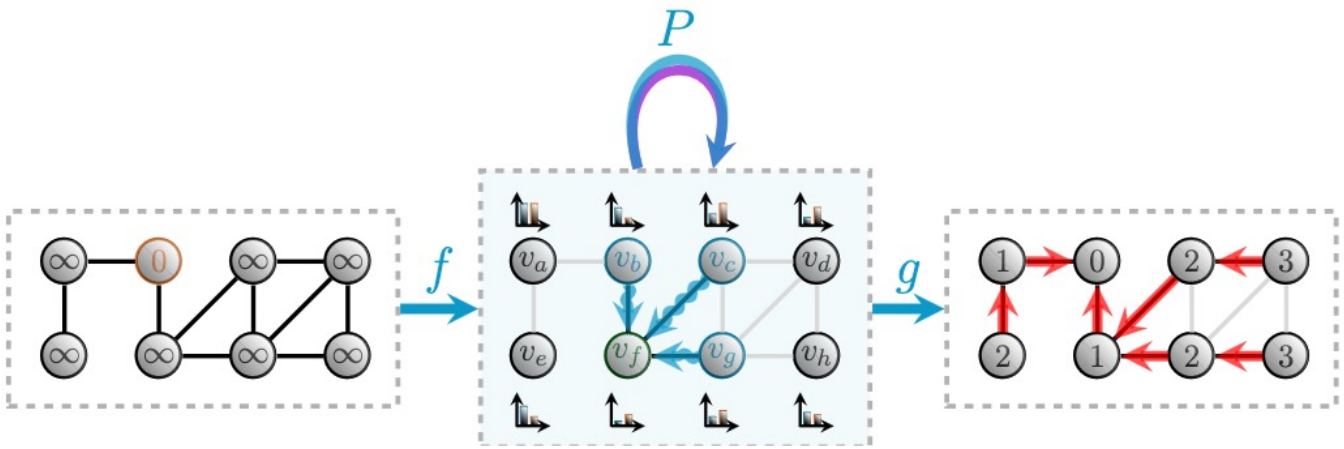
$\bar{y} \approx g\left(P\left(f(x)\right)\right)$

**1.** On abstract inputs, learn encode-process-decode functions

Figure by Cappart et al.

# Neural algorithmic pipeline

Abstract inputs $\bar{x}$



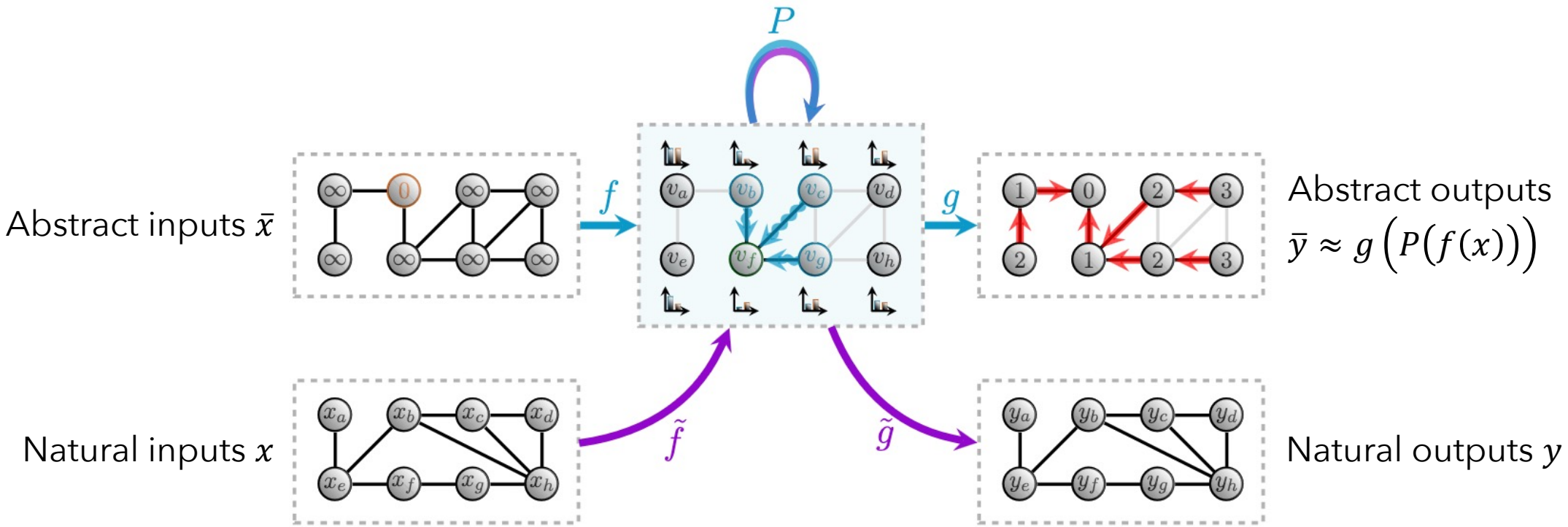

Abstract outputs
$\bar{y} \approx g\left(P\left(f(x)\right)\right)$

After training on abstract inputs, processor $P$:

1. Is aligned with computations of target algorithm
2. Admits useful gradients
3. Operates over high-dim latent space (better use of data)

Figure by Cappart et al.

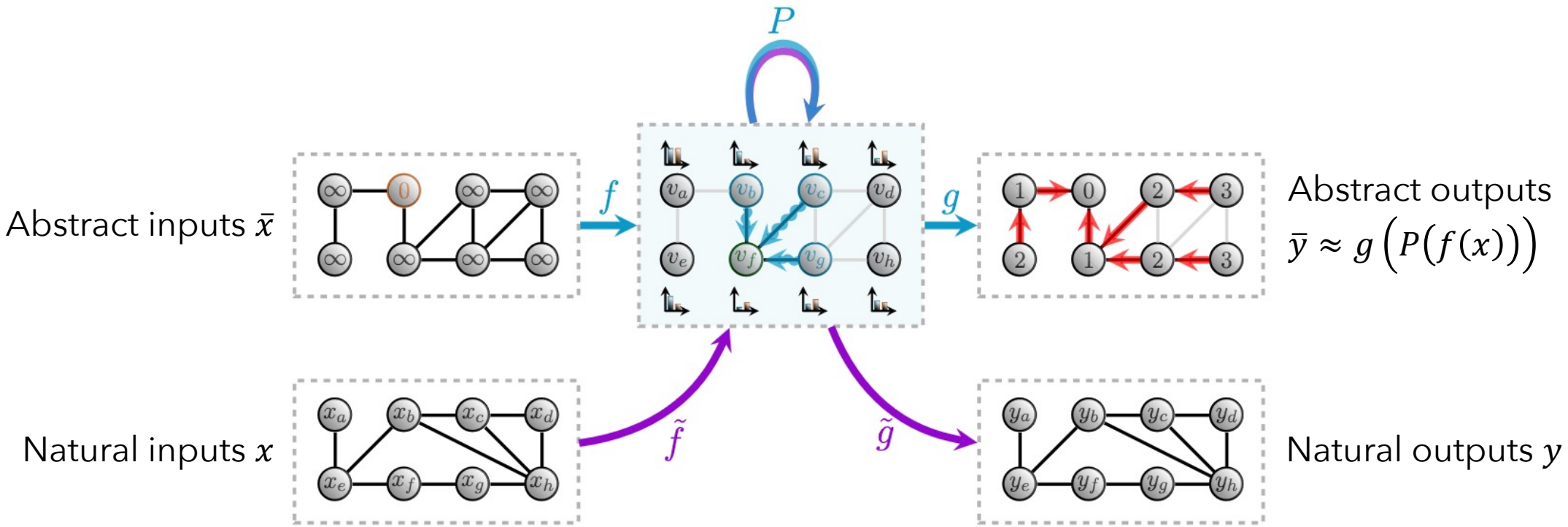
# Neural algorithmic pipeline




**2.** Set up encode-decode functions for natural inputs/outputs

Figure by Cappart et al.

# Neural algorithmic pipeline




**3.** Learn parameters using loss that compares $\tilde{g}\left(P\left(\tilde{f}(x)\right)\right)$ to $y$

Figure by Cappart et al.

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
   i. Example algorithms
   ii. Experiments
   iii. Additional motivation
   **iv. Additional research**
3. Integer programming with GNNs

# Additional research

Lots of research in the past few years! E.g.:

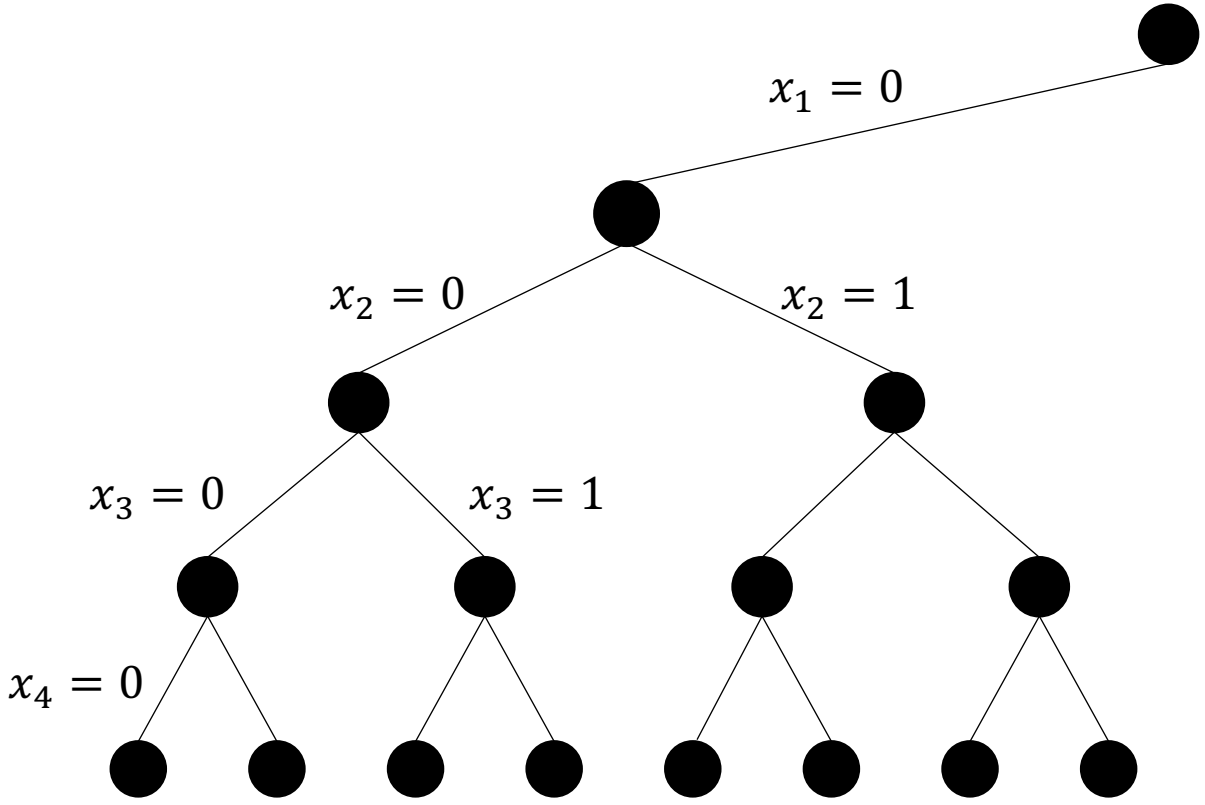
- How to achieve **algorithmic alignment** & **theory guarantees**
  - Xu et al., ICLR'20; Dudzik, Veličković, NeurIPS'22
- **CLRS** benchmark
  - Sorting, searching, dynamic programming, graph algorithms, etc.
  - Veličković et al. ICML'22; Ibarz et al. LoG'22; Bevilacqua et al. ICML'23
- **Primal-dual** algorithms
  - Numeroso et al., ICLR'23

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
3. **Integer programming with GNNs**

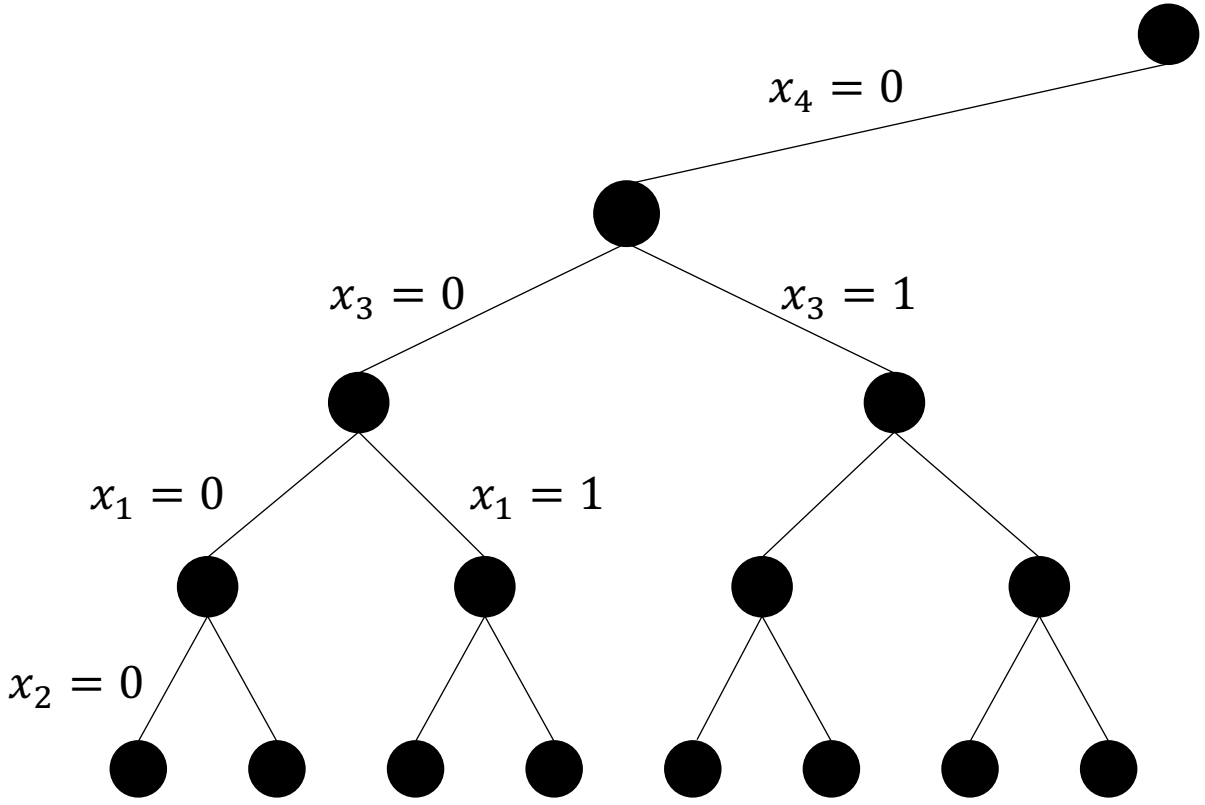
Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

# Variable selection policy (VSP)




Better branching order than $x_1, x_2, x_3, x_4$?

# Variable selection policy (VSP)

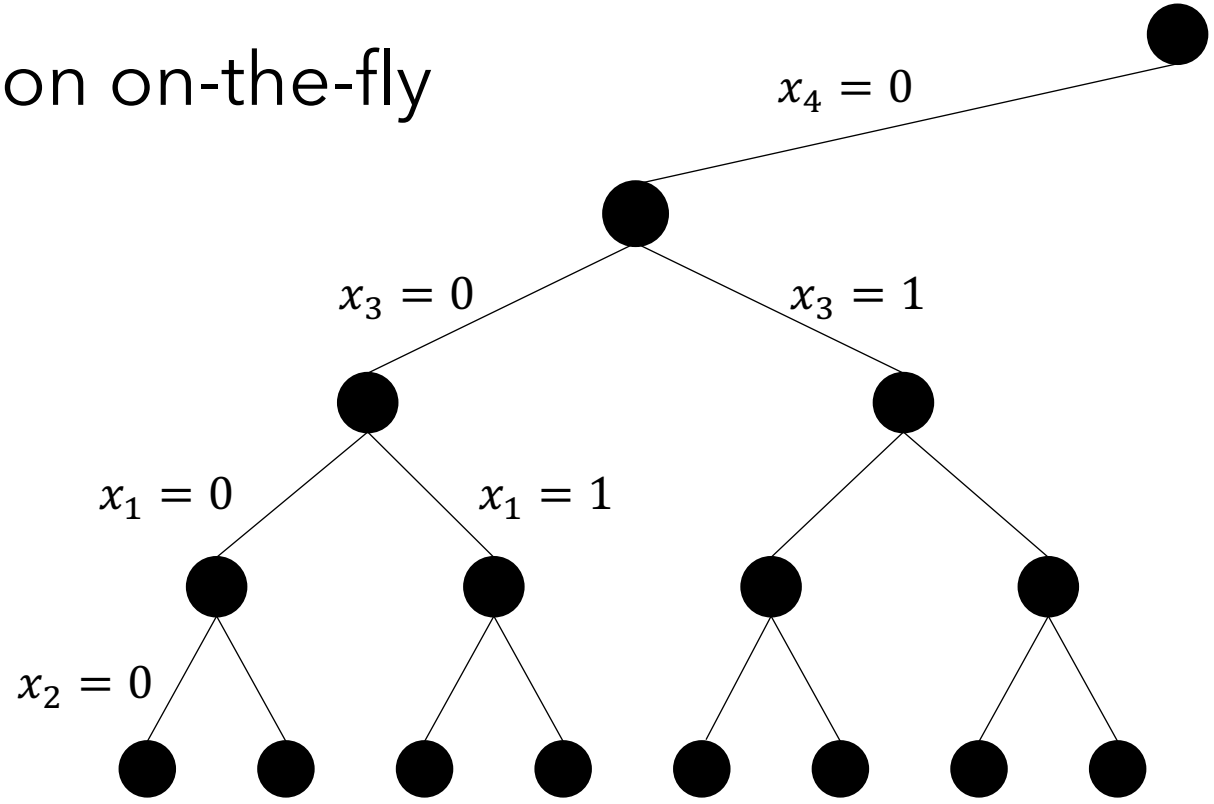



Better branching order than $x_1, x_2, x_3, x_4$? E.g., $x_4, x_3, x_1, x_2$

# Variable selection policy (VSP)

Chooses variables to branch on on-the-fly

*Rather than pre-defined order*



# Variable selection policy (VSP)

At node $j$ with LP objective value $z(j)$:

- Let $z_i^+(j)$ be the LP objective value after setting $x_i = 1$
- Let $z_i^-(j)$ be the LP objective value after setting $x_i = 0$

**VSP example:**

Branch on the variable $x_i$ that maximizes

$$\max\{z(j) - z_i^+(j), 10^{-6}\} \cdot \max\{z(j) - z_i^-(j), 10^{-6}\}$$

If score was $\left(z(j) - z_i^+(j)\right)\left(z(j) - z_i^-(j)\right)$ and $z(j) - z_i^+(j) = 0$:

would lose information stored in $z(j) - z_i^-(j)$

# Strong branching

**Challenge:** Computing $z_i^-(j), z_i^+(j)$ requires solving a lot of LPs

- Computing all LP relaxations referred to as ***strong-branching***
- Very **time intensive**

**Pro:** Strong branching leads to small search trees

**Idea:** Train an ML model to imitate strong-branching

Khalil et al. [AAAI'16], Alvarez et al. [INFORMS JoC'17], Hansknecht et al. [arXiv'18]

**This paper:** using a GNN

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
3. Integer programming with GNNs
   i. **Machine learning formulation**
   ii. Baselines
   iii. Experiments
   iv. Additional research

# Problem formulation

**Goal:** learn a policy $\pi(\, a_t \mid s_t \,)$

Probability of branching on variable $a_t$ when solver is in state $s_t$

**Approach** (imitation learning)**:**

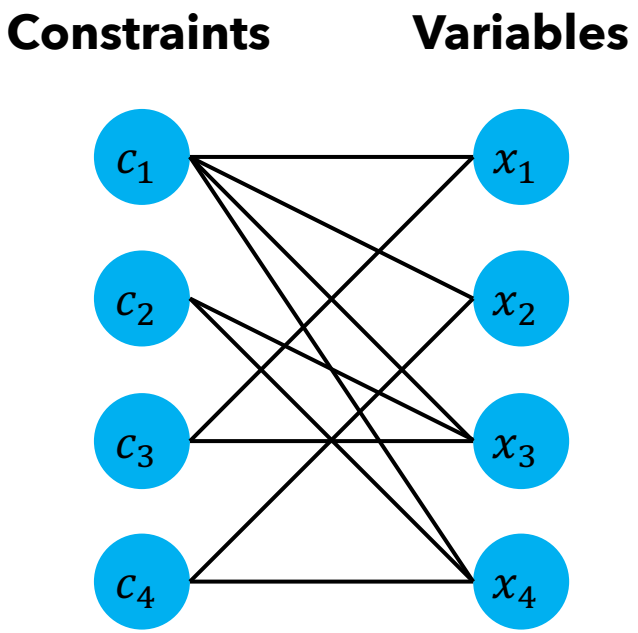
- Run strong branching on training set of instances
- Collect dataset of (state, variable) pairs $S = \{(s_i, a_i^*)\}_{i=1}^N$
- Learn policy $\pi_{\boldsymbol{\theta}}$ with training set $S$

Gasse et al.; NeurIPS'19

# State encoding

State $s_t$ of B&B encoded as a **bipartite graph**
with **node** and **edge features**

$$
\begin{aligned}
\max \quad & 9x_1 + 5x_2 + 6x_3 + 4x_4 \\
\text{s.t.} \quad & 6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10 && (c_1) \\
& x_3 + x_4 \leq 10 && (c_2) \\
& -x_1 + x_3 \leq 0 && (c_3) \\
& -x_2 + x_4 \leq 0 && (c_4) \\
& x_1, x_2, x_3, x_4 \in \{0,1\}
\end{aligned}
$$

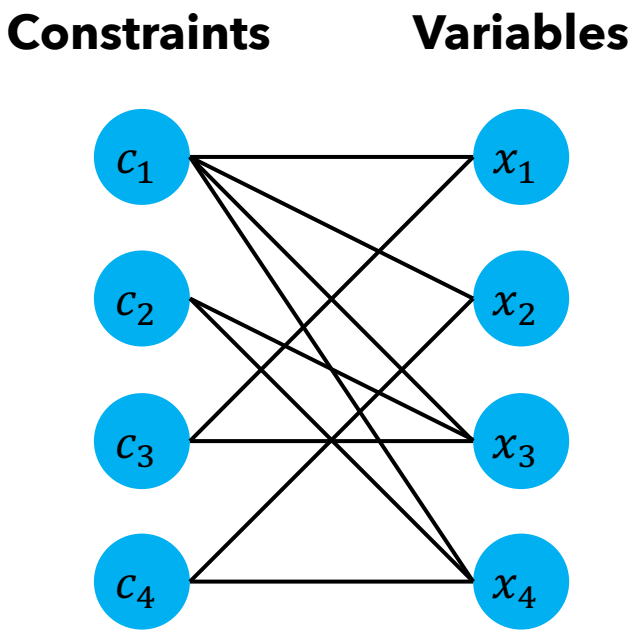



Gasse et al.; NeurIPS'19

# State encoding

State $s_t$ of B&B encoded as a **bipartite graph** with **node** and **edge features**

- **Edge feature:** constraint coefficient
- **Example node features:**
  - Constraints:
    - Cosine similarity with objective
    - Tight in LP solution?
  - Variables:
    - Objective coefficient
    - Solution value equals upper/lower bound?




Gasse et al.; NeurIPS'19

# GNN structure

1. Pass from variables → constraints

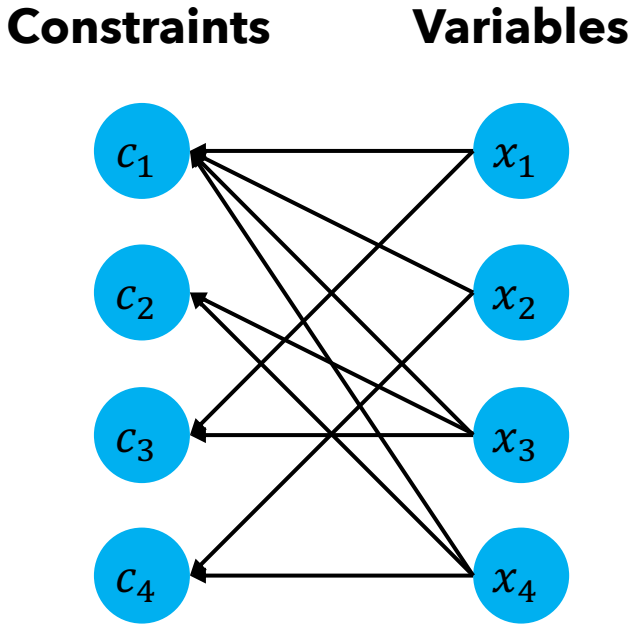
$$\boldsymbol{c}_i \leftarrow f_C\left(\boldsymbol{c}_i, \sum_{j:(i,j)\in E} g_C(\boldsymbol{c}_i, \boldsymbol{v}_j, \boldsymbol{e}_{ij})\right)$$

Constraint features

2-layer MLP with relu activations

Variable features

Edge features




Gasse et al.; NeurIPS'19

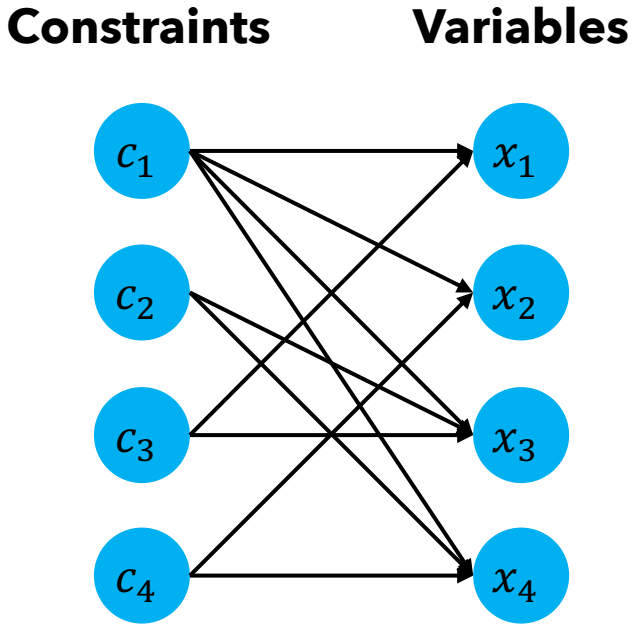
# GNN structure

1. Pass from variables → constraints

$$\boldsymbol{c}_i \leftarrow f_C\left(\boldsymbol{c}_i, \sum_{j:(i,j)\in E} g_C(\boldsymbol{c}_i, \boldsymbol{v}_j, \boldsymbol{e}_{ij})\right)$$

2. Pass from constraints → variables

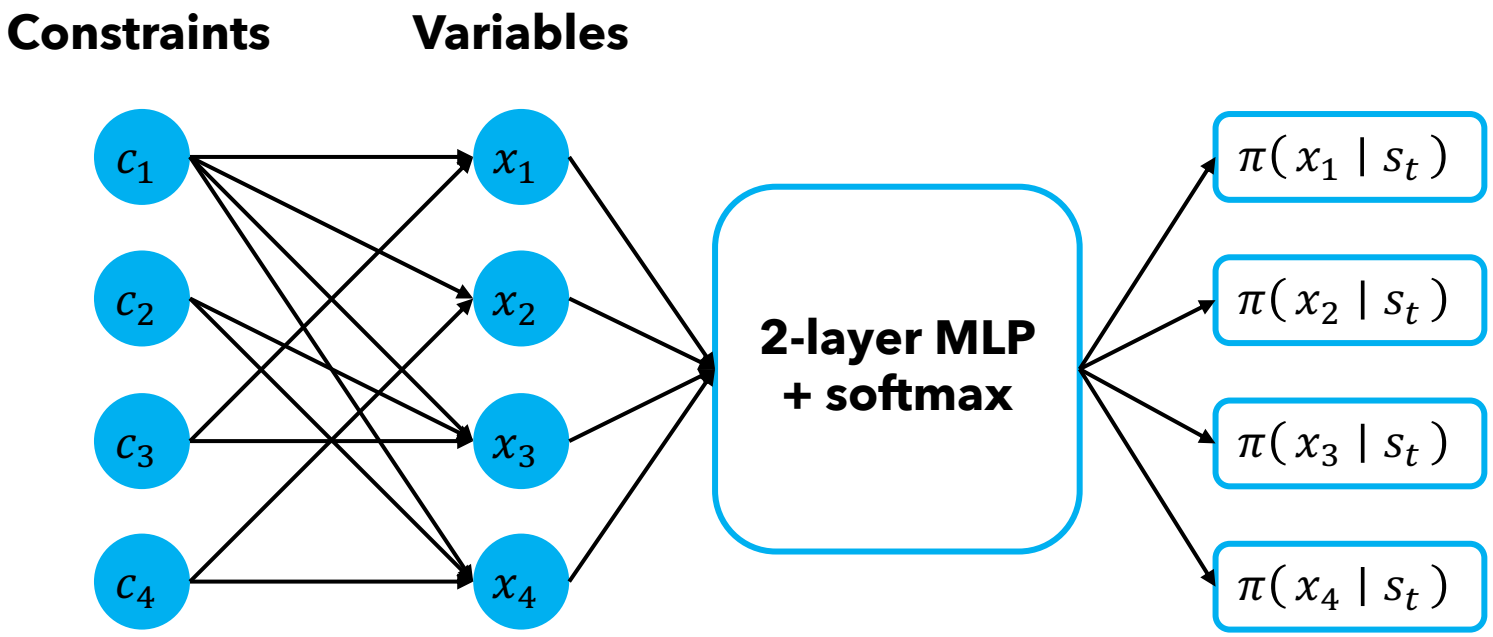
$$\boldsymbol{v}_j \leftarrow f_V\left(\boldsymbol{v}_j, \sum_{i:(i,j)\in E} g_V(\boldsymbol{c}_i, \boldsymbol{v}_j, \boldsymbol{e}_{ij})\right)$$




Gasse et al.; NeurIPS'19

# GNN structure

3. Compute distribution over variables




Gasse et al.; NeurIPS'19

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
3. Integer programming with GNNs
   i. Machine learning formulation
   **ii. Baselines**
   iii. Experiments
   iv. Additional research

# Reliability pseudo-cost branching (RPB)

**Rough idea:**

- Goal: estimate $z(j) - z_i^+(j)$ w/o solving the LP with $x_i = 1$
- Estimate = avg change after setting $x_i = 1$ elsewhere in tree

  *This is the "pseudo-cost"*
- "Reliability": do strong branching if estimate is "unreliable"

  *E.g., early in the tree*

**Default branching rule of SCIP** (leading open-source solver):

$$\max\{\underbrace{\widetilde{\Delta}_i^+(j)}_{\text{Estimate of } z(j) - z_i^+(j)}, 10^{-6}\} \cdot \max\{\underbrace{\widetilde{\Delta}_i^-(j)}_{\text{Estimate of } z(j) - z_i^-(j)}, 10^{-6}\}$$

Achterberg and Berthold, CPAIOR'09

# Learning to rank approaches

- Predict which variable **strong branching** would rank highest
- Using a **linear model** instead of a GNN

- Khalil et al. [AAAI'16]:
  Use learning-to-rank algorithm **SVM$^{\text{rank}}$** [Joachims, KDD'06]
- Hansknecht et al. [arXiv'18]
  Use learning-to-rank alg **lambdaMART** [Burges, Learning'10]

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
3. Integer programming with GNNs
   i. Machine learning formulation
   ii. Baselines
   **iii. Experiments**
   iv. Additional research

# Set covering instances

Always train on “easy” instances

| | 1000 columns, 500 rows | | | 1000 columns, 2000 rows | | |
|---|---|---|---|---|---|---|
| | Easy | | | Hard | | |
| Model | Time | Wins | Nodes | Time | Wins | Nodes |
| FSB | 17.30 ± 6.1% | 0 / 100 | 17 ±13.7% | 3600.00 ± 0.0% | 0 / 0 | n/a ± n/a % |
| RPB | 8.98 ± 4.8% | 0 / 100 | **54** ±20.8% | 1677.02 ± 3.0% | 4 / 65 | 47 299 ± 4.9% |
| TREES | 9.28 ± 4.9% | 0 / 100 | 187 ± 9.4% | 2869.21 ± 3.2% | 0 / 35 | 59 013 ± 9.3% |
| SVMRANK | 8.10 ± 3.8% | 1 / 100 | 165 ± 8.2% | 2389.92 ± 2.3% | 0 / 47 | 42 120 ± 5.4% |
| LMART | 7.19 ± 4.2% | 14 / 100 | 167 ± 9.0% | 2165.96 ± 2.0% | 0 / 54 | 45 319 ± 3.4% |
| GCNN | **6.59** ± 3.1% | **85** / 100 | 134 ± 7.6% | **1489.91** ± 3.3% | **66** / 70 | **29 981** ± 4.9% |

Gasse et al.; NeurIPS’19

# Set covering instances

Runtime in seconds with a timeout of 1 hour

Number instances with fastest runtime / number solved

Size of B&B tree

| Model | Easy Time | Easy Wins | Easy Nodes | Hard Time | Hard Wins | Hard Nodes |
|---|---|---|---|---|---|---|
| FSB | 17.30 ± 6.1% | 0 / 100 | 17 ±13.7% | 3600.00 ± 0.0% | 0 / 0 | n/a ± n/a % |
| RPB | 8.98 ± 4.8% | 0 / 100 | **54** ±20.8% | 1677.02 ± 3.0% | 4 / 65 | 47 299 ± 4.9% |
| TREES | 9.28 ± 4.9% | 0 / 100 | 187 ± 9.4% | 2869.21 ± 3.2% | 0 / 35 | 59 013 ± 9.3% |
| SVMRANK | 8.10 ± 3.8% | 1 / 100 | 165 ± 8.2% | 2389.92 ± 2.3% | 0 / 47 | 42 120 ± 5.4% |
| LMART | 7.19 ± 4.2% | 14 / 100 | 167 ± 9.0% | 2165.96 ± 2.0% | 0 / 54 | 45 319 ± 3.4% |
| GCNN | **6.59** ± 3.1% | **85** / 100 | 134 ± 7.6% | **1489.91** ± 3.3% | **66** / 70 | **29 981** ± 4.9% |

Gasse et al.; NeurIPS'19

# Set covering instances

- GNN is **faster than SCIP** default VSP (RPB)
- Performance generalizes to **larger instances**
- Similar results for auction design & facility location problems

| | Easy | | | Hard | | |
|---|---|---|---|---|---|---|
| Model | Time | Wins | Nodes | Time | Wins | Nodes |
| FSB | 17.30 ± 6.1% | 0 / 100 | 17 ±13.7% | 3600.00 ± 0.0% | 0 / 0 | n/a ± n/a % |
| RPB | 8.98 ± 4.8% | 0 / 100 | **54** ±20.8% | 1677.02 ± 3.0% | 4 / 65 | 47 299 ± 4.9% |
| TREES | 9.28 ± 4.9% | 0 / 100 | 187 ± 9.4% | 2869.21 ± 3.2% | 0 / 35 | 59 013 ± 9.3% |
| SVMRANK | 8.10 ± 3.8% | 1 / 100 | 165 ± 8.2% | 2389.92 ± 2.3% | 0 / 47 | 42 120 ± 5.4% |
| LMART | 7.19 ± 4.2% | 14 / 100 | 167 ± 9.0% | 2165.96 ± 2.0% | 0 / 54 | 45 319 ± 3.4% |
| GCNN | **6.59** ± 3.1% | **85** / 100 | 134 ± 7.6% | **1489.91** ± 3.3% | **66** / 70 | **29 981** ± 4.9% |

Gasse et al.; NeurIPS'19

# Max independent set instances

RPB is catching up to GNN on MIS instances

| Model | Easy Time | Easy Wins | Easy Nodes | Hard Time | Hard Wins | Hard Nodes |
|---|---|---|---|---|---|---|
| FSB | 23.58 ±29.9% | 9 / 100 | 7 ±35.9% | 3600.00 ± 0.0% | 0 / 0 | n/a ± n/a % |
| RPB | 8.77 ±11.8% | 7 / 100 | **20** ±36.1% | 2045.61 ±18.3% | 22 / 42 | **2675** ±24.0% |
| TREES | 10.75 ±22.1% | 1 / 100 | 76 ±44.2% | 3565.12 ± 1.2% | 0 / 3 | 38 296 ± 4.1% |
| SVMRANK | 8.83 ±14.9% | 2 / 100 | 46 ±32.2% | 2902.94 ± 9.6% | 1 / 18 | 6256 ±15.1% |
| LMART | 7.31 ±12.7% | 30 / 100 | 52 ±38.1% | 3044.94 ± 7.0% | 0 / 12 | 8893 ± 3.5% |
| GCNN | **6.43** ±11.6% | **51** / 100 | 43 ±40.2% | **2024.37** ±30.6% | **25** / 29 | 2997 ±26.3% |

Gasse et al.; NeurIPS'19

# Outline (applied techniques)

1. GNNs overview
2. Neural algorithmic alignment
3. Integer programming with GNNs
   i. Machine learning formulation
   ii. Baselines
   iii. Experiments
   **iv. Additional research**

# Additional research

**CPU-friendly** approaches

Gupta et al., NeurIPS'20

**Bipartite representation** inspired many follow-ups

Nair et al., '20; Sonnerat et al., '21; Wu et al., NeurIPS'21; Huang et al. ICML'23; …

**Survey** on *Combinatorial Optimization & Reasoning w/ GNNs*:

Cappart, Chételat, Khalil, Lodi, Morris, Veličković, JMLR'23

# Conclusions and future directions

# Overview

1. **Theoretical guarantees**
    a. Statistical guarantees for algorithm configuration
        i. Broadly applicable theory for deriving generalization guarantees
        ii. Proved using connections between primal and dual classes
    b. Online algorithm configuration
        a. Impossible in the worst cases
        b. Introduced *dispersion* to provide no-regret guarantees

# Overview

1. **Theoretical guarantees**
   a. Statistical guarantees for algorithm configuration
   b. Online algorithm configuration

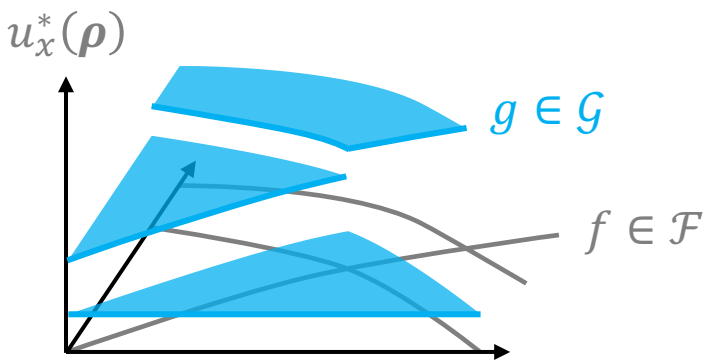
2. **Applied techniques:** Graph neural networks
   a. Neural algorithmic alignment
   b. GNNs for variable selection in branch-and-bound

# Future work: Tighter statistical bounds

WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$

given training set of size $\tilde{O}\left(\frac{1}{\epsilon^2}\left(\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VCdim}(\mathcal{F}^*)\log k\right)\right)$

Number of boundary functions




$k$ is often exponential

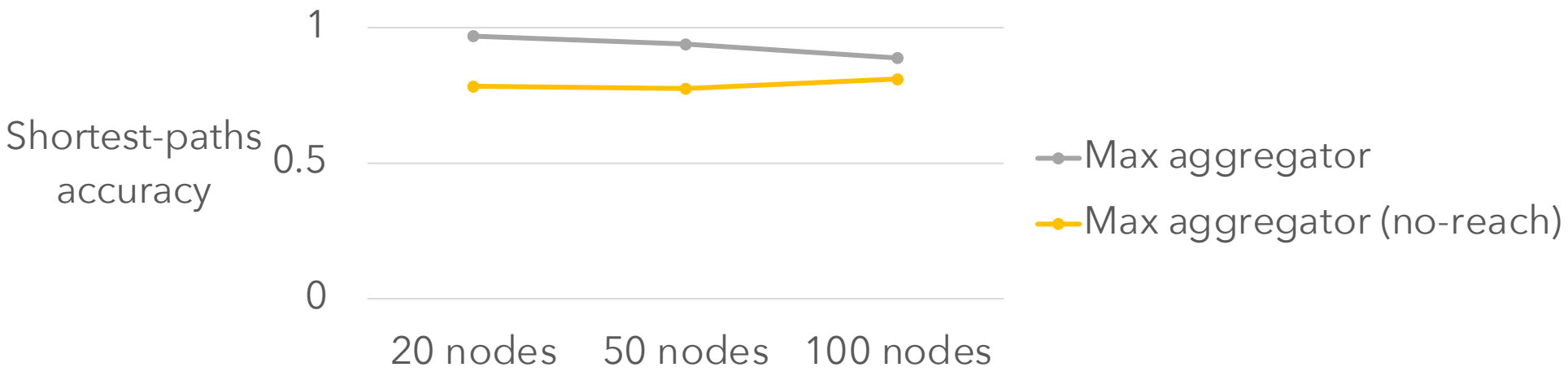
*Can lead to large bounds*

I expect this can sometimes be avoided!

*Would require more information about duals*

# Future work: Knowledge transfer

- Training a GNN to solve multiple related problems… can sometimes lead to better **single-task** performance
- E.g., training reachability and shortest-paths (grey line) v.s. just training shortest-paths (yellow line)




Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

# Future work: Knowledge transfer

- Training a GNN to solve multiple related problems… can sometimes lead to better **single-task** performance
- Can we understand **theoretically** why this happens?
  - For which sets of algorithms can we expect **knowledge transfer**?

# Future work: Size generalization

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

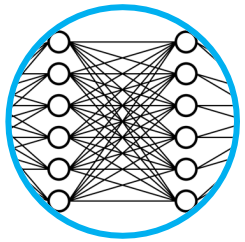
Applied research: Dai et al., NeurIPS'17; Veličković, et al., ICLR'20; …

**Goal:** eventually, solve problems **no one's ever been able to solve**

However, size generalization is not immediate! It depends on:

- The **machine-learned algorithm**

  *Is the algorithm scale sensitive?*



**Example** [Xu et al., ICLR'21]:

- Algorithms represents by GNNs **do generalize**
- Algs represented by MLPs **don't generalize** across size

# Future work: Size generalization

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

Applied research: Dai et al., NeurIPS'17; Veličković, et al., ICLR'20; …

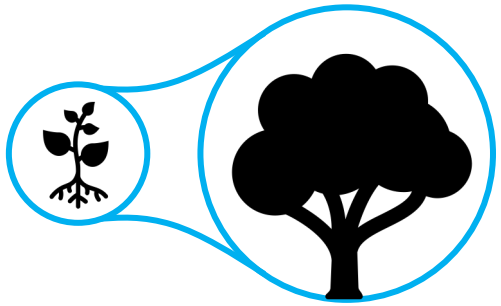
**Goal:** eventually, solve problems **no one's ever been able to solve**

However, size generalization is not immediate! It depends on:

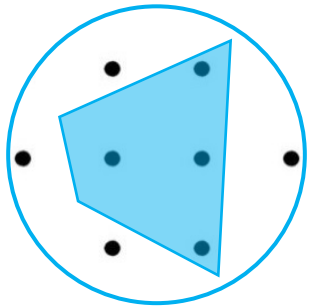
- The **machine-learned algorithm**

  *Is the algorithm scale sensitive?*
- The **problem instances**

  *As size scales, what features must be preserved?*



# Future work: Size generalization



Can you:

1. **Shrink** a set of big integer programs
   graphs
   …
2. **Learn** a good algorithm on the **small** instances
3. **Apply** what you learned to the **big** instances?

# Future work: ML as a toolkit for theory

Which algorithm classes to optimize over?

**Classical algorithm design & analysis**

**Data-driven algorithm design**

**Q:** *Why* are some machine-learned algs so dominant?

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

"New and interesting" greedy strategies for MAXCUT and MVC

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

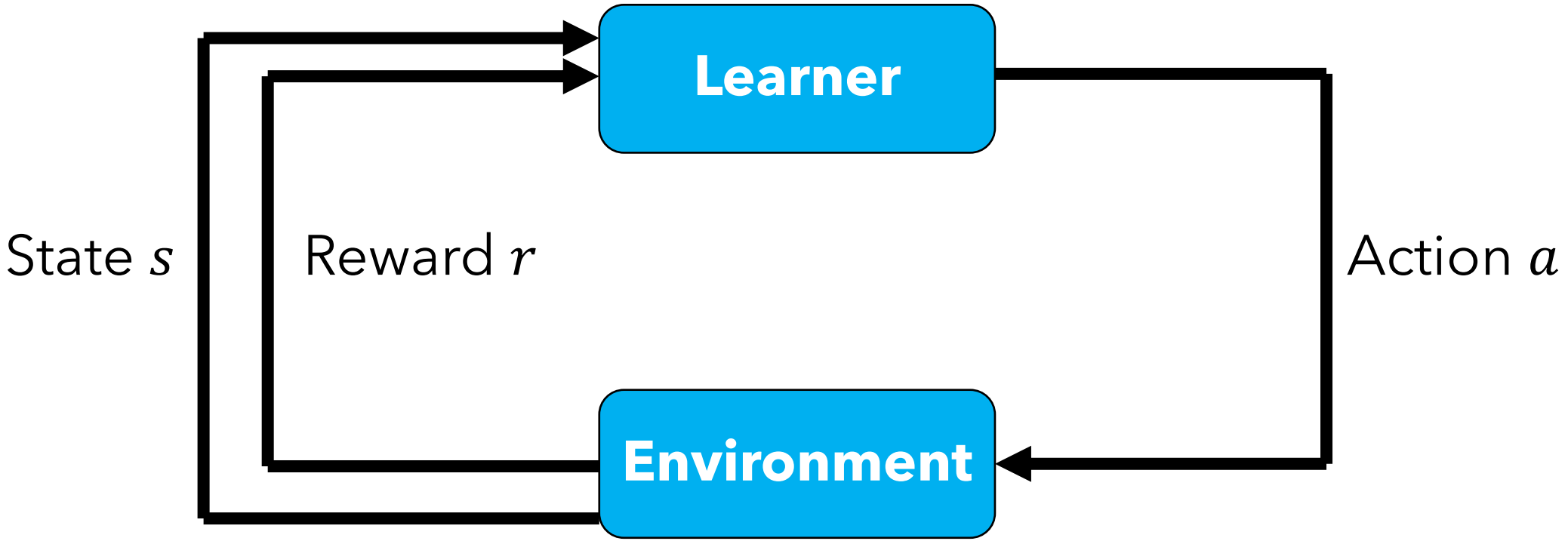
thus could be a "good **assistive tool** for discovering new algorithms."

# Additional slides

# Outline (additional applied techniques)

1. **Reinforcement learning overview**
2. Learning greedy heuristics with RL

# Learner interaction with environment




Slide by Kolter

# Markov decision processes

$S$: set of states (assumed for now to be discrete)

$A$: set of actions

Transition probability distribution $P(s' \mid s, a)$

*Probability of entering state $s'$ from state $s$ after taking action $a$*

Reward function $R: S \to \mathbb{R}$

**Goal:** Policy $\pi: S \to A$ that maximizes total (discounted) reward

Slide by Kolter

# Policies and value functions

Policy is a mapping from states to actions $\pi: S \rightarrow A$

**Value function for a policy:**

Expected sum of discounted rewards

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid s_0 = s, a_t = \pi(s_t), s_{t+1} | s_t, a_t \sim P\right]$$

Discount factor

Slide by Kolter

# Optimal policy and value function

**Optimal policy** $\pi^\star$ achieves the highest value for every state

$$V^{\pi^\star}(s) = \max_{\pi} V^{\pi}(s)$$

Value function is written $V^\star = V^{\pi^\star}$

Several different ways to find $\pi^\star$

- Value iteration
- Policy iteration

Slide by Kolter

# Challenge of RL

**MDP $(S, A, P, R)$:**

- $S$: set of states (assumed for now to be discrete)
- $A$: set of actions
- Transition probability distribution $P(s_{t+1} \mid s_t, a_t)$
- Reward function $R: S \to \mathbb{R}$

**RL twist:** We don't know $P$ or $R$, or too big to enumerate

Slide by Kolter

# Q-learning

**Q functions:**

Like value functions but defined over state-action pairs

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s' \in S} P(\, s' \mid s, a\,) Q^{\pi}\big(s', \pi(s')\big)$$

I.e., Q function is the value of:

1. Starting in state $s$
2. Taking action $a$
3. Then acting according to $\pi$

Slide by Kolter

# Q-learning

$$Q^\star(s,a) = R(s) + \gamma \sum_{s' \in S} P(\, s' \mid s, a\,) \max_{a'} Q^\star(s', a')$$
$$= R(s) + \gamma \sum_{s' \in S} P(\, s' \mid s, a\,) V^\star(s')$$

$Q^\star$ is the value of:

1. Starting in state $s$
2. Taking action $a$
3. Then acting optimally

Slide by Kolter

# Q-learning

**(High-level) Q-learning algorithm**

**initialize** $\hat{Q}(s,a) \leftarrow 0, \forall s, a$

**repeat**

Observe current state $s$ and reward $r$

Take action $a = \operatorname{argmax} \hat{Q}(s,\cdot)$ and observe next state $s'$

Improve estimate $\hat{Q}$ based on $s, r, a, s'$

Can use *function approximation* to represent $\hat{Q}$ compactly

$$\hat{Q}(s,a) = f_\theta(s,a)$$

# Outline (additional applied techniques)

1. Reinforcement learning overview
2. **Learning greedy heuristics with RL**

Dai, Khalil, Zhang, Dilkina, Song; NeurIPS'17

# RL for combinatorial optimization

Tons of research in this area

**Travelling salesman**

Bello et al., ICLR'17; Dai et al., NeurIPS'17; Nazari et al., NeurIPS'18; …

**Bin packing**

Hu et al., '17; Laterre et al., '18; Cai et al., DRL4KDD'19; Li et al., '20; …

**Maximum cut**

Dai et al., NeurIPS'17; Cappart et al., AAAI'19; Barrett et al., AAAI'20; …

**Minimum vertex cover**

Dai et al., NeurIPS'17; Song et al., UAI'19; …

**This section:** Example of a pioneering work in this space

# Overview

**Goal:** use RL to learn new *greedy strategies* for graph problems

Feasible solution constructed by successively adding nodes to solution

**Input:** Graph $G = (V, E)$, weights $w(u, v)$ for $(u, v) \in E$

**RL state representation:** Graph embedding

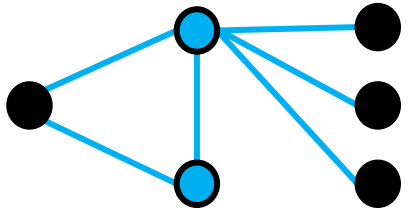
Dai, Khalil, et al.; NeurIPS'17

# Outline (additional applied techniques)

1. Reinforcement learning overview
2. Learning greedy heuristics with RL
   i. **Examples: Min vertex cover and max cut**
   ii. RL formulation
   iii. Experiments

# Minimum vertex cover

Find smallest vertex subset such that each edge is covered

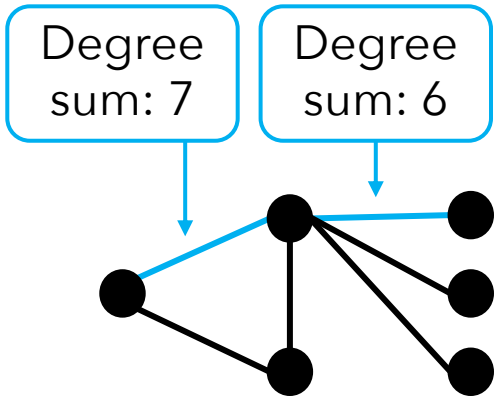


# Minimum vertex cover

Find smallest vertex subset such that each edge is covered

**2-approximation:**

Greedily add vertices of edge with **maximum degree sum**



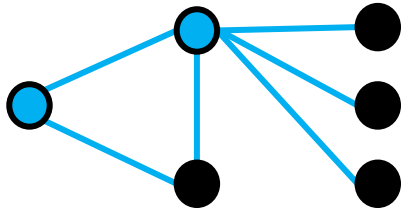
# Minimum vertex cover

Find smallest vertex subset such that each edge is covered

**2-approximation:**

Greedily add vertices of edge with **maximum degree sum**

**Scoring function** that guides greedy algorithm



# Maximum cut

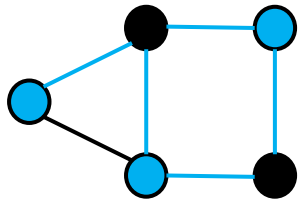
Find partition $(S, V \setminus S)$ of nodes that maximizes

$$\sum_{(u,v)\in C} w(u,v)$$

where $C = \{(u,v) \in E : u \in S, v \notin S\}$

If $w(u,v) = 1$ for all $(u,v) \in E$:

$$\sum_{(u,v)\in C} w(u,v) = 5$$



# Maximum cut

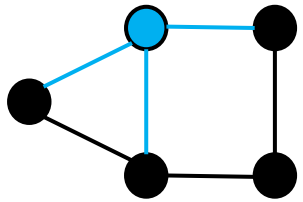
Find partition $(S, V \setminus S)$ of nodes that maximizes

$$\sum_{(u,v) \in C} w(u,v)$$

where $C = \{(u,v) \in E : u \in S, v \notin S\}$

**Greedy:** move node from one side of cut to the other

Move node that results in the largest improvement in cut weight



# Maximum cut

Find partition $(S, V \setminus S)$ of nodes that maximizes

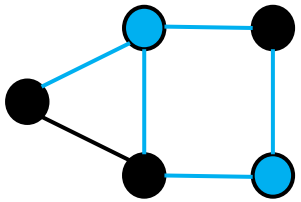
$$\sum_{(u,v) \in C} w(u,v)$$

where $C = \{(u,v) \in E : u \in S, v \notin S\}$

**Greedy:** move node from one side of cut to the other

Move node that results in the largest improvement in cut weight

**Scoring function** that guides greedy algorithm



# Outline (additional applied techniques)

1. Reinforcement learning overview
2. Learning greedy heuristics with RL
   i. Example: Min vertex cover and max cut
   **ii. RL formulation**
   iii. Experiments

# Reinforcement learning formulation

**State:**

- *Goal:* encode partial solution $S = (v_1, v_2, \ldots, v_{|S|}), v_i \in V$

E.g., nodes in independent set, nodes on one side of cut

Dai, Khalil, et al.; NeurIPS'17

# Reinforcement learning formulation

**State:**

- *Goal:* encode partial solution $S = (v_1, v_2, \dots, v_{|S|}), v_i \in V$
- Use GNN to compute graph embedding $\boldsymbol{\mu}$

Initial node features $x_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{else} \end{cases}$

**Action:** Choose vertex $v \in V \setminus S$ to add to solution

**Transition** (deterministic): For chosen $v \in V \setminus S$, set $x_v = 1$

Dai, Khalil, et al.; NeurIPS'17

# Reinforcement learning formulation

**Reward:** $r(S, v)$ is change in objective when transition $S \to (S, v)$

**Policy** (deterministic): $\pi(v|S) = \begin{cases} 1 & \text{if } v = \underset{v' \notin S}{\operatorname{argmax}}\, \hat{Q}(\boldsymbol{\mu}, v') \\ 0 & \text{else} \end{cases}$

Dai, Khalil, et al.; NeurIPS'17

# Outline (additional applied techniques)

1. Reinforcement learning overview
2. Learning greedy heuristics with RL
   i. Example: Min vertex cover and max cut
   ii. RL formulation
   **iii. Experiments**

# Min vertex cover

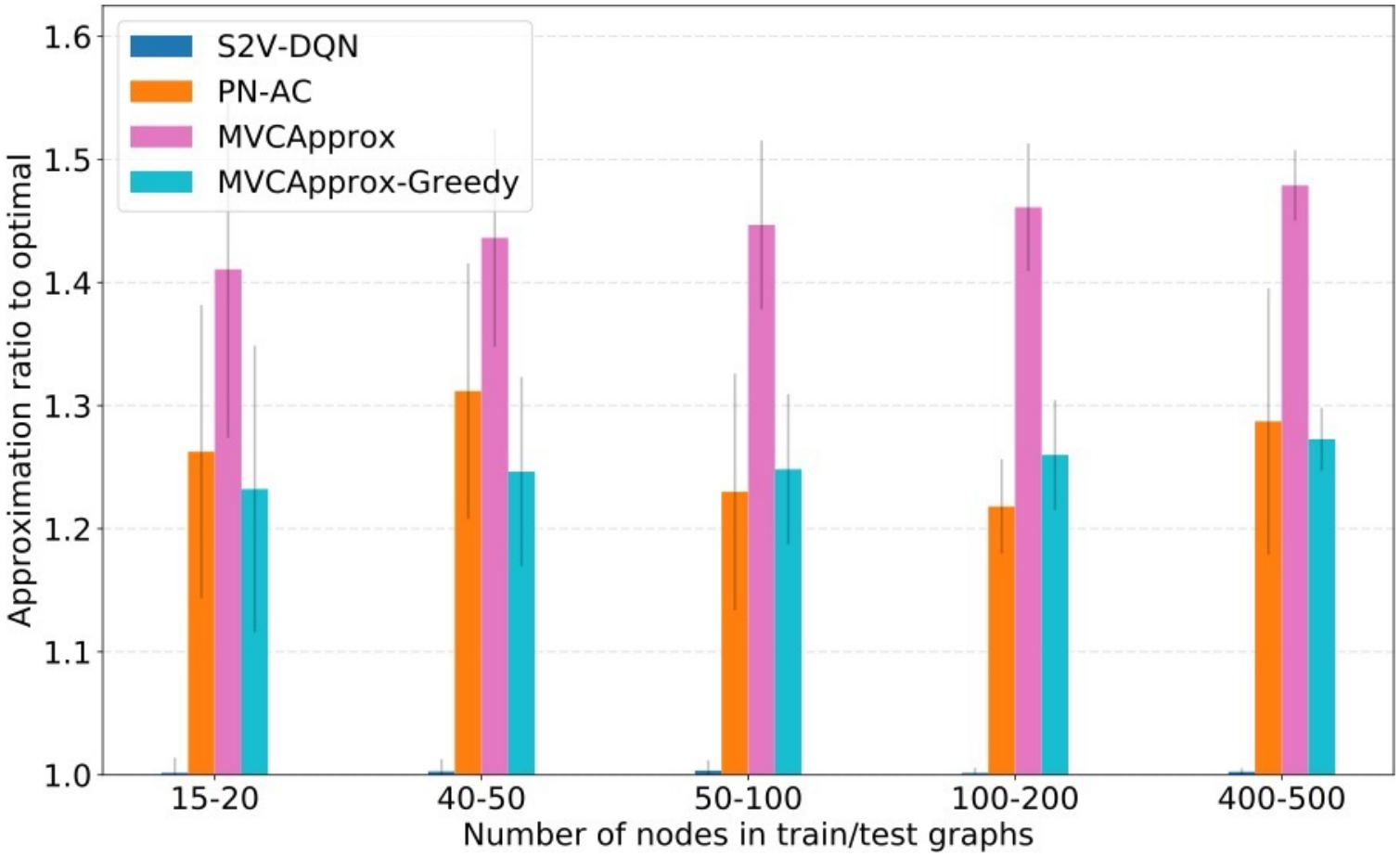
Barabasi-Albert random graphs

Paper's approach

Another DL approach [Bello et al., arXiv'16]

2-approximation algorithm

Greedy algorithm from first few slides




Dai, Khalil, et al.; NeurIPS'17

# Max cut

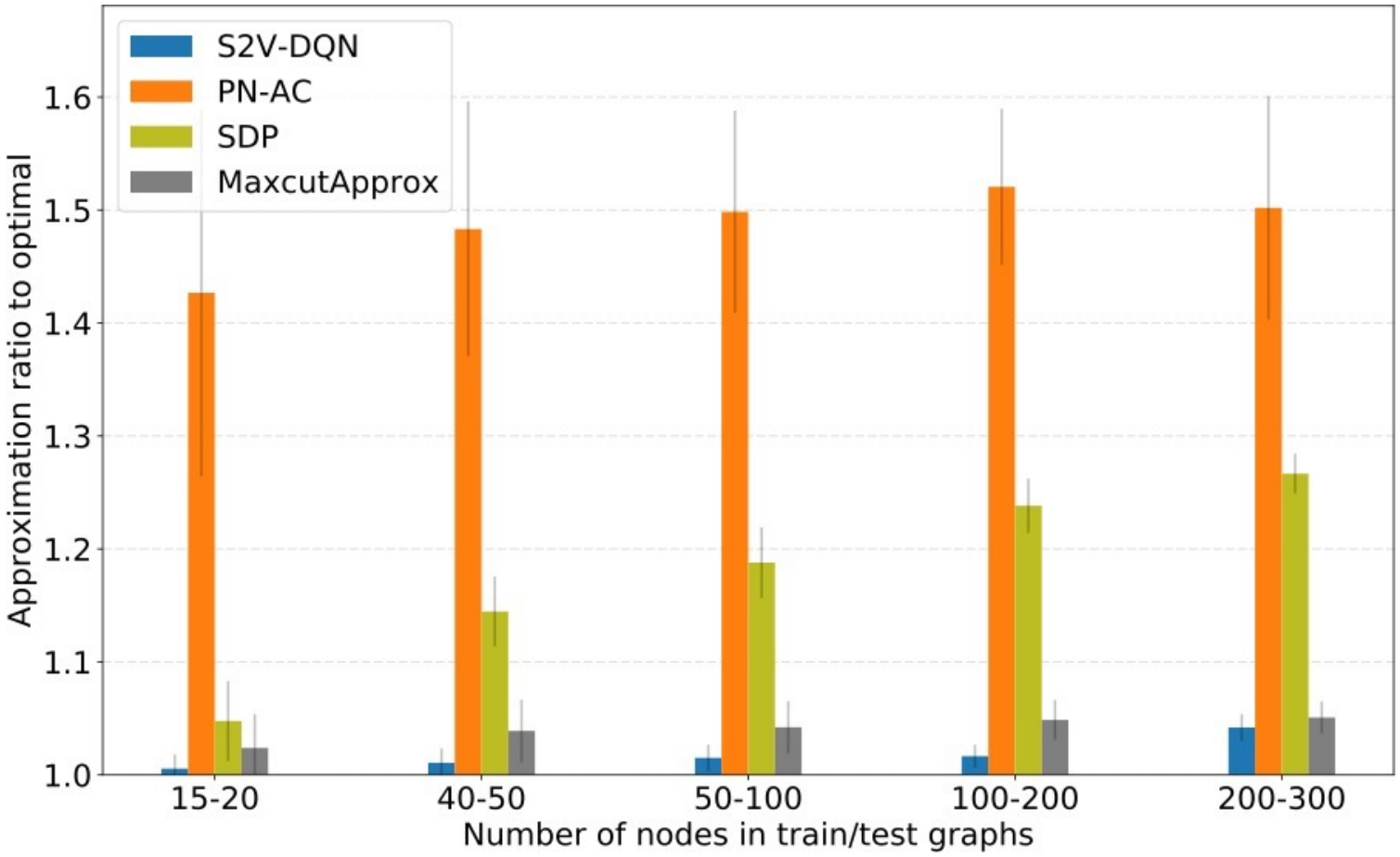
Barabasi-Albert random graphs

Paper's approach

Another DL approach [Bello et al., arXiv'16]

Goemans-Williamson algorithm

Greedy algorithm from first few slides




Dai, Khalil, et al.; NeurIPS'17

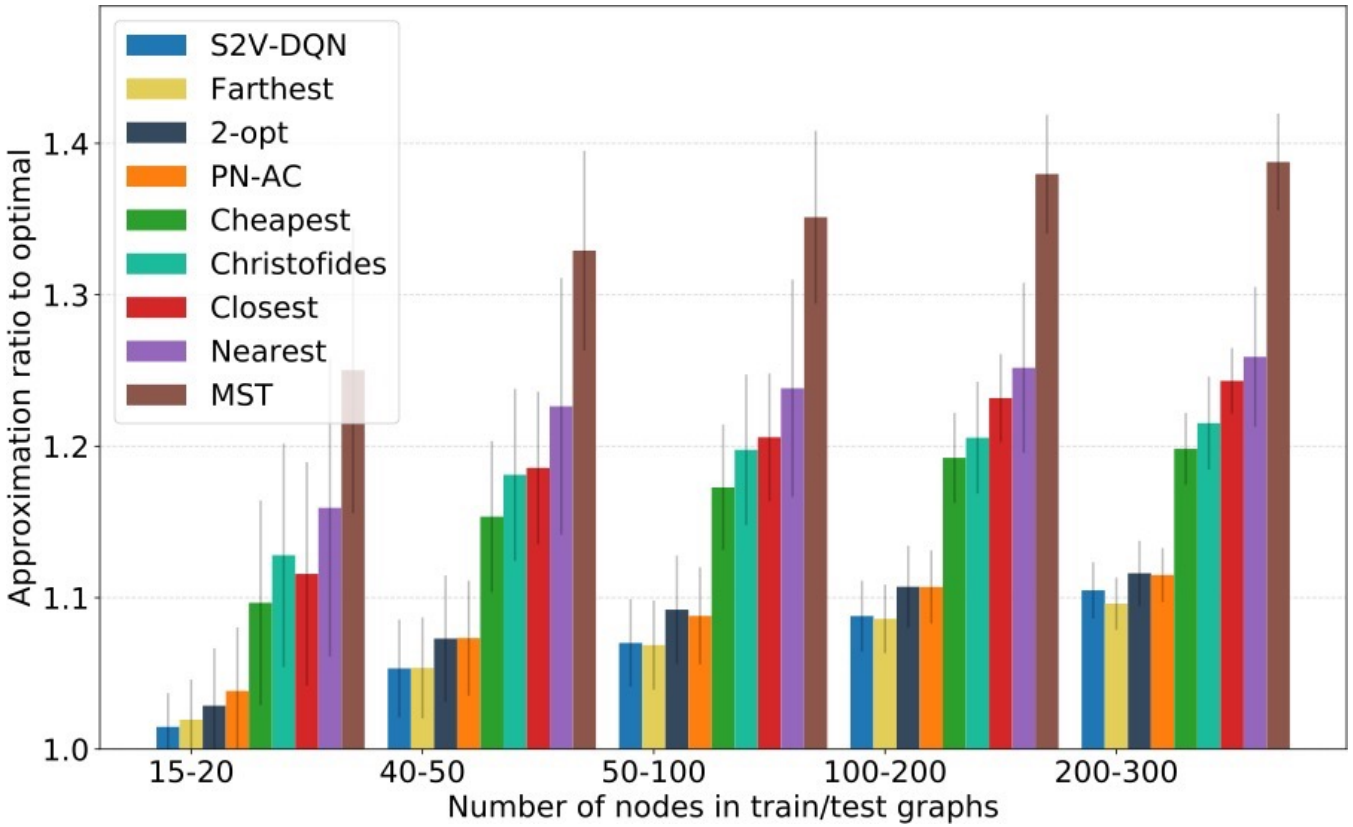
# TSP

Uniform random points on 2-D grid

**Paper's approach**

- Initial subtour: 2 cities that are farthest apart
- Repeat the following:
  - Choose city that's *farthest* from any city in the subtour
  - Insert in position where it causes the smallest distance increase

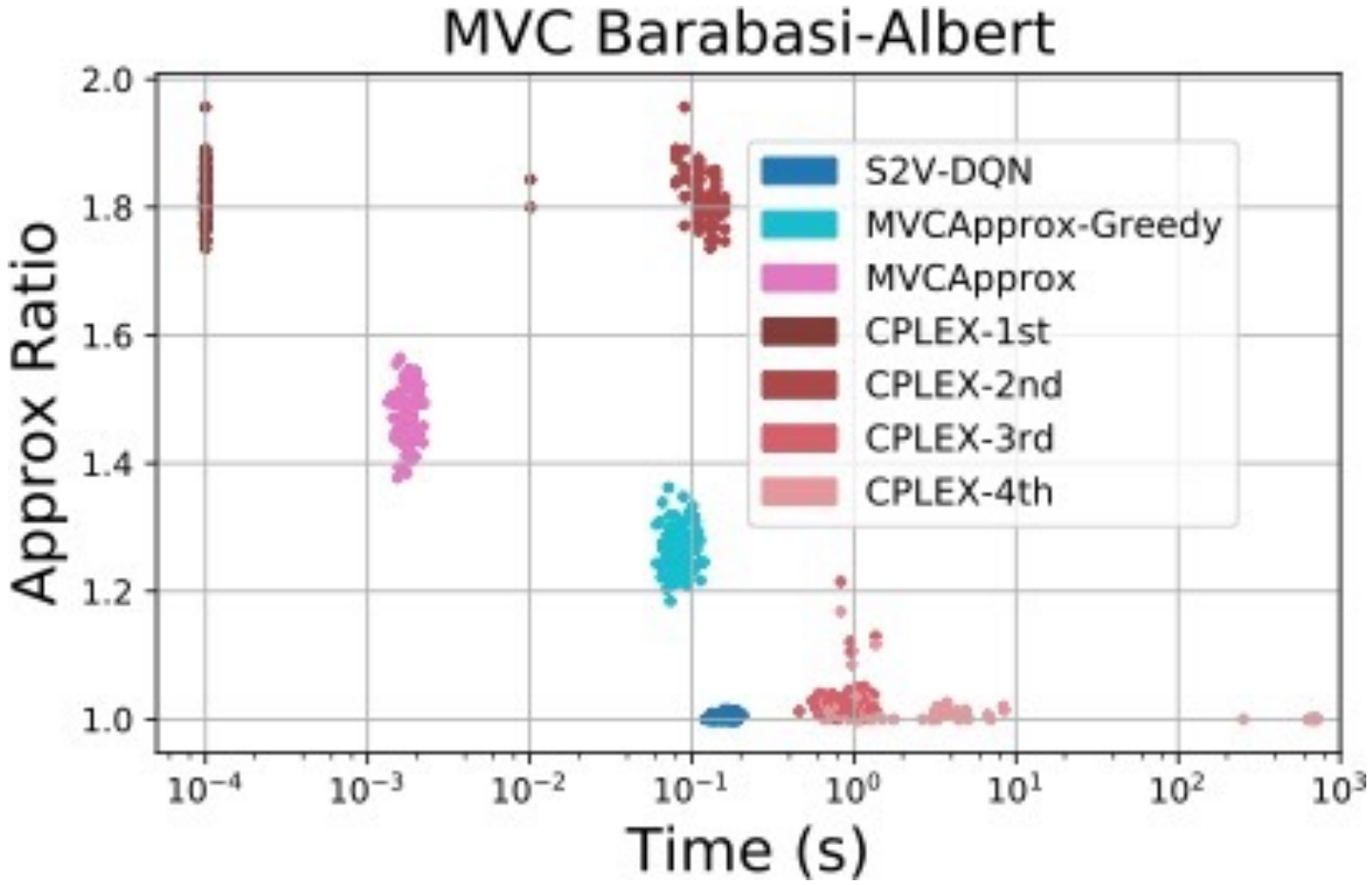
[Rosenkrantz et al., SIAM JoC'77]




Dai, Khalil, et al.; NeurIPS'17

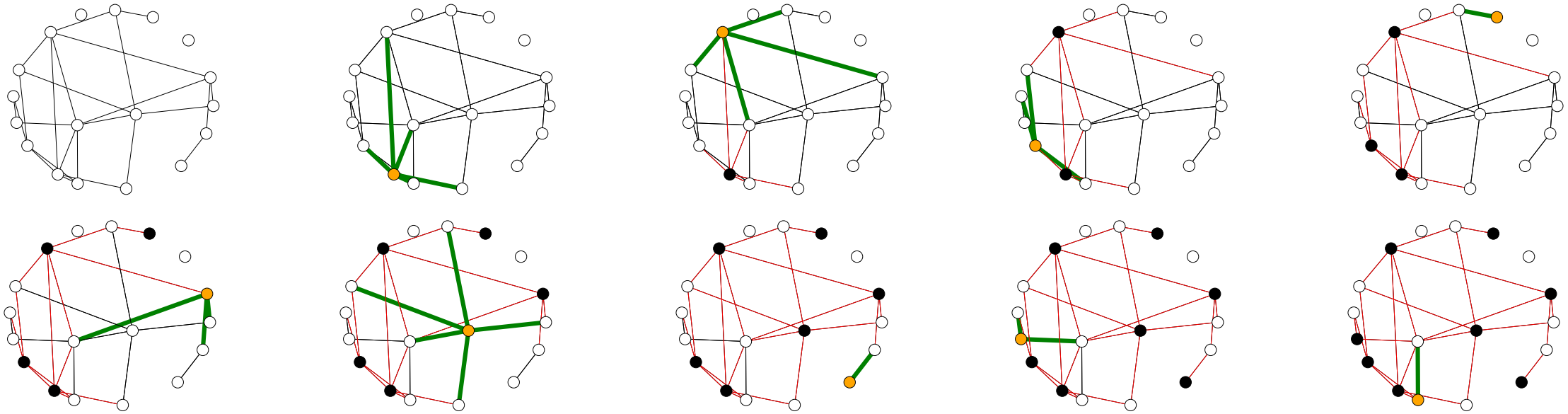
# Runtime comparisons

**CPLEX-1st:** 1st feasible solution found by CPLEX




Dai, Khalil, et al.; NeurIPS'17

# Min vertex cover visualization



Nodes seem to be selected to balance between:

- Degree
- Connectivity of the remaining graph

Dai, Khalil, et al.; NeurIPS'17