Machine learning for discrete optimization: Theoretical guarantees and applied frontiers

Ellen Vitercik

Stanford University

How to integrate **machine learning** into **discrete optimization**?

How to integrate **machine learning** into **discrete optimization**?

- **Algorithm configuration**

  How to tune an algorithm's parameters?

How to integrate **machine learning** into **discrete optimization**?

Algorithm configuration

How to tune an algorithm's parameters?

Algorithm selection

Given a variety of algorithms, which to use?

How to integrate **machine learning** into **discrete optimization**?

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 **discrete optimization**?

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

Algorithm configuration

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

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

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**

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?

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 **discrete optimization**?

Algorithm configuration

How to tune an algorithm's parameters?

Algorithm selection

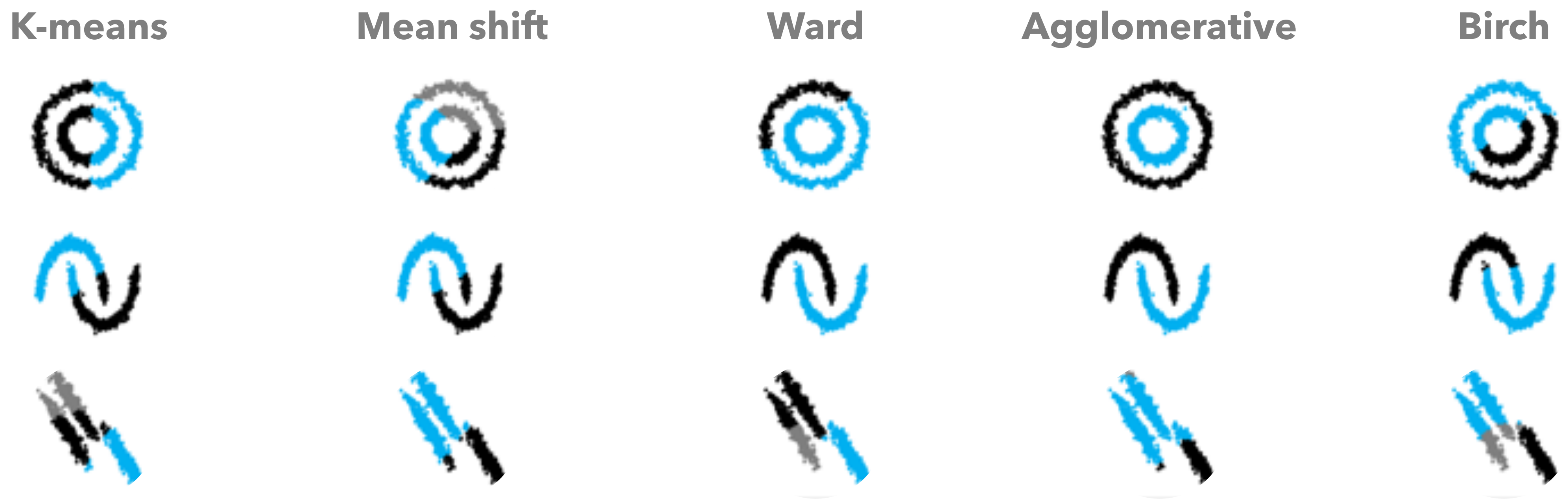
Given a variety of algorithms, which to use?

Algorithm design

Can machine learning guide algorithm discovery?

Example: Clustering

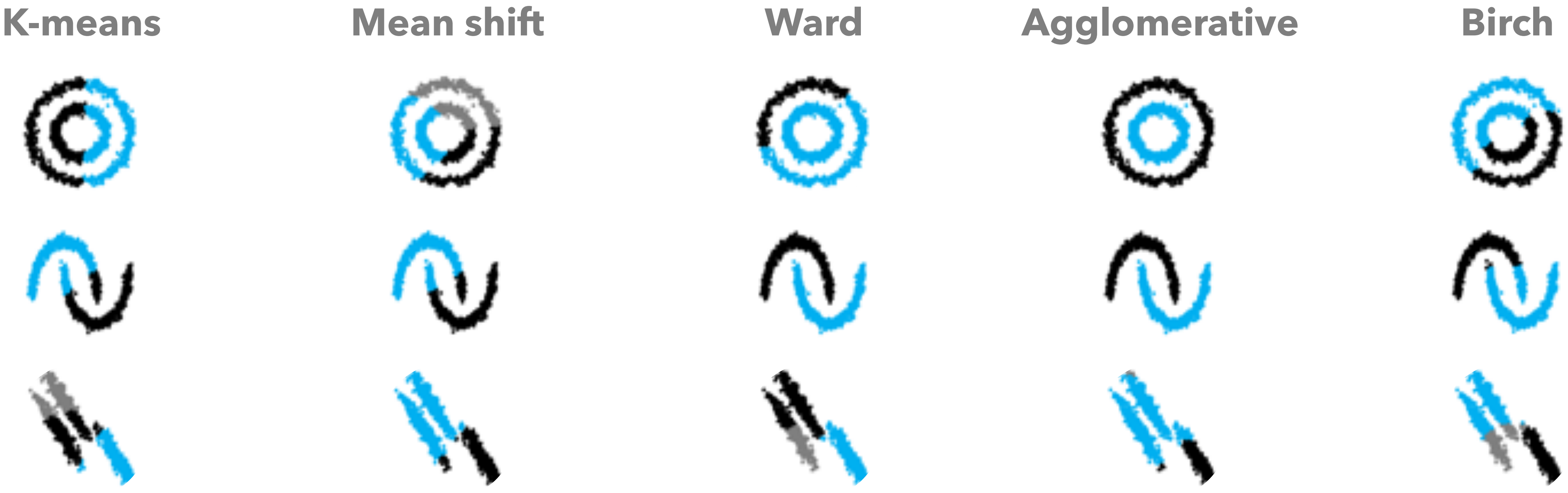
Many different algorithms



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Example: Clustering

Many different algorithms



How to **select** the best algorithm for the application at hand?

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Algorithm selection in theory

Worst-case analysis has been the main framework for decades

Has led to beautiful, practical algorithms

Algorithm selection in theory

Worst-case analysis has been the main framework for decades

Has led to beautiful, practical algorithms

Worst-case analysis's approach to **algorithm selection**:

Select the algorithm that's best in worst-case scenarios

Algorithm selection in theory

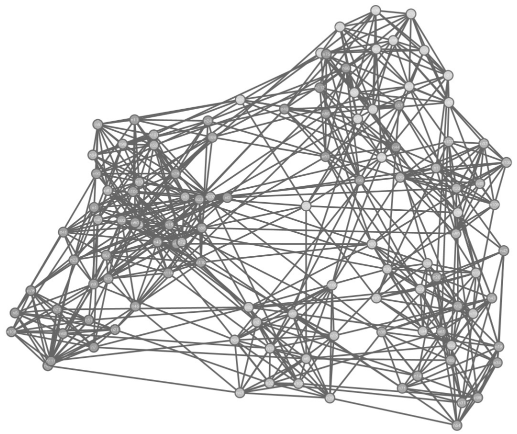
Worst-case analysis has been the main framework for decades

Has led to beautiful, practical algorithms

Worst-case analysis's approach to **algorithm selection**:

Select the algorithm that's best in worst-case scenarios

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



How to integrate **machine learning** into **discrete optimization**?

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 **discrete optimization**?

Long-term goal:

Researchers will be empowered with **data-driven tools** to

How to integrate **machine learning** into **discrete optimization**?

Long-term goal:

Researchers will be empowered with **data-driven tools** to

- Conceive

How to integrate **machine learning** into **discrete optimization**?

Long-term goal:

Researchers will be empowered with **data-driven tools** to

- Conceive
- Prototype

How to integrate **machine learning** into **discrete optimization**?

Long-term goal:

Researchers will be empowered with **data-driven tools** to

- Conceive
- Prototype
- Validate

How to integrate **machine learning** into **discrete optimization**?

Long-term goal:

Researchers will be empowered with **data-driven tools** to

- Conceive
- Prototype
- Validate

algorithmic ideas…

How to integrate **machine learning** into **discrete optimization**?

Long-term goal:

Researchers will be empowered with **data-driven tools** to

- Conceive
- Prototype
- Validate

algorithmic ideas…

and provide theoretical guarantees for their discoveries

How to integrate **machine learning** into **discrete optimization**?

Research area is built on a key observation:

How to integrate **machine learning** into **discrete optimization**?

Research area is built on a key observation:

In practice, we have data about the application domain

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

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

In practice, we have data about the application domain

Scheduling problems an airline solves

In practice, we have data about the application domain

How can we use this data to guide:

- **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?

ML + discrete opt: Potential impact

Example: integer programming

- Used heavily throughout industry and science





ML + discrete opt: Potential impact

Example: integer programming

- Used heavily throughout industry and science
- **Many** different ways to incorporate **learning** into solving

ML + discrete opt: 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



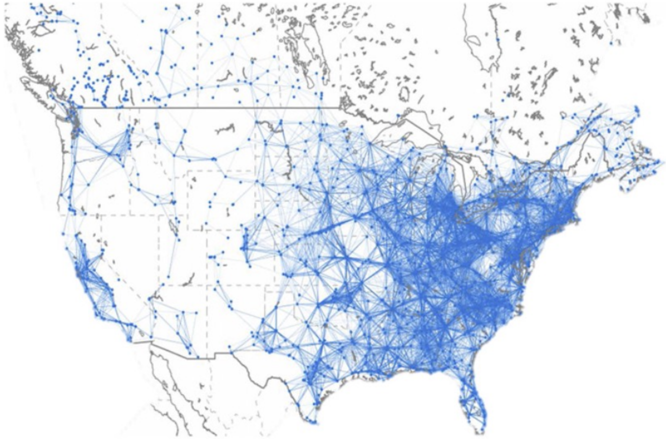


Example: Spectrum auctions

- In ‘16–’17, FCC held a $19.8 billion radio spectrum auction

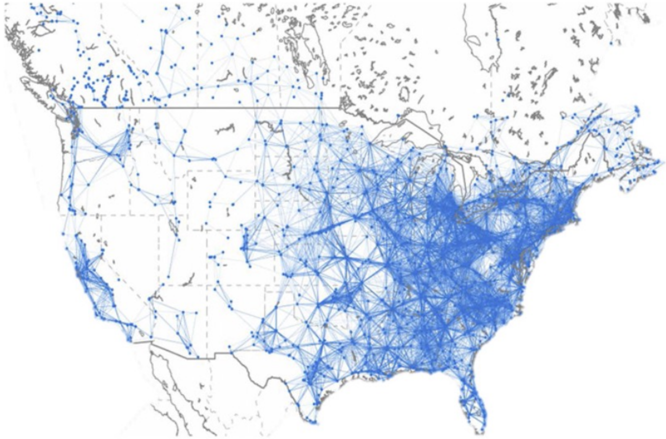
Example: Spectrum auctions

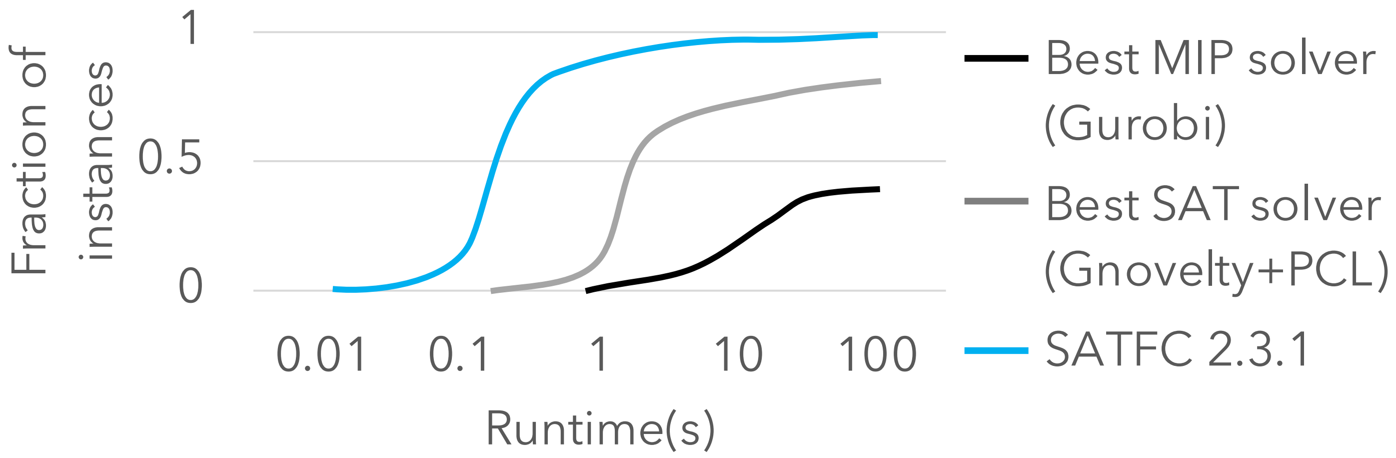
- In ‘16–’17, FCC held a $19.8 billion radio spectrum auction
  - Involves solving huge graph-coloring problems



Example: Spectrum auctions

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

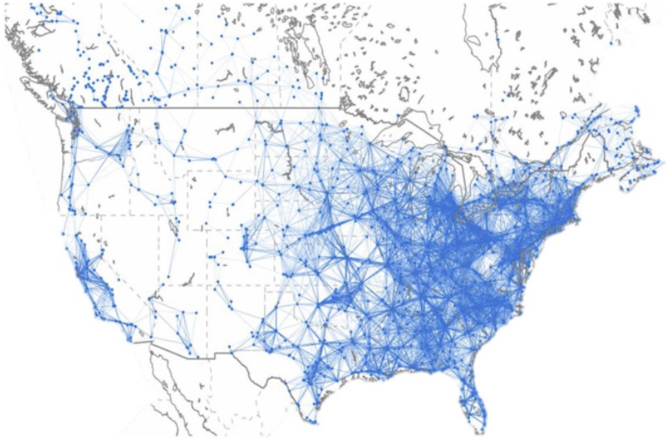


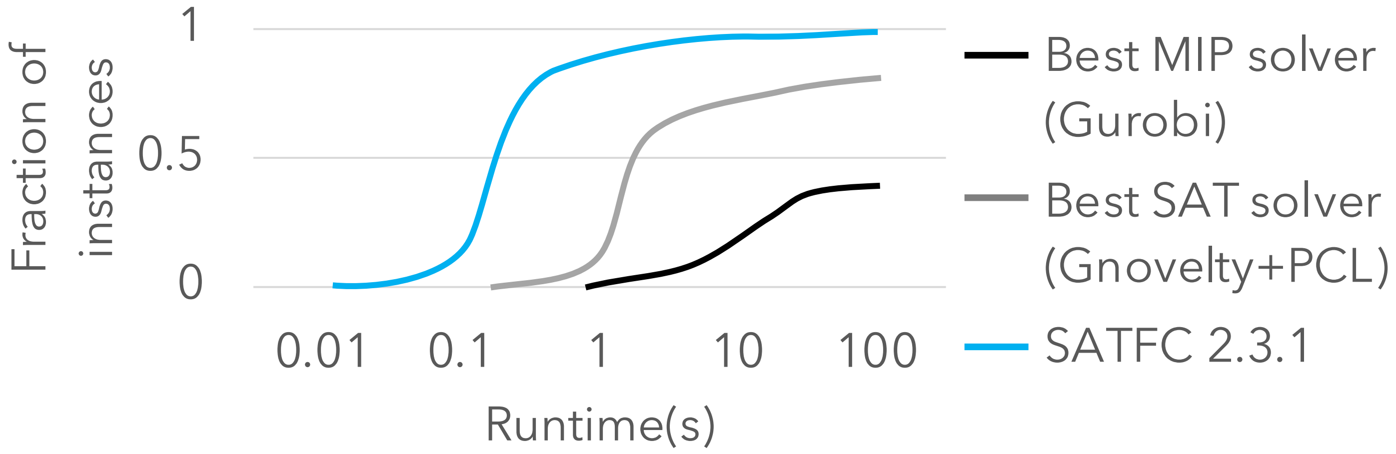


- SATFC uses algorithm configuration + selection

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

A bit of history

Important research direction in artificial intelligence for decades

A bit of history

Important research direction in artificial intelligence for decades

Has led to **breakthroughs** in

- Combinatorial auction winner determination

A bit of history

Important research direction in artificial intelligence for decades

Has led to **breakthroughs** in
- Combinatorial auction winner determination
- SAT

A bit of history

Important research direction in artificial intelligence for decades

Has led to **breakthroughs** in

- Combinatorial auction winner determination
- SAT
- Constraint satisfaction

A bit of history

Important research direction in artificial intelligence for decades

Has led to **breakthroughs** in
- Combinatorial auction winner determination
- SAT
- Constraint satisfaction
- Integer programming

A bit of history

Important research direction in artificial intelligence for decades

Has led to **breakthroughs** in

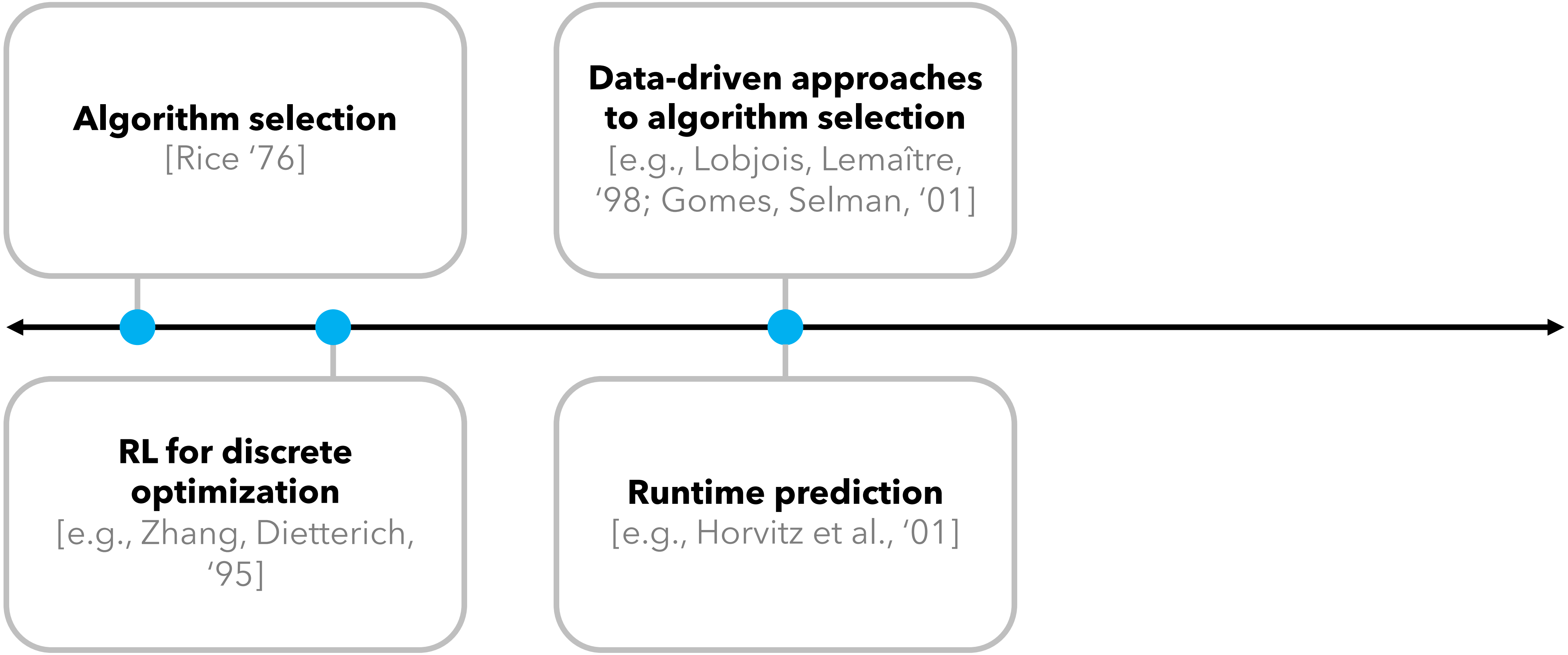
- Combinatorial auction winner determination
- SAT
- Constraint satisfaction
- Integer programming
- Many other areas

A bit of history

A bit of history

A bit of history

A bit of history



A bit of history

Algorithm selection
[Rice '76]

RL for discrete optimization
[e.g., Zhang, Dietterich, '95]

Data-driven approaches to algorithm selection
[e.g., Lobjois, Lemaître, '98; Gomes, Selman, '01]

Runtime prediction
[e.g., Horvitz et al., '01]

Machine learning for algorithm selection
[e.g., Kadioglu et al., 2010, Leyton-Brown et al., 2009, Sandholm, 2013, Xu et al., 2008, 2010, 2011]

A bit of history

Algorithm selection
[Rice ’76]

RL for discrete optimization
[e.g., Zhang, Dietterich, ’95]

Data-driven approaches to algorithm selection
[e.g., Lobjois, Lemaître, ’98; Gomes, Selman, ’01]

Runtime prediction
[e.g., Horvitz et al., ’01]

Machine learning for algorithm selection
[e.g., Kadioglu et al., 2010, Leyton-Brown et al., 2009, Sandholm, 2013, Xu et al., 2008, 2010, 2011]

Machine learning for algorithm configuration
[e.g., Hutter et al. ’09, ’10, ’11, Ansótegui et al. ’09, Sandholm, 2013]

A bit of history

Plan for tutorial

1. **Applied techniques**
   a. Graph neural networks
   b. Reinforcement learning

2. **Theoretical guarantees**
   a. Statistical guarantees for algorithm configuration
   b. Algorithms with predictions

Where much of my research has been

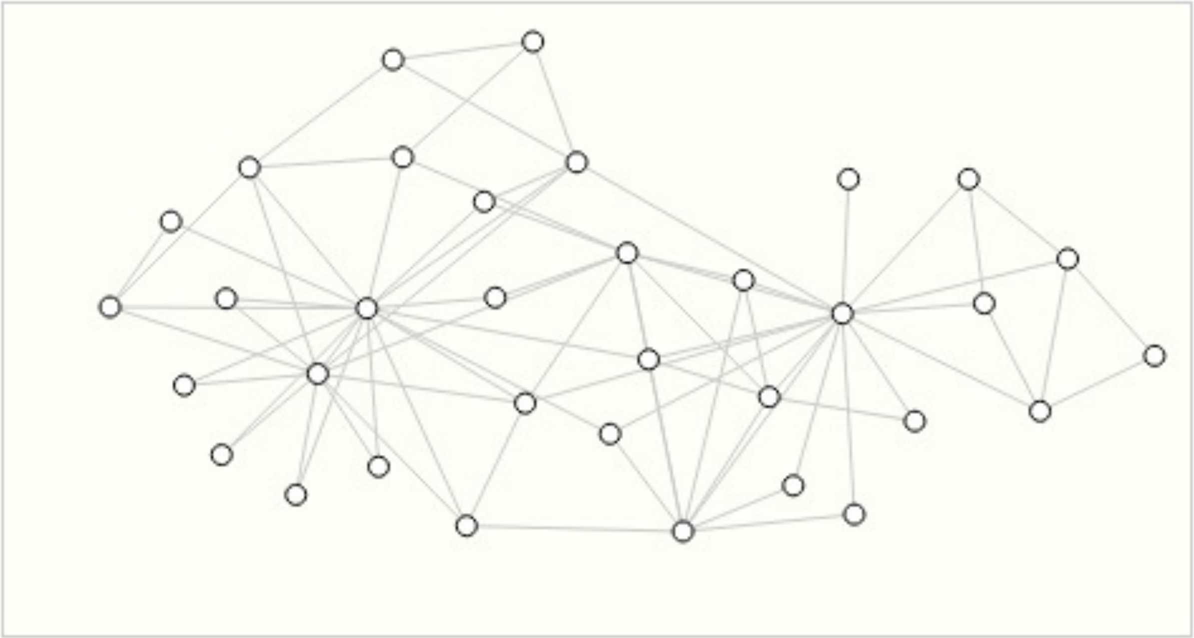
Outline (applied techniques)

1. **GNNs overview**
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL

GNN motivation

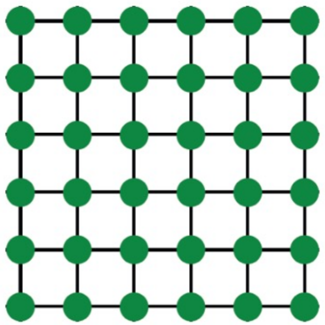
Main question:

How to utilize relational structure for better prediction?

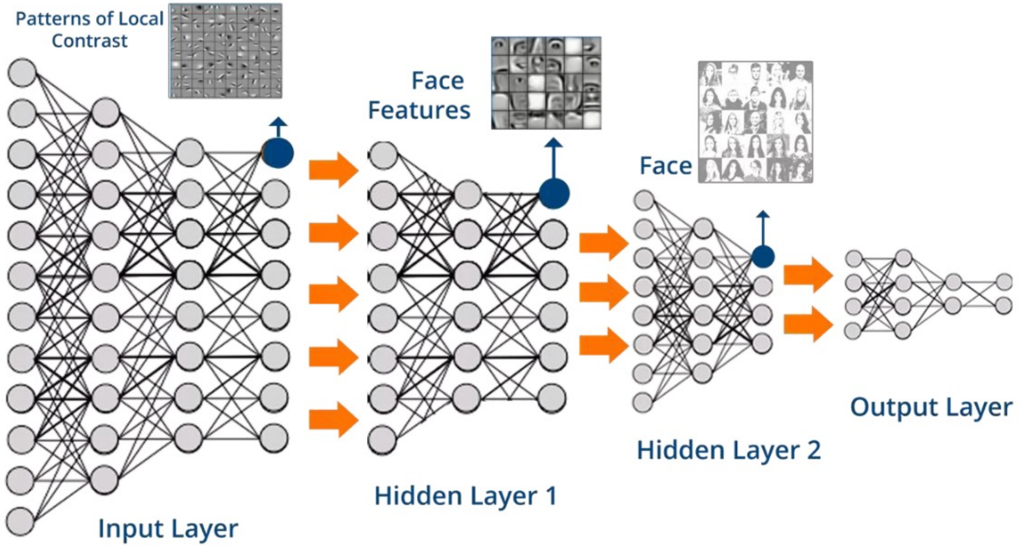


Today: Modern ML toolbox

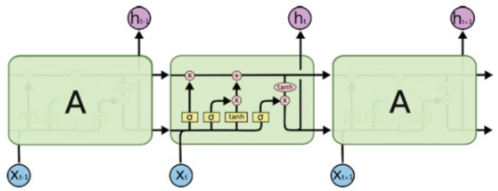
Modern DL toolbox is designed for simple sequences & grids



Images



Text/Speech

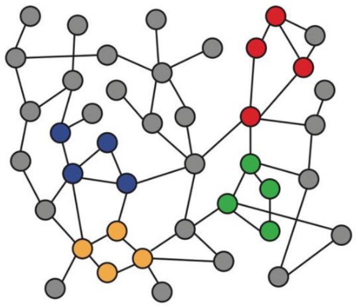


Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Why is graph deep learning hard?

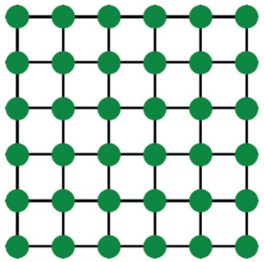
Networks are complex

- Arbitrary size and complex topological structure

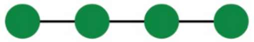


Networks

versus



Images



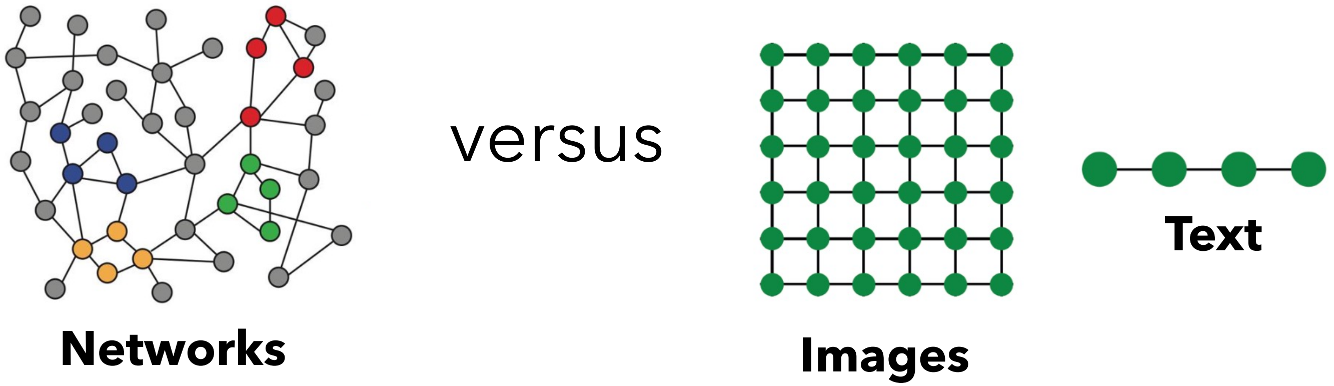
Text

Figure by Leskovec

Why is graph deep learning hard?

Networks are complex

- Arbitrary size and complex topological structure



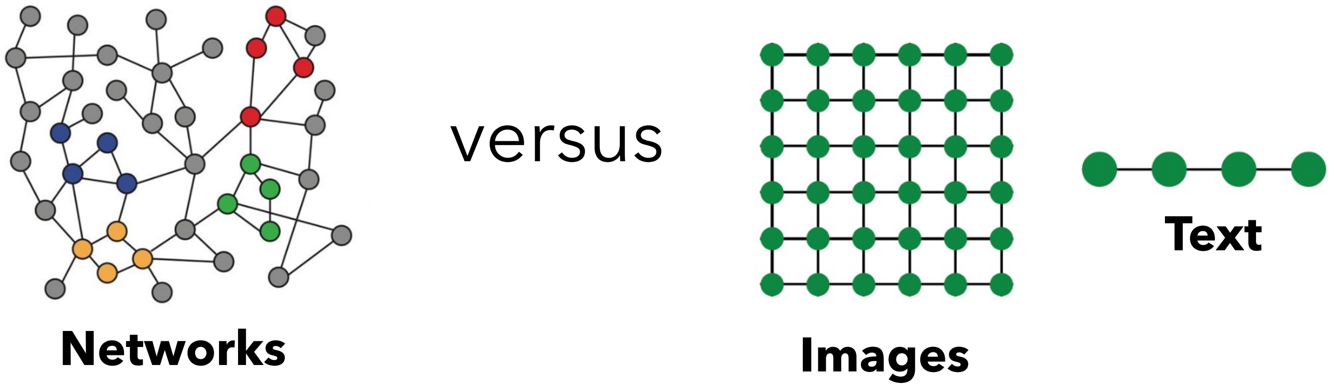
- No fixed node ordering or reference point

Figure by Leskovec

Why is graph deep learning hard?

Networks are complex

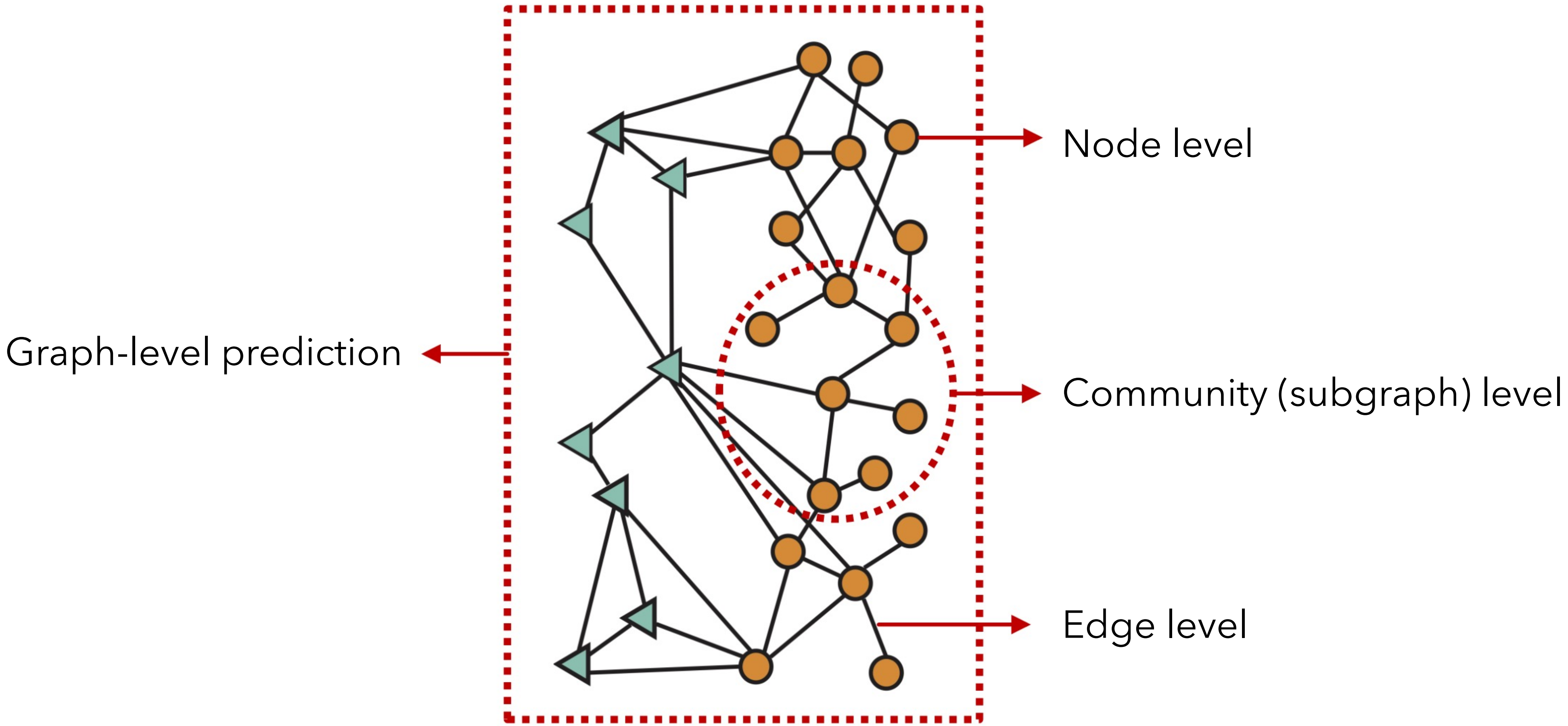
- Arbitrary size and complex topological structure



- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Figure by Leskovec

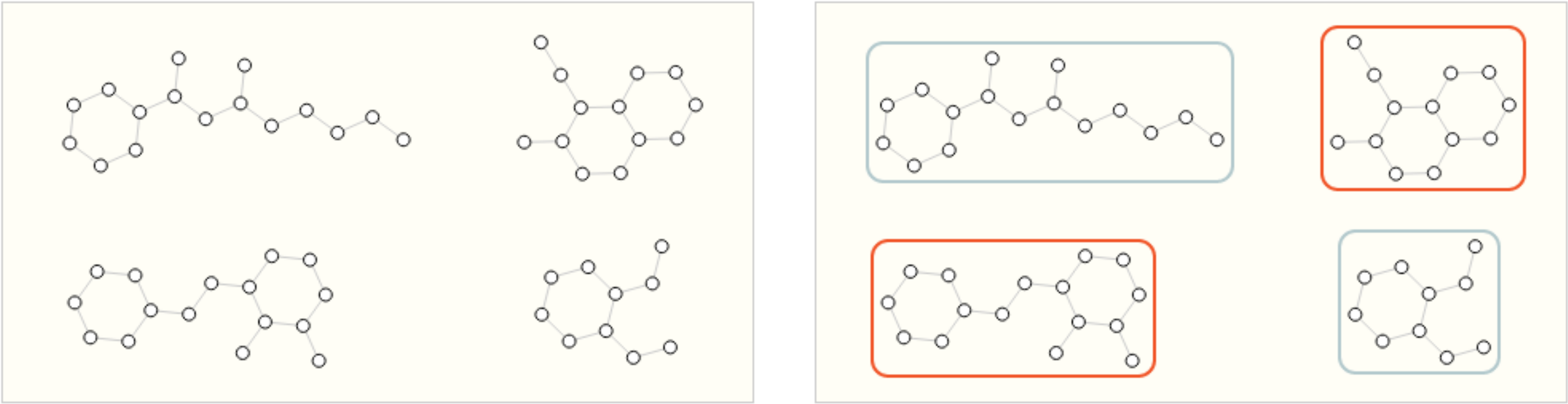
Different types of tasks



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Figure by Leskovec

Prediction with graphs: Examples

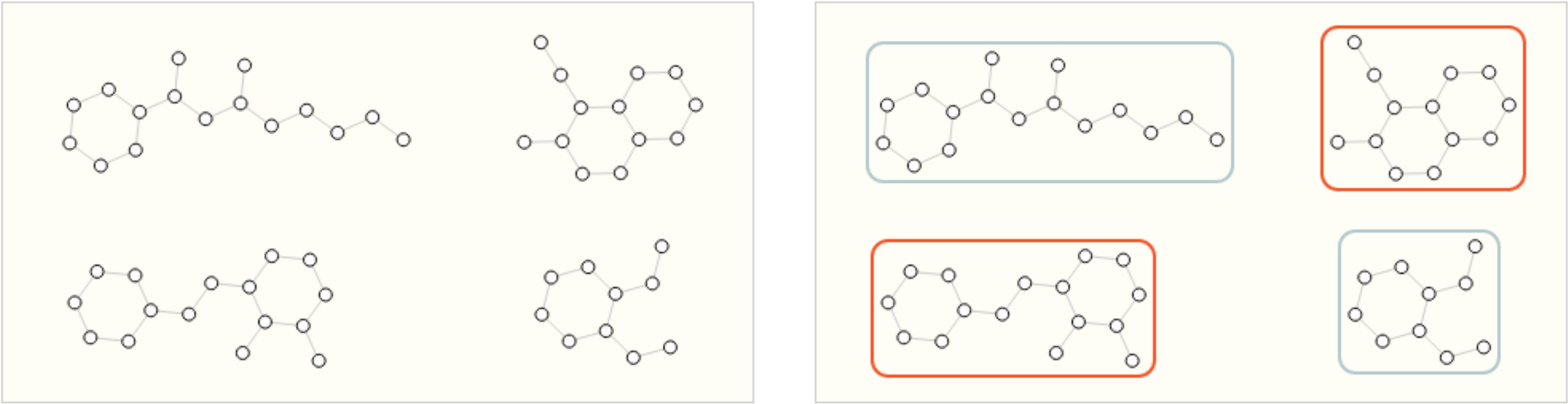


Graph-level tasks:

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

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

Prediction with graphs: Examples



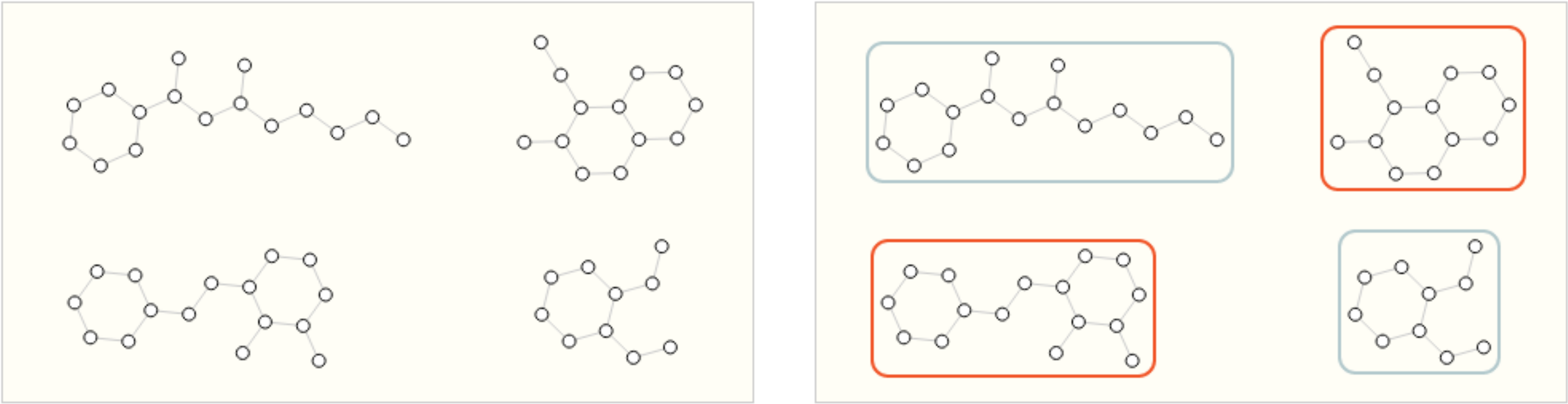
Graph-level tasks:

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

- What the molecule smells like

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

Prediction with graphs: Examples



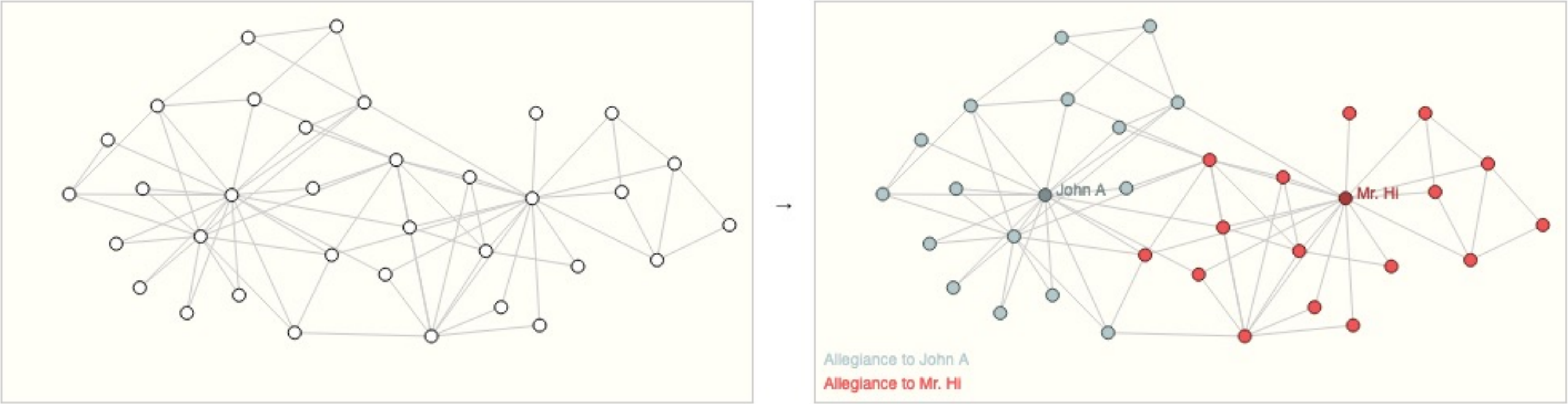
Graph-level tasks:

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]

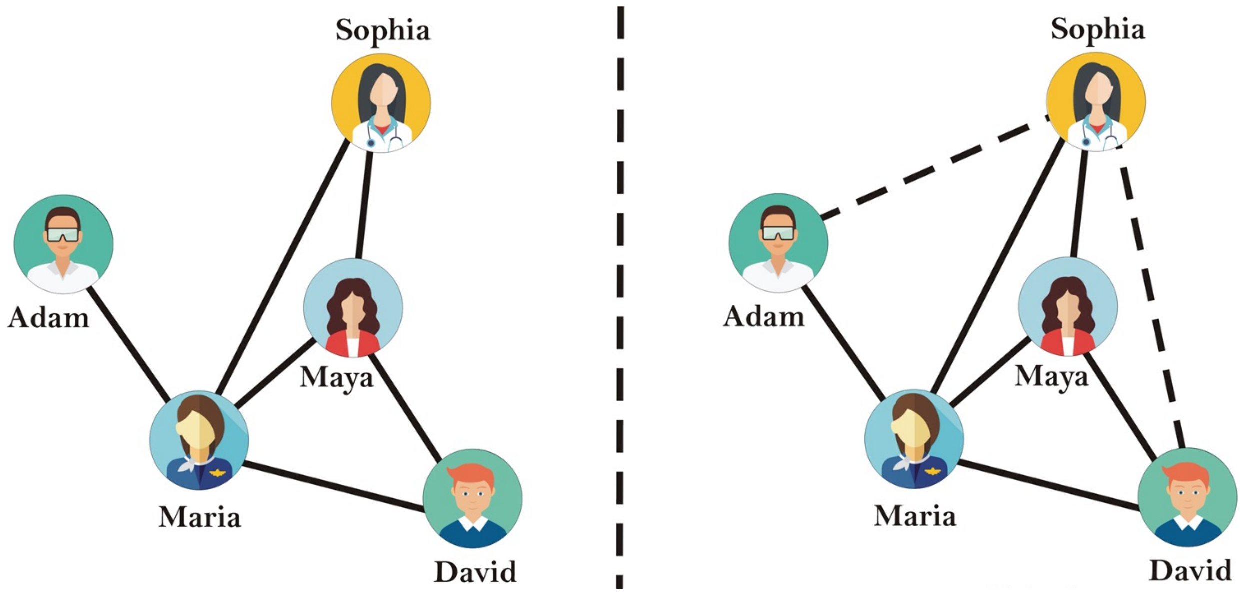
Prediction with graphs: Examples



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, …

Example: Traffic routing

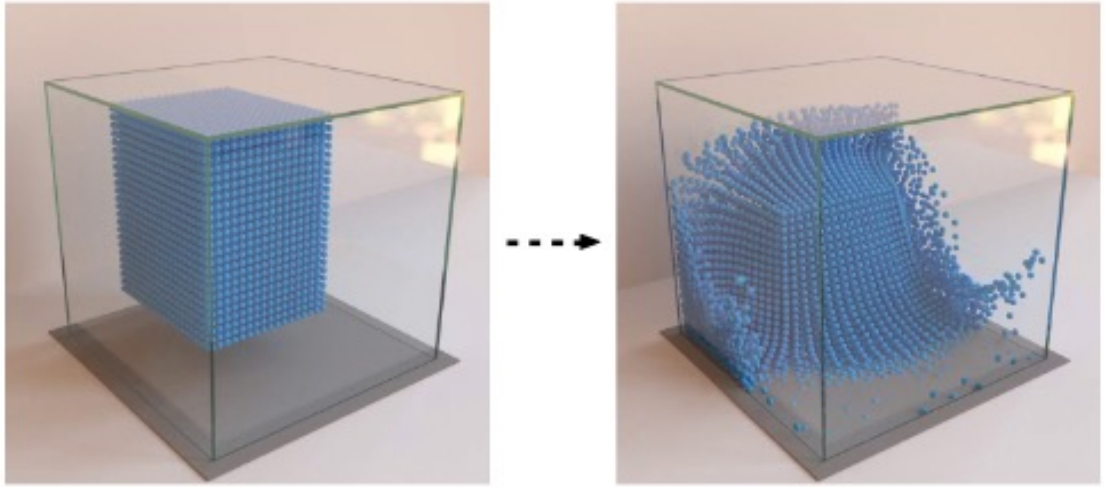
E.g., Google maps

deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks

Example: Learning to simulate physics

Nodes: Particles

Edges: Interaction between particles

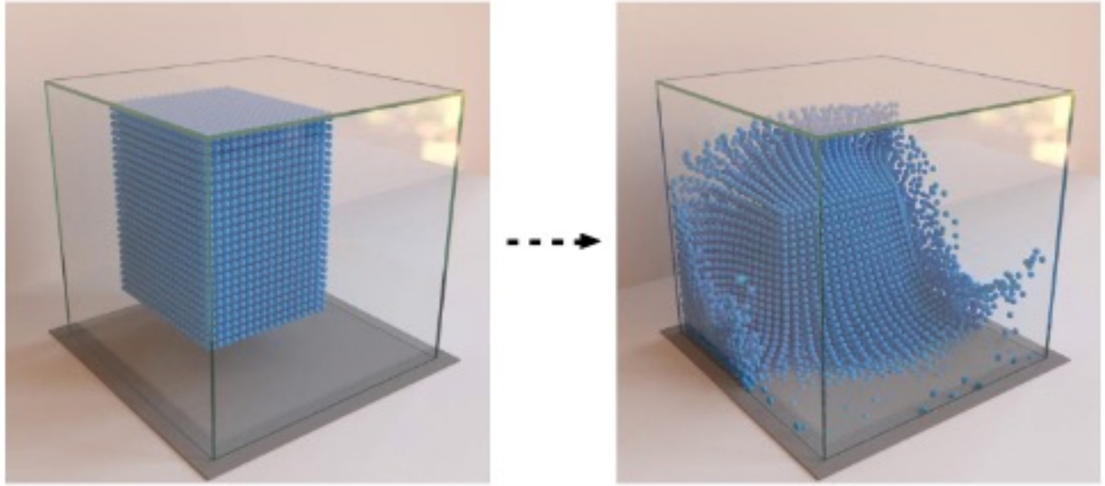


Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Example: Learning to simulate physics

Nodes: Particles

Edges: Interaction between particles

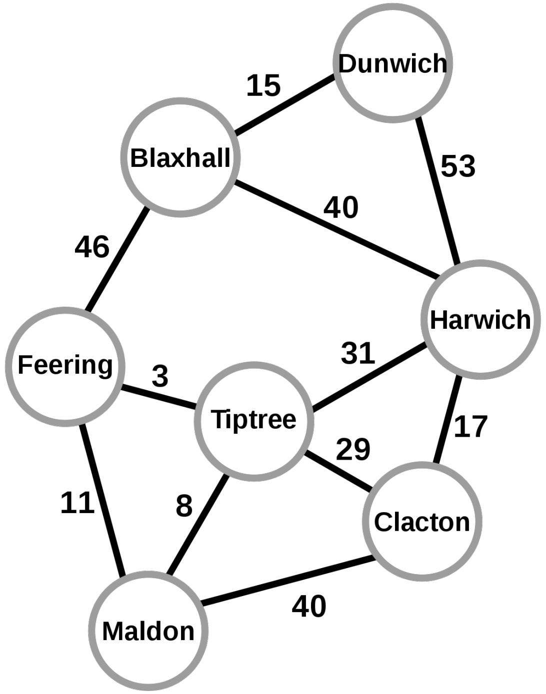


Goal: Predict how a graph will evolve over time

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Example: Combinatorial optimization

Replace full algorithm or learn steps (e.g., branching decision)



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

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Graph neural networks: First step

- Design features for nodes/links/graphs

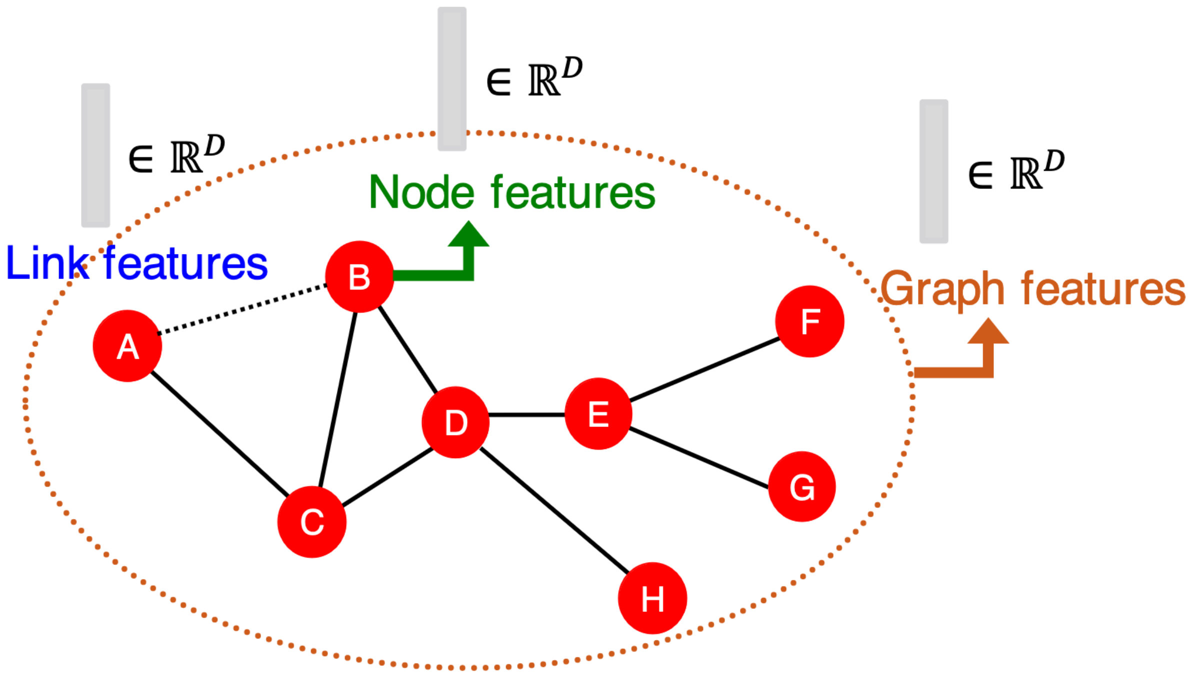


Figure by Leskovec

Graph neural networks: First step

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

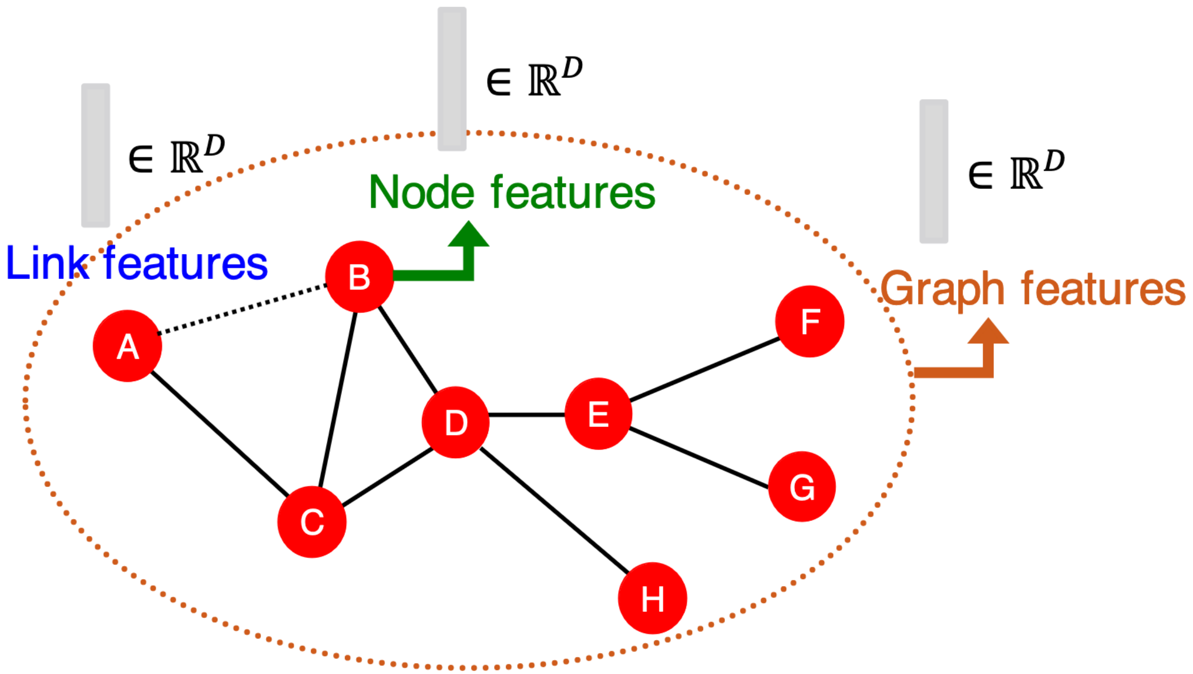


Figure by Leskovec

Graph neural networks: Objective

Idea:

1. Encode each node and its neighborhood with embedding

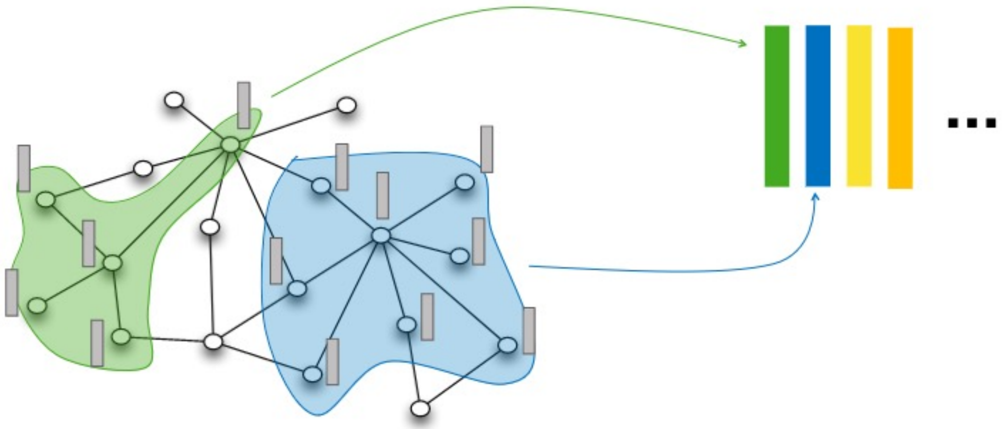


Figure by Jegelka

Graph neural networks: Objective

Idea:

1. Encode each node and its neighborhood with embedding
2. Aggregate set of node embeddings into graph embedding

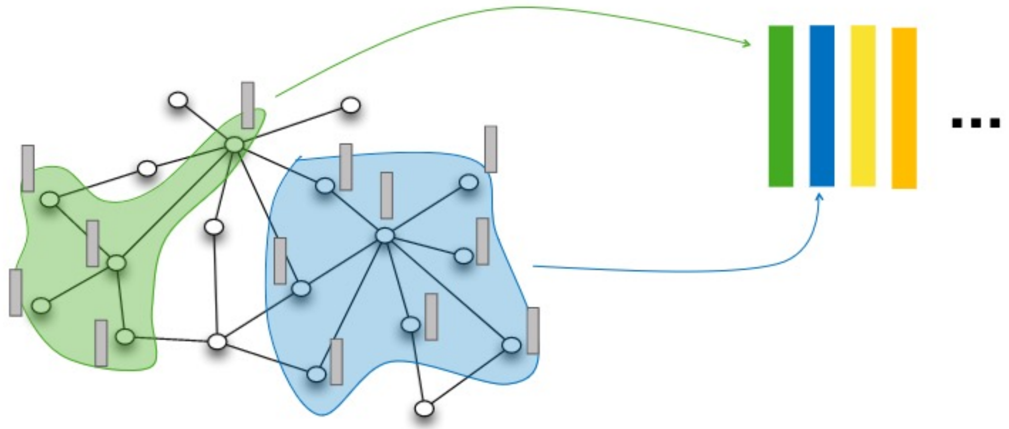


Figure by Jegelka

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

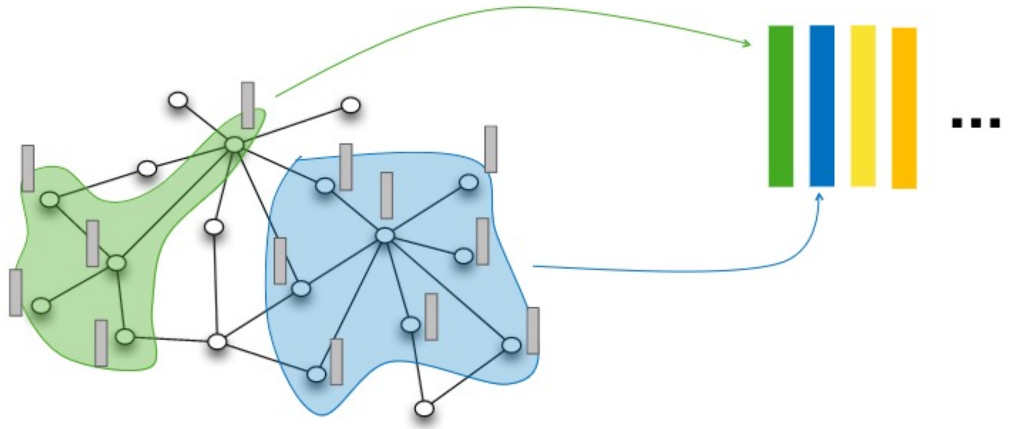
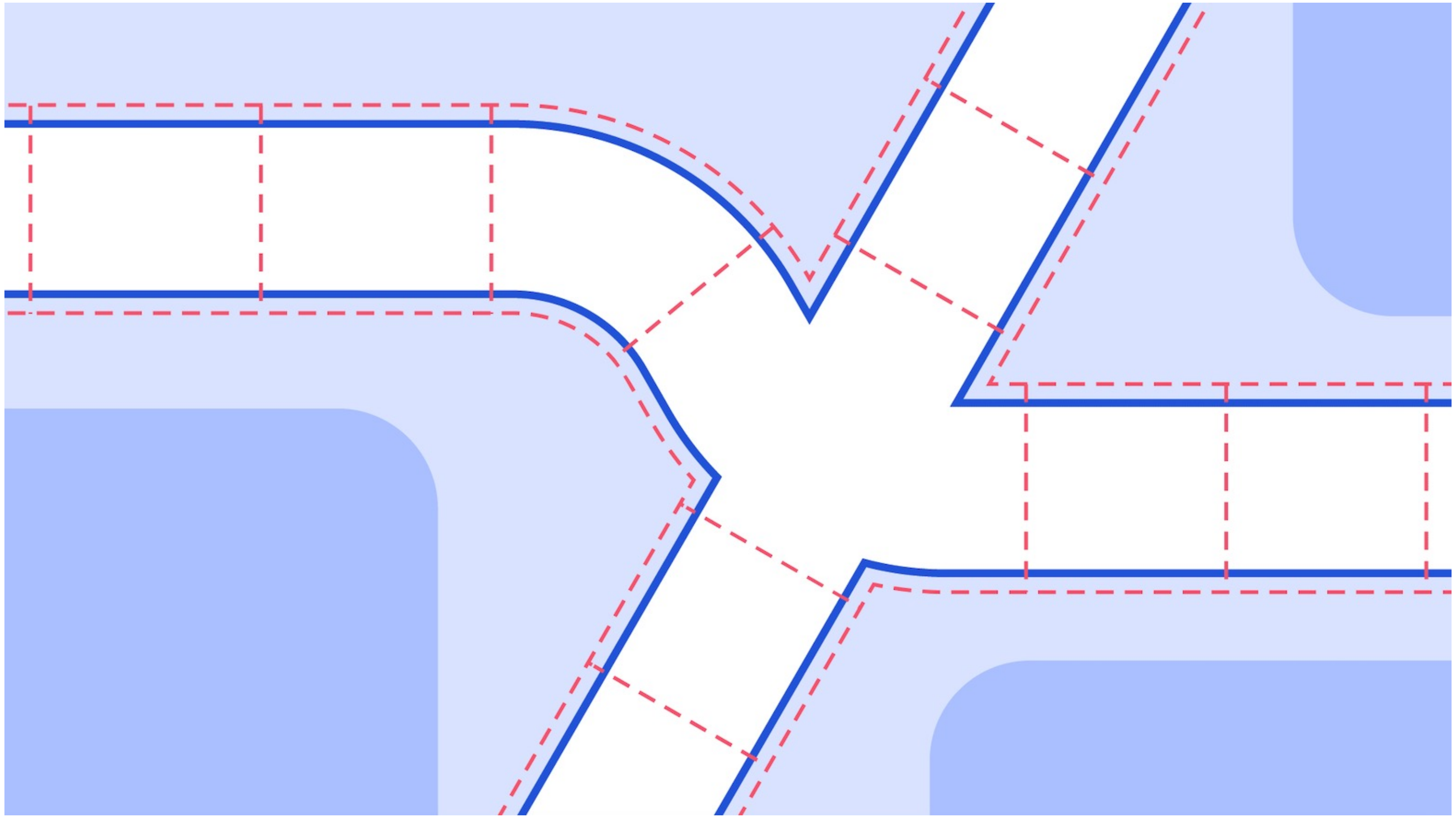


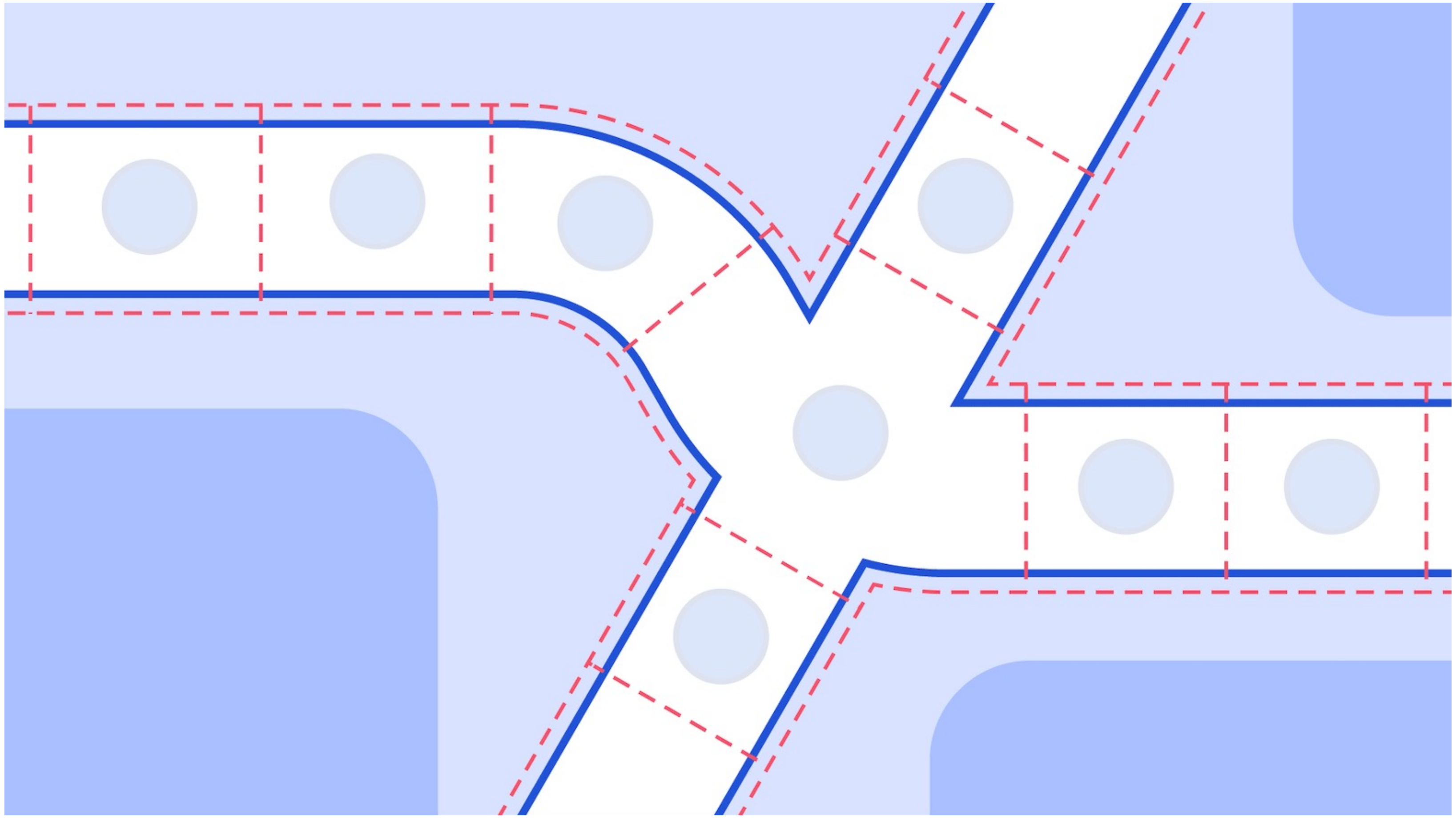
Figure by Jegelka

Ellen Vitercik, AAAI’24 tutorial
vitercik.github.io/slides/AAAI24.pdf

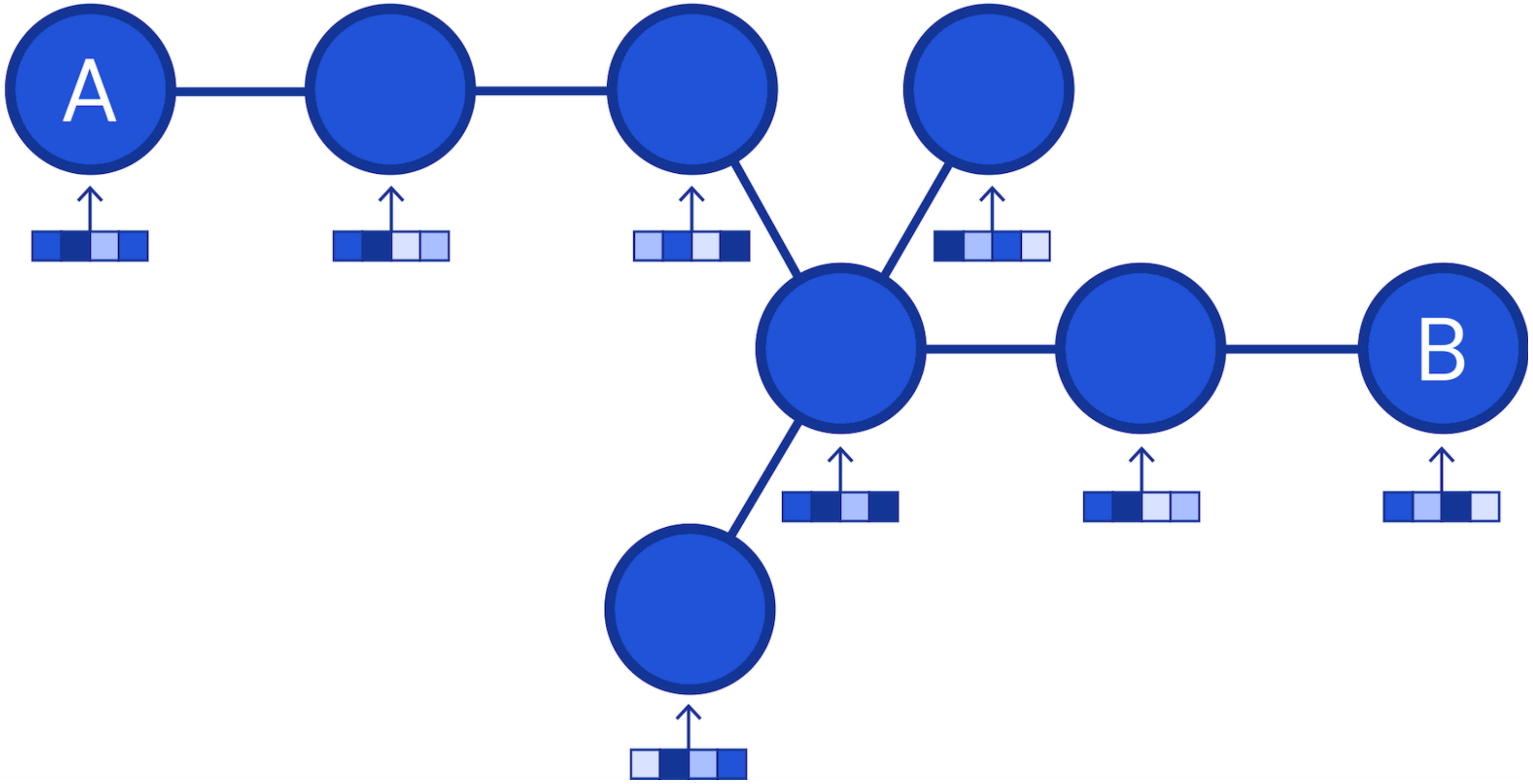
https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks



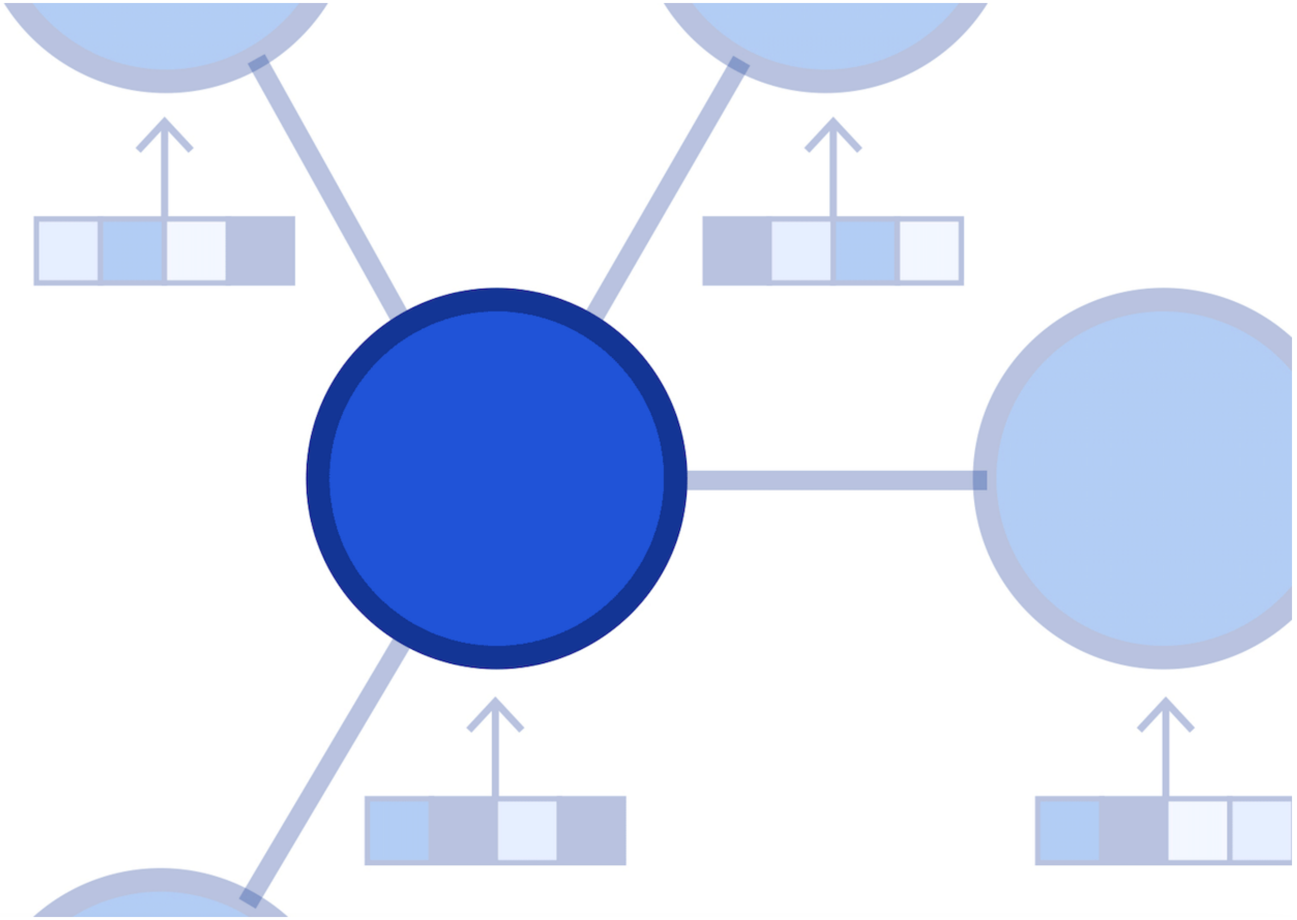
https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks



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

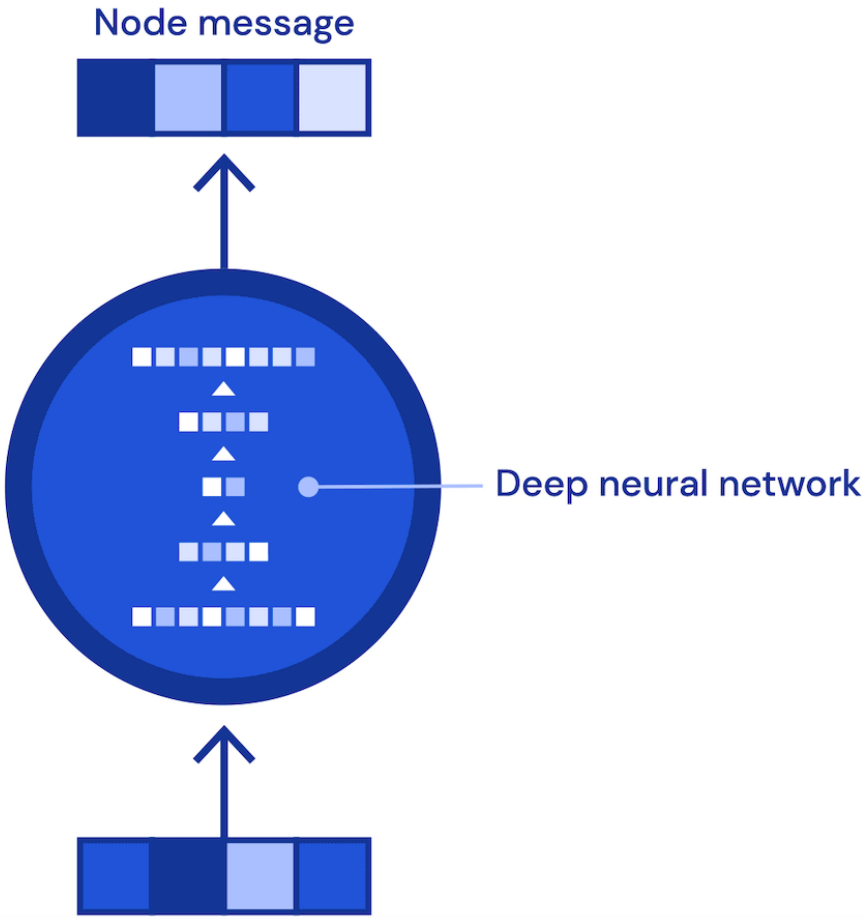


https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks

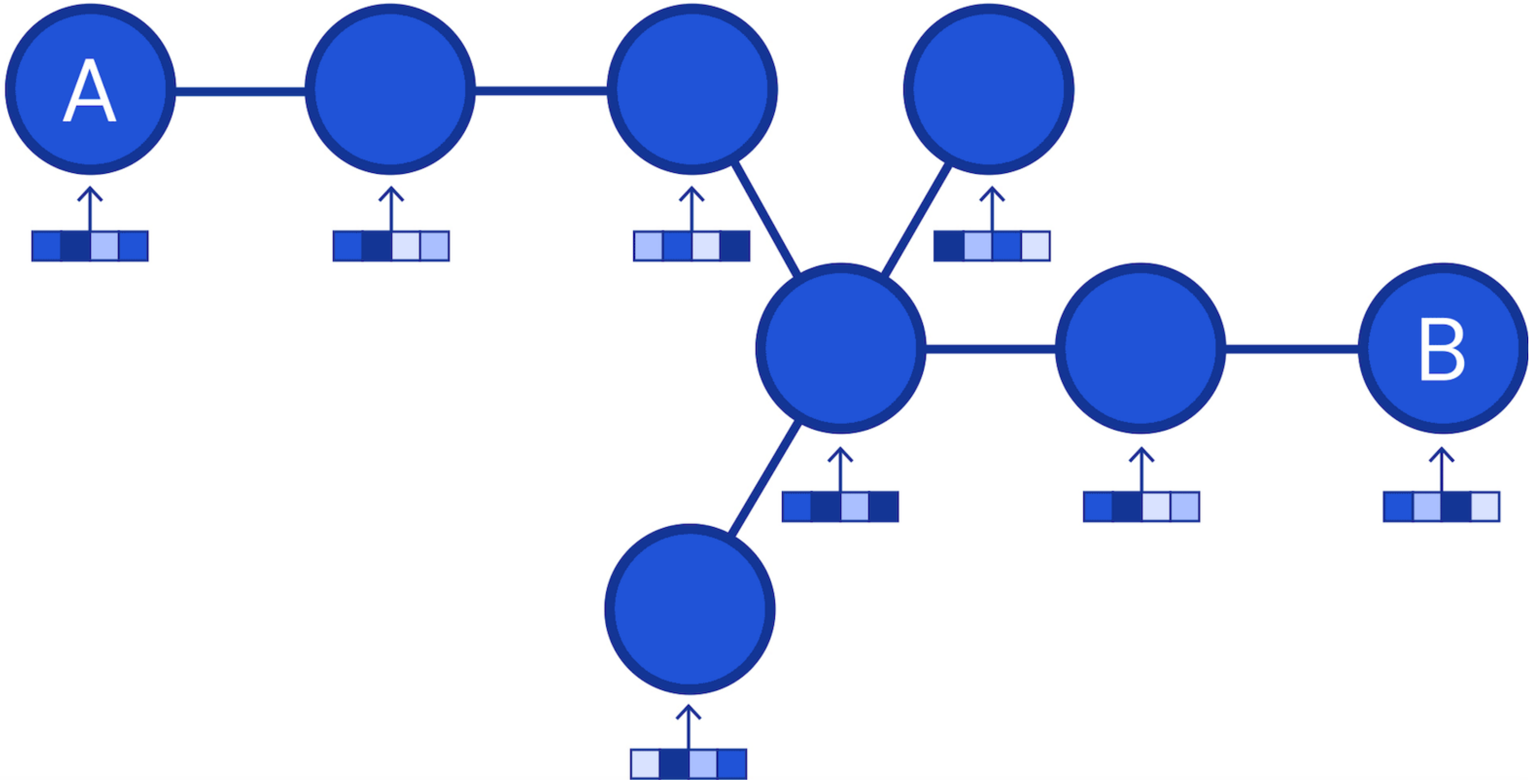


Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

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

Encoding neighborhoods: General form

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

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

Encoding neighborhoods: General form

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

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

1. **Aggregate** over neighbors

Encoding neighborhoods: General form

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

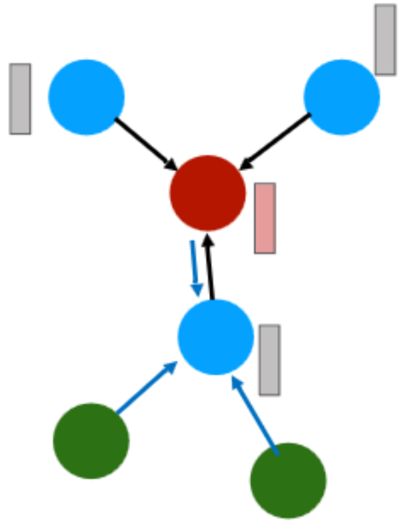
In each round $k \in [K]$, for each node v:

1. **Aggregate** over neighbors

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

Encoding neighborhoods: General form

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

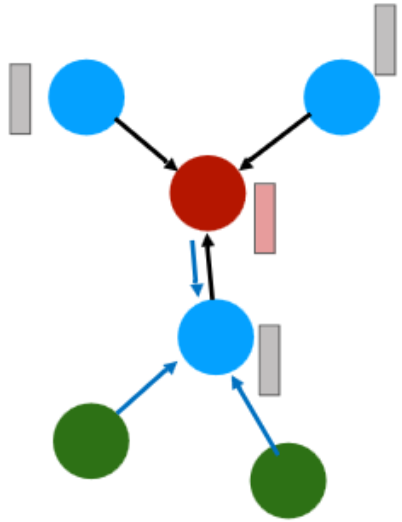


In each round $k \in [K]$, for each node v:

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

Encoding neighborhoods: General form

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

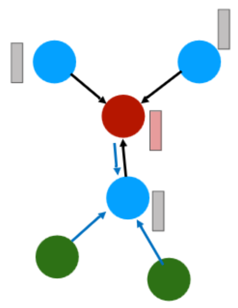


In each round $k \in [K]$, for each node v:

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

The basic GNN

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

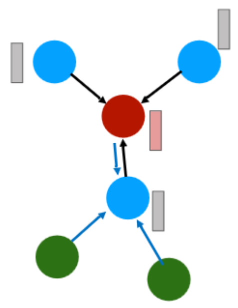


$$\boldsymbol{m}_{N(v)} = \text{AGGREGATE}(\{\boldsymbol{h}_u : u \in N(v)\}) = \sum_{u \in N(v)} \boldsymbol{h}_u$$

Figure by Jegelka

The basic GNN

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



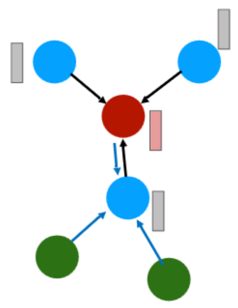
$$\boldsymbol{m}_{N(v)} = \text{AGGREGATE}(\{\boldsymbol{h}_u : u \in N(v)\}) = \sum_{u \in N(v)} \boldsymbol{h}_u$$

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

Figure by Jegelka

The basic GNN

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



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

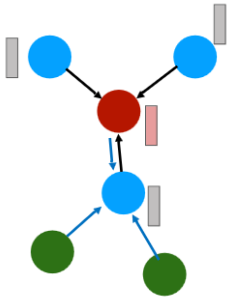
$$\mathrm{COMBINE}\left(\boldsymbol{h}_v, \boldsymbol{m}_{N(v)}\right) = \sigma\left(W_{\mathrm{self}} \boldsymbol{h}_v + W_{\mathrm{neigh}} \boldsymbol{m}_{N(v)} + \boldsymbol{b}\right)$$

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

Figure by Jegelka

The basic GNN

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



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

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

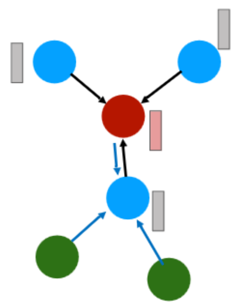
Trainable parameters

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

Ellen Vitercik, AAAI’24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Figure by Jegelka

Aggregation functions

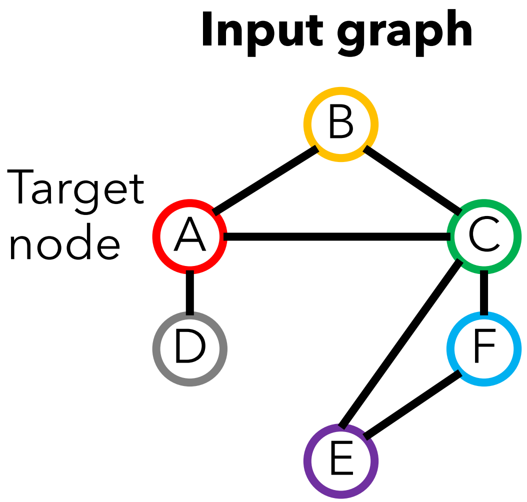


$$\boldsymbol{m}_{N(v)} = \text{AGGREGATE}(\{\boldsymbol{h}_u : u \in N(v)\}) = \underbrace{\bigoplus_{u \in N(v)}}_{} \boldsymbol{h}_u$$

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

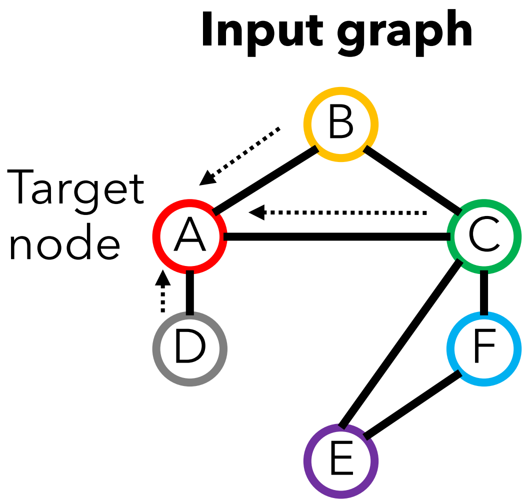
Figure by Jegelka

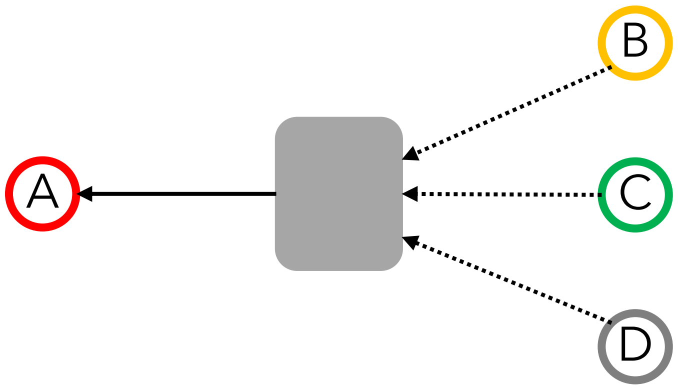
Node embeddings unrolled



Figures by Leskovec

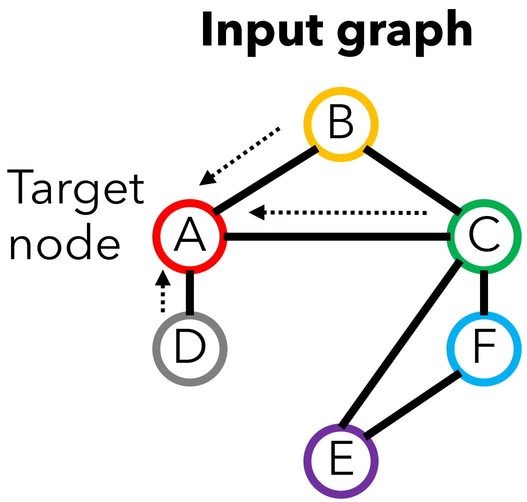
Node embeddings unrolled

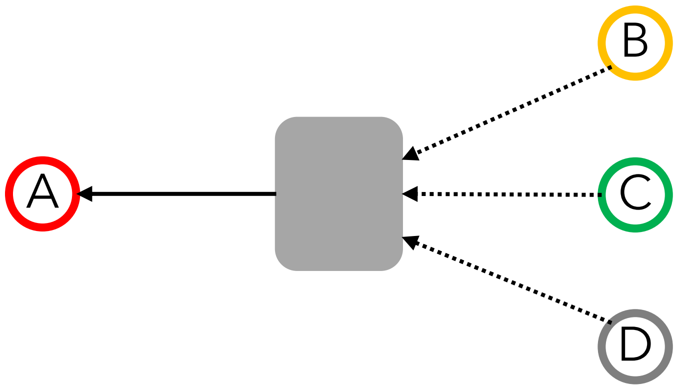




Figures by Leskovec

Node embeddings unrolled

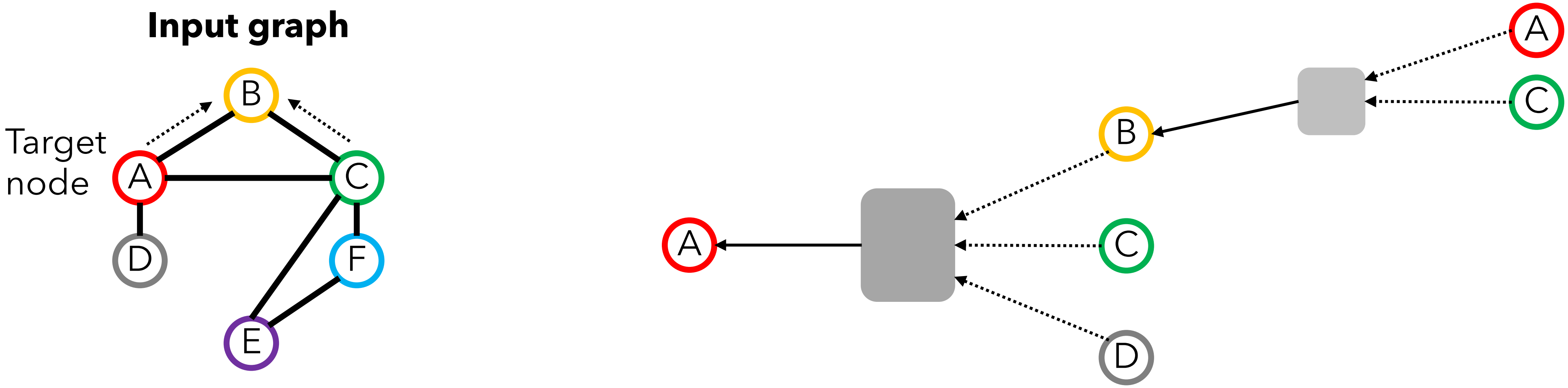




Grey boxes: aggregation functions that we learn

Figures by Leskovec

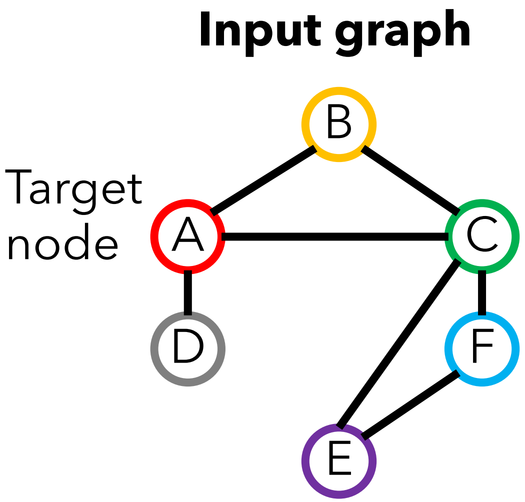
Node embeddings unrolled

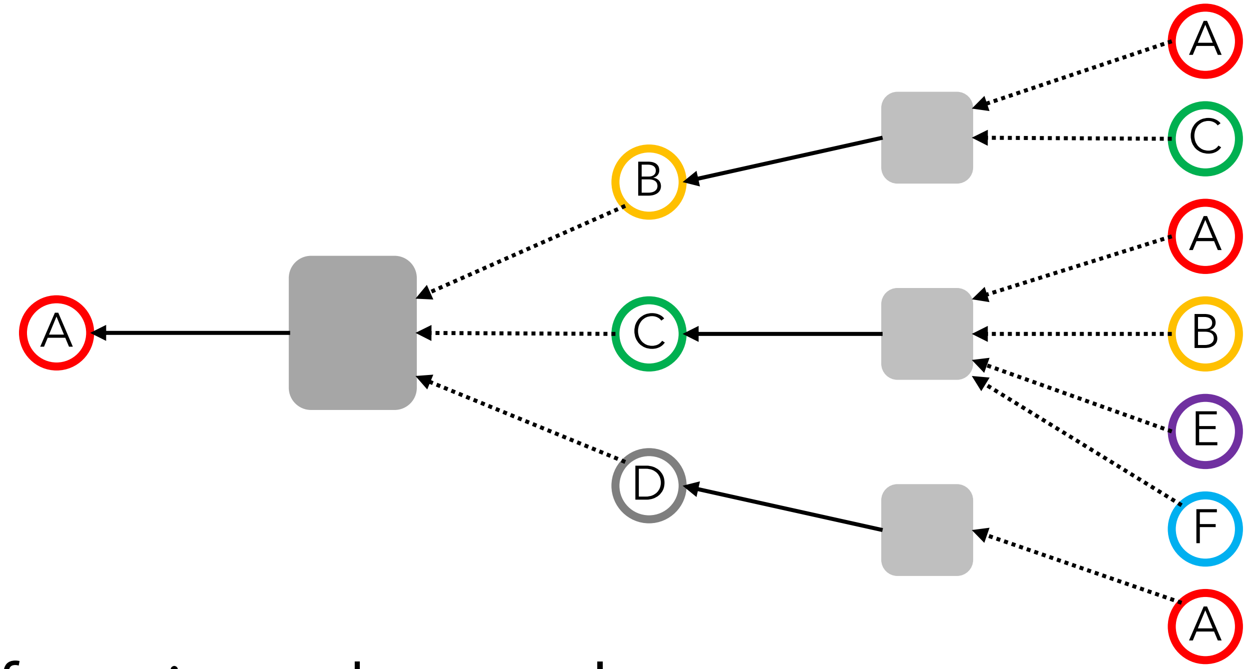


Grey boxes: aggregation functions that we learn

Figures by Leskovec

Node embeddings unrolled

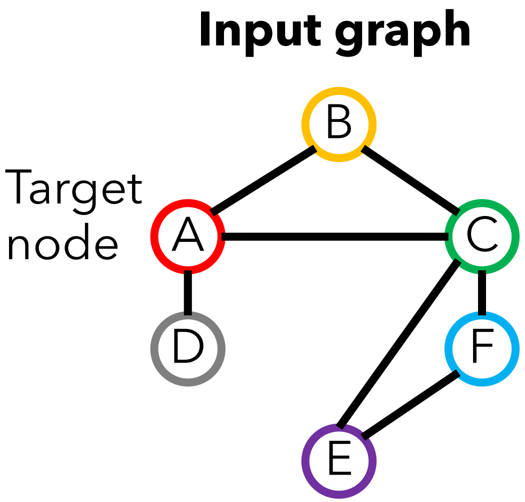


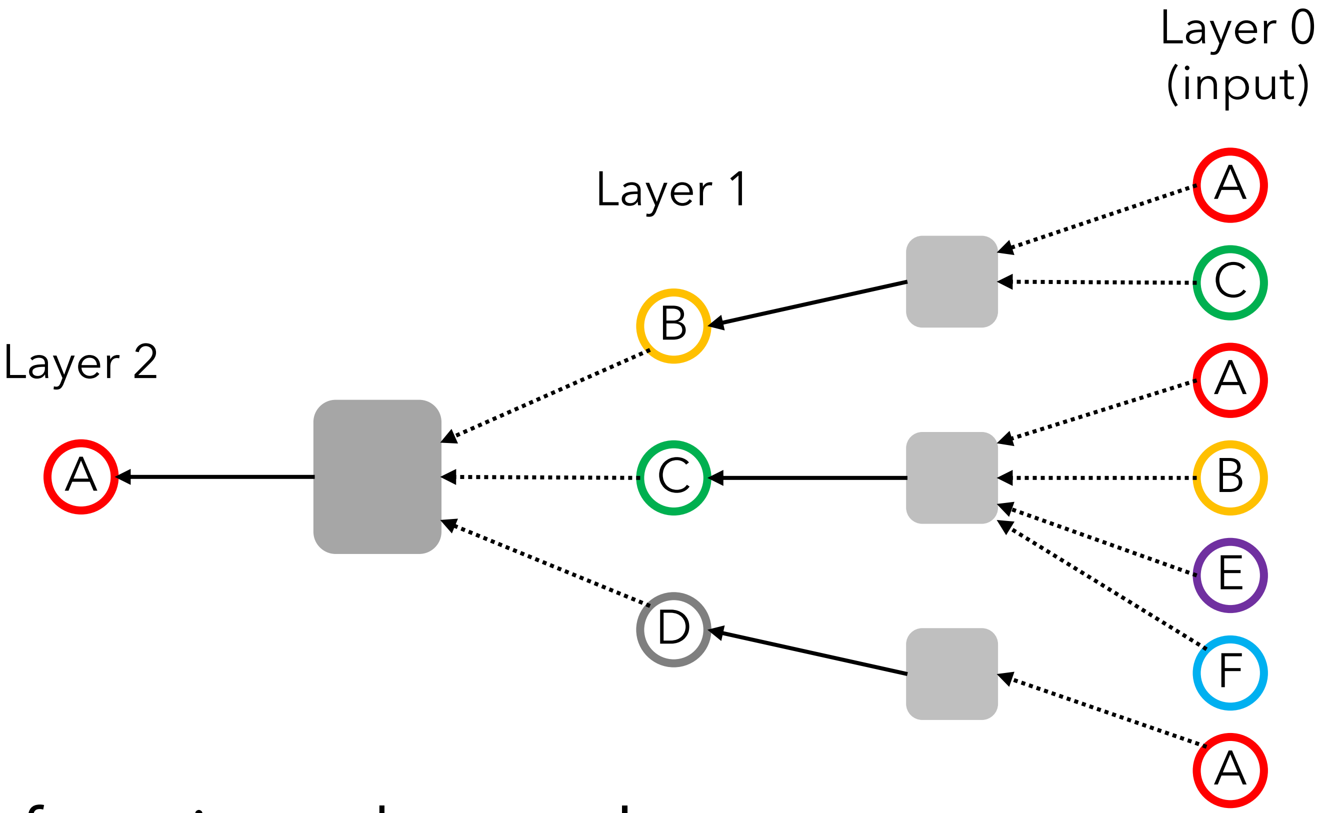


Grey boxes: aggregation functions that we learn

Figures by Leskovec

Node embeddings unrolled

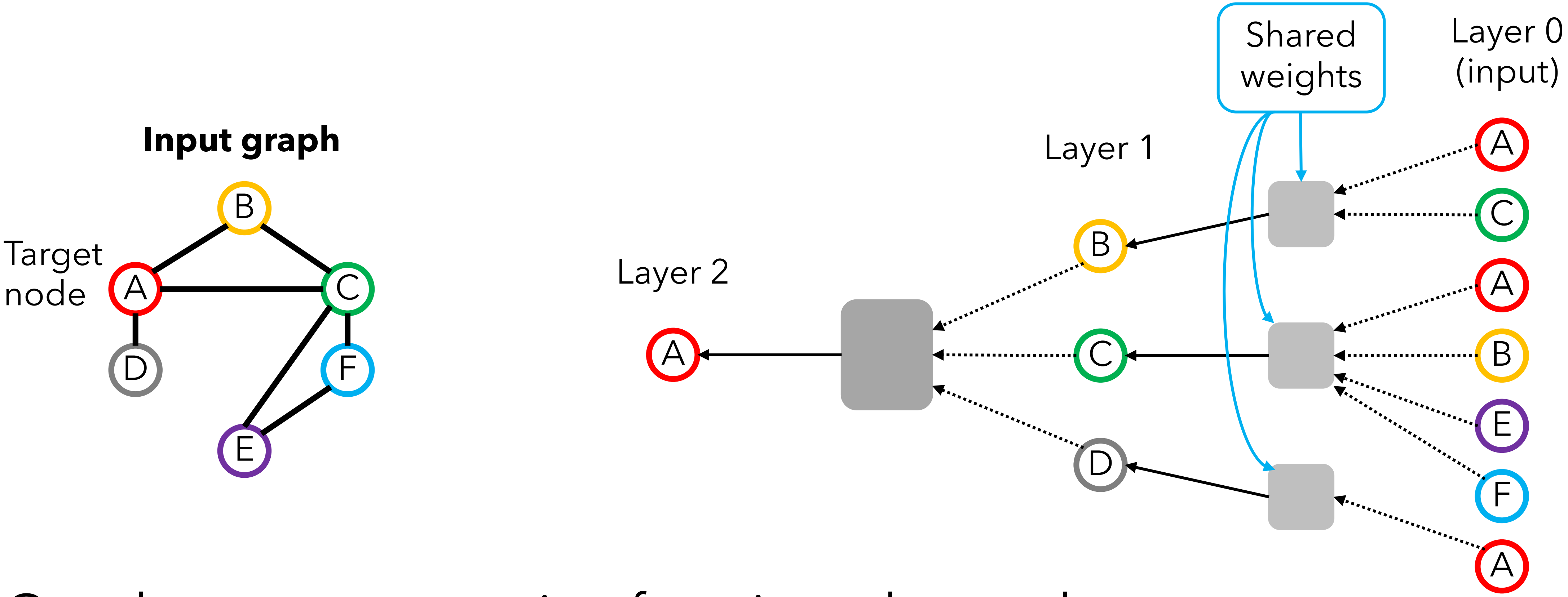




Grey boxes: aggregation functions that we learn

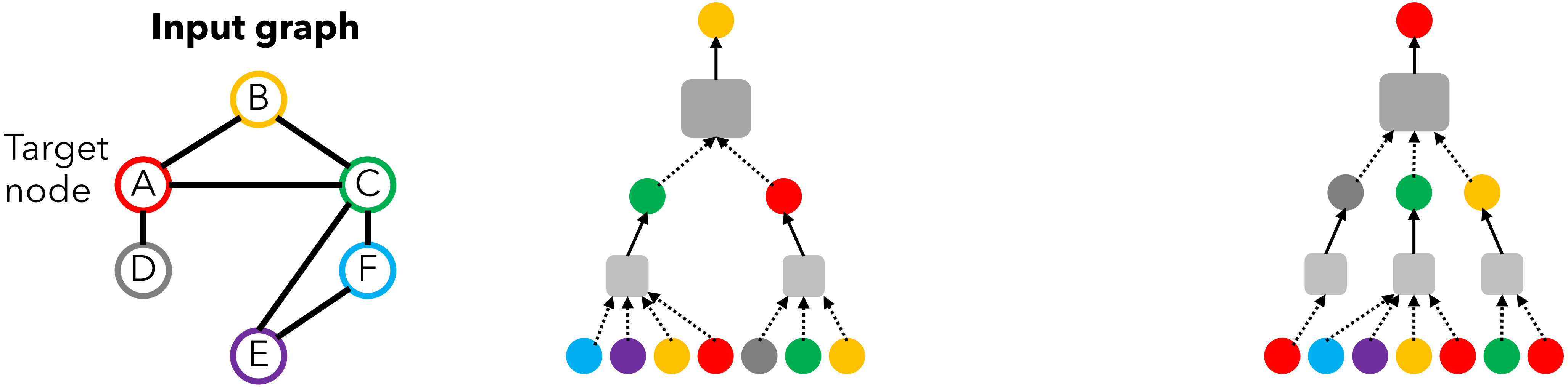
Figures by Leskovec

Node embeddings unrolled



Grey boxes: aggregation functions that we learn

Node embeddings unrolled

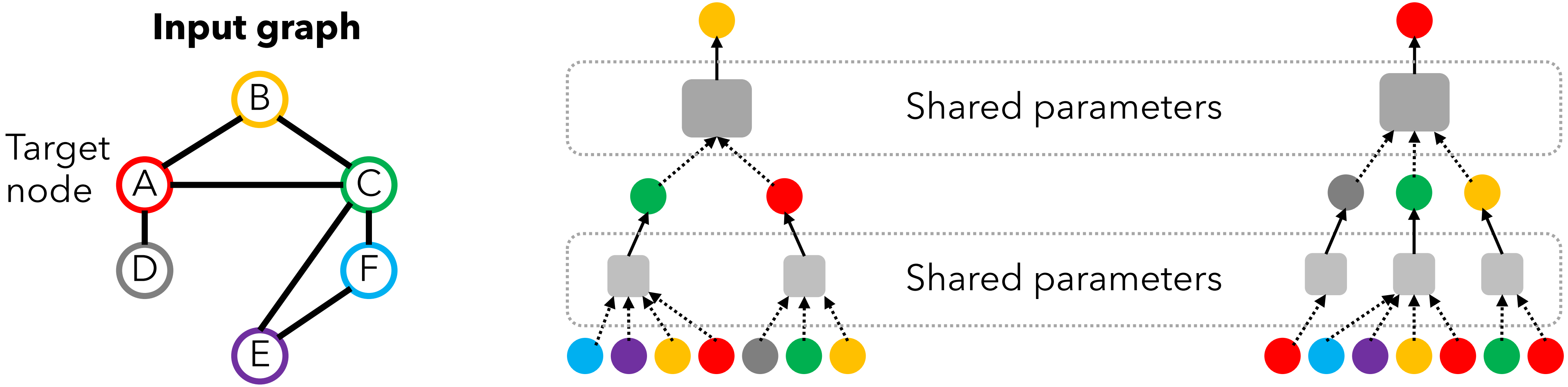


Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Figures by Leskovec

Node embeddings unrolled

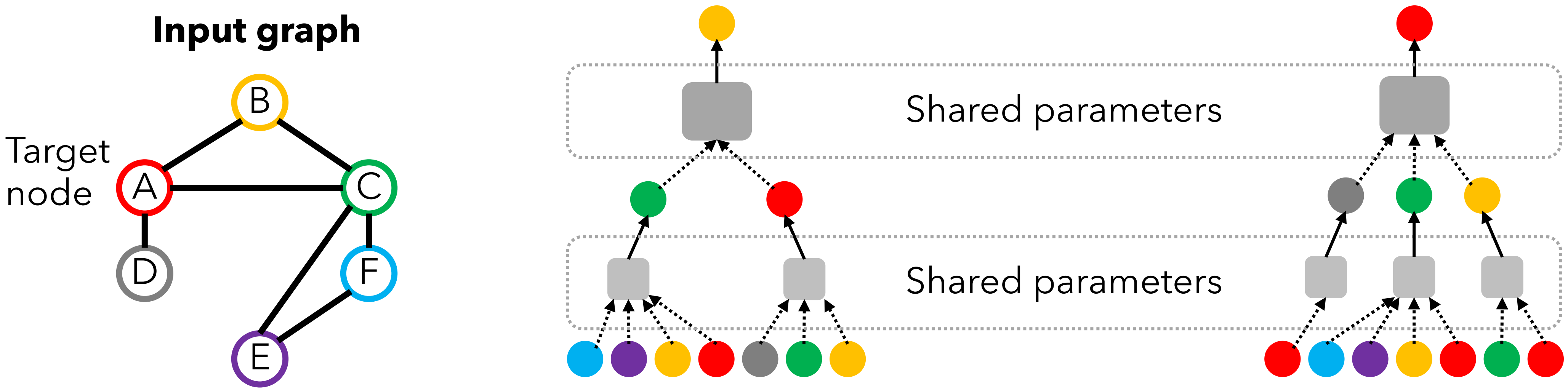
Use the same aggregation functions for all nodes



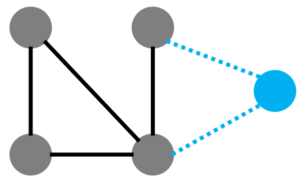
Figures by Leskovec

Node embeddings unrolled

Use the same aggregation functions for all nodes



Can generate encodings for previously unseen nodes & graphs!



Figures by Leskovec

Outline (applied techniques)

1. GNNs overview
2. **Integer programming with GNNs**
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL

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

Integer programming solvers

Most popular tool for solving combinatorial problems

Routing

Manufacturing

Scheduling

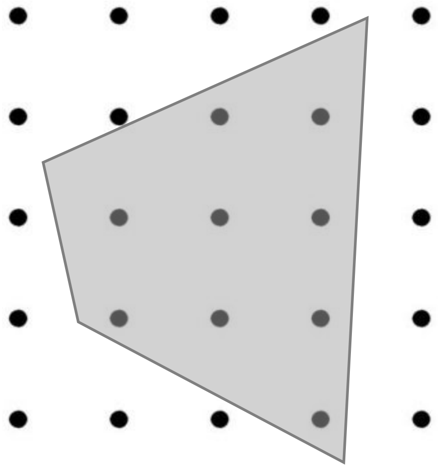
Planning

Finance

Integer and linear programming

Integer program (IP)

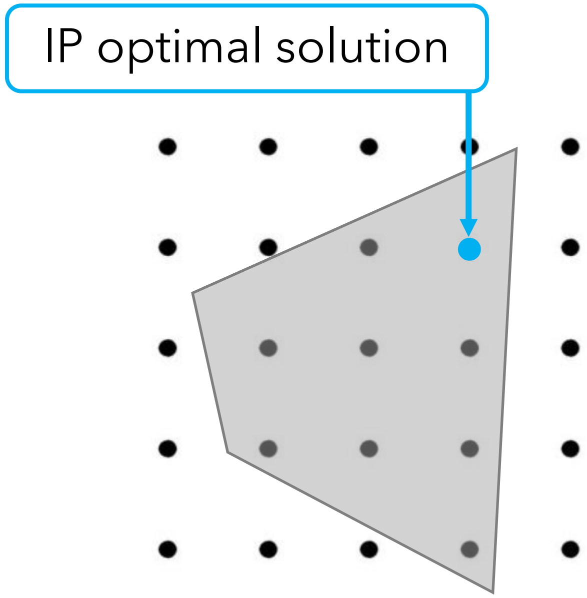
$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \boldsymbol{z} \in \mathbb{Z}^n \end{aligned}$$



Integer and linear programming

Integer program (IP)

$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \boldsymbol{z} \in \mathbb{Z}^n \end{aligned}$$



Integer and linear programming

Integer program (IP)

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

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

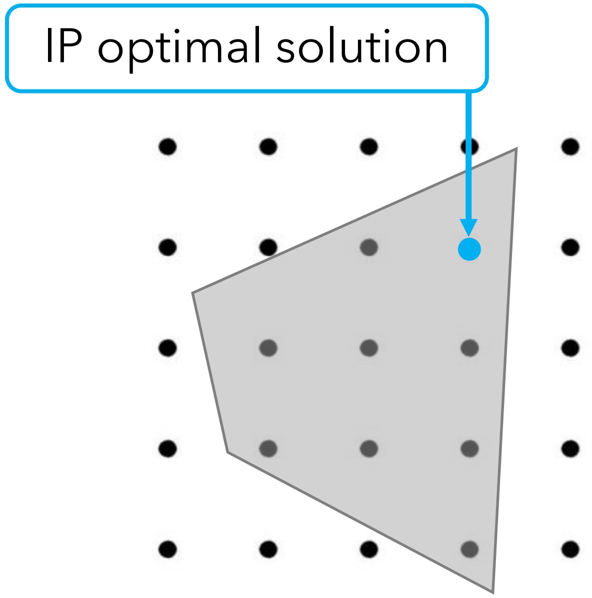
$\boldsymbol{z} \in \mathbb{Z}^n$

Linear program (LP)

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

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

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



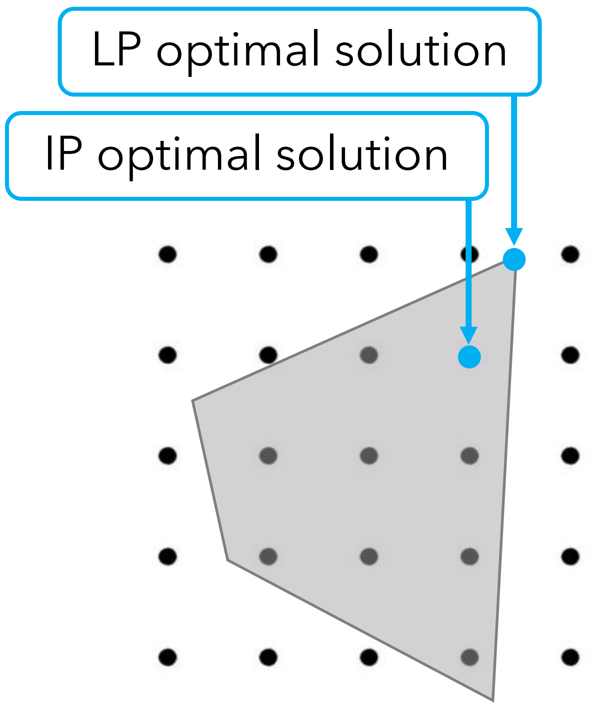
Integer and linear programming

Integer program (IP)

$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \boldsymbol{z} \in \mathbb{Z}^n \end{aligned}$$

Linear program (LP)

$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \cancel{\boldsymbol{z} \in \mathbb{Z}^n} \end{aligned}$$



Integer and linear programming

Integer program (IP)

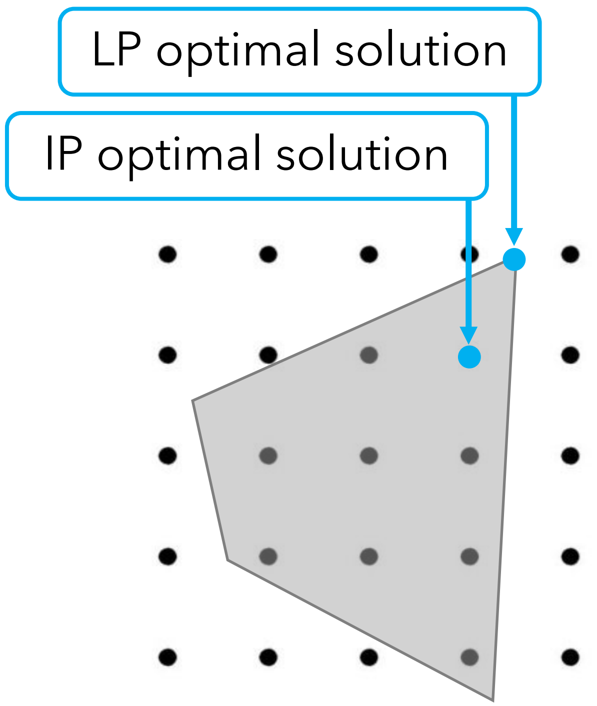
$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \le \boldsymbol{b} \\ & \boldsymbol{z} \in \mathbb{Z}^n \end{aligned}$$

NP-hard

Linear program (LP)

$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \le \boldsymbol{b} \\ & \cancel{\boldsymbol{z} \in \mathbb{Z}^n} \end{aligned}$$

Efficiently solvable



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Integer and linear programming

Integer program (IP)

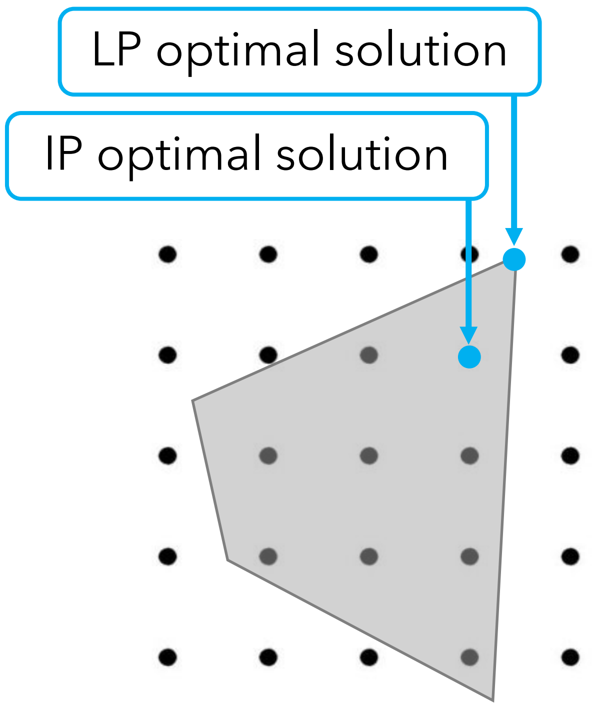
$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \boldsymbol{z} \in \mathbb{Z}^n \end{aligned}$$

NP-hard

Linear program (LP)

$$\begin{aligned} \max \quad & \boldsymbol{c} \cdot \boldsymbol{z} \\ \text{s.t.} \quad & A\boldsymbol{z} \leq \boldsymbol{b} \\ & \cancel{\boldsymbol{z} \in \mathbb{Z}^n} \end{aligned}$$

Efficiently solvable



LP provides valuable guidance in B&B

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

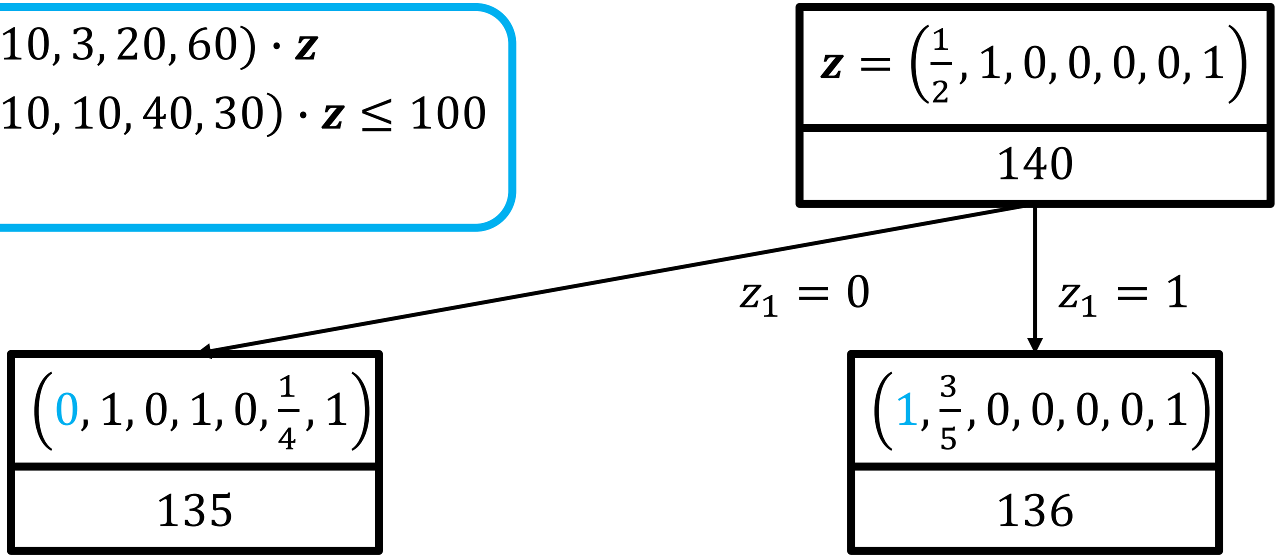
Branch and bound (B&B)

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

$\boldsymbol{z} = \left(\frac{1}{2}, 1, 0, 0, 0, 0, 1\right)$
140

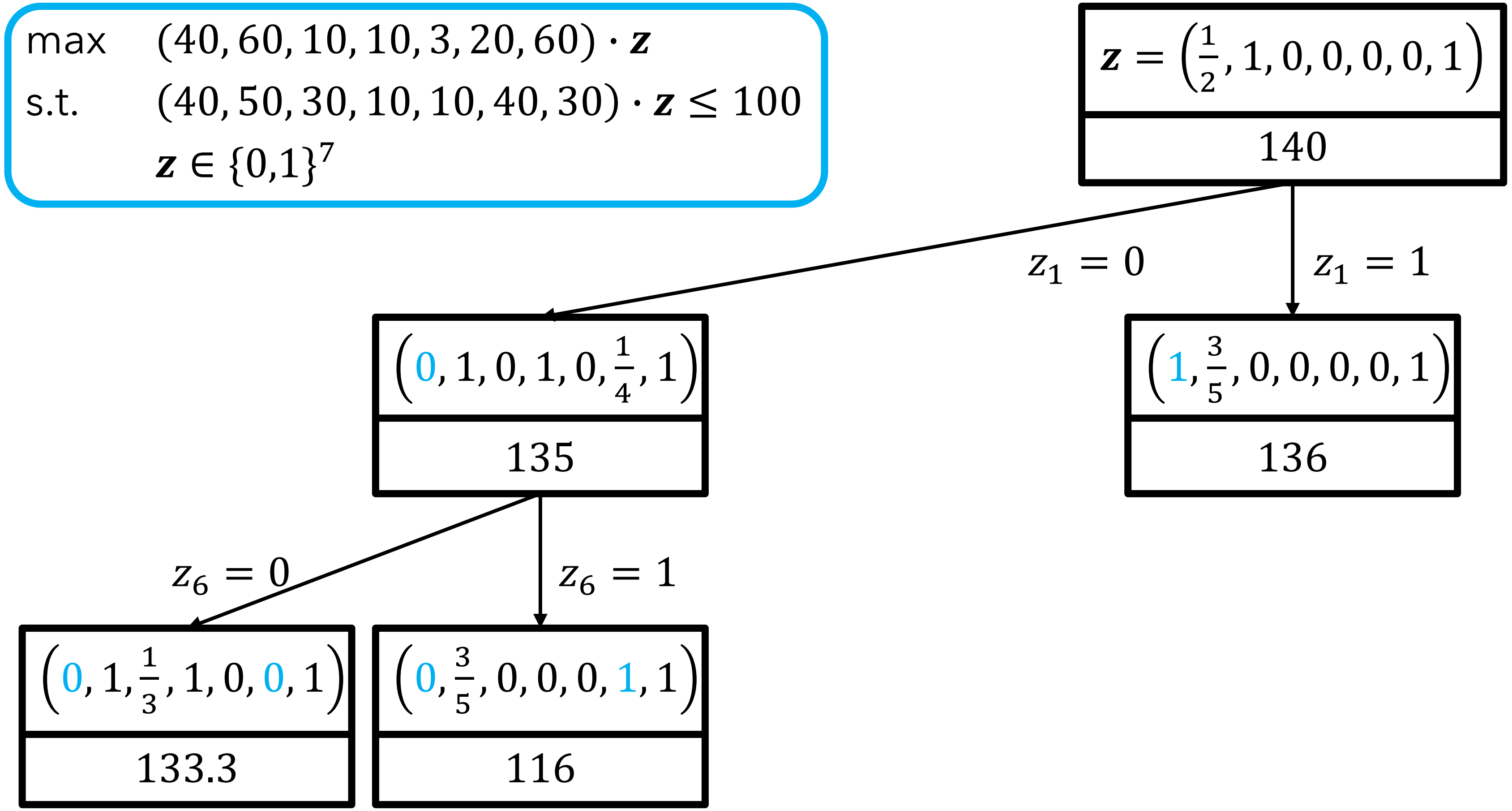
Branch and bound (B&B)

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



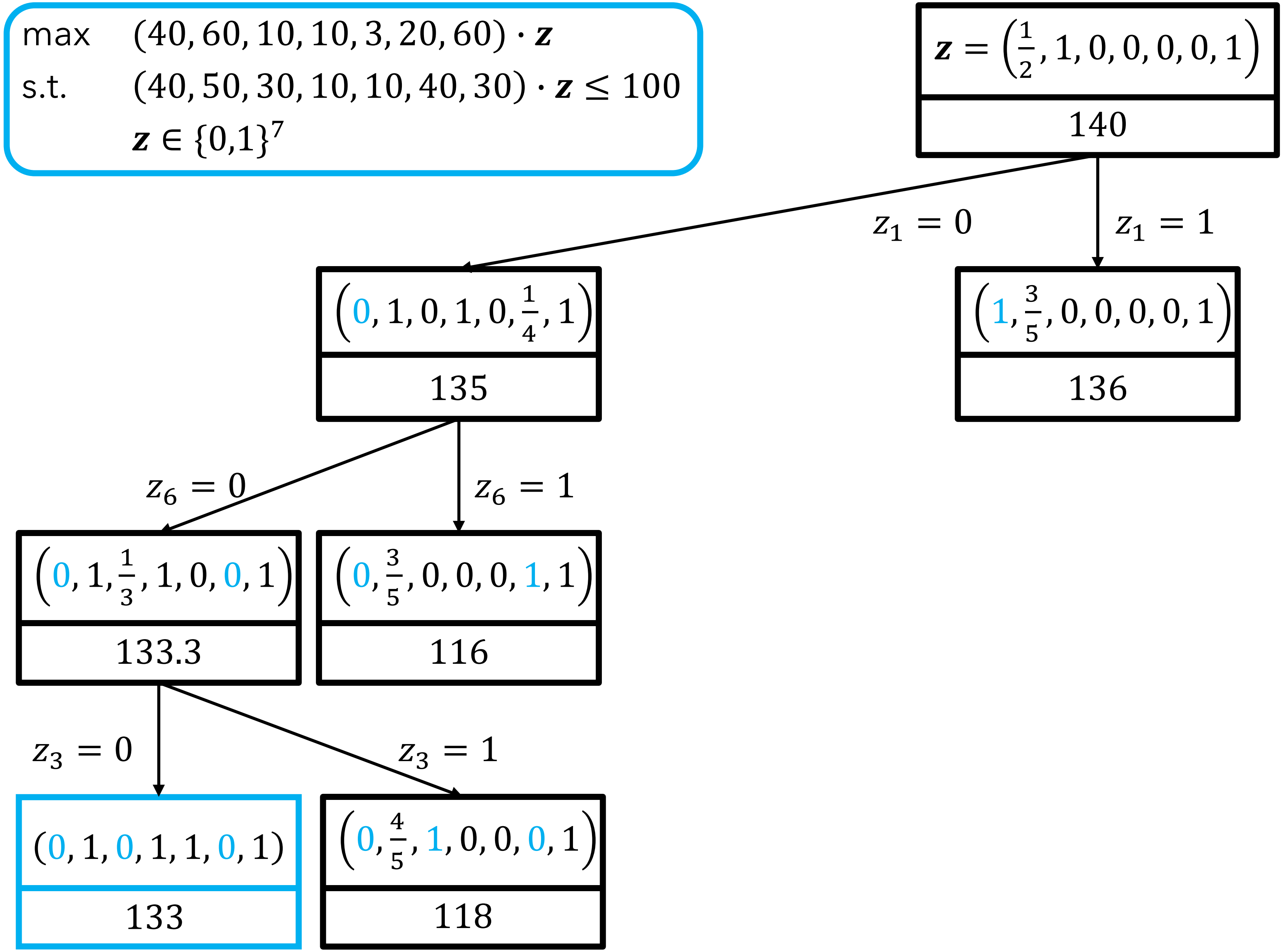
Branch and bound (B&B)

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



Branch and bound (B&B)

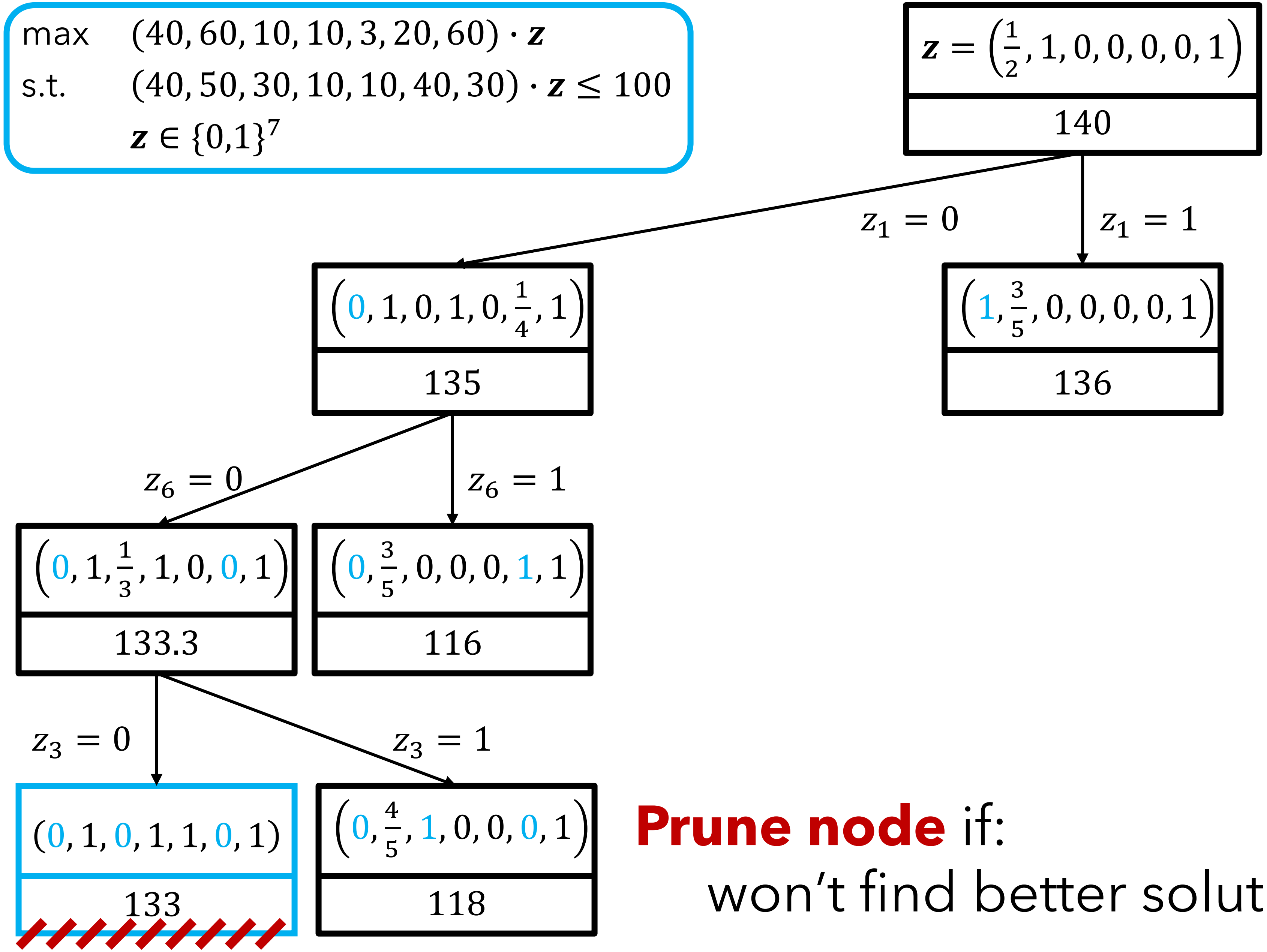
$$
\begin{aligned}
\text{max} \quad & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{z} \\
\text{s.t.} \quad & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{z} \leq 100 \\
& \boldsymbol{z} \in \{0,1\}^7
\end{aligned}
$$



Branch and bound (B&B)

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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



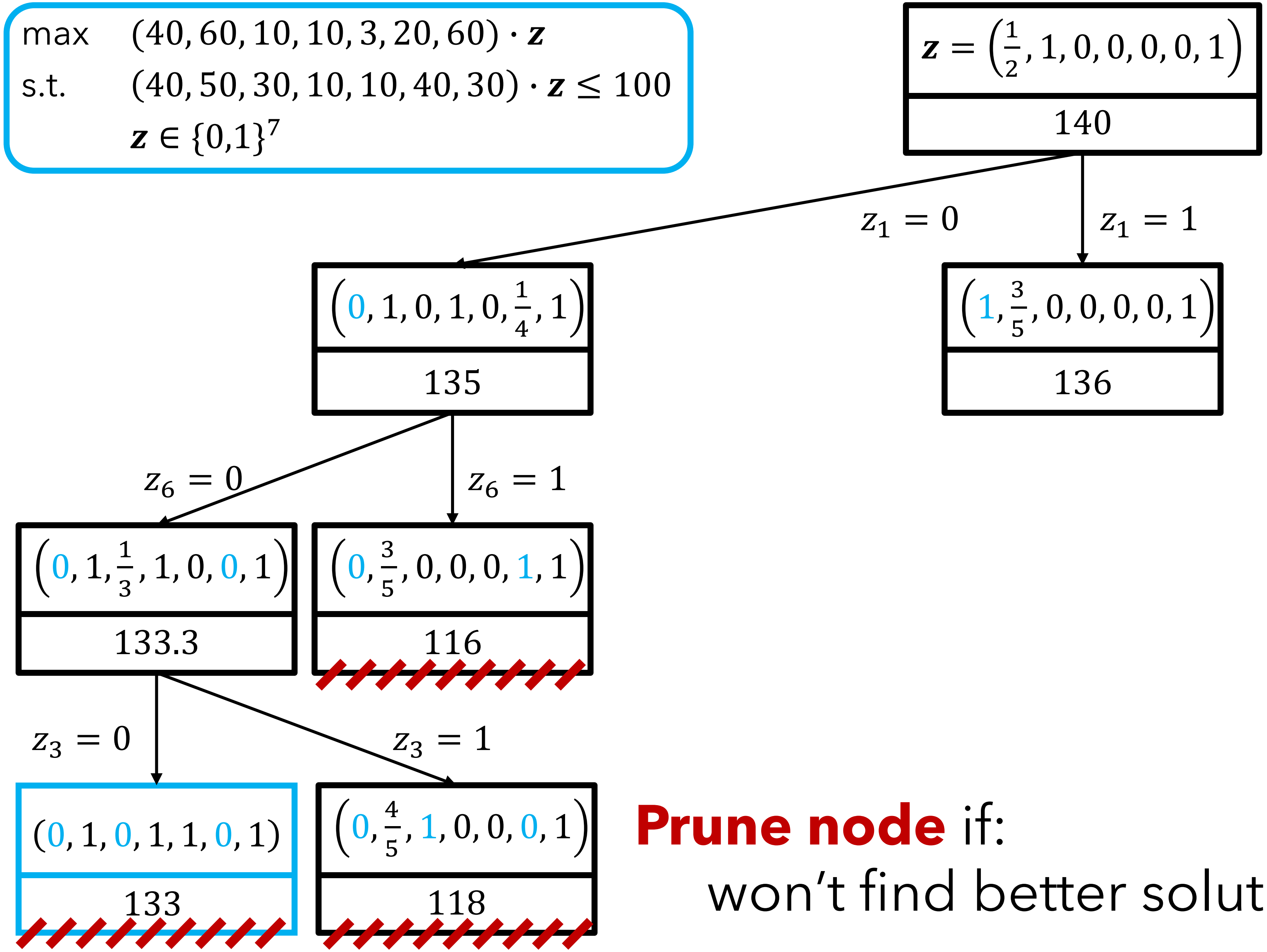
Branch and bound (B&B)

Prune node if:

won't find better solution along branch

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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



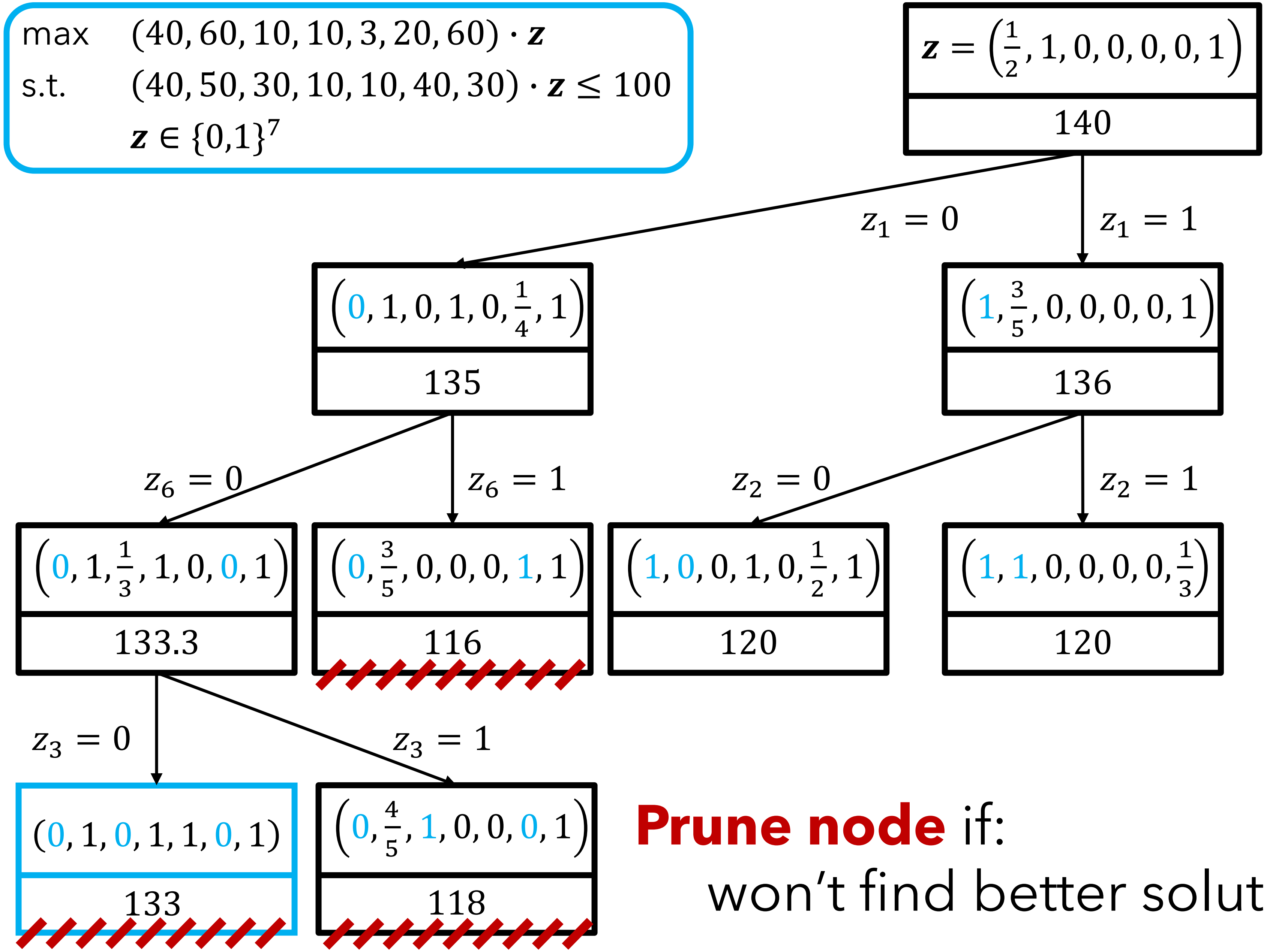
Branch and bound (B&B)

Prune node if:

won't find better solution along branch

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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



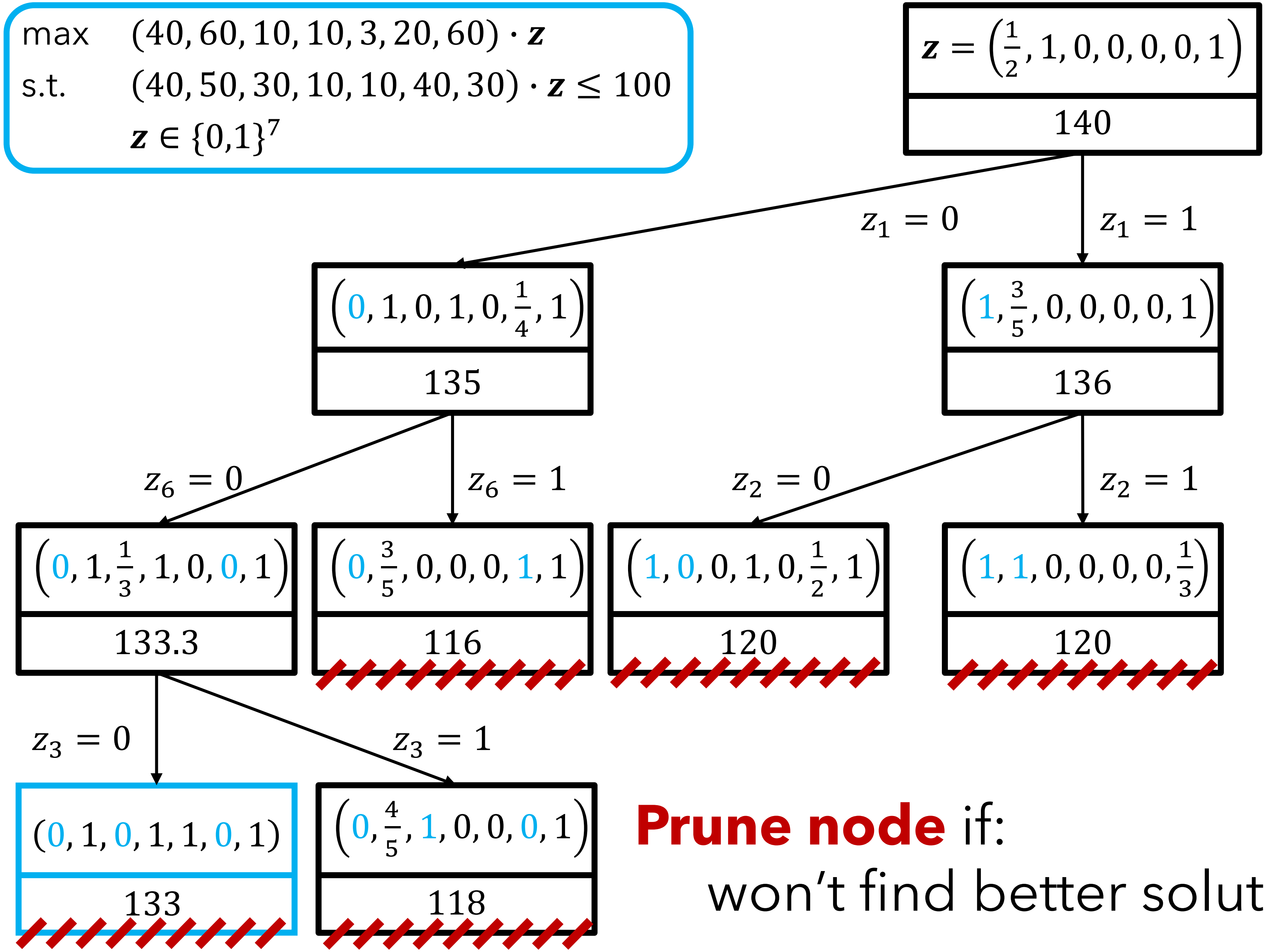
Branch and bound (B&B)

Prune node if:

won't find better solution along branch

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

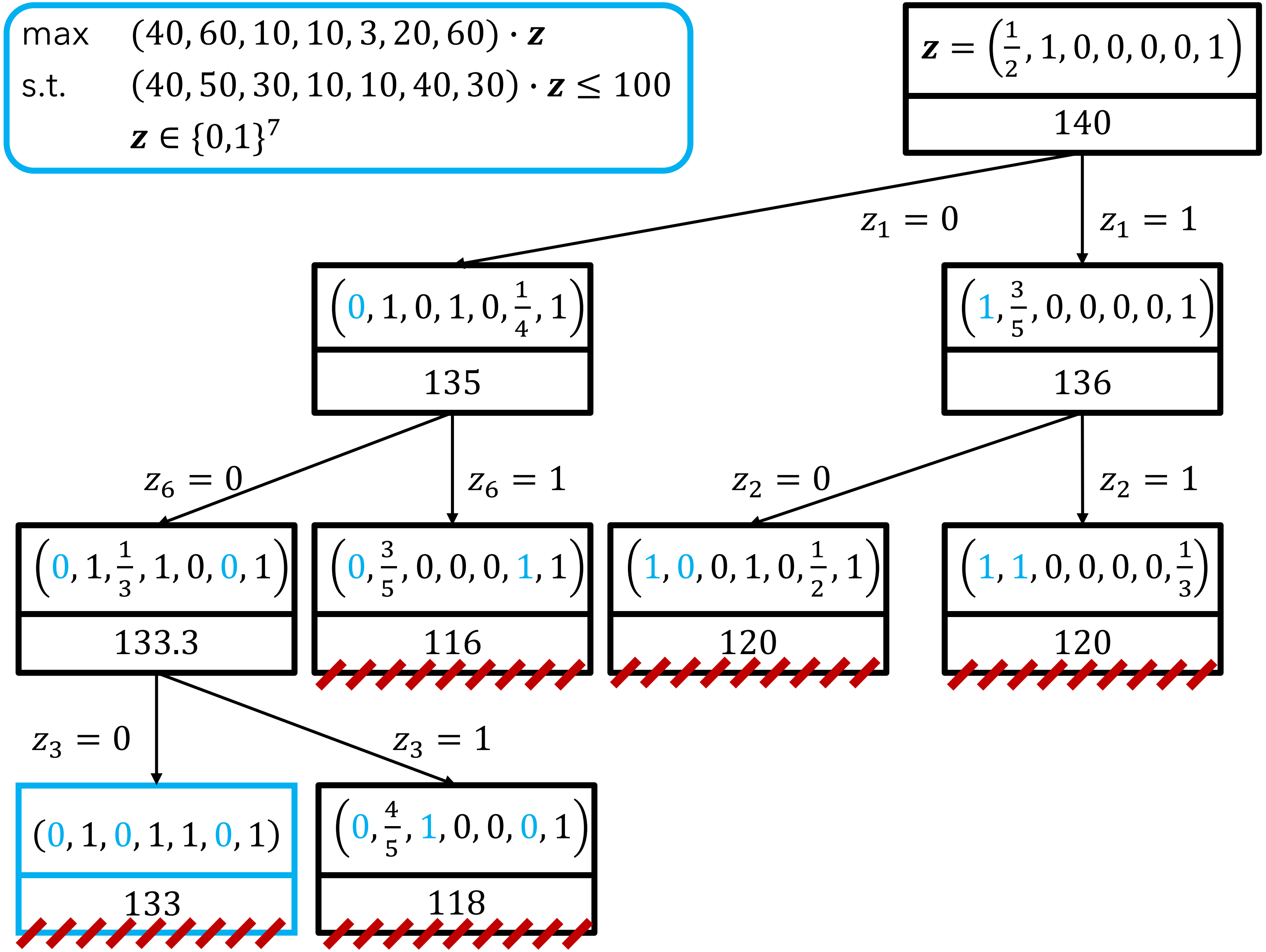


Branch and bound (B&B)

Prune node if:

won't find better solution along branch

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



This section: Variable selection

Variable selection policies (VSPs)

Score-based variable selection policies:

At leaf Q, branch on variable z_i maximizing $\mathbf{score}(\boldsymbol{Q}, \boldsymbol{i}) \in \mathbb{R}$

Variable selection policies (VSPs)

Score-based variable selection policies:

At leaf Q, branch on variable z_i maximizing $\mathbf{score}(\boldsymbol{Q}, \boldsymbol{i}) \in \mathbb{R}$

Many options! Little known about which to use when

Gauthier, Ribière, Math. Prog. '77; Beale, Annals of Discrete Math. '79; Linderoth, Savelsbergh, INFORMS JoC '99; Achterberg, Math. Prog. Computation '09; Gilpin, Sandholm, Disc. Opt. '11; …

Variable selection policy example

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

Variable selection policy example

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

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

Variable selection policy example

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 $x_i = 0$

Variable selection policy example

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

Variable selection policy example

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

$$\left(z(j) - z_i^+(j)\right)\left(z(j) - z_i^-(j)\right)$$

In more detail, scoring rule is $\max\{z(j) - z_i^+(j), 10^{-6}\} \cdot \max\{z(j) - z_i^-(j), 10^{-6}\}$:

If $z(j) - z_i^+(j) = 0$, 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

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

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***

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

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**

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

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

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

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]

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

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 section: using a GNN

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

Outline (applied techniques)

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

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

Problem formulation

Goal: learn a policy $\pi(x_i \mid s_t)$

Probability of branching on variable x_i when solver is in state s_t

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

Problem formulation

Goal: learn a policy $\underbrace{\pi(x_i \mid s_t)}$

Probability of branching on variable x_i when solver is in state s_t

Approach (imitation learning)**:**

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

Problem formulation

Goal: learn a policy $\underbrace{\pi(x_i \mid s_t)}_{\boxed{\text{Probability of branching on variable } x_i \text{ when solver is in state } s_t}}$

Approach (imitation learning)**:**

- Run strong branching on training set of instances

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

Problem formulation

Goal: learn a policy $\underbrace{\pi(x_i \mid s_t)}$

Probability of branching on variable x_i when solver is in state s_t

Approach (imitation learning)**:**

- Run strong branching on training set of instances
- Collect dataset of (state, variable) pairs $S = \{(s_i, x_{i^*})\}_{i=1}^N$

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

Problem formulation

Goal: learn a policy $\pi(x_i \mid s_t)$

Probability of branching on variable x_i when solver is in state s_t

Approach (imitation learning)**:**

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

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

State encoding

State s_t of B&B encoded as a **bipartite graph**

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

State encoding

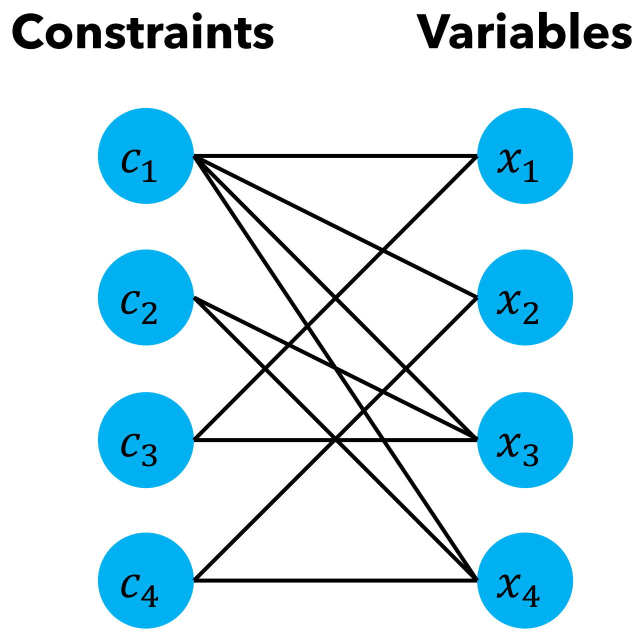
State s_t of B&B encoded as a **bipartite graph** with **node** and **edge features**

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

State encoding

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

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

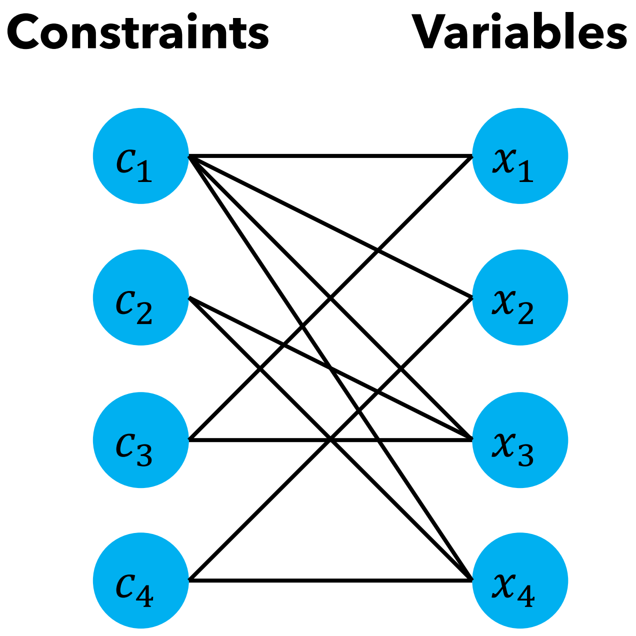


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

State encoding

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

- **Edge feature:** constraint coefficient

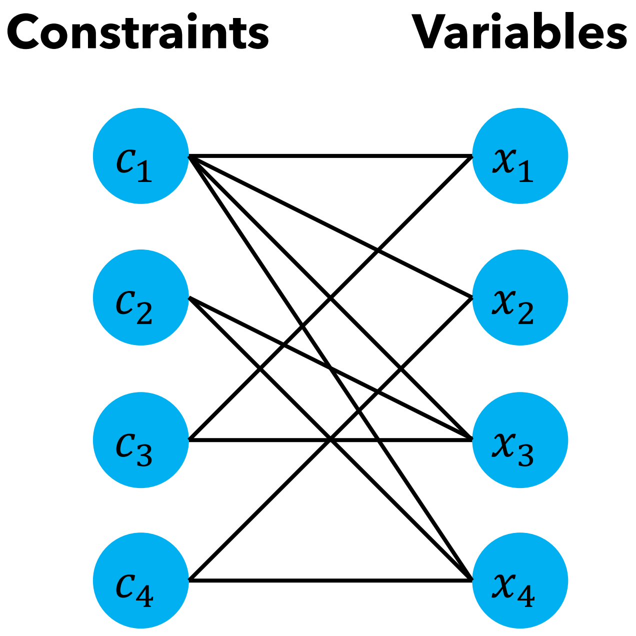


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

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:**

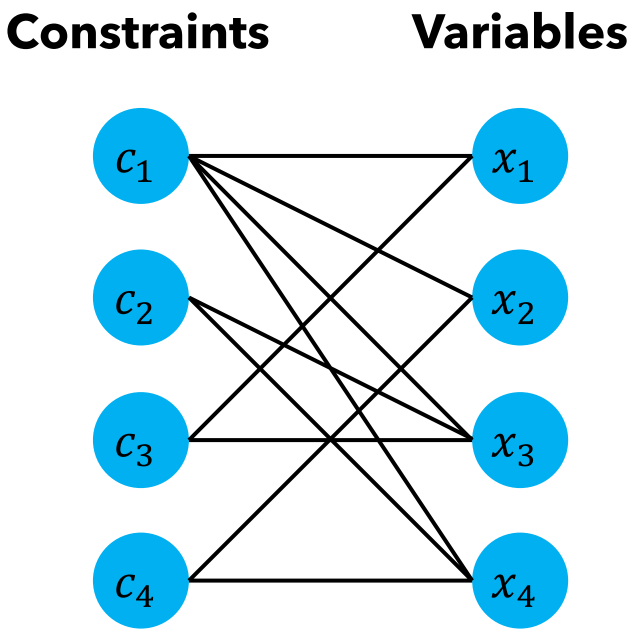


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

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

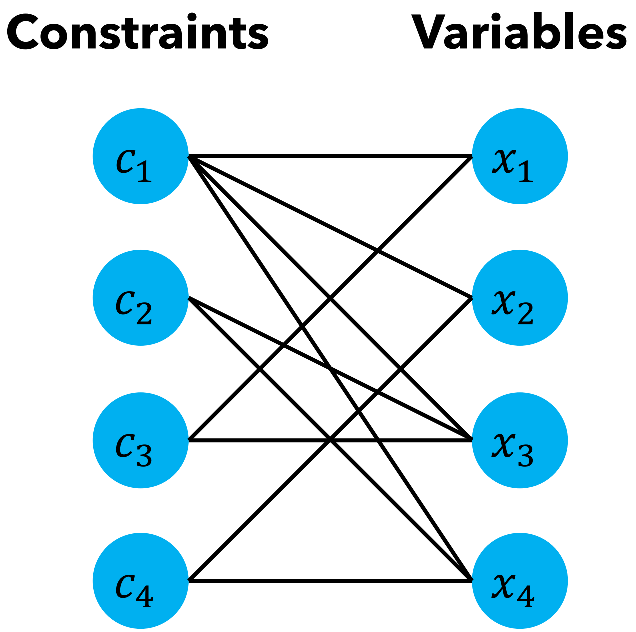


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

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?

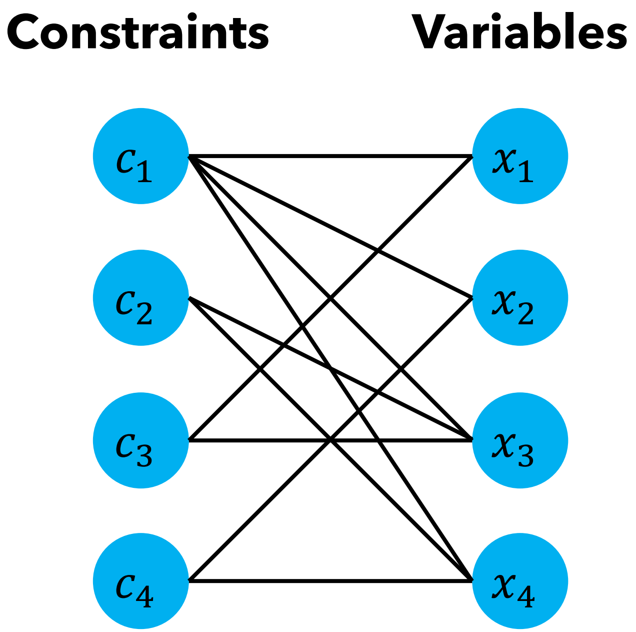


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

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

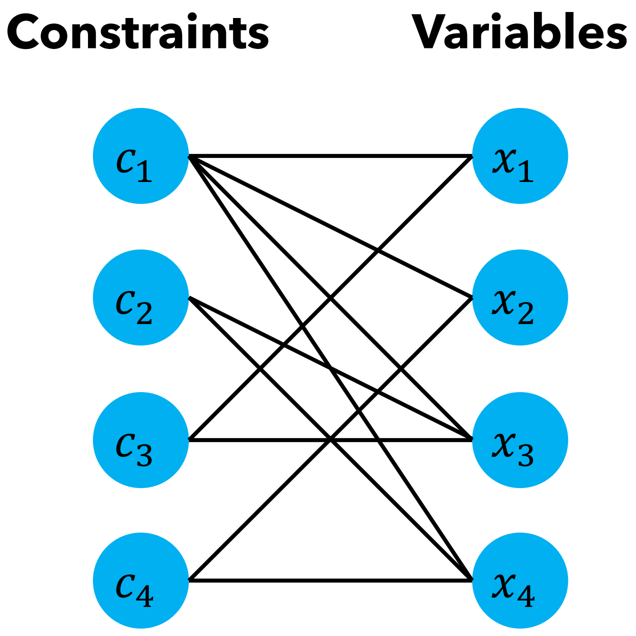


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

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?

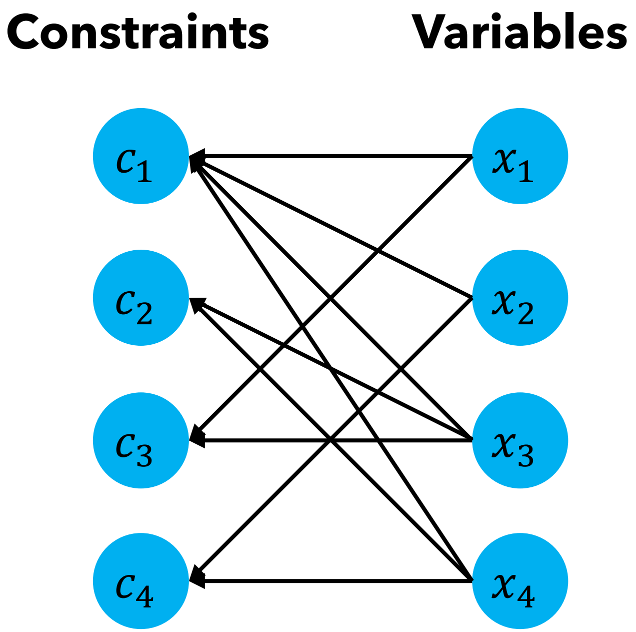


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

GNN structure

1. Pass from variables → constraints

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



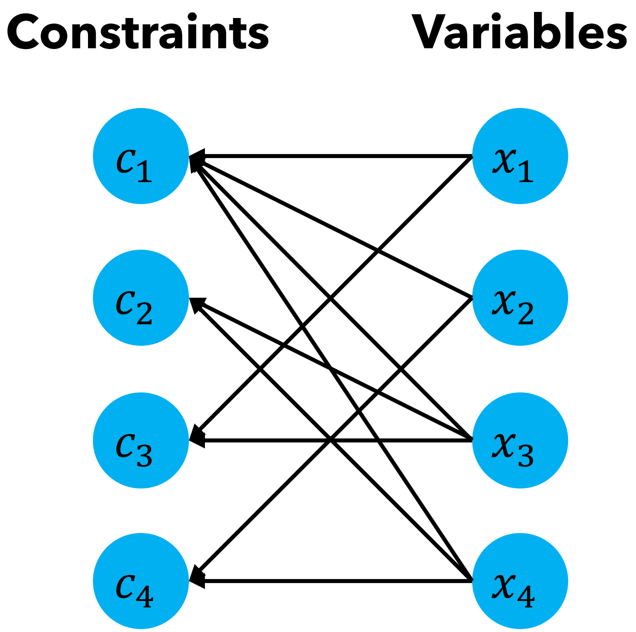
Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

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



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

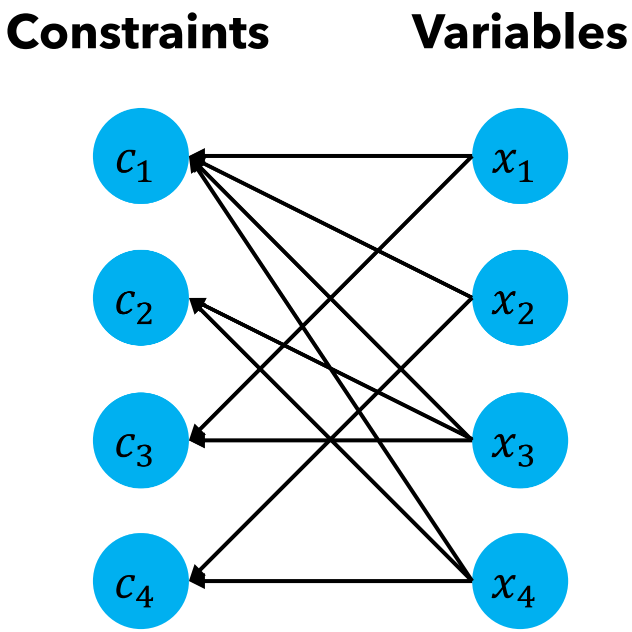
GNN structure

1. Pass from variables → constraints

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

Constraint features

2-layer MLP with relu activations



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

GNN structure

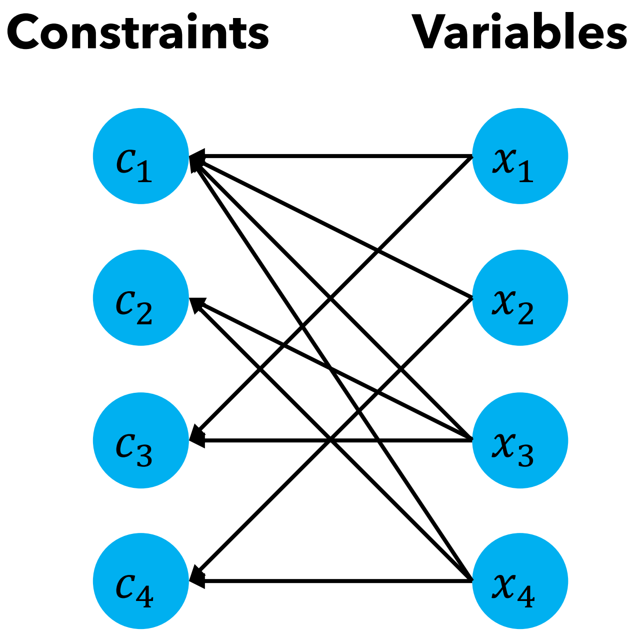
1. Pass from variables → constraints

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

Constraint features

2-layer MLP with relu activations

Variable features

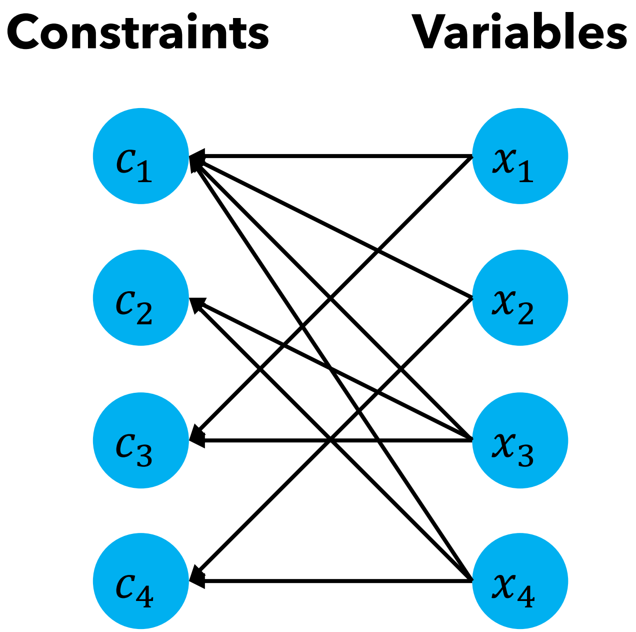


Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS'19

GNN structure

1. Pass from variables → constraints

$$\underbrace{\boldsymbol{c}_i}_{\text{Constraint features}} \leftarrow \underbrace{f_C}_{\text{2-layer MLP with relu activations}}\left(\boldsymbol{c}_i, \sum_{j:(i,j)\in E} g_C\left(\boldsymbol{c}_i, \underbrace{\boldsymbol{v}_j}_{\text{Variable features}}, \underbrace{\boldsymbol{e}_{ij}}_{\text{Edge features}}\right)\right)$$



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

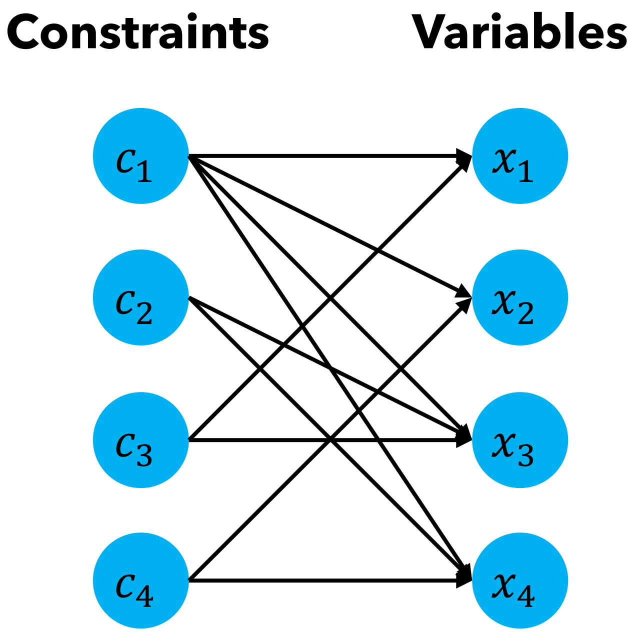
GNN structure

1. Pass from variables → constraints

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

2. Pass from constraints → variables

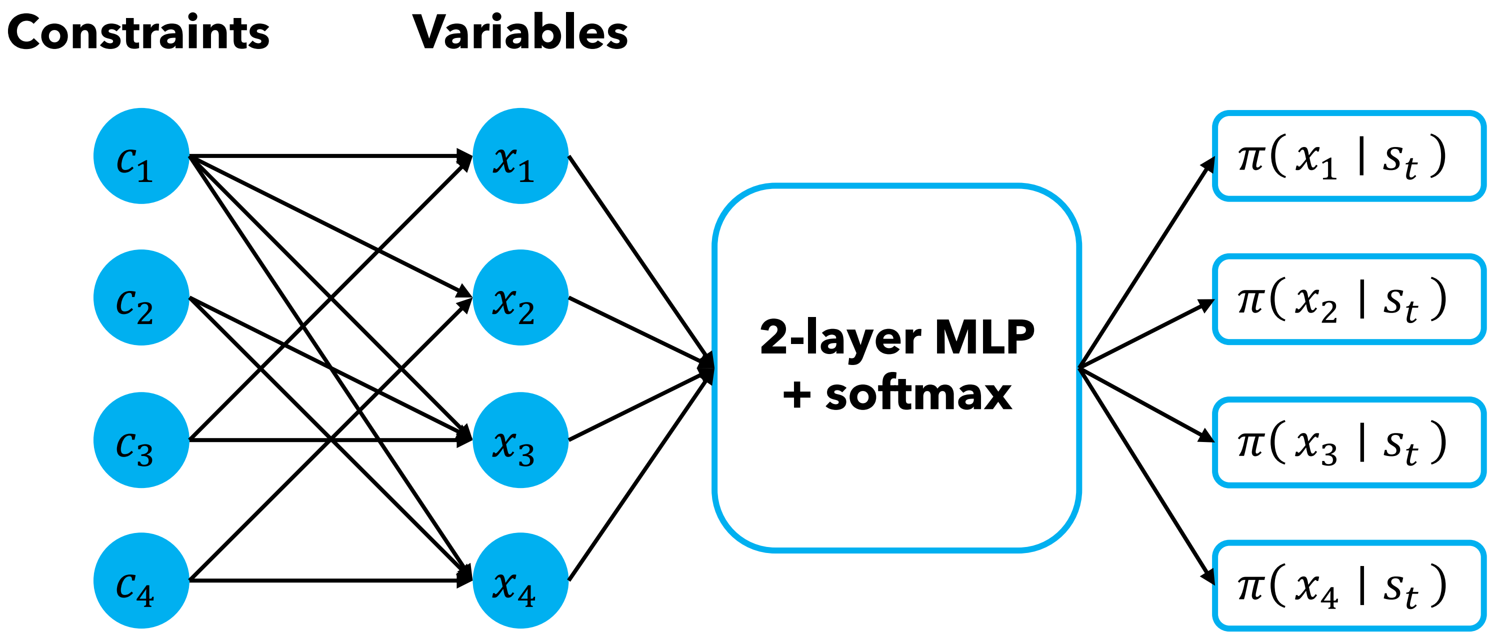
$$\boldsymbol{v}_j \leftarrow f_V\left(\boldsymbol{v}_j, \sum_{i:(i,j)\in E} g_V\left(\boldsymbol{c}_i, \boldsymbol{v}_j, \boldsymbol{e}_{ij}\right)\right)$$



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

GNN structure

3. Compute distribution over variables



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

Outline (applied techniques)

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

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

Reliability pseudo-cost branching (RPB)

Rough idea:

- Goal: estimate $z(j) - z_i^+(j)$ w/o solving the LP with $x_i = 1$

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

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

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

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"

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

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"

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

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

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

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

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

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):

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

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):

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

Technically,
$\max\{\widetilde{\Delta}_i^+(j), 10^{-6}\} \cdot \max\{\widetilde{\Delta}_i^-(j), 10^{-6}\}$

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

Learning to rank approaches

Predict which variable **strong branching** would rank highest using models other than GNNs

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

Learning to rank approaches

Predict which variable **strong branching** would rank highest using models other than GNNs

- Khalil et al. [AAAI'16]:
  Use learning-to-rank algorithm **SVM$^{\text{rank}}$** [Joachims, KDD'06]

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

Learning to rank approaches

Predict which variable **strong branching** would rank highest using models other than GNNs

- 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]

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

Learning to rank approaches

Predict which variable **strong branching** would rank highest using models other than GNNs

- 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]
- Alvarez et al. [INFORMS JoC'17]:
  Use **regression trees**

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

Outline (applied techniques)

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

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

Set covering instances

Train and test on “easy” instances: 1000 columns, 500 rows

Model	**Time**	**Wins**	**Nodes**
Full strong branching	17.30±6.1%	0/100	17±13.7%
Reliability pseudo-cost	8.98±4.8%	0/100	**54**±20.8%
Regression trees	9.28±4.9%	0/100	187±9.4%
SVMrank	8.10±3.8%	1/100	165±8.2%
lambdaMART	7.19±4.2%	14/100	167±9.0%
GNN	**6.59**±3.1%	**85**/100	134±7.6%

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

Set covering instances

Train and test on "easy" instances: 1000 columns, 500 rows

Runtime in seconds with a timeout of 1 hour

Model	Time	Wins	Nodes
[illegible]	[illegible]±6.1%	0/100	17±13.7%
Reliability pseudo-cost	8.98±4.8%	0/100	**54**±20.8%
Regression trees	9.28±4.9%	0/100	187±9.4%
SVMrank	8.10±3.8%	1/100	165±8.2%
lambdaMART	7.19±4.2%	14/100	167±9.0%
GNN	**6.59**±3.1%	**85**/100	134±7.6%

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

Set covering instances

Train and test on “easy” instances: 1000 columns, 500 rows

Runtime in seconds with a timeout of 1 hour

Number instances with fastest runtime / number solved

Model	**Time**	**Wins**	**Nodes**
[illegible]	[illegible]±6.1%	0/100	17±13.7%
Reliability [illegible]	[illegible]	[illegible]	**54**±20.8%
Regression trees	9.28±4.9%	0/100	187±9.4%
SVMrank	8.10±3.8%	1/100	165±8.2%
lambdaMART	7.19±4.2%	14/100	167±9.0%
GNN	**6.59**±3.1%	**85**/100	134±7.6%

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

Set covering instances

Train and test on "easy" instances: 1000 columns, 500 rows

Model	**Time**	**Wins**	**Nodes**
[illegible]	[illegible]±6.1%	0/100	17±13.7%
Reliability [illegible]	[illegible]	[illegible]	**54**±20.8%
Regression trees	9.28±4.9%	0/10[illegible]	[illegible]4%
SVMrank	8.10±3.8%	1/100	165±8.2%
lambdaMART	7.19±4.2%	14/100	167±9.0%
GNN	**6.59**±3.1%	**85**/100	134±7.6%

Time: Runtime in seconds with a timeout of 1 hour

Wins: Number instances with fastest runtime / number solved

Nodes: Size of B&B tree

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

Set covering instances

Train and test on "easy" instances: 1000 columns, 500 rows

Model	Time	Wins	Nodes
Full strong branching	17.30±6.1%	0/100	17±13.7%
Reliability pseudo-cost	8.98±4.8%	0/100	**54**±20.8%
Regression trees	9.28±4.9%	0/100	187±9.4%
SVMrank	8.10±3.8%	1/100	165±8.2%
lambdaMART	7.19±4.2%	14/100	167±9.0%
GNN	**6.59**±3.1%	**85**/100	134±7.6%

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

Set covering instances

GNN is **faster than SCIP** default VSP (reliability pseudo-cost)

Model	Time	Wins	Nodes
Full strong branching	17.30±6.1%	0/100	17±13.7%
Reliability pseudo-cost	8.98±4.8%	0/100	**54**±20.8%
Regression trees	9.28±4.9%	0/100	187±9.4%
SVMrank	8.10±3.8%	1/100	165±8.2%
lambdaMART	7.19±4.2%	14/100	167±9.0%
GNN	**6.59**±3.1%	**85**/100	134±7.6%

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

Set covering instances

Train: “easy”; test: “**hard**” instances w/ 1000 columns, 2000 rows

Model	**Time**	**Wins**	**Nodes**
Full strong branching	Timed out	0/0	N/A
Reliability pseudo-cost	1677.98±3.0%	4/65	47299±4.9%
Regression trees	2869.21±3.2%	0/35	59013±9.3%
SVMrank	2389.92±2.3%	0/47	42120±5.4%
lambdaMART	2165.96±2.0%	0/54	45319±3.4%
GNN	**1489.91**±3.3%	**66/70**	**29981**±4.9%

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

Set covering instances

Performance generalizes to **larger instances**

Model	Time	Wins	Nodes
Full strong branching	Timed out	0/0	N/A
Reliability pseudo-cost	1677.98±3.0%	4/65	47299±4.9%
Regression trees	2869.21±3.2%	0/35	59013±9.3%
SVMrank	2389.92±2.3%	0/47	42120±5.4%
lambdaMART	2165.96±2.0%	0/54	45319±3.4%
GNN	**1489.91**±3.3%	**66/70**	**29981**±4.9%

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

Set covering instances

Similar results for auction design & facility location problems

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

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
   i. Machine learning formulation
   ii. Baselines
   iii. Experiments
   iv. Additional research
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL

Additional research

CPU-friendly approaches

Gupta et al., NeurIPS'20

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; …

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*:

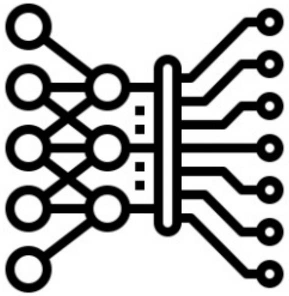
Cappart, Chételat, Khalil, Lodi, Morris, Veličković, JMLR'23

Outline (applied techniques)

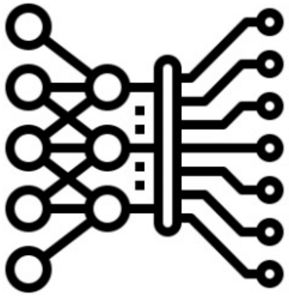
1. GNNs overview
2. Integer programming with GNNs
3. **Neural algorithmic alignment**
4. Learning greedy heuristics with RL

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20
Cappart, Chételat, Khalil, Lodi, Morris, Veličković, JMLR'23

Problem-solving approaches

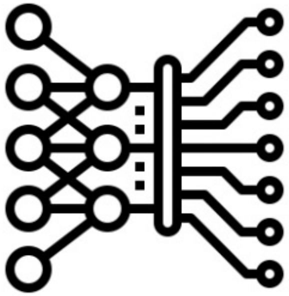


Problem-solving approaches



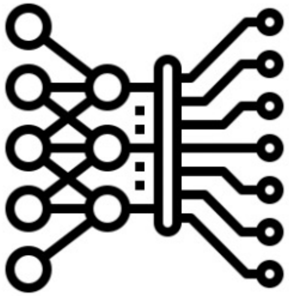
+ Operate on raw inputs

Problem-solving approaches



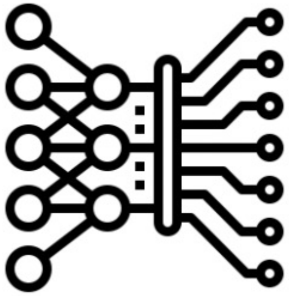
- + Operate on raw inputs
- + Generalize on noisy conditions

Problem-solving approaches



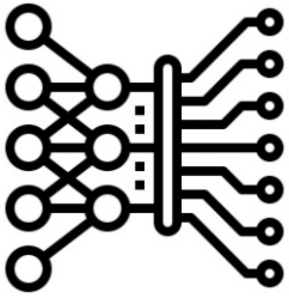
+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks

Problem-solving approaches



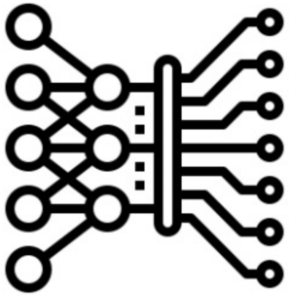
- \+ Operate on raw inputs
- \+ Generalize on noisy conditions
- \+ Models reusable across tasks
- \- Require big data

Problem-solving approaches



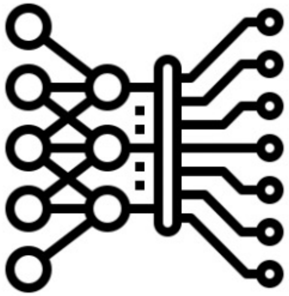
+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating

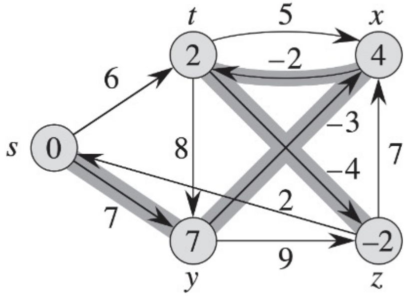
Problem-solving approaches



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

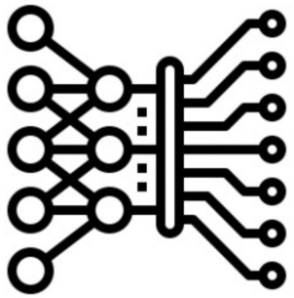
Problem-solving approaches

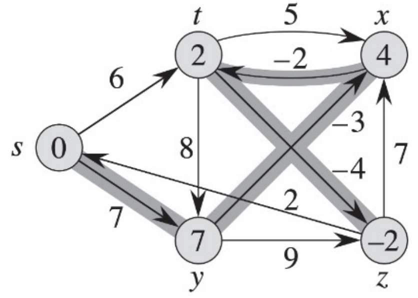




- \+ Operate on raw inputs
- \+ Generalize on noisy conditions
- \+ Models reusable across tasks
- \- Require big data
- \- Unreliable when extrapolating
- \- Lack of interpretability

Problem-solving approaches

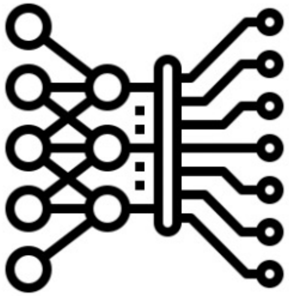


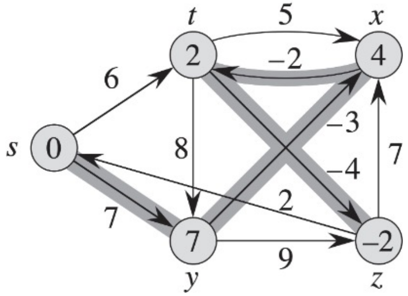


+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

+ Trivially strong generalization

Problem-solving approaches

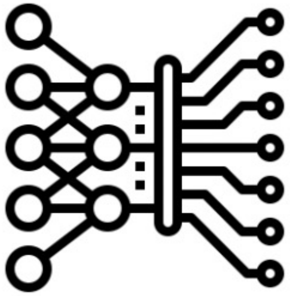




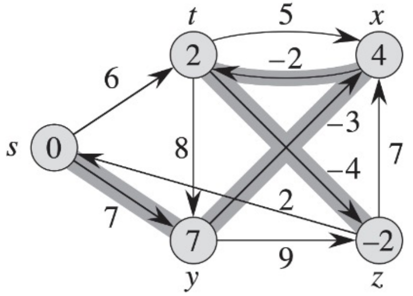
+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

+ Trivially strong generalization
+ Compositional (subroutines)

Problem-solving approaches

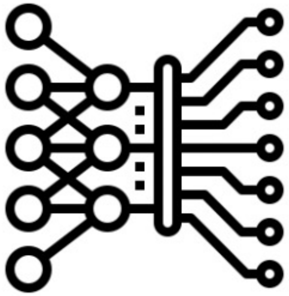


+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

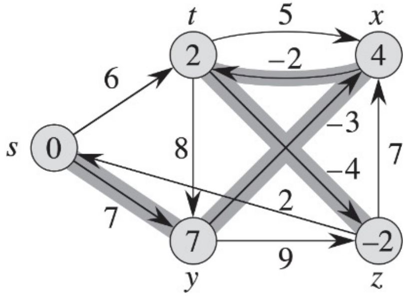


+ Trivially strong generalization
+ Compositional (subroutines)
+ Guaranteed correctness

Problem-solving approaches

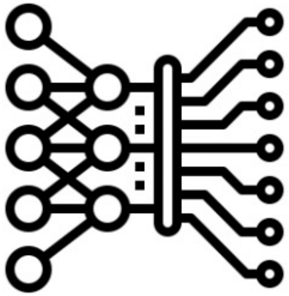


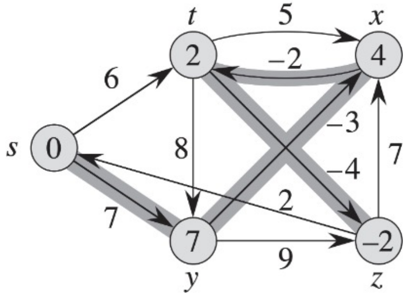
+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability



+ Trivially strong generalization
+ Compositional (subroutines)
+ Guaranteed correctness
+ Interpretable operations

Problem-solving approaches

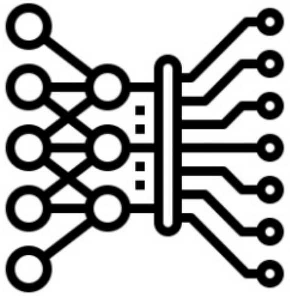


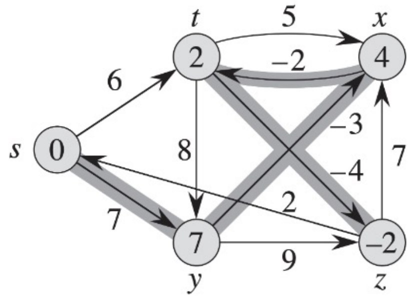


+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

+ Trivially strong generalization
+ Compositional (subroutines)
+ Guaranteed correctness
+ Interpretable operations
- Input must match spec

Problem-solving approaches

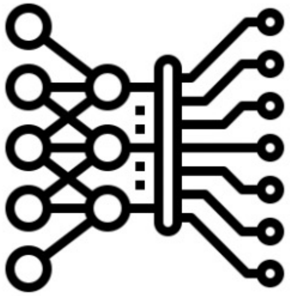


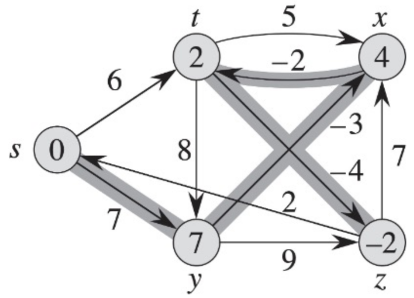


+ Operate on raw inputs
+ Generalize on noisy conditions
+ Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

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

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?

GNNs + combinatorial optimization

Lots of awesome research! E.g.,

Traveling salesman problem

E.g., Vinyals et al., ‘15; Joshi et al., ’19; …

$$(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3)$$

Boolean satisfiability

E.g., Selsam et al., ‘19; Cameron et al., ’20; …

GNNs + combinatorial optimization

Lots of awesome research! E.g.,

Traveling salesman problem

E.g., Vinyals et al., '15; Joshi et al., '19; …

$(x_1 \vee x_2)$

$\wedge$

$(x_1 \vee \bar{x}_3)$

Boolean satisfiability

E.g., Selsam et al., '19; Cameron et al., '20; …

This section: Neural graph algorithm execution

Aligns well with theoretical sections of this tutorial

Neural graph algorithm execution

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

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

Neural graph algorithm execution

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

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

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

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**

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

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**

If we already have a classical algorithm for the problem…

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

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**

If we already have a classical algorithm for the problem…

Why not just run that algorithm?

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

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**

If we already have a classical algorithm for the problem…

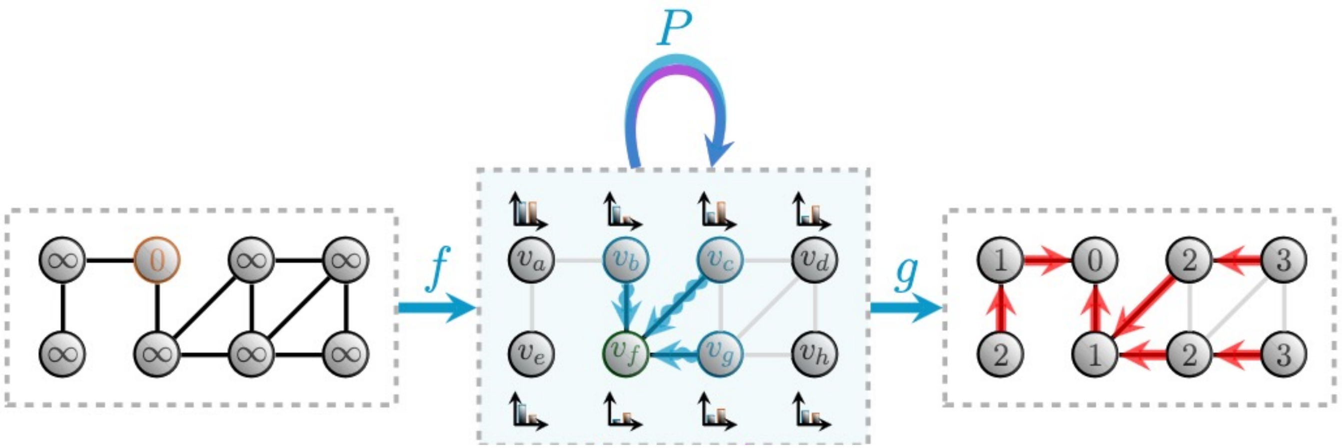
Why not just run that algorithm?

Will answer soon, but first: a few words on the pipeline

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

Neural algorithmic pipeline

Input $\boldsymbol{x}^{(1)}$

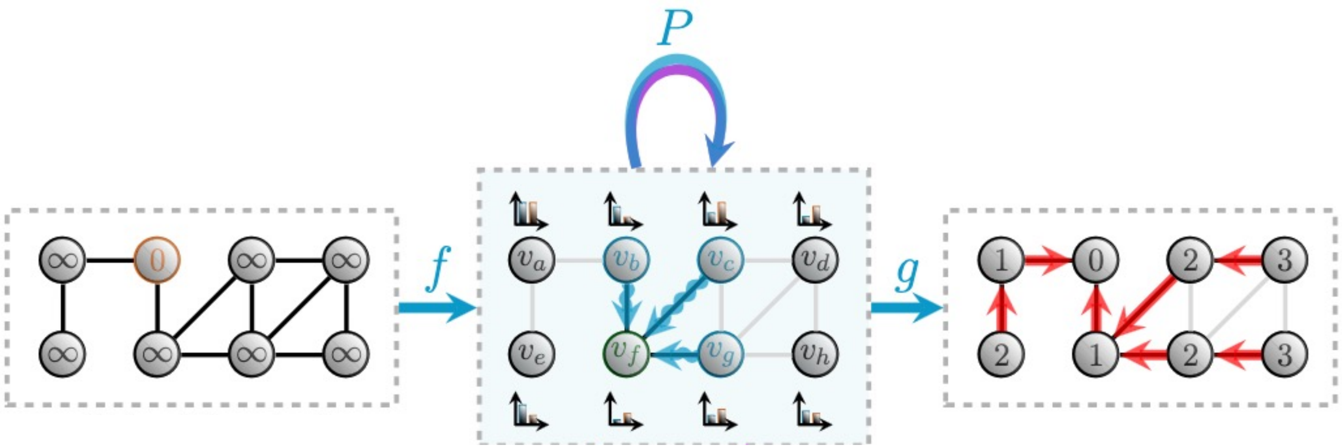


Output $\boldsymbol{y}^{(T)}$

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

Neural algorithmic pipeline

Input $\boldsymbol{x}^{(1)}$



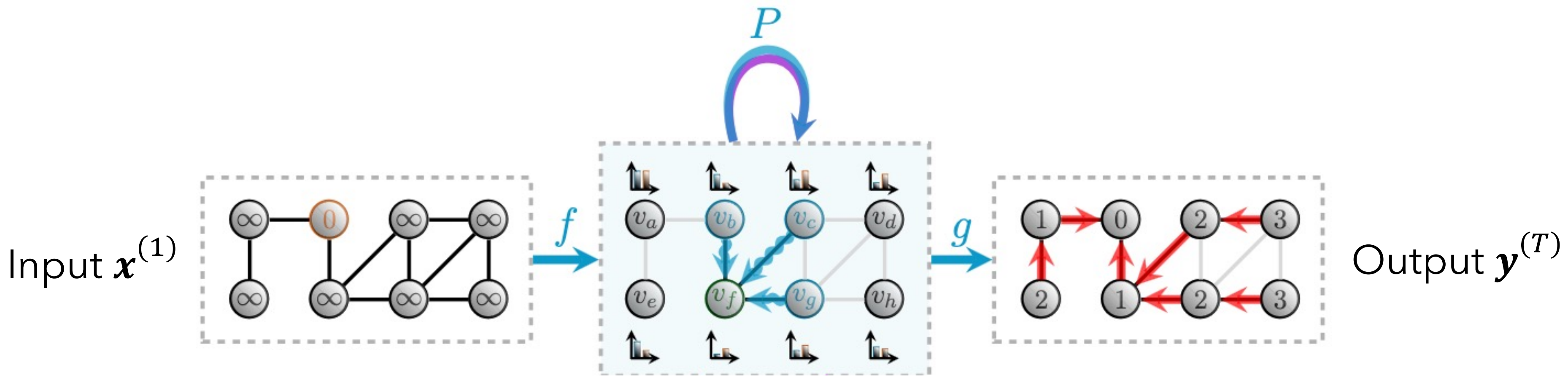
Output $\boldsymbol{y}^{(T)}$

Encoder network f

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

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

Neural algorithmic pipeline



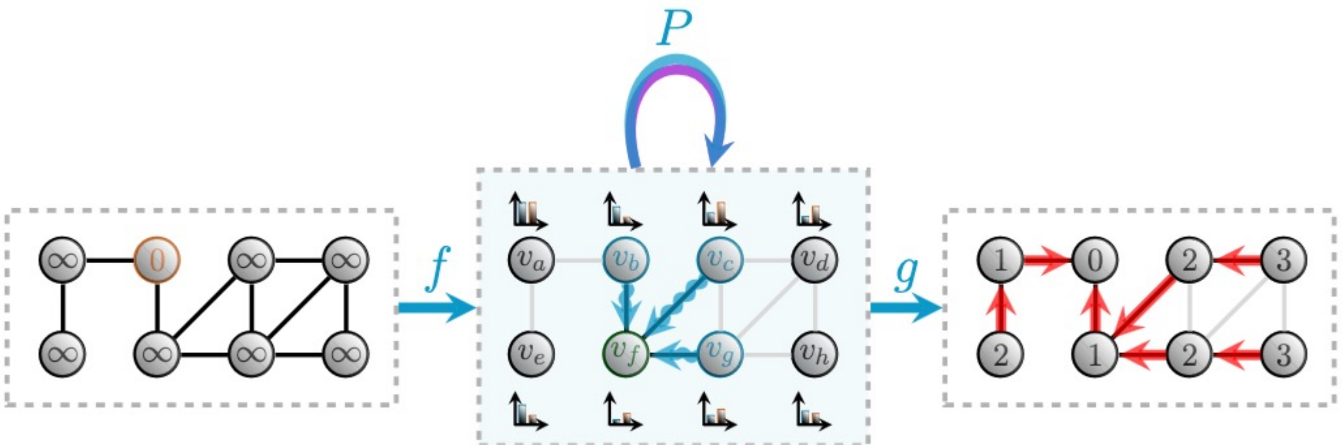
Processor network P

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

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

Neural algorithmic pipeline

Input $\boldsymbol{x}^{(1)}$



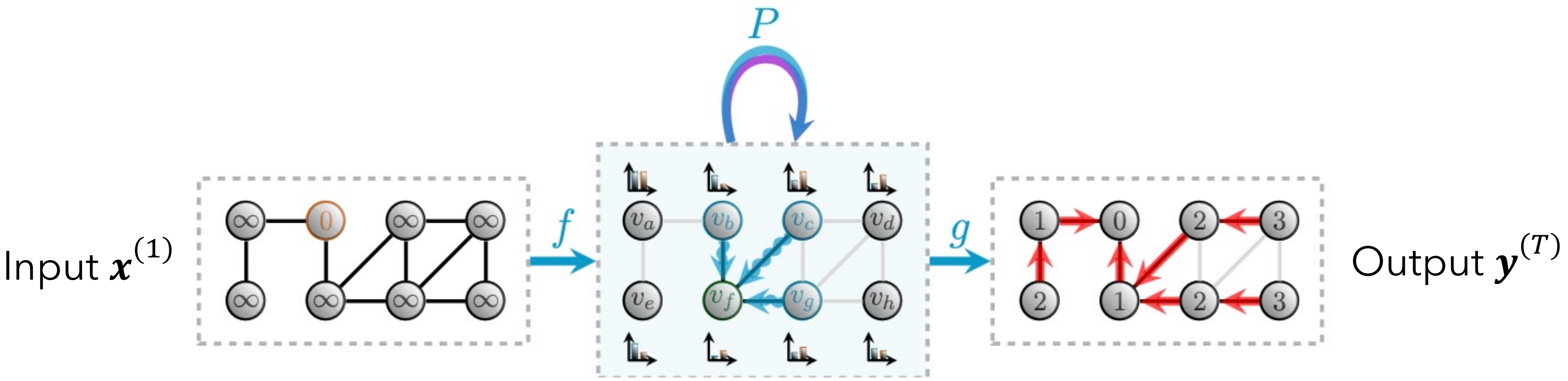
Output $\boldsymbol{y}^{(T)}$

Decoder network $\boldsymbol{g}$

- Transform's GNNs output into algorithmic output

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

Neural algorithmic pipeline



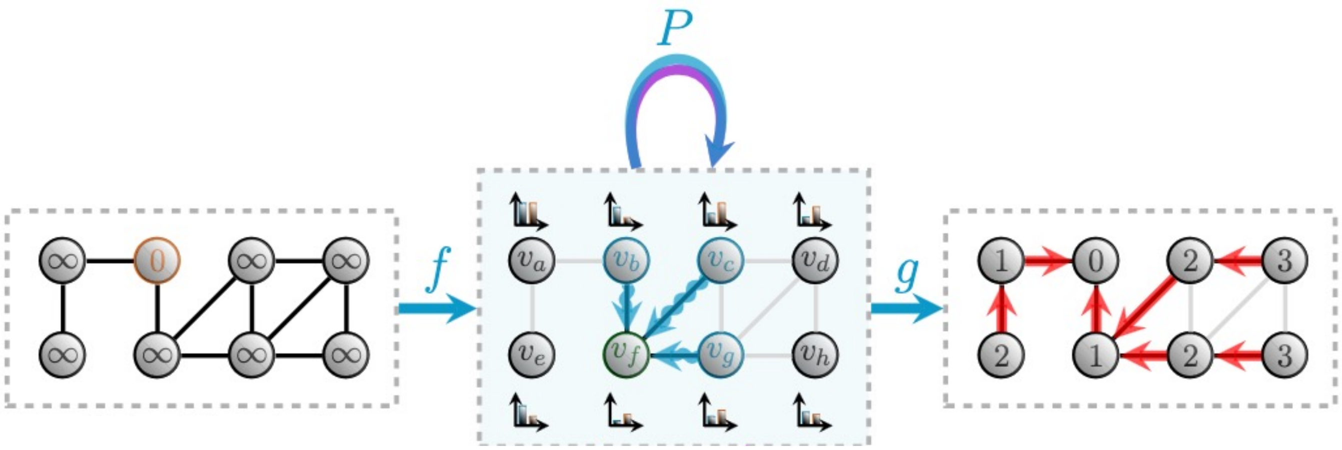
Multi-task approach

- Learn a **single** processor network P for related problems

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

Neural algorithmic pipeline

Input $\boldsymbol{x}^{(1)}$



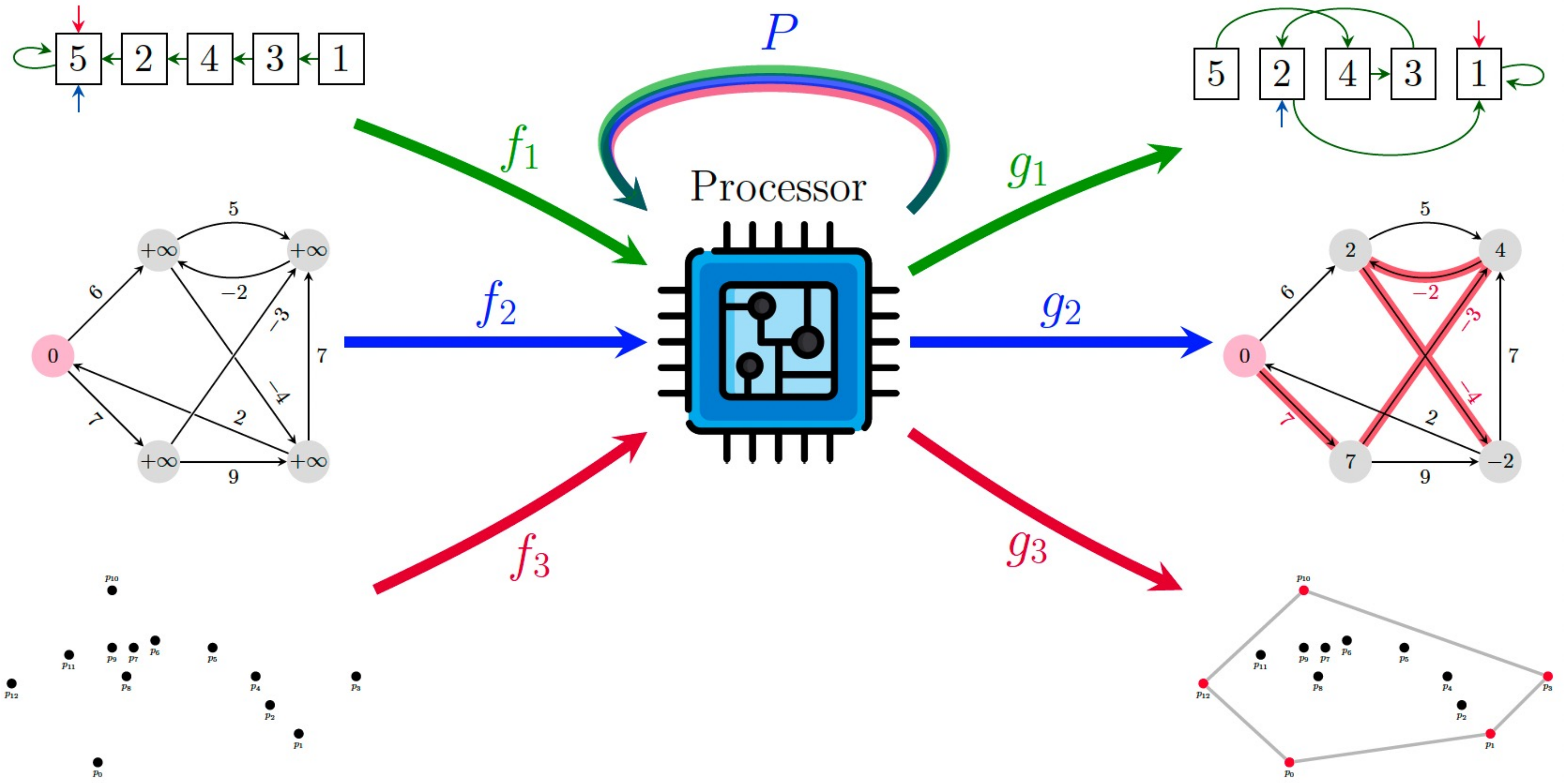
Output $\boldsymbol{y}^{(T)}$

Multi-task approach

- Learn a **single** processor network P for related problems
- Learn **task-specific** encoder, decoder functions f_A, g_A

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

Neural algorithmic pipeline



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Figure by Ibarz et al., LoG'22

Why use GNNs for algorithm design?

Why use GNNs for algorithm design?

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

Why use GNNs for algorithm design?

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

Why use GNNs for algorithm design?

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

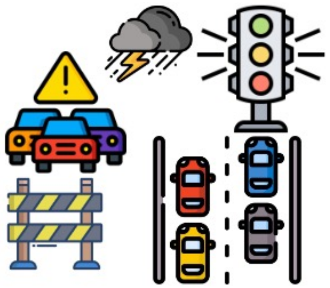
- Enforce their inputs to conform to stringent preconditions

Why use GNNs for algorithm design?

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

Enforce their inputs to conform to stringent preconditions

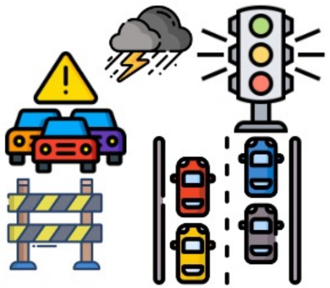
However, we design algorithms to solve **real-world** problems!



Natural inputs

Why use GNNs for algorithm design?

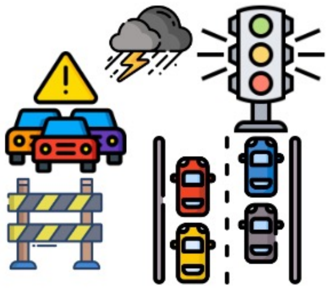
- Assume we have real-world inputs



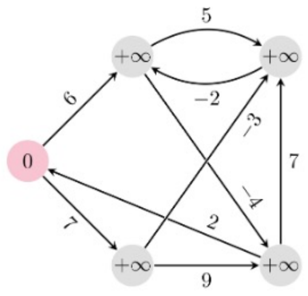
Natural inputs

Why use GNNs for algorithm design?

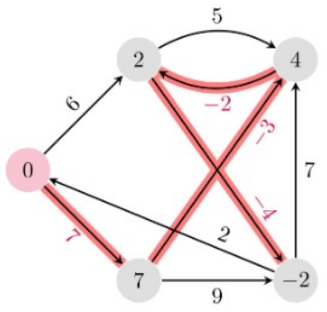
- Assume we have real-world inputs
  …but algorithm only admits abstract inputs



Natural inputs



Abstract inputs



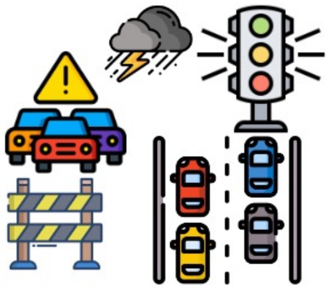
Abstract outputs

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

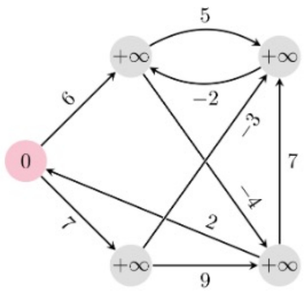
Slide by Veličković

Why use GNNs for algorithm design?

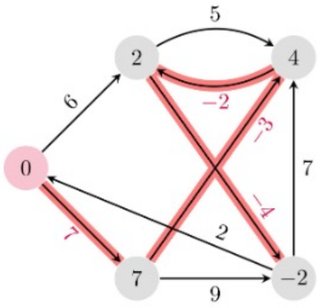
- Assume we have real-world inputs
  …but algorithm only admits abstract inputs
- Could try **manually** converting from one input to another



Natural inputs



Abstract inputs

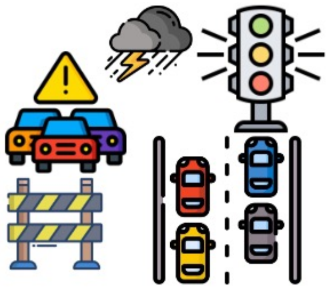


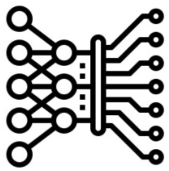
Abstract outputs

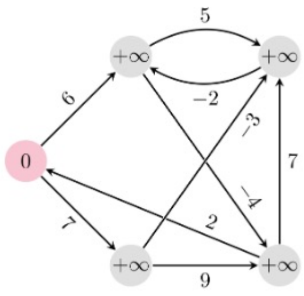
Slide by Veličković

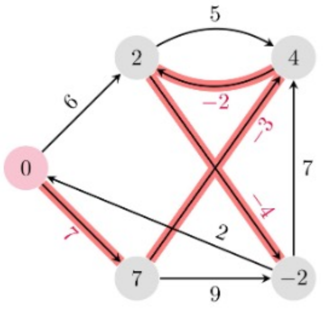
Why use GNNs for algorithm design?

- Alternatively, **replace** human feature extractor with NN





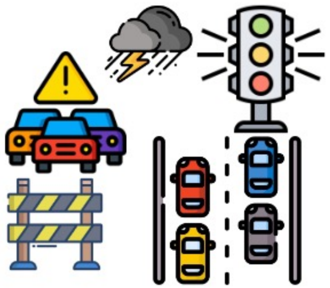




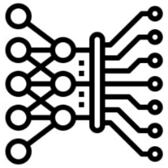
Natural inputs → **Abstract inputs** → **Abstract outputs**

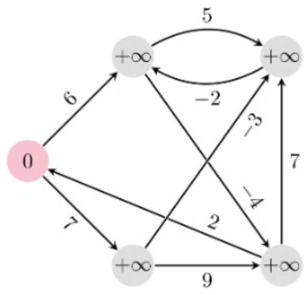
Why use GNNs for algorithm design?

- Alternatively, **replace** human feature extractor with NN
  - Still apply same combinatorial algorithm

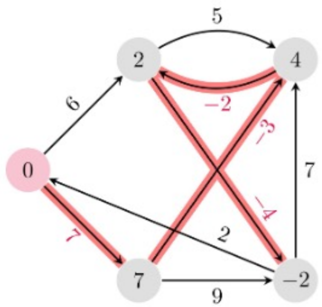


Natural inputs





Abstract inputs

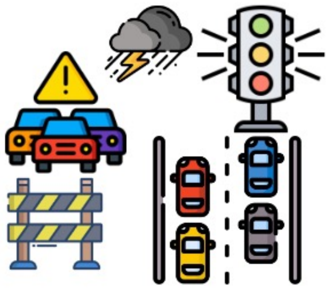


Abstract outputs

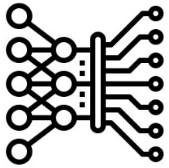
Slide by Veličković

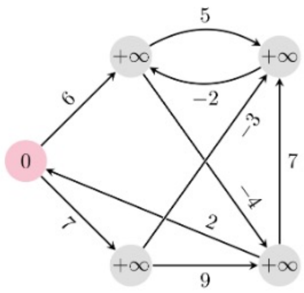
Why use GNNs for algorithm design?

- Alternatively, **replace** human feature extractor with NN
  - Still apply same combinatorial algorithm
- Issue: algorithms typically perform **discrete optimization**

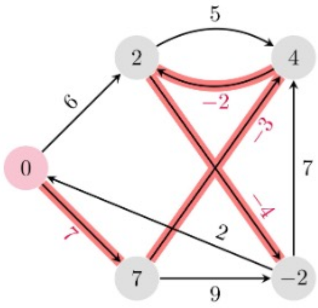


Natural inputs





Abstract inputs



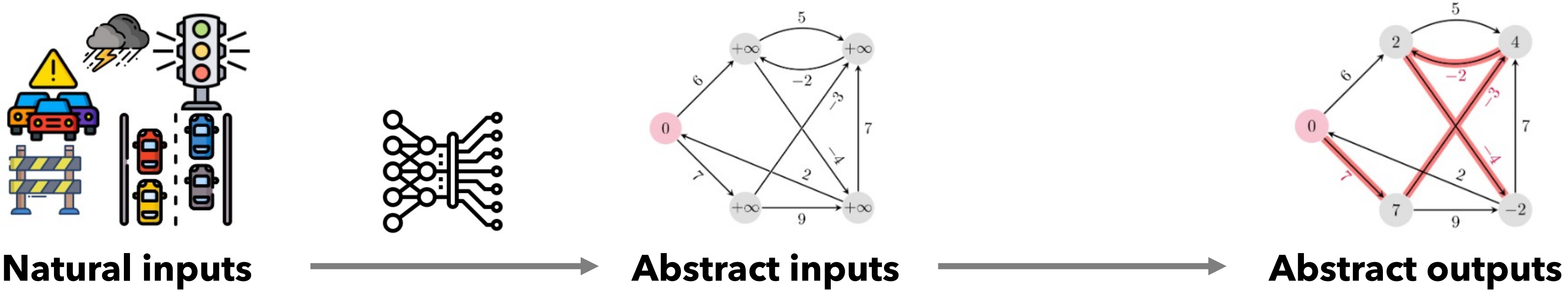
Abstract outputs

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Slide by Veličković

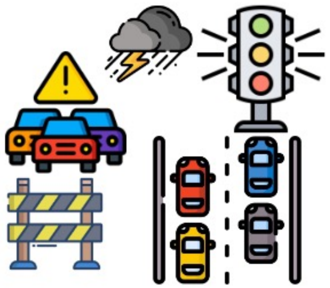
Why use GNNs for algorithm design?

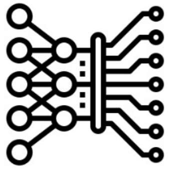
- Alternatively, **replace** human feature extractor with NN
  - Still apply same combinatorial algorithm
- Issue: algorithms typically perform **discrete optimization**
  - Doesn't play nicely with **gradient-based** optimization of NNs

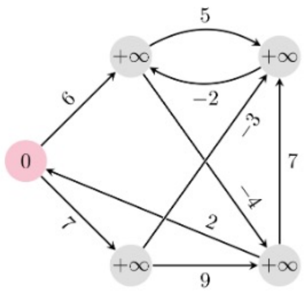


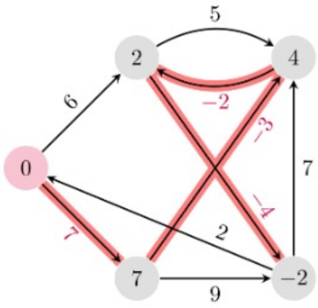
Why use GNNs for algorithm design?

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





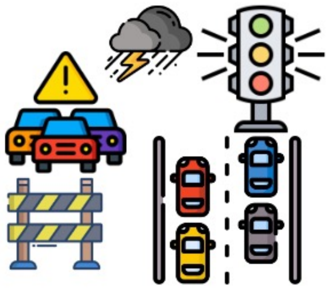


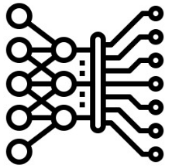


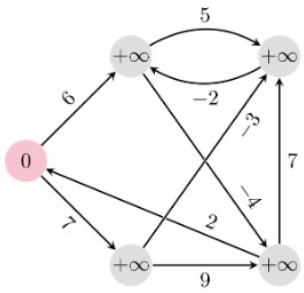
Natural inputs → **Abstract inputs** → **Abstract outputs**

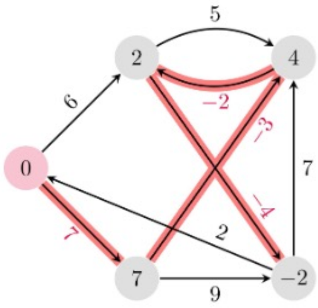
Why use GNNs for algorithm design?

- Second (more fundamental) issue: **data efficiency**
  - Real-world data is often incredibly rich





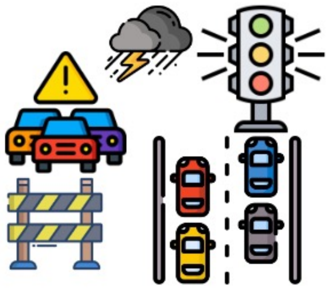


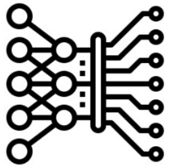


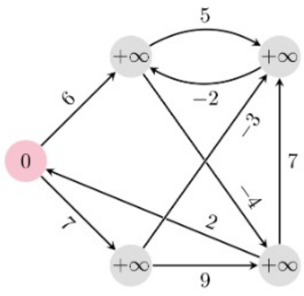
Natural inputs → **Abstract inputs** → **Abstract outputs**

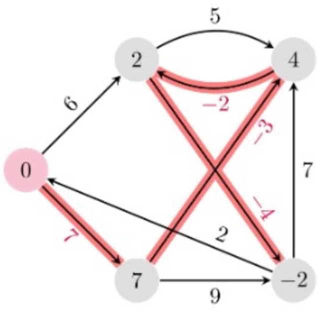
Why use GNNs for algorithm design?

- Second (more fundamental) issue: **data efficiency**
  - Real-world data is often incredibly rich
  - We still have to compress it down to scalar values







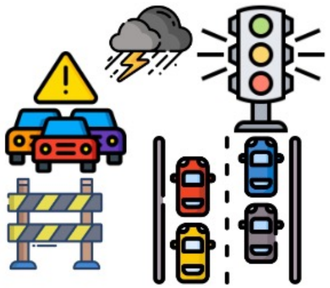


Natural inputs → **Abstract inputs** → **Abstract outputs**

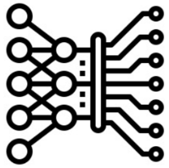
Slide by Veličković

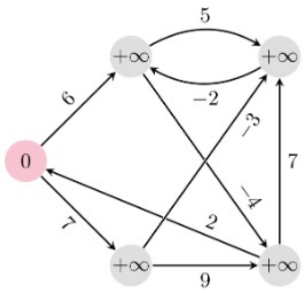
Why use GNNs for algorithm design?

- 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

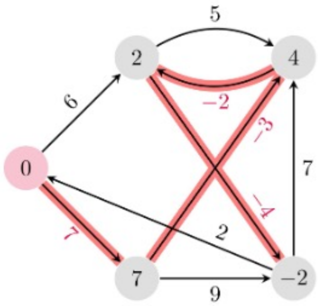


Natural inputs





Abstract inputs

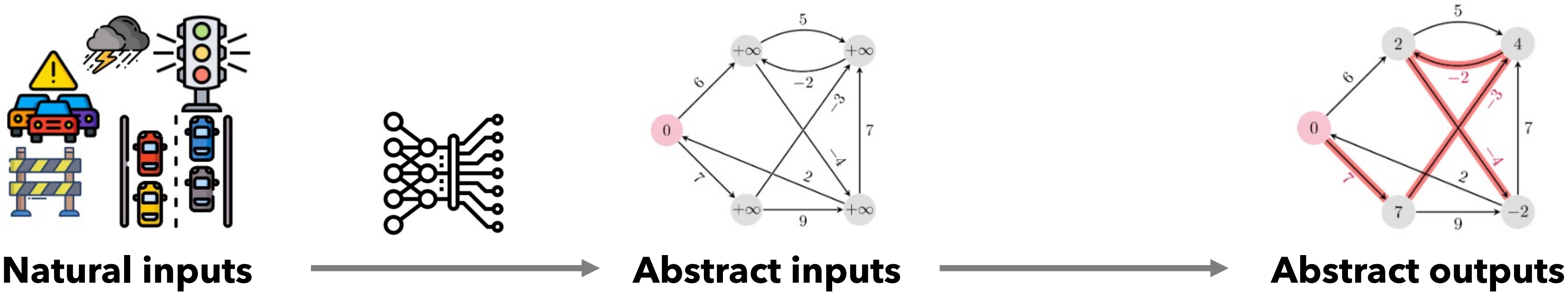


Abstract outputs

Why use GNNs for algorithm design?

- 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!



Why use GNNs for algorithm design?

- 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:

Why use GNNs for algorithm design?

- 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**

Why use GNNs for algorithm design?

- 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

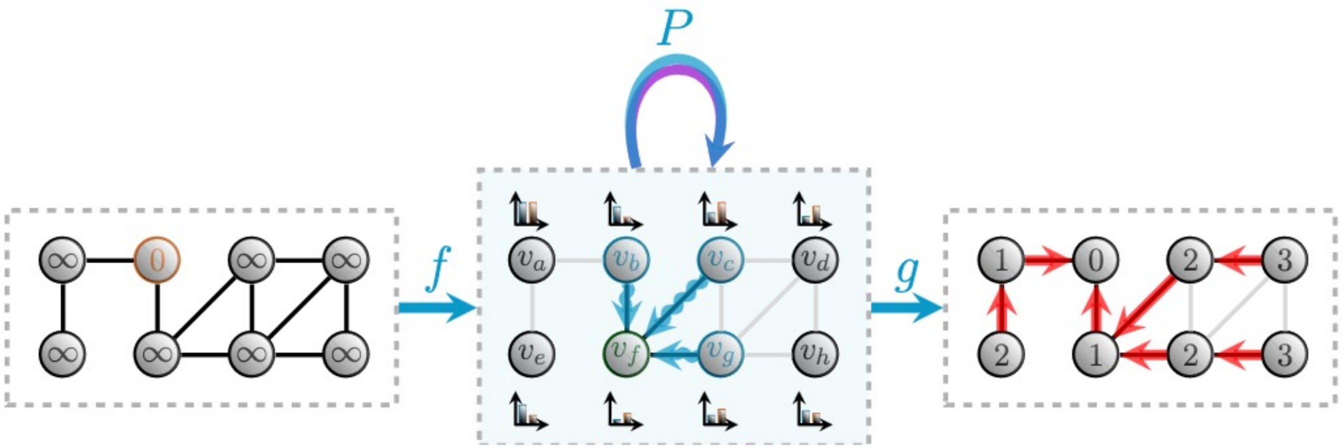
  Assumes it is perfect!

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

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

Neural algorithmic pipeline

Abstract inputs $\bar{x}$



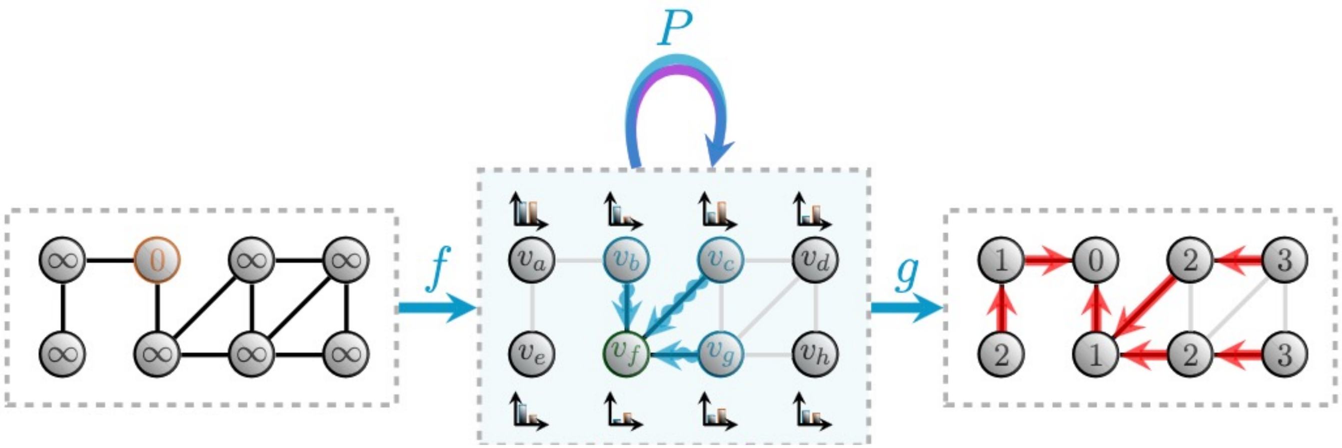
Abstract outputs
$\bar{y} \approx g\Big(P\big(f(x)\big)\Big)$

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

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

Neural algorithmic pipeline

Abstract inputs $\bar{x}$



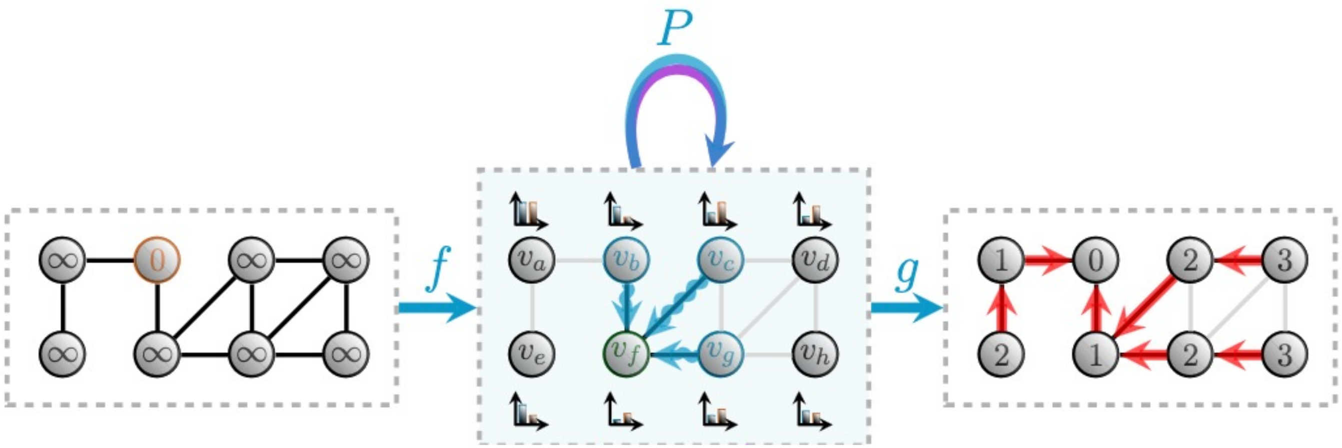
Abstract outputs
$\bar{y} \approx g\left(P\left(f(x)\right)\right)$

After training on abstract inputs, processor P:

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

Neural algorithmic pipeline

Abstract inputs $\bar{x}$



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

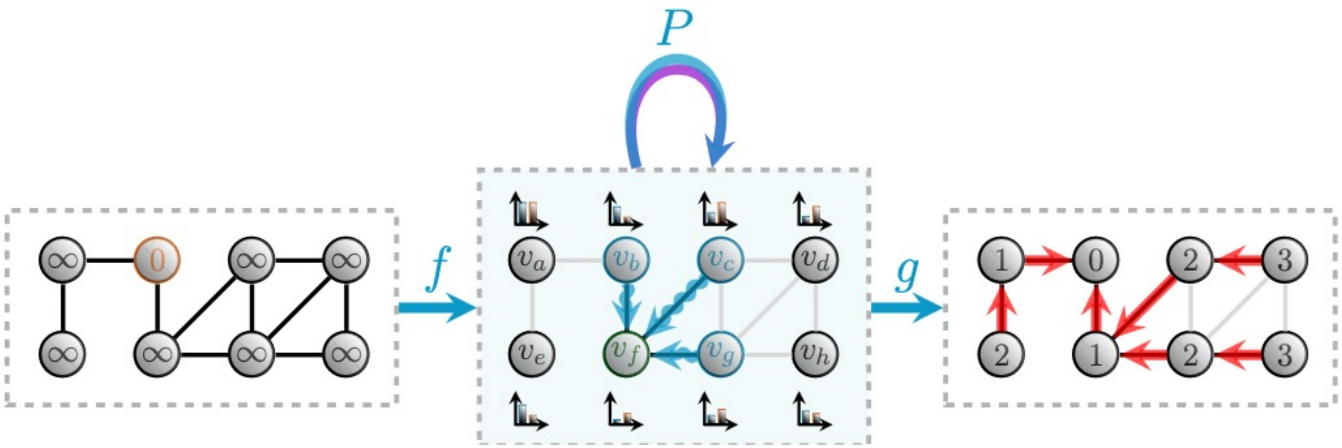
After training on abstract inputs, processor P:

1. Admits useful gradients

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

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:

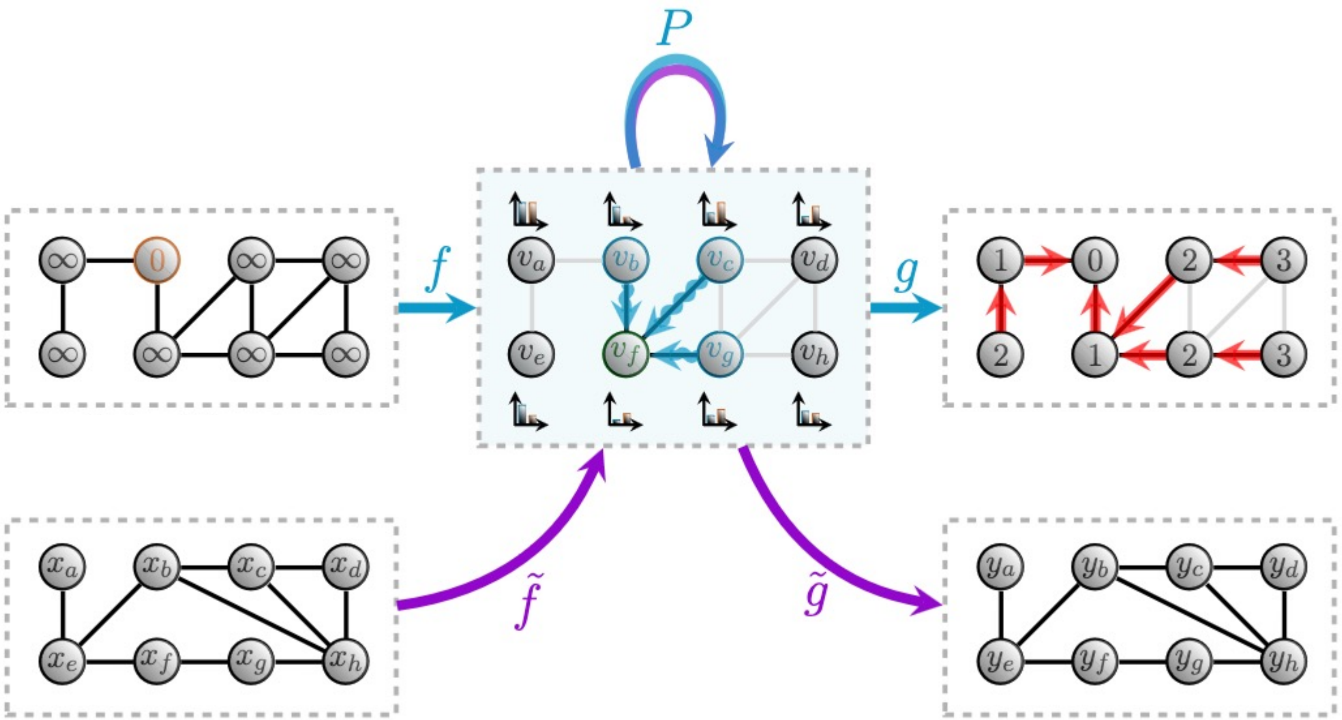
1. Admits useful gradients
2. Operates over high-dim latent space (better use of data)

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

Neural algorithmic pipeline

Abstract inputs $\bar{x}$

Natural inputs x



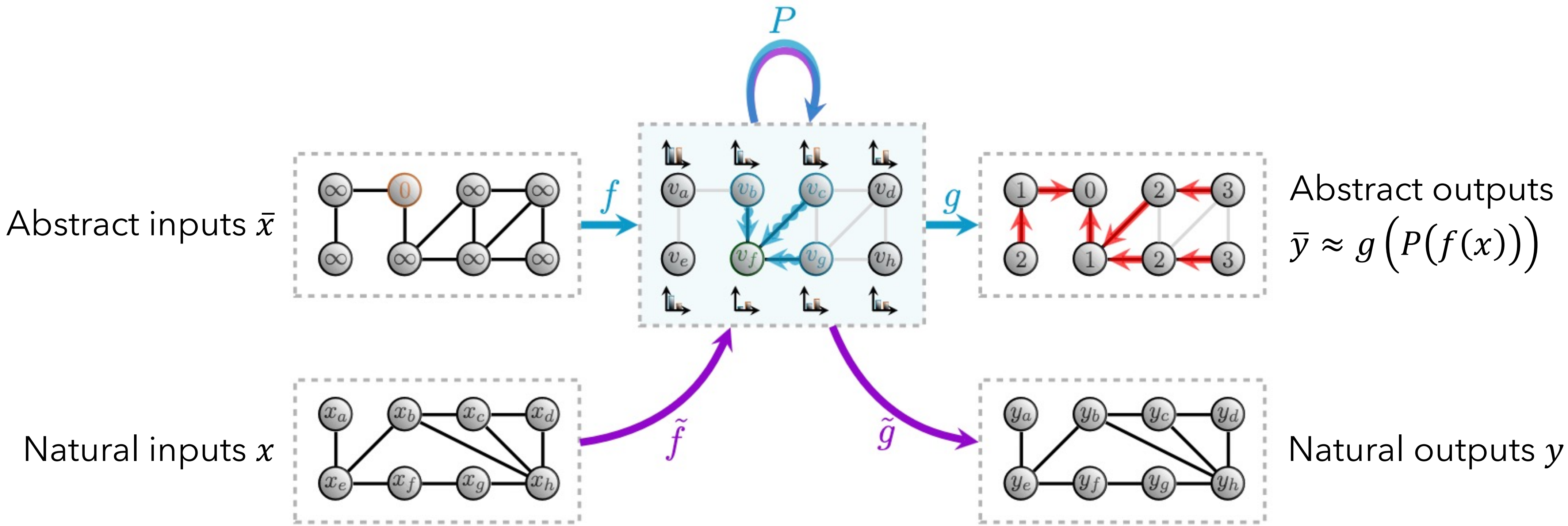
Abstract outputs
$\bar{y} \approx g\left(P\left(f(x)\right)\right)$

Natural outputs y

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

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

Neural algorithmic pipeline



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

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
   i. Motivation
   ii. Example algorithms
   iii. Experiments
   iv. Understanding max-aggregation
   v. Additional research
4. Learning greedy heuristics with RL

Breadth-first search

- Source node s

Breadth-first search

- Source node s
- Initial input $x_i^{(1)} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s \end{cases}$

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:

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:

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

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 $x_i^{(1)} = \begin{cases} 0 & \text{if } i = s \\ \infty & \text{if } i \neq s \end{cases}$

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

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

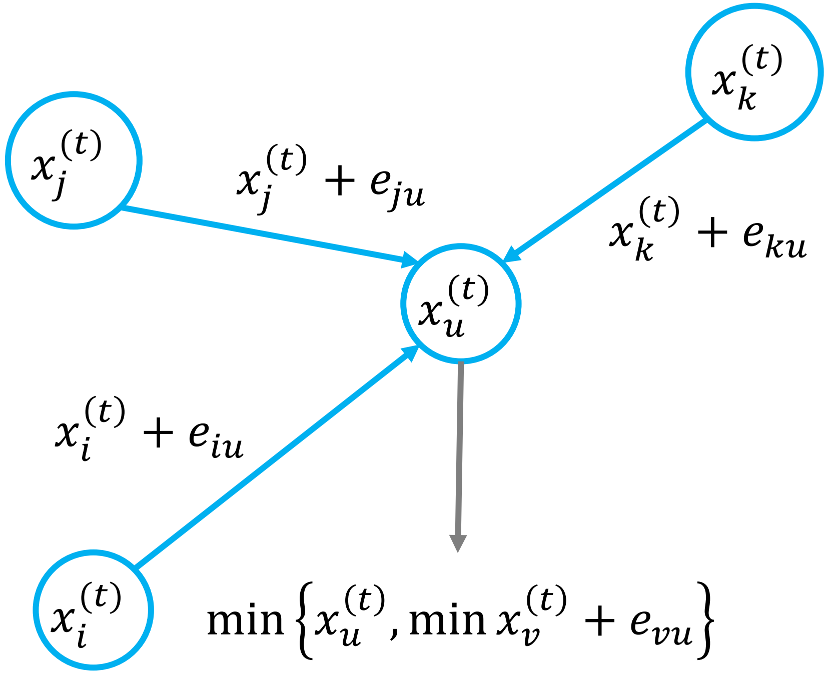
Bellman-Ford (shortest path)

- 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

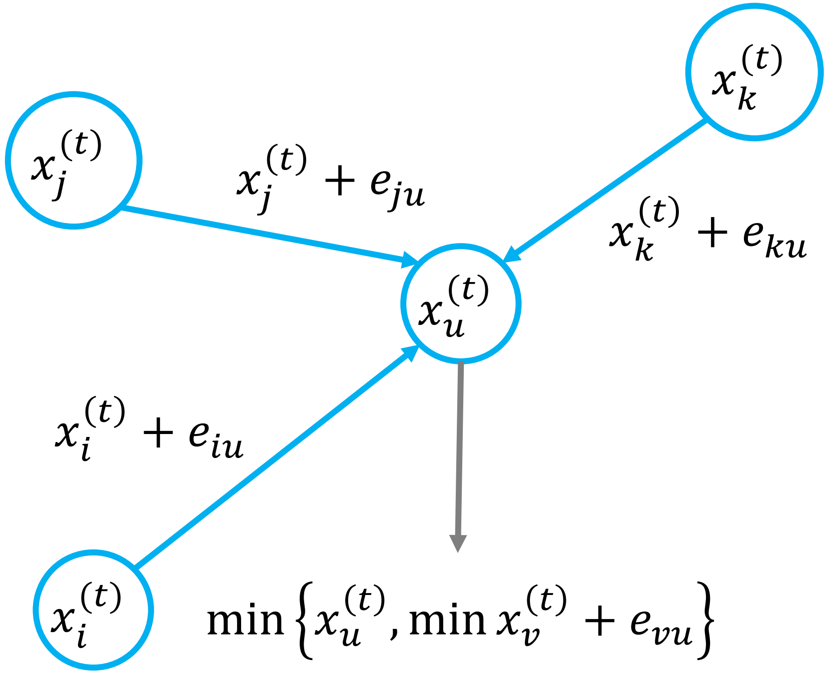
$$x_i^{(t+1)} = \min \left\{ x_i^{(t)}, \min_{(j,i) \in E} x_j^{(t)} + e_{ji} \right\}$$

Bellman-Ford: Message passing



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

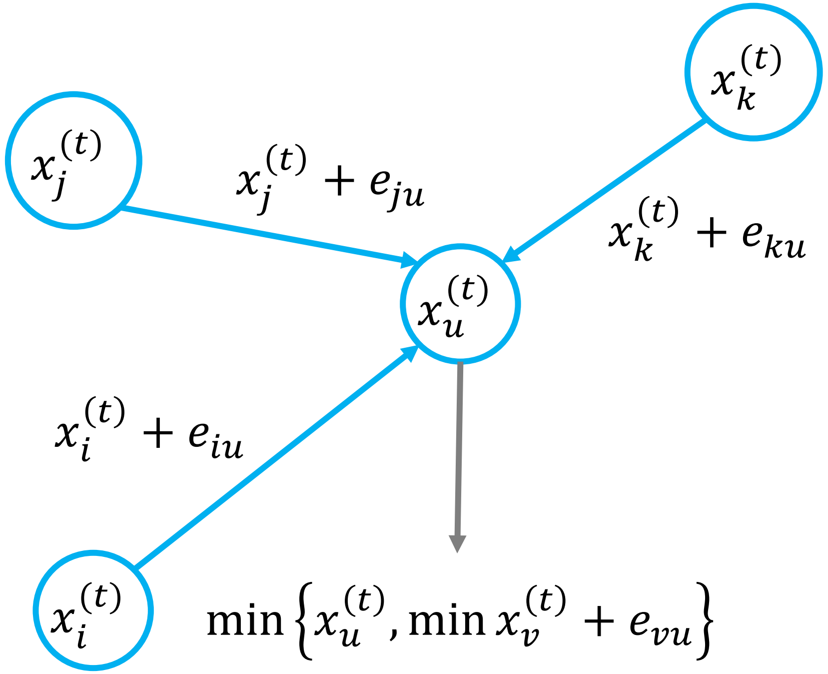
Bellman-Ford: Message passing



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

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

Bellman-Ford: Message passing



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

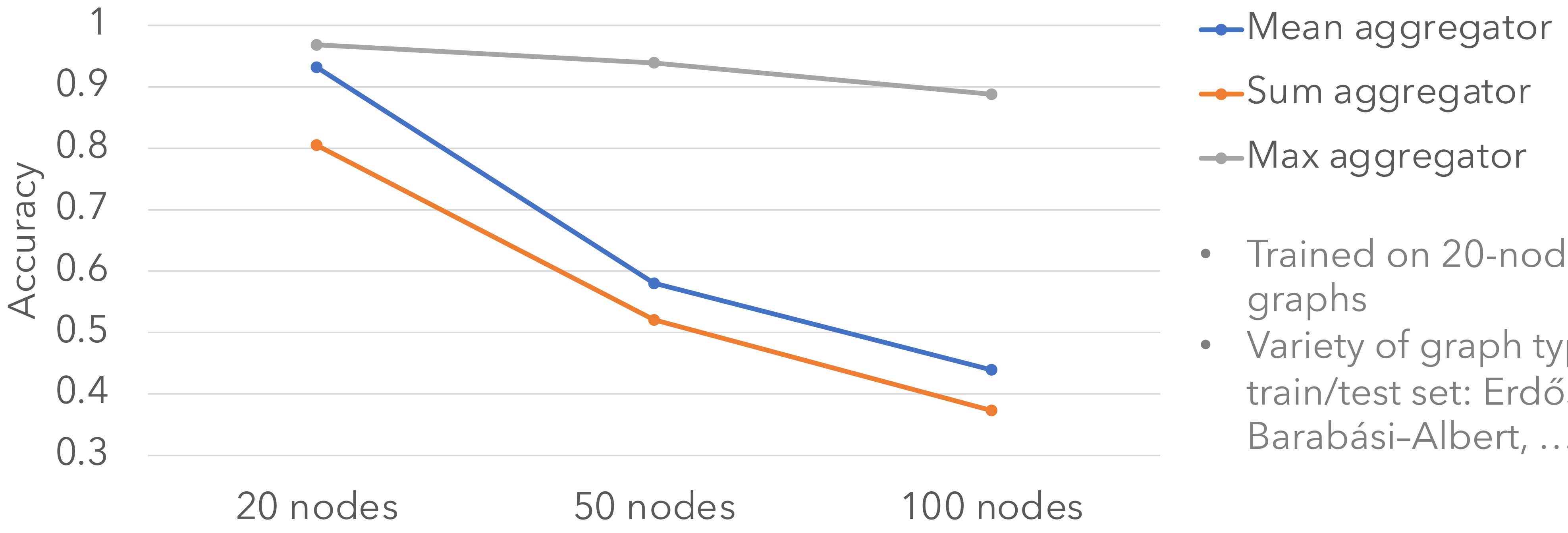
(Really, so that a function of $\boldsymbol{h}_u^{(t)} \approx x_u^{(t)}$)

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

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
   i. Motivation
   ii. Example algorithms
   iii. Experiments
   iv. Understanding max-aggregation
   v. Additional research
4. Learning greedy heuristics with RL

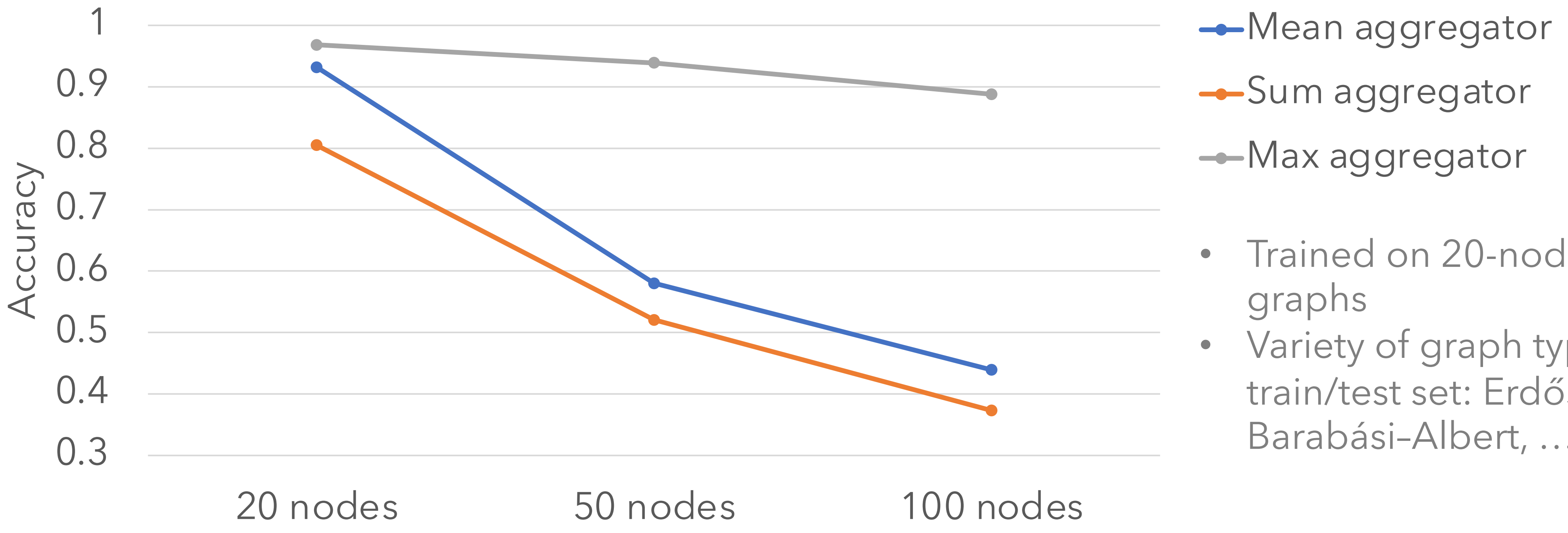
Shortest-path predecessor prediction



- Trained on 20-node graphs
- Variety of graph types in train/test set: Erdős-Reyni, Barabási–Albert, …

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

Shortest-path predecessor prediction

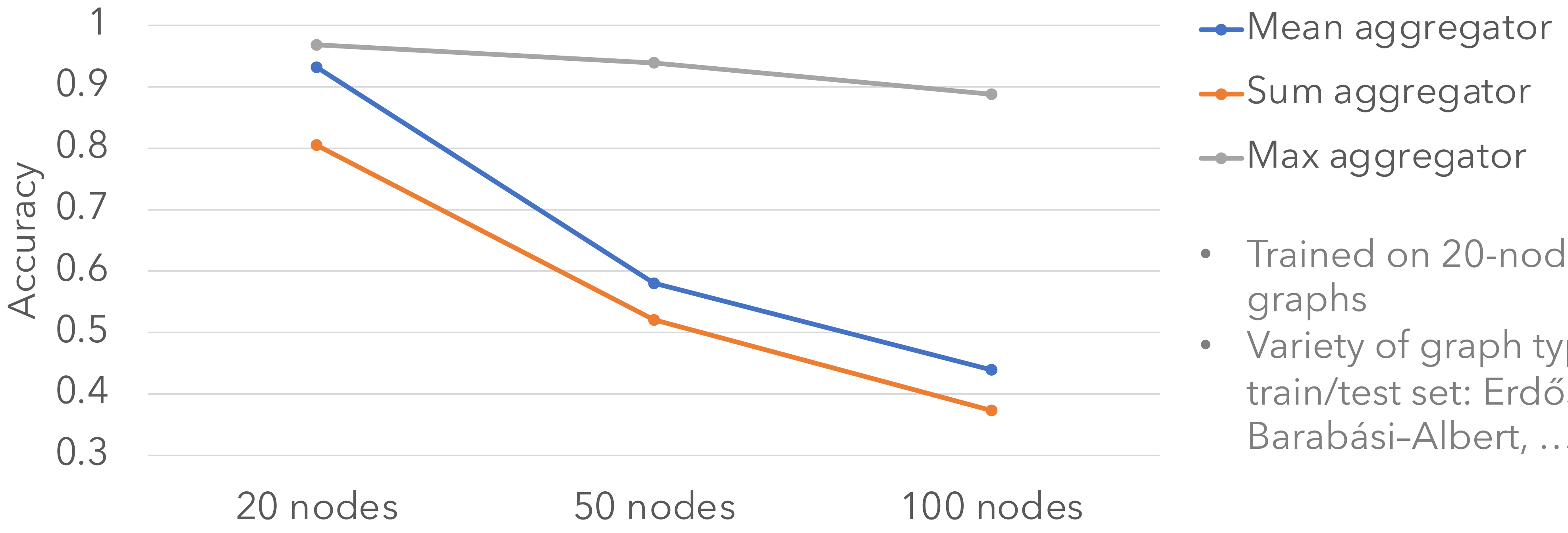


- Trained on 20-node graphs
- Variety of graph types in train/test set: Erdős-Reyni, Barabási–Albert, …

Improvement of max-aggregator increases with size

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

Shortest-path predecessor prediction



- Trained on 20-node graphs
- Variety of graph types in train/test set: Erdős-Reyni, Barabási–Albert, …

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**

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

Learning multiple algorithms

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

Comparisons

- *(no-reach)*: Learn Bellman-Ford alone

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

Learning multiple algorithms

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

Comparisons

- *(no-reach):* Learn Bellman-Ford alone
    - Doesn’t simultaneously learn reachability

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

Learning multiple algorithms

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

Comparisons

- *(no-reach)*: Learn Bellman-Ford alone
    - Doesn't simultaneously learn reachability
- *(no-algo)*:

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

Learning multiple algorithms

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

Comparisons

- *(no-reach):* Learn Bellman-Ford alone
  - Doesn't simultaneously learn reachability
- *(no-algo):*
  - Don't supervise intermediate steps

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

Learning multiple algorithms

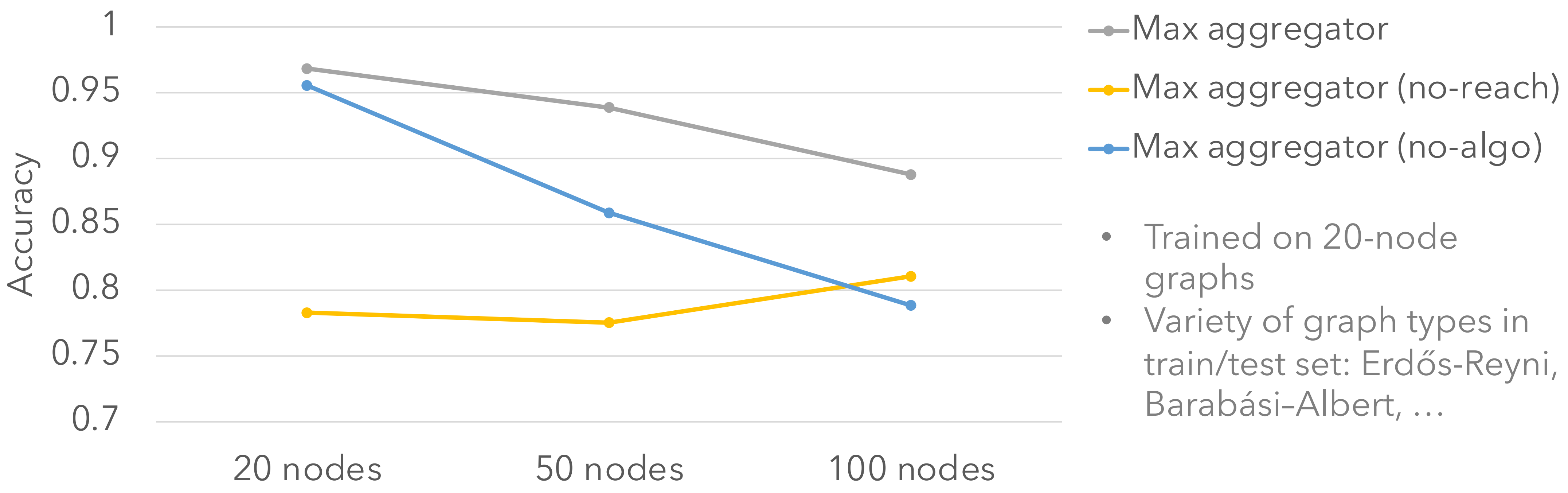
Learn to execute both BFS and Bellman-Ford **simultaneously**

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

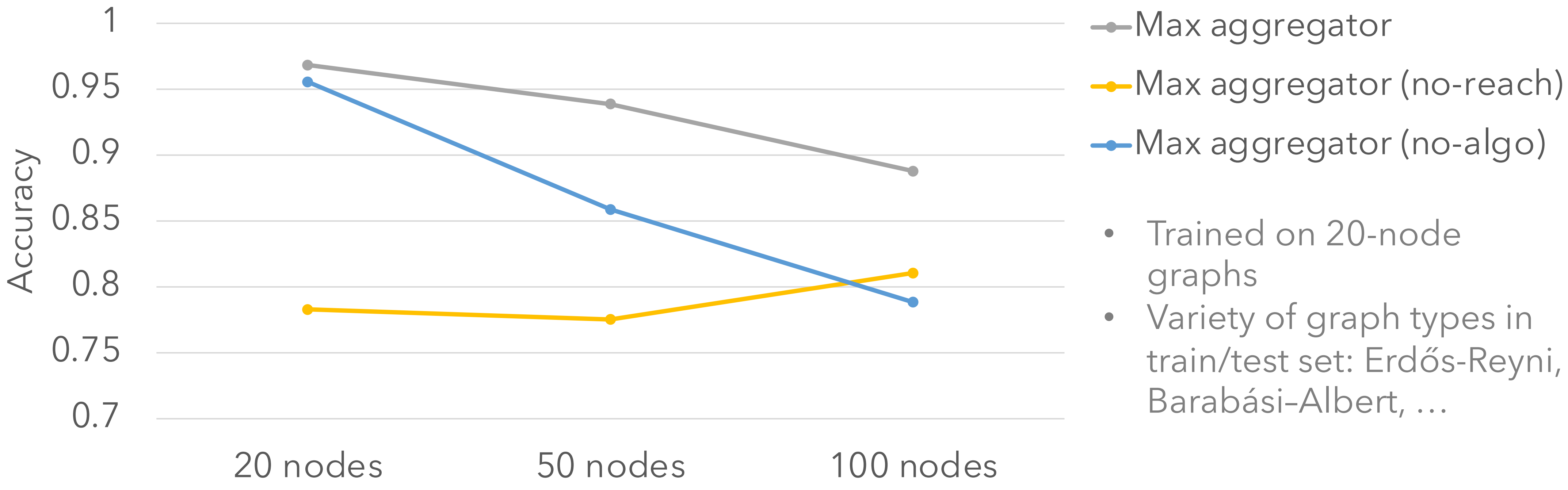


- Trained on 20-node graphs
- Variety of graph types in train/test set: Erdős-Reyni, Barabási–Albert, …

- **(no-reach) results**: positive knowledge transfer

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

Shortest-path predecessor prediction



- Trained on 20-node graphs
- Variety of graph types in train/test set: Erdős-Reyni, Barabási–Albert, …

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

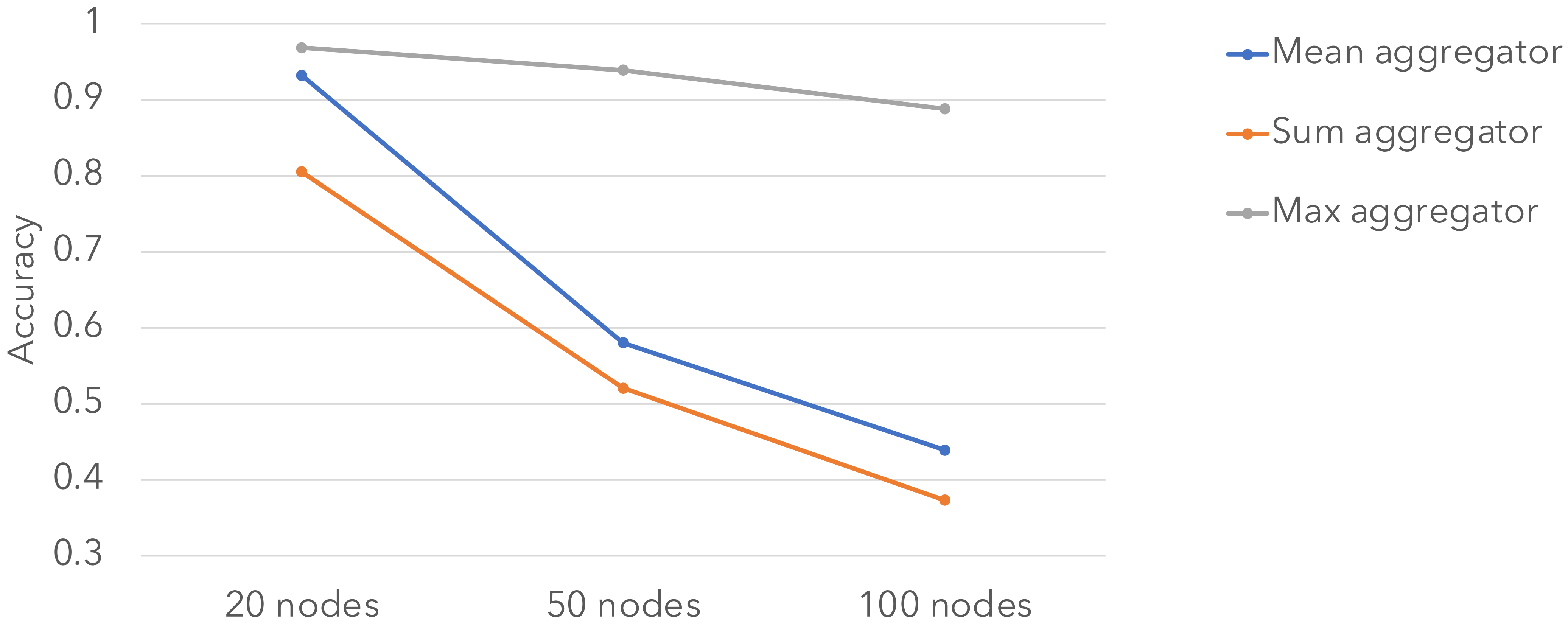
Veličković, Ying, Padovano, Hadsell, Blundell, ICLR'20

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
   i. Motivation
   ii. Example algorithms
   iii. Experiments
   iv. **Understanding max-aggregation**
   v. Additional research
4. Learning greedy heuristics with RL

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

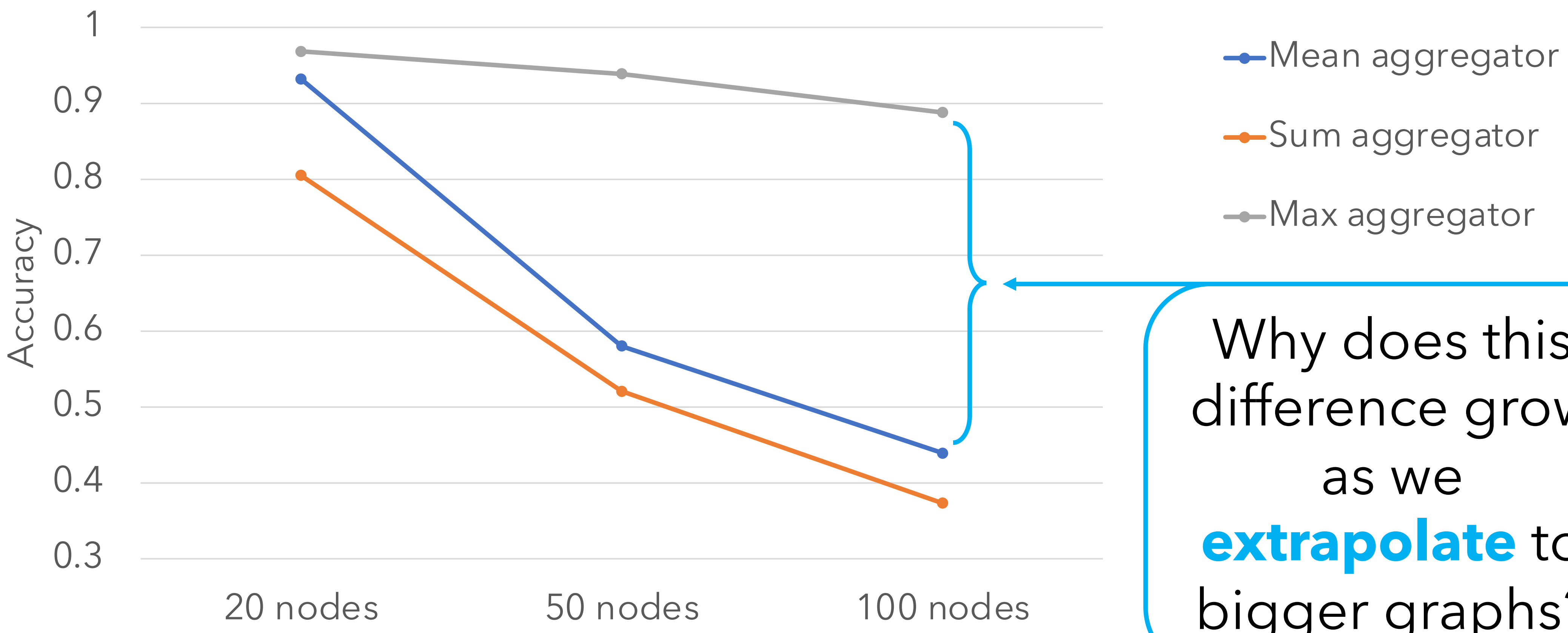
Shortest-path predecessor prediction



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

Shortest-path predecessor prediction



Why does this difference grow as we **extrapolate** to bigger graphs?

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

Extrapolation error

- $f: \mathcal{X} \to \mathbb{R}$ is a model trained on $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{D}$

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Extrapolation error

- $f: \mathcal{X} \to \mathbb{R}$ is a model trained on $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{D}$
  $y_i = g(x_i)$ for some ground-truth function g

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Extrapolation error

- $f: \mathcal{X} \to \mathbb{R}$ is a model trained on $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{D}$
  $y_i = g(x_i)$ for some ground-truth function g
- $\mathcal{P}$ is a **distribution over $\mathcal{X} \setminus \mathcal{D}$**

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

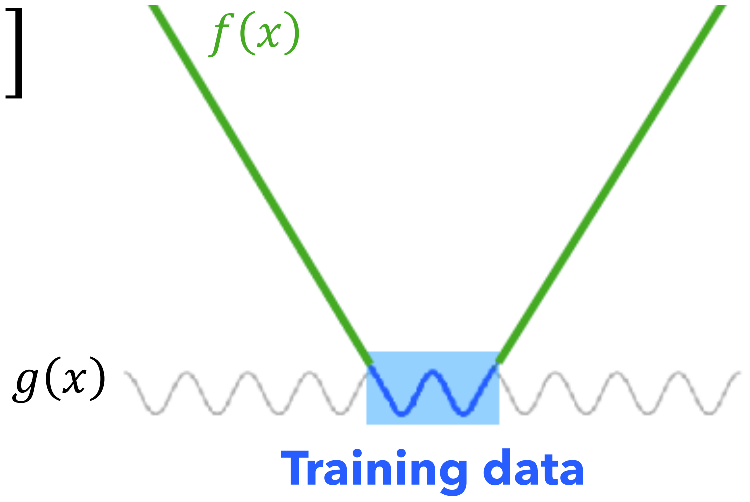
Extrapolation error

- $f: \mathcal{X} \to \mathbb{R}$ is a model trained on $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{D}$
  $y_i = g(x_i)$ for some ground-truth function g
- $\mathcal{P}$ is a **distribution over $\mathcal{X} \setminus \mathcal{D}$**
- $\ell: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

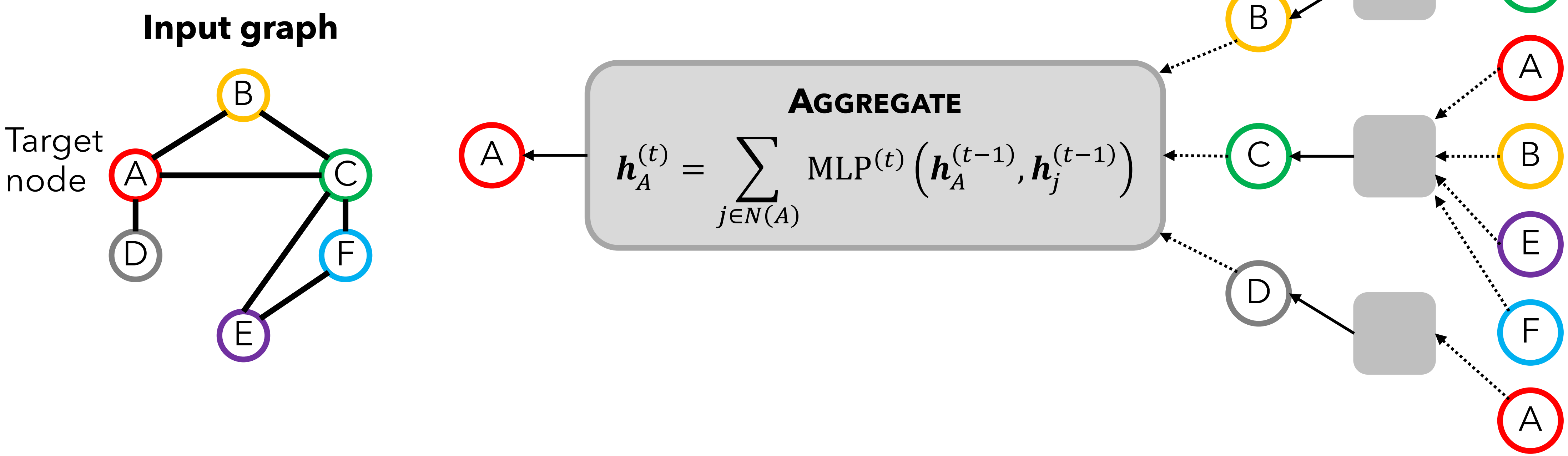
Extrapolation error

- $f: \mathcal{X} \to \mathbb{R}$ is a model trained on $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{D}$
  $y_i = g(x_i)$ for some ground-truth function g
- $\mathcal{P}$ is a **distribution over $\mathcal{X} \setminus \mathcal{D}$**
- $\ell: \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ is a loss function
- **Extrapolation error:** $\mathbb{E}_{x\sim\mathcal{P}}\left[\ell\left(f(x), g(x)\right)\right]$



Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Aggregation functions



ReLU MLP extrapolate linearly

Theorem [Xu et al., ICLR'21, informal]:

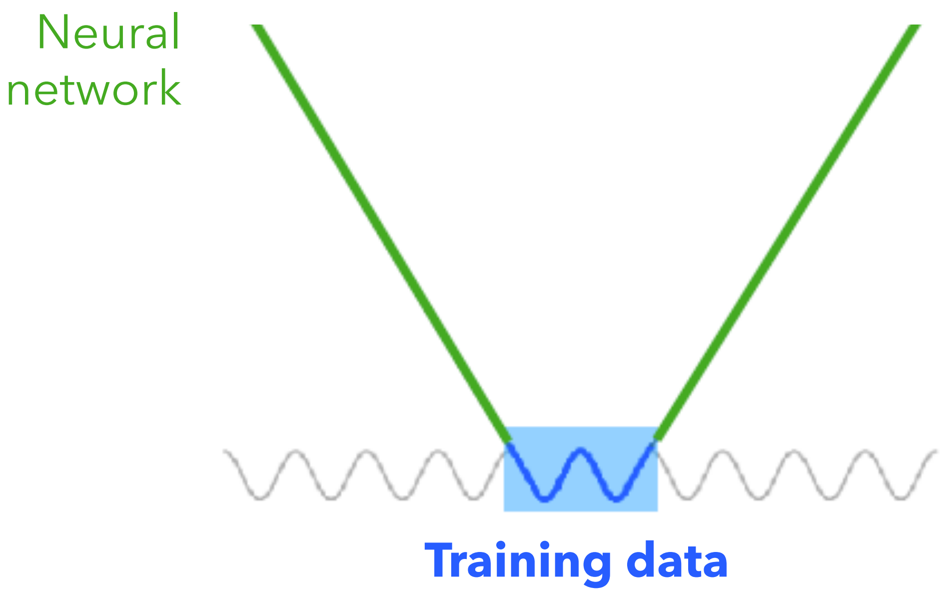
- $f: \mathbb{R}^d \to \mathbb{R}$, a 2-layer ReLU MLP trained w/ gradient descent

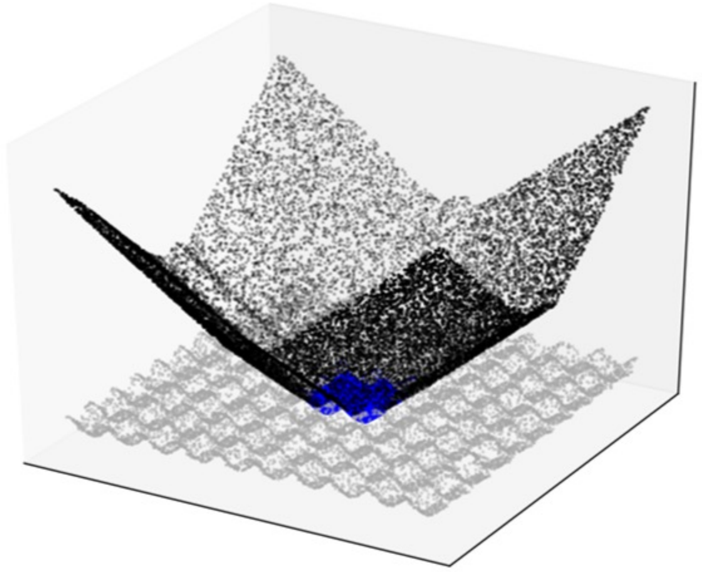
Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

ReLU MLP extrapolate linearly

Theorem [Xu et al., ICLR'21, informal]:

- $f: \mathbb{R}^d \rightarrow \mathbb{R}$, a 2-layer ReLU MLP trained w/ gradient descent
- Along any direction $\boldsymbol{v} \in \mathbb{R}^d$, f approaches a **linear** function





Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

ReLU MLP extrapolate linearly

Theorem [Xu et al., ICLR'21, informal]:

- $f: \mathbb{R}^d \to \mathbb{R}$, a 2-layer ReLU MLP trained w/ gradient descent
- Along any direction $\boldsymbol{v} \in \mathbb{R}^d$, f approaches a **linear** function
- More formally, let $\boldsymbol{x} = t\boldsymbol{v}$

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

ReLU MLP extrapolate linearly

Theorem [Xu et al., ICLR'21, informal]:

- $f: \mathbb{R}^d \rightarrow \mathbb{R}$, a 2-layer ReLU MLP trained w/ gradient descent
- Along any direction $\boldsymbol{v} \in \mathbb{R}^d$, f approaches a **linear** function
- More formally, let $\boldsymbol{x} = t\boldsymbol{v}$
  - Then $f(\boldsymbol{x} + h\boldsymbol{v}) - f(\boldsymbol{x}) = f(t\boldsymbol{v} + h\boldsymbol{v}) - f(t\boldsymbol{v}) \rightarrow \beta_{\boldsymbol{v}} h$ at a rate $O\left(\frac{1}{t}\right)$

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Implications for GNNs

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

GNN: $\boldsymbol{h}_i^{(t)} = \sum_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Implications for GNNs

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

GNN: $\boldsymbol{h}_i^{(t)} = \boxed{\sum_{j\in N(i)} \mathrm{MLP}}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

MLP must learn a **non-linearity**

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Implications for GNNs

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

GNN: $\boldsymbol{h}_i^{(t)} = \sum_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

GNN 2: $\boldsymbol{h}_i^{(t)} = \max_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

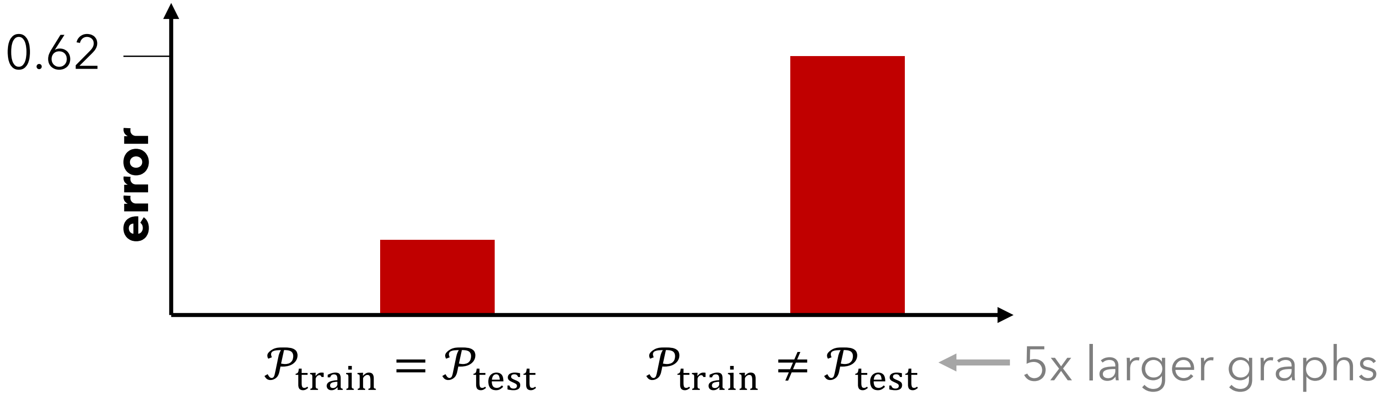
Implications for GNNs

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

$\boxed{\text{GNN:}}$ $\boldsymbol{h}_i^{(t)} = \sum_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

GNN 2: $\boldsymbol{h}_i^{(t)} = \max_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

Predicting shortest path predecessor:
[Veličković et al. ICLR'20]



Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

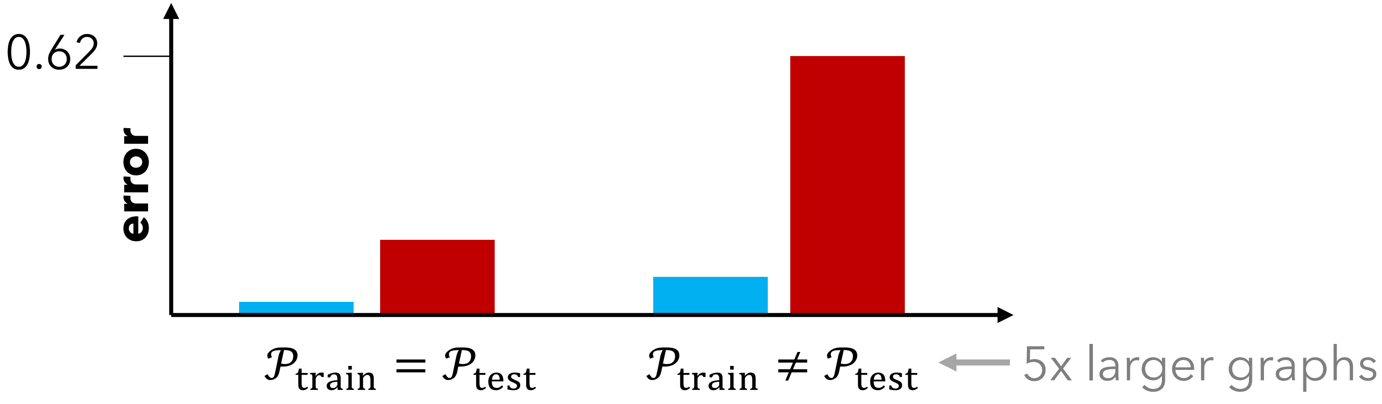
Implications for GNNs

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

GNN: $\boldsymbol{h}_i^{(t)} = \sum_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

GNN 2: $\boldsymbol{h}_i^{(t)} = \max_{j\in N(i)} \mathrm{MLP}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

Predicting shortest path predecessor:
[Veličković et al. ICLR'20]



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Xu, Zhang, Li, Du, Kawarabayashi, Jegelka, ICLR'21

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
    i. Motivation
    ii. Example algorithms
    iii. Experiments
    iv. Understanding max-aggregation
    v. Additional research
4. Learning greedy heuristics with RL

Additional research

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

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

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

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)

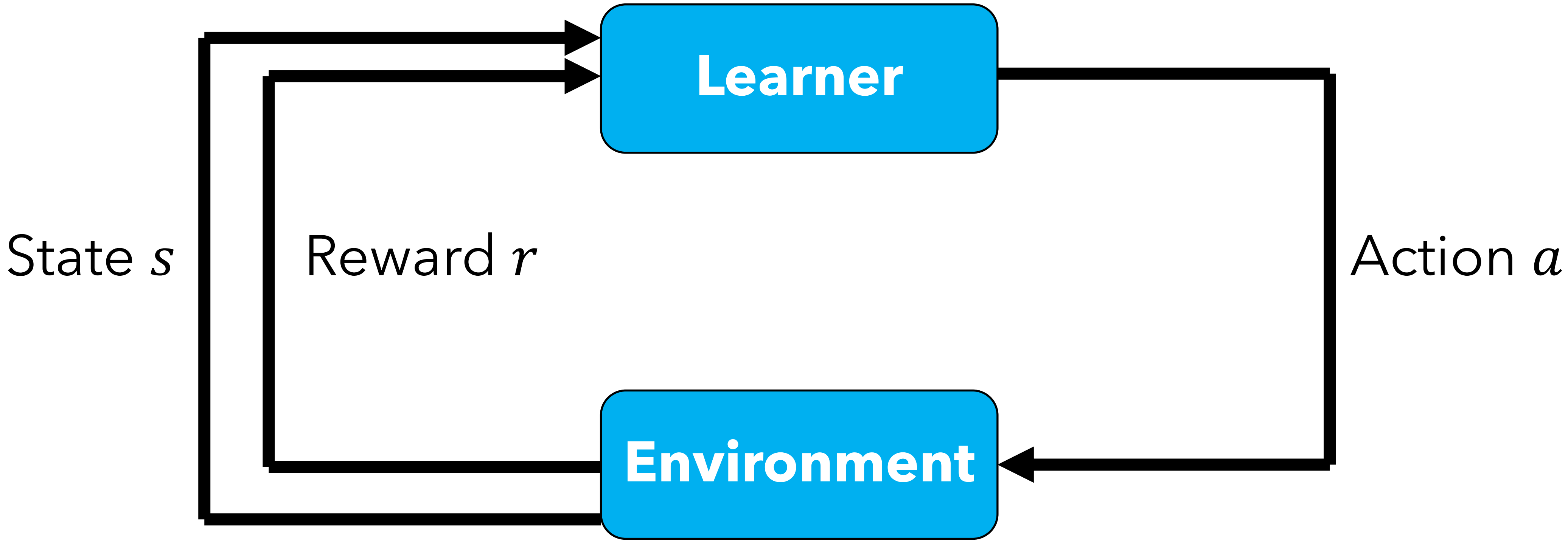
1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. **Learning greedy heuristics with RL**

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

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL
   i. **Reinforcement learning refresher**
      a. **Markov decision processes**
      b. Reinforcement learning
   ii. Overview: RL for combinatorial optimization
   iii. Examples: Min vertex cover and max cut
   iv. RL formulation
   v. Experiments

Learner interaction with environment



Markov decision processes

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

Markov decision processes

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

A: set of actions

Markov decision processes

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

A: set of actions

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

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

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

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

Policies and value functions

Value function for a policy:

Expected sum of discounted rewards

Policies and value functions

Value function for a policy:

Expected sum of discounted rewards

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

Policies and value functions

Value function for a policy:

Expected sum of discounted rewards

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

Policies and value functions

Value function for a policy:

Expected sum of discounted rewards

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

Discount factor

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

(Bellman equation)

Optimal policy and value function

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

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

Optimal policy and value function

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

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

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

- Value iteration
- Policy iteration

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL
    i. Reinforcement learning refresher
        a. Markov decision processes
        b. **Reinforcement learning**
    ii. Overview: RL for combinatorial optimization
    iii. Examples: Min vertex cover and max cut
    iv. RL formulation
    v. Experiments

Challenge of RL

MDP $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R})$:

- 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 $R: S \rightarrow \mathbb{R}$

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

Q-learning

Q functions:

Like value functions but defined over state-action pairs

Q-learning

Q functions:

Like value functions but defined over state-action pairs

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

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:

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

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

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 π

Q-learning

Q function of the optimal policy $\pi^{\star}$:

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

Q-learning

Q function of the optimal policy $\pi^\star$:

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

Q-learning

Q function of the optimal policy $\pi^\star$:

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

$Q^\star$ is the value of:

Q-learning

Q function of the optimal policy $\pi^\star$:

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

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

$Q^\star$ is the value of:

1. Starting in state s

Q-learning

Q function of the optimal policy $\pi^\star$:

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

$Q^\star$ is the value of:

1. Starting in state s
2. Taking action a

Q-learning

Q function of the optimal policy $\pi^\star$:

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

$Q^\star$ is the value of:

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

Q-learning

(High-level) Q-learning algorithm

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

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

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'

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'

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 $a = \text{argmax}\, \hat{Q}(s, \cdot)$ 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 (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL
   i. Reinforcement learning refresher
   ii. Overview: RL for combinatorial optimization
   iii. Examples: Min vertex cover and max cut
   iv. RL formulation
   v. Experiments

RL for combinatorial optimization

Tons of research in this area

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; …

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; …

Maximum cut

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

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; …

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; …

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

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

Overview

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

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

Overview

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

Feasible solution constructed by successively adding nodes to solution

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

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 $(u, v) \in E$

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

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, Zhang, Dilkina, Song; NeurIPS'17

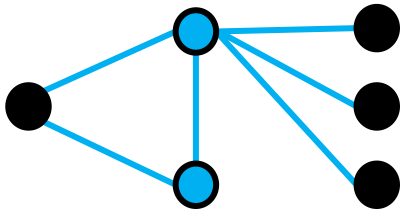
Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL
   i. Reinforcement learning refresher
   ii. Overview: RL for combinatorial optimization
   iii. **Examples: Min vertex cover and max cut**
   iv. RL formulation
   v. Experiments

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

Minimum vertex cover

Find smallest vertex subset such that each edge is covered

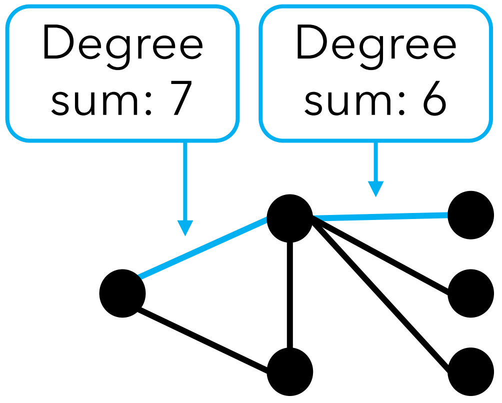


Minimum vertex cover

Find smallest vertex subset such that each edge is covered

2-approximation:

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

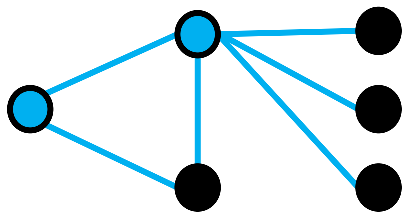


Minimum vertex cover

Find smallest vertex subset such that each edge is covered

2-approximation:

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



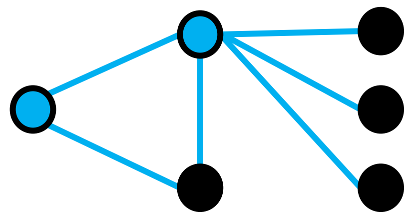
Minimum vertex cover

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

Maximum cut

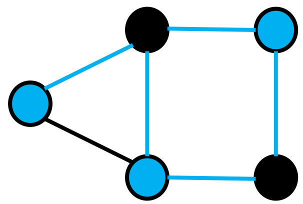
Find partition $(S, V \setminus S)$ of nodes that maximizes

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

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

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

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



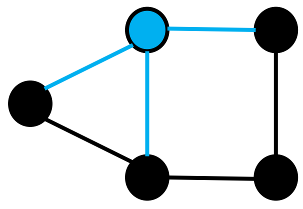
Maximum cut

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



Maximum cut

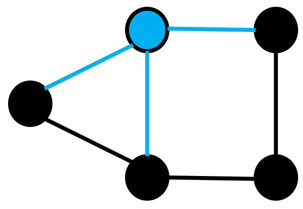
Find partition $(S, V \setminus S)$ of nodes that maximizes

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

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

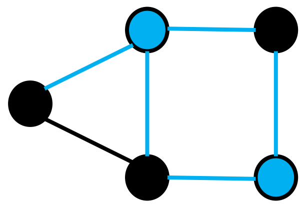
Find partition $(S, V \setminus S)$ of nodes that maximizes

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

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



Maximum cut

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

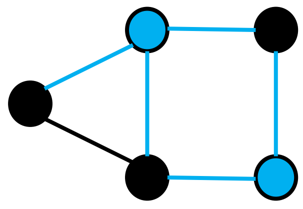
$$\sum_{(u,v)\in C} w(u,v)$$

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



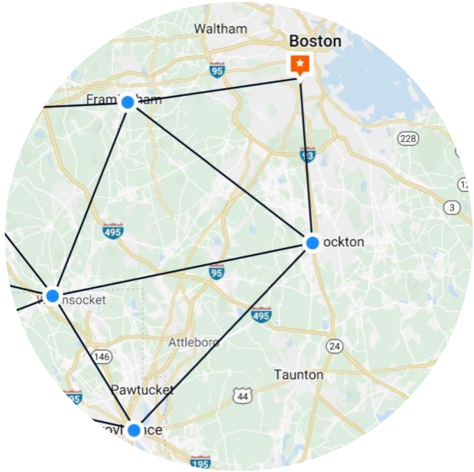
Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL
   i. Reinforcement learning refresher
   ii. Overview: RL for combinatorial optimization
   iii. Examples: Min vertex cover and max cut
   iv. RL formulation
   v. Experiments

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

RL for combinatorial optimization

Goal: learn a scoring function to guide greedy algorithm

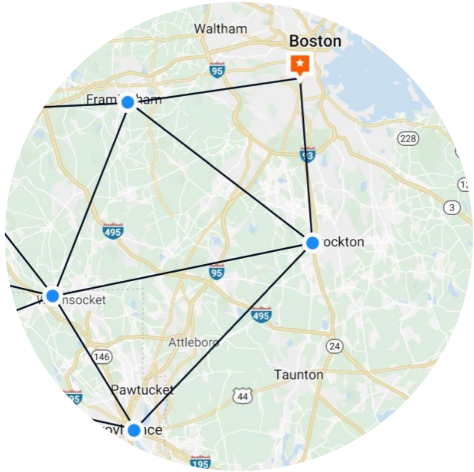


Dai, Khalil, Zhang, Dilkina, Song; NeurIPS'17

RL for combinatorial optimization

Goal: learn a scoring function to guide greedy algorithm

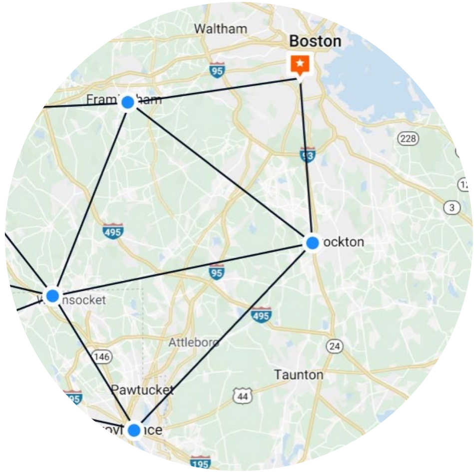
Problem	**Greedy operation**
Min vertex cover	Insert node into cover
Max cut	Insert node into subset
Traveling salesman	Insert node into sub-tour



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

RL for combinatorial optimization

Greedy algorithm	**Reinforcement learning**
Partial solution	State
Scoring function	Q-function
Select best node	Greedy policy

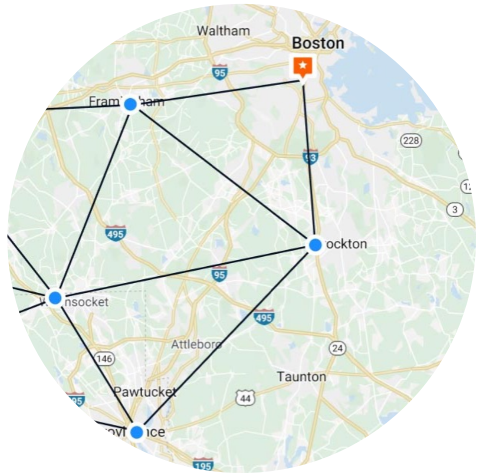


Dai, Khalil, Zhang, Dilkina, Song; NeurIPS'17

RL for combinatorial optimization

Greedy algorithm	**Reinforcement learning**
Partial solution	State
Scoring function	Q-function
Select best node	Greedy policy

Repeat until all edges are covered:



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

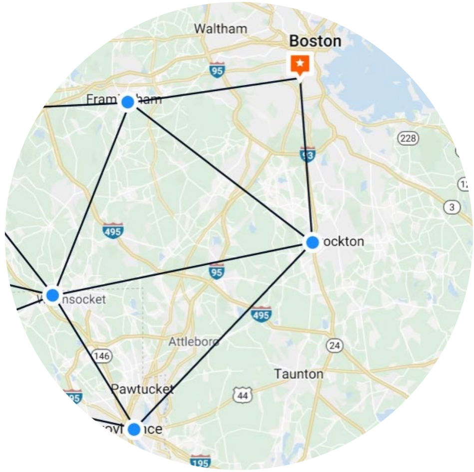
Dai, Khalil, Zhang, Dilkina, Song; NeurIPS'17

RL for combinatorial optimization

Greedy algorithm	Reinforcement learning
Partial solution	State
Scoring function	Q-function
Select best node	Greedy policy

Repeat until all edges are covered:

1. Compute node scores



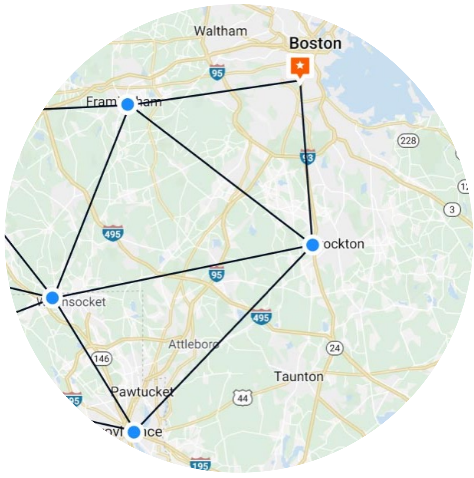
Dai, Khalil, Zhang, Dilkina, Song; NeurIPS'17

RL for combinatorial optimization

Greedy algorithm	**Reinforcement learning**
Partial solution	State
Scoring function	Q-function
Select best node	Greedy policy

Repeat until all edges are covered:

1. Compute node scores
2. Select best node with respect to score



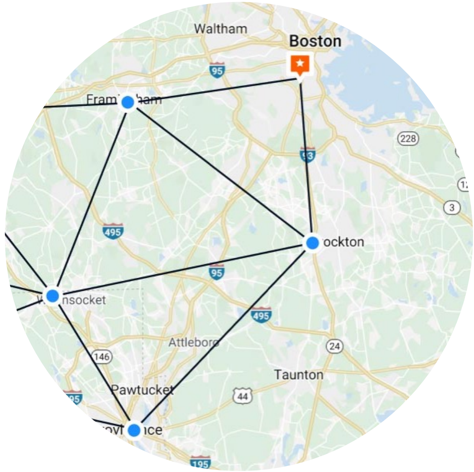
Dai, Khalil, Zhang, Dilkina, Song; NeurIPS'17

RL for combinatorial optimization

Greedy algorithm	**Reinforcement learning**
Partial solution	State
Scoring function	Q-function
Select best node	Greedy policy

Repeat until all edges are covered:

1. Compute node scores
2. Select best node with respect to score
3. Add best node to partial solution



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

Reinforcement learning formulation

State:

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

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

Reinforcement learning formulation

State:

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

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

Dai, Khalil, Zhang, Dilkina, Song; 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 $\boldsymbol{\mu}$

Dai, Khalil, Zhang, Dilkina, Song; 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 $x_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{else} \end{cases}$

Dai, Khalil, Zhang, Dilkina, Song; 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

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

Reinforcement learning formulation

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

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

Reinforcement learning formulation

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

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

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

Outline (applied techniques)

1. GNNs overview
2. Integer programming with GNNs
3. Neural algorithmic alignment
4. Learning greedy heuristics with RL
   i. Reinforcement learning refresher
   ii. Overview: RL for combinatorial optimization
   iii. Examples: Min vertex cover and max cut
   iv. RL formulation
   v. Experiments

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

Min vertex cover

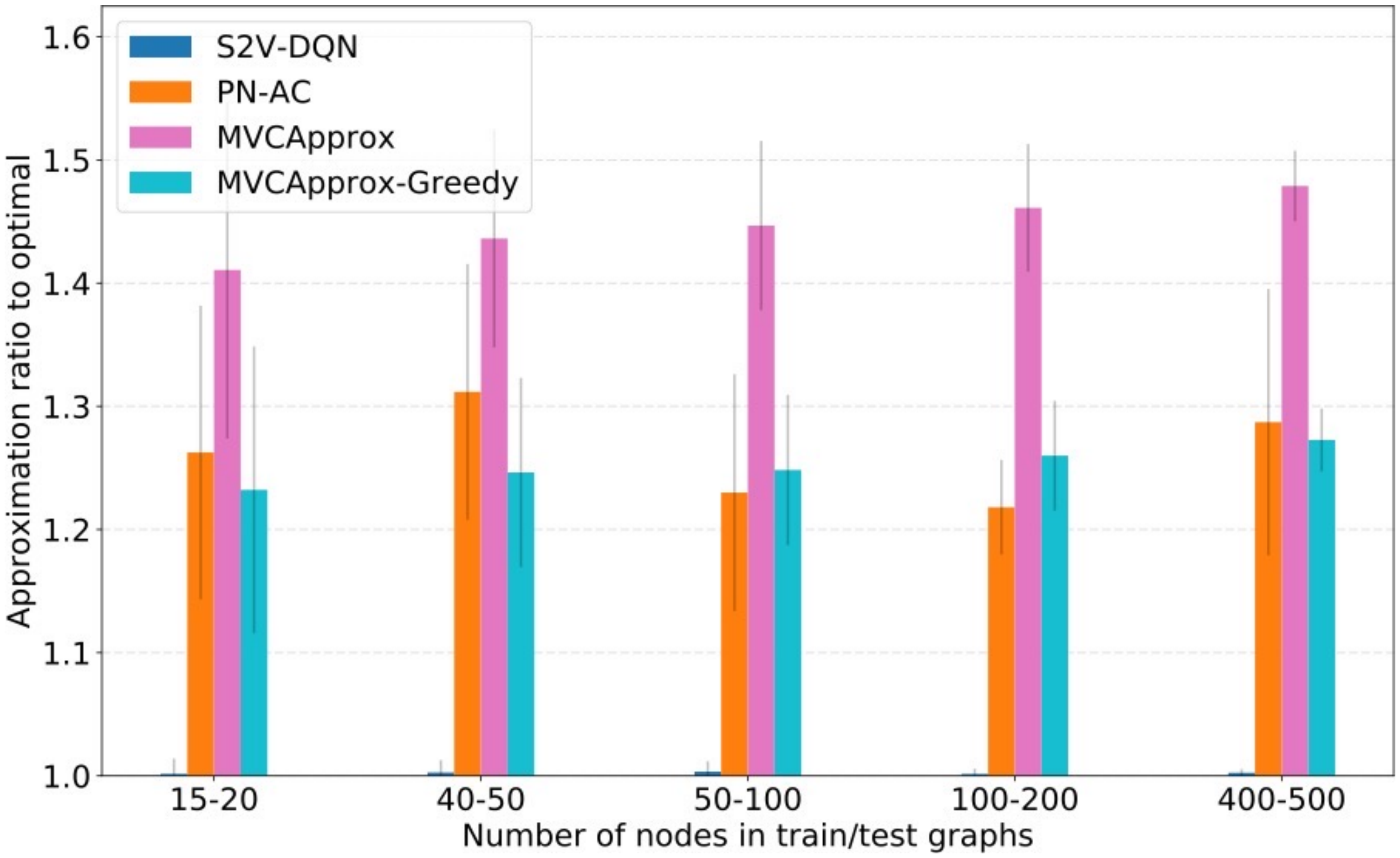
Barabasi-Albert random graphs

Paper's approach

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

2-approximation algorithm

Greedy algorithm from first few slides



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

Max cut

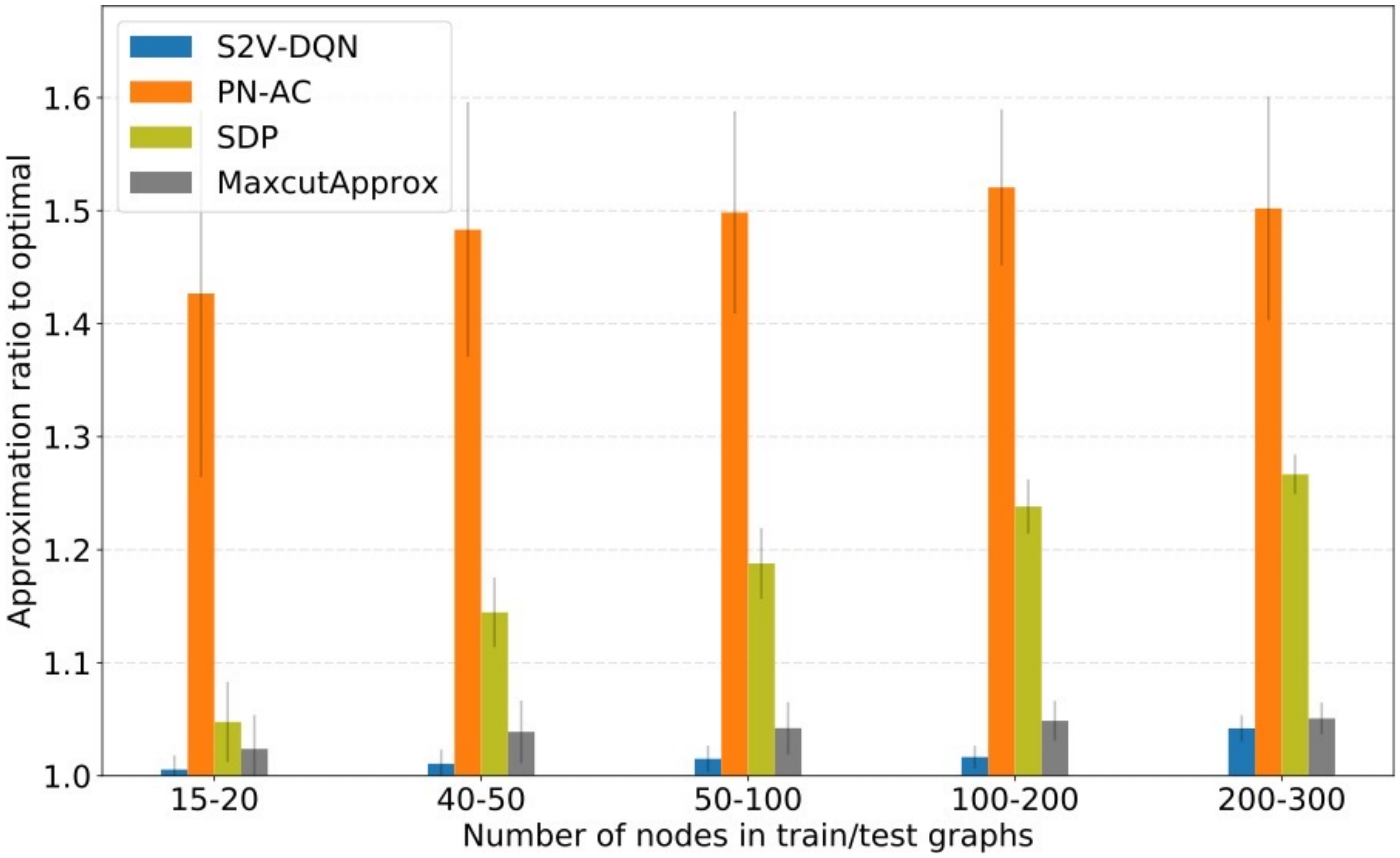
Barabasi-Albert random graphs

Paper's approach

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

Goemans-Williamson algorithm

Greedy algorithm from first few slides



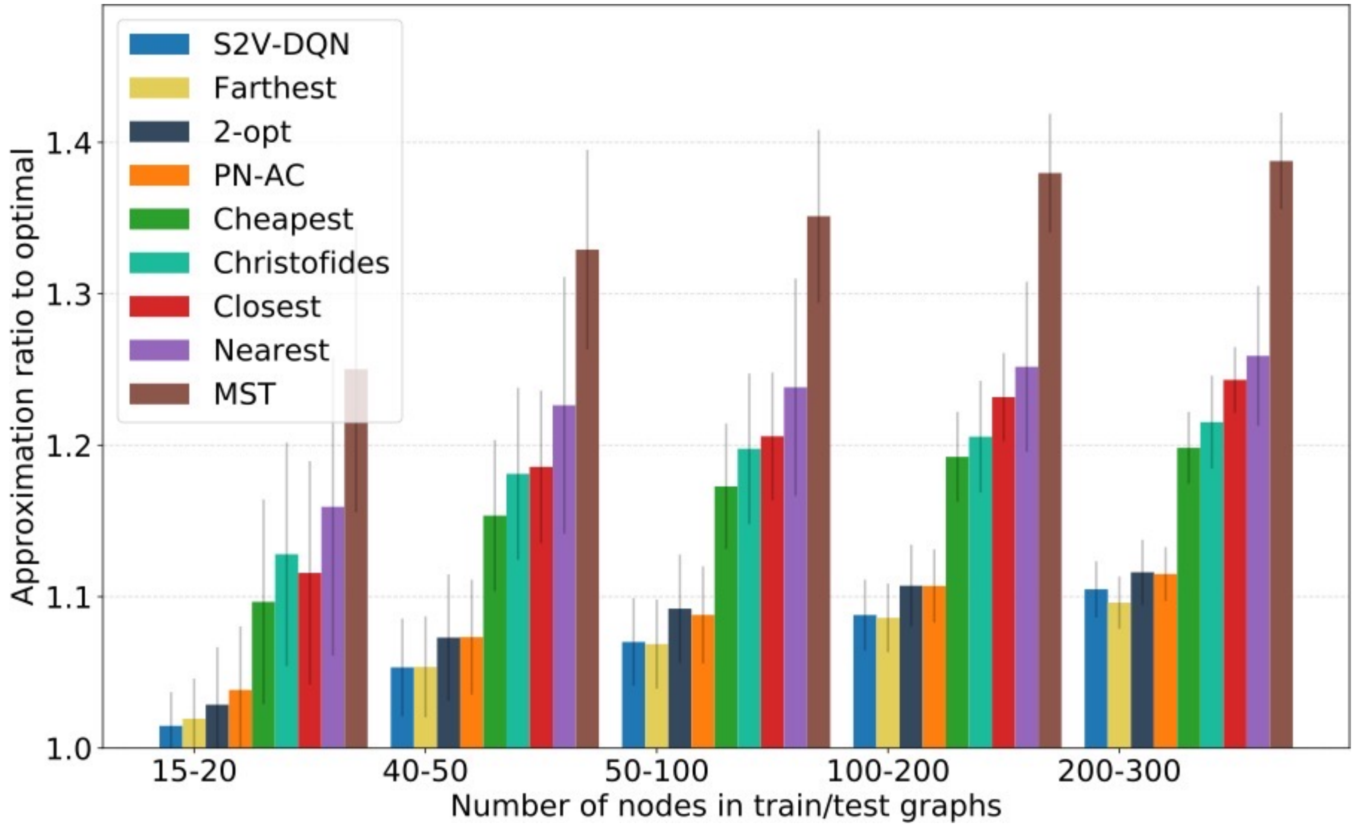
Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

TSP

Uniform random points on 2-D grid

Paper's approach



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

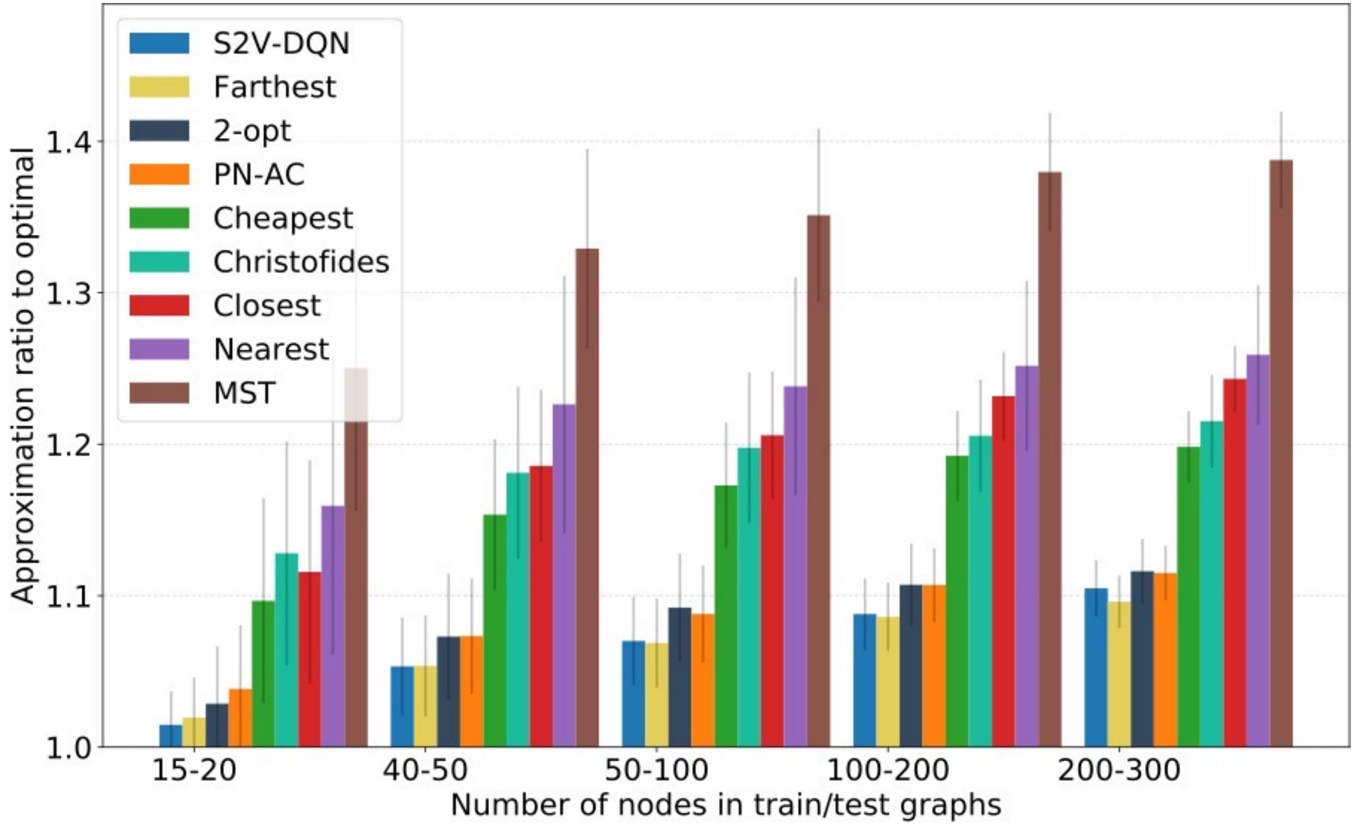
TSP

Uniform random points on 2-D grid

Paper's approach

- Initial subtour: 2 cities that are farthest apart

[Rosenkrantz et al., SIAM JoC'77]



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

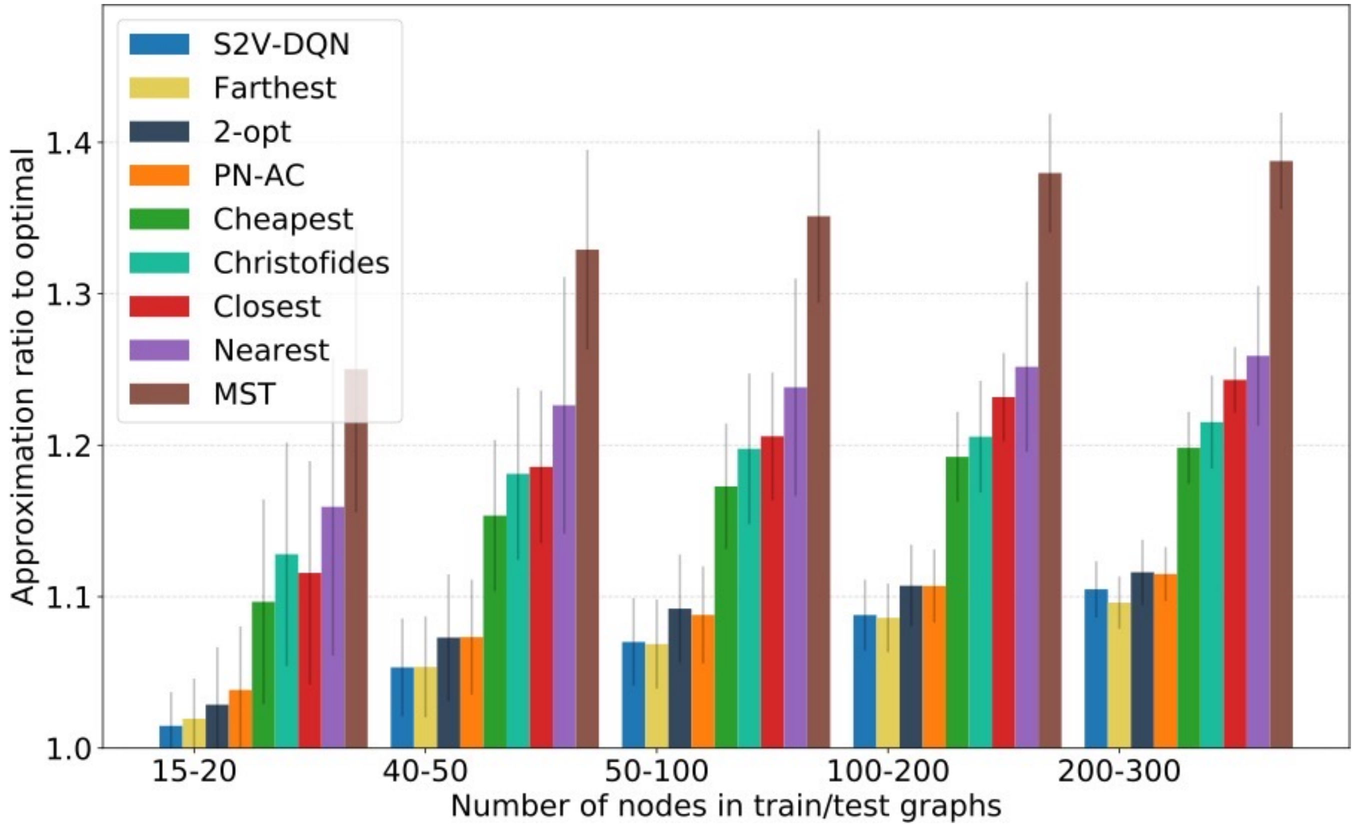
TSP

Uniform random points on 2-D grid

Paper's approach

- Initial subtour: 2 cities that are farthest apart
- Repeat the following:

[Rosenkrantz et al., SIAM JoC'77]



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

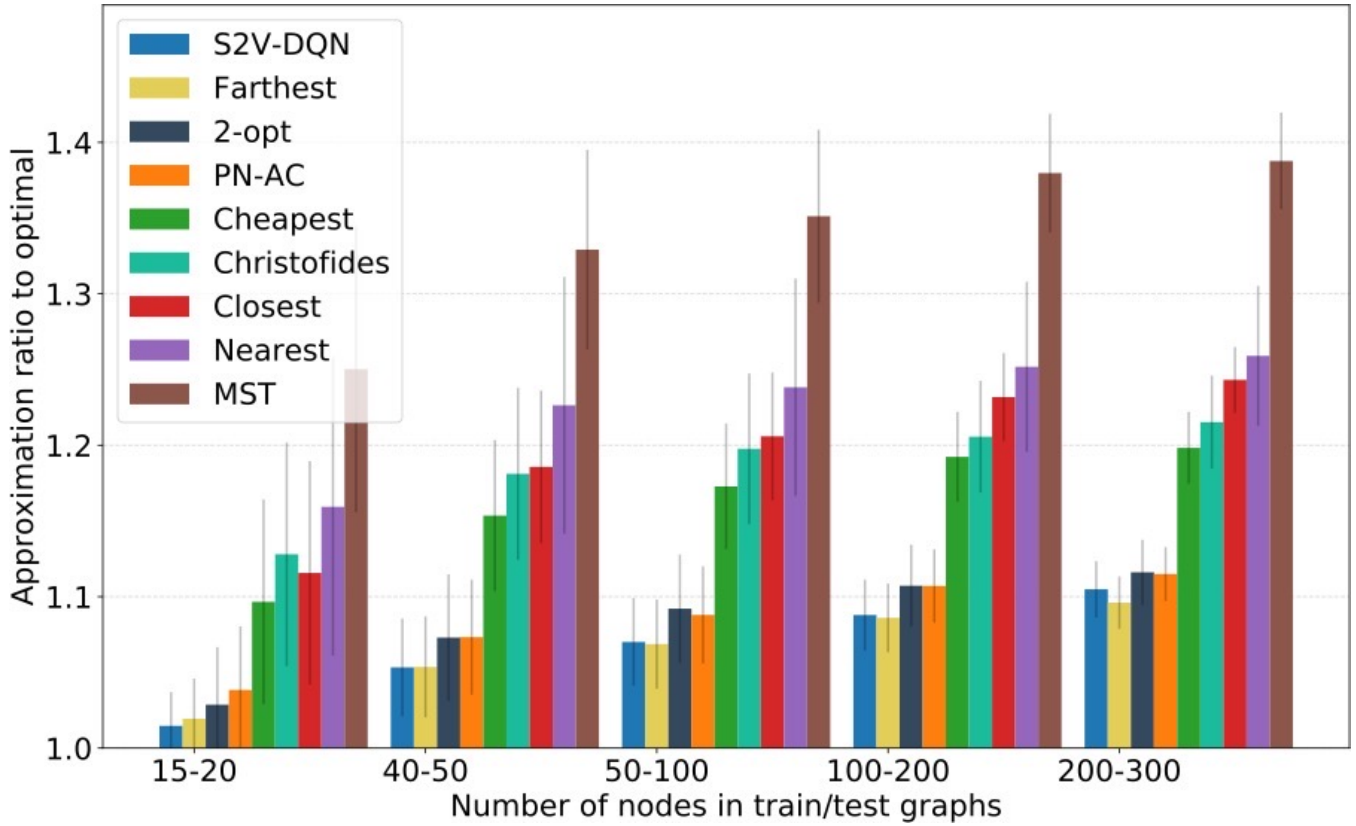
TSP

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

[Rosenkrantz et al., SIAM JoC'77]



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

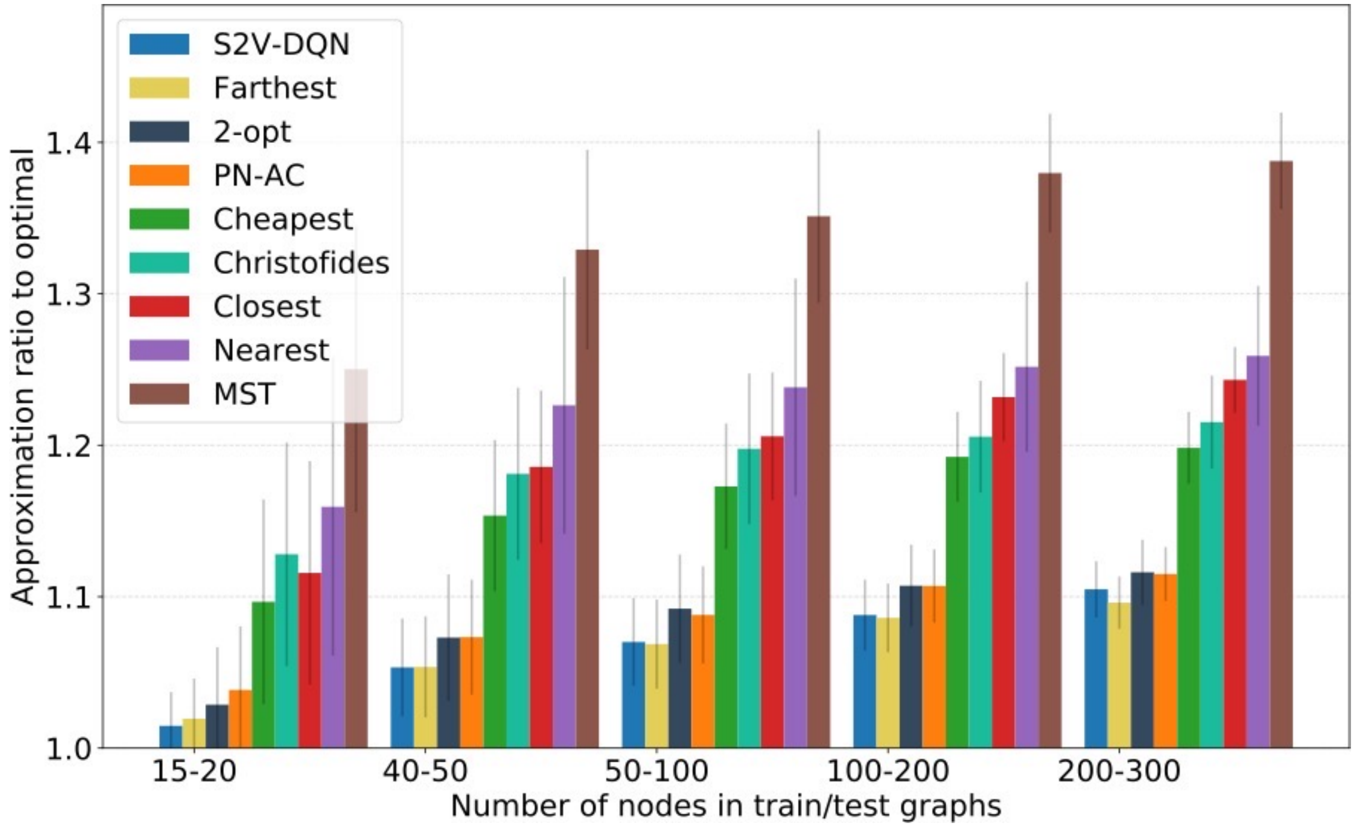
TSP

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, Zhang, Dilkina, Song; NeurIPS'17

Runtime comparisons

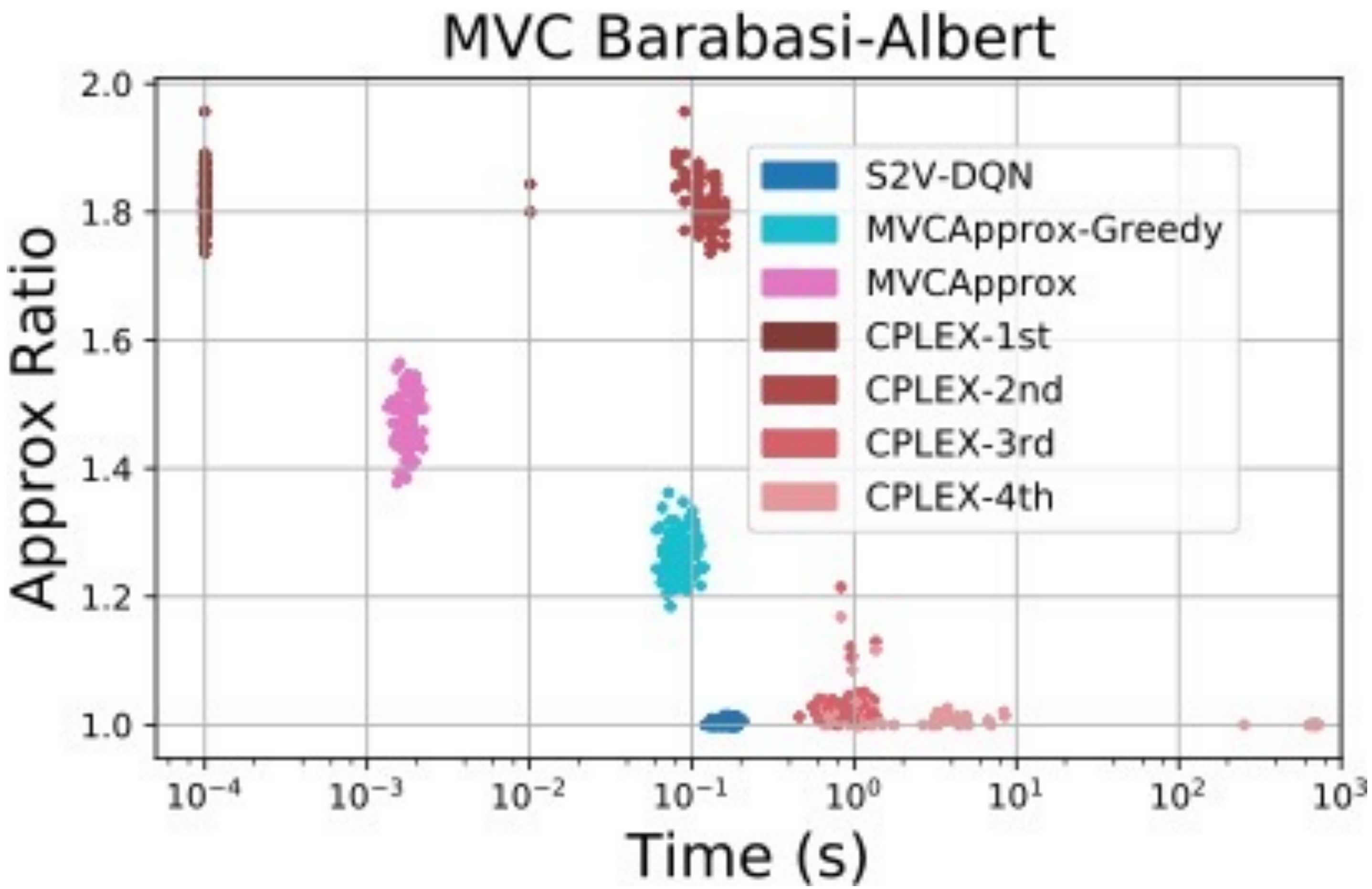
Paper's approach

Greedy algorithm from first few slides

2-approximation algorithm

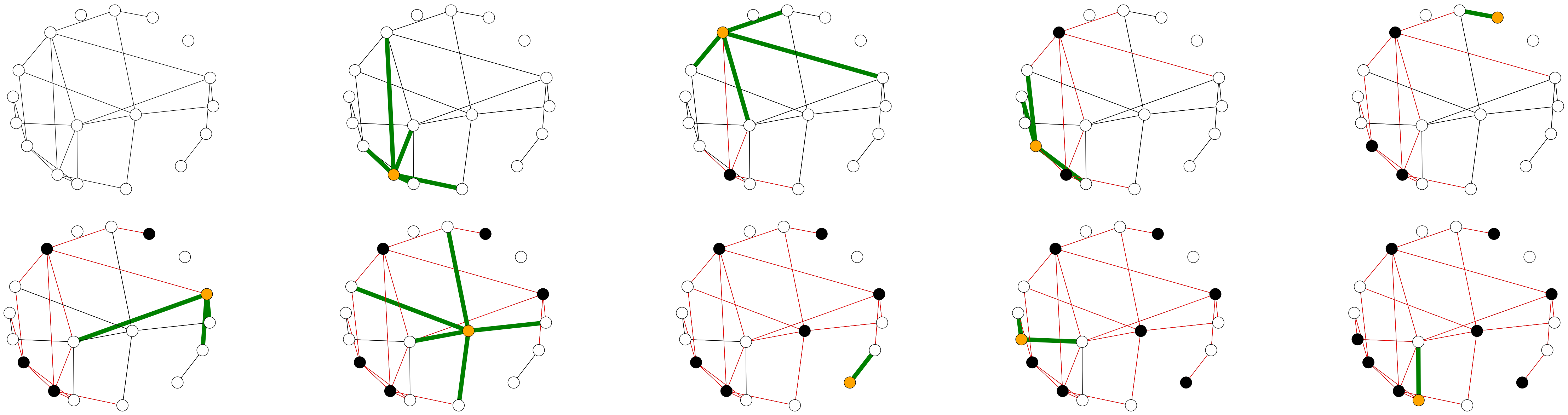
CPLEX-1st: 1st feasible solution found by CPLEX

CPLEX-2nd: 2nd feasible solution found by CPLEX



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

Min vertex cover visualization



Nodes seem to be selected to balance between:

- Degree
- Connectivity of the remaining graph

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

Summary

1 **Applied techniques**

a. Graph neural networks
    a. Neural algorithmic alignment
    b. Variable selection for integer programming
b. Learning greedy heuristics with RL

2 **After the break: Theoretical guarantees**

a. Statistical guarantees for algorithm configuration
b. Algorithms with predictions

Where much of my research has been

Summary

1 Applied techniques

a. Graph neural networks
   a. Neural algorithmic alignment
   b. Variable selection for integer programming
b. Learning greedy heuristics with RL

2 Theoretical guarantees

a. Statistical guarantees for algorithm configuration
b. Algorithms with predictions

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

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

Algorithm configuration

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

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

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**

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?

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**

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'

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

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'

$$\begin{array}{cccccc} A & - & - & C & T & G \\ - & G & T & C & A & - \end{array}$$

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'

A	–	–	C	T	G
–	G	T	C	A	–

Match

Sequence alignment algorithms

Input: Two sequences S and S'

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

Output: Alignment of S and S'

$$\begin{array}{cccccc} \mathrm{A} & - & - & \mathrm{C} & \mathrm{T} & \mathrm{G} \\ - & \mathrm{G} & \mathrm{T} & \mathrm{C} & \mathrm{A} & - \end{array}$$

Mismatch

Match

Sequence alignment algorithms

Input: Two sequences S and S'

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

Output: Alignment of S and S'

$$
\begin{array}{cccccc}
\mathtt{A} & - & - & \mathtt{C} & \mathtt{T} & \mathtt{G} \\
- & \mathtt{G} & \mathtt{T} & \mathtt{C} & \mathtt{A} & -
\end{array}
$$

Insertion/deletion (*indel*)

Match

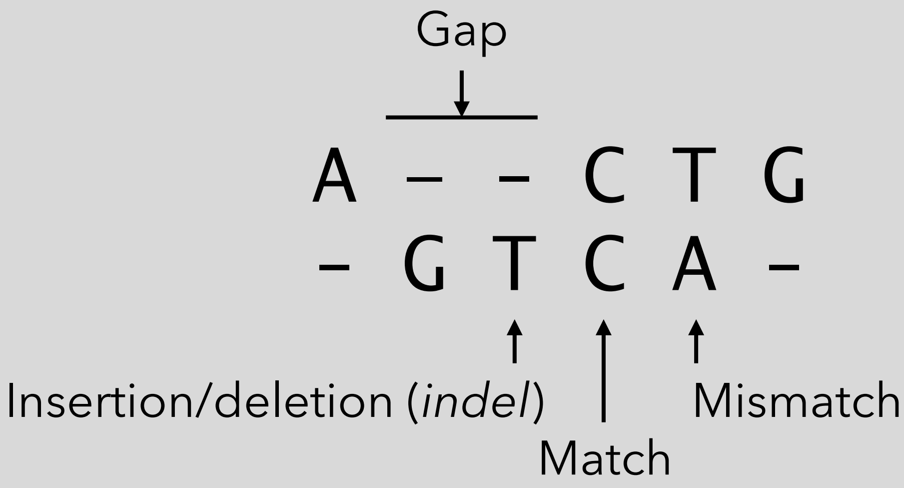
Mismatch

Sequence alignment algorithms

Input: Two sequences S and S'

Output: Alignment of S and S'

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



Sequence alignment algorithms

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

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

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:

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

$S =$ A C T G

$S' =$ G T C A

Gap

A – – C T G

– G T C A –

Insertion/deletion (*indel*)

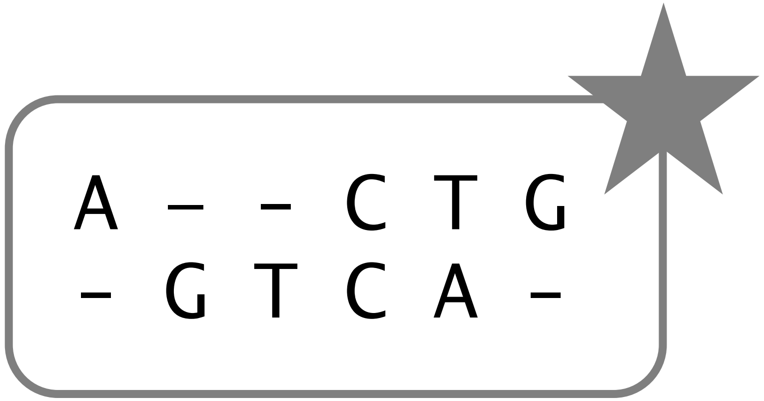
Match

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

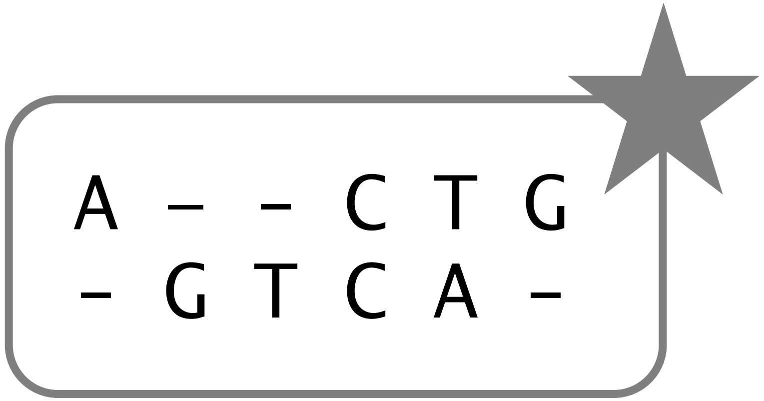


Sequence alignment algorithms

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



Sequence alignment algorithms

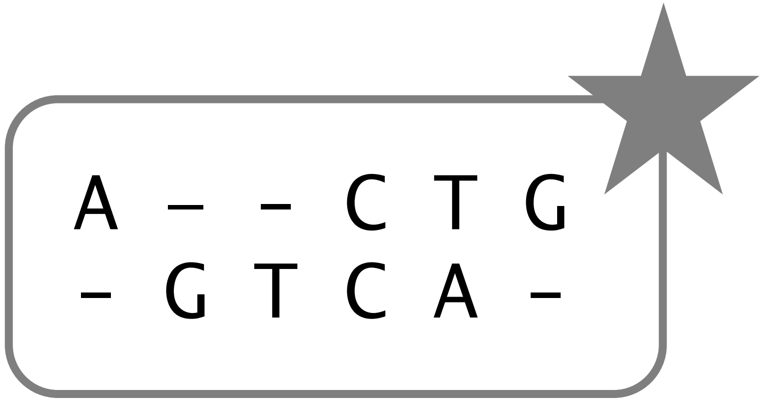
Can sometimes access **ground-truth, reference** alignment

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?



Sequence alignment algorithms

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

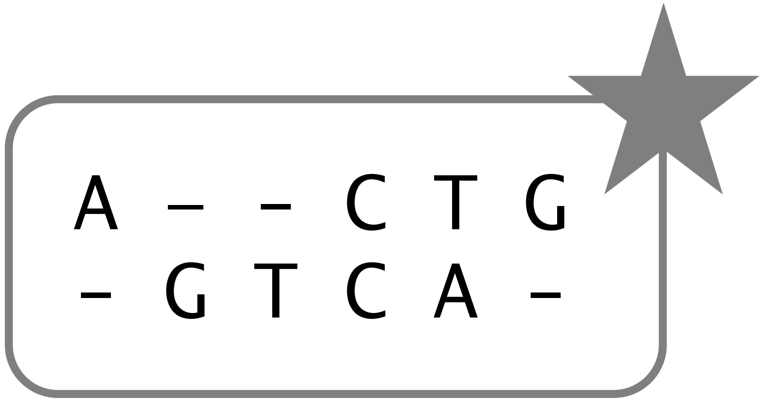
E.g., in computational biology: Bahr et al., Nucleic Acids Res.'01; Raghava et al., BMC Bioinformatics '03; Edgar, Nucleic Acids Res.'04; Walle et al., Bioinformatics'04

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]



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

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

…

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

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/ good avg performance 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]

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

Seen

Sequence S_1
Sequence S_1'
Reference alignment A_1

Sequence S_2
Sequence S_2'
Reference alignment A_2

...

Unseen

Sequence S
Sequence S'
Unknown alignment A

Key question (focus of this section):

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

Generalization

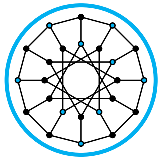
Key question **(focus of this section):**

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

Generalization

Key question (focus of this section):

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



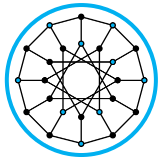
Greedy algorithms

Gupta, Roughgarden, ITCS'16

Generalization

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

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



Greedy algorithms

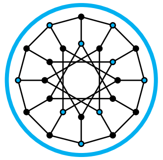
Gupta, Roughgarden, ITCS'16

First to ask question for algorithm configuration

Generalization

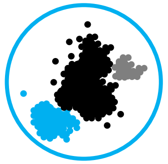
Key question **(focus of this section):**

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



Greedy algorithms

Gupta, Roughgarden, ITCS'16 ← First to ask question for algorithm configuration



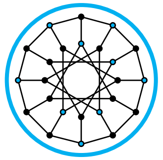
Clustering

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

Generalization

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

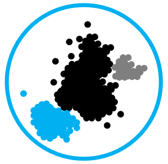
Good performance on **average** over **training set** implies good **future** performance?



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

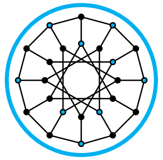
Search

Sakaue, Oki, NeurIPS'22

Generalization

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

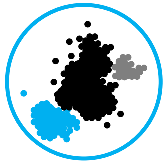
Good performance on **average** over **training set** implies good **future** performance?



Greedy algorithms

Gupta, Roughgarden, ITCS'16

First to ask question for algorithm configuration

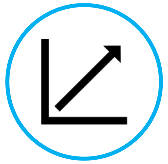


Clustering

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:

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

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, …

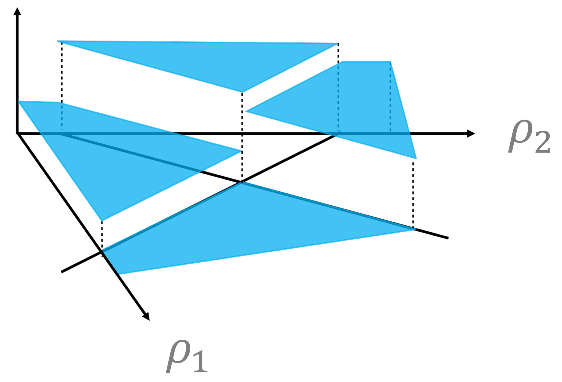
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

This section: Main result

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

Piecewise constant, linear, quadratic, …

Algorithmic performance on fixed input



Piecewise constant

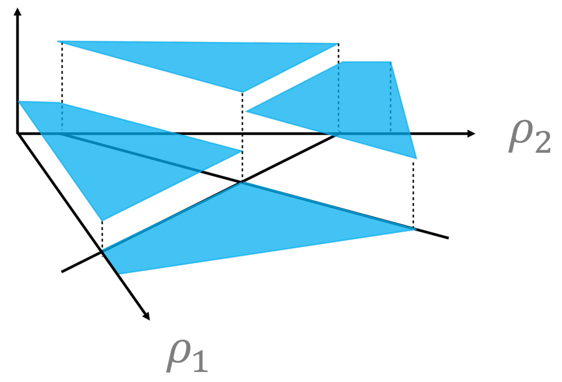
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

This section: Main result

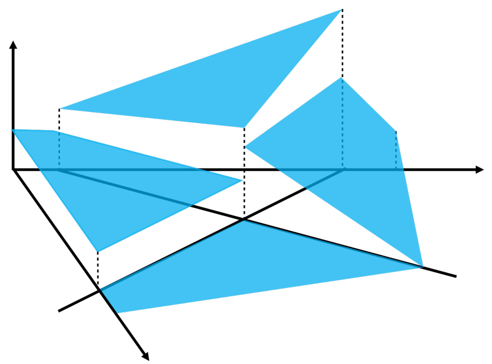
Performance is **piecewise-structured** function of parameters

Piecewise constant, linear, quadratic, …

Algorithmic performance on fixed input



Piecewise constant



Piecewise linear

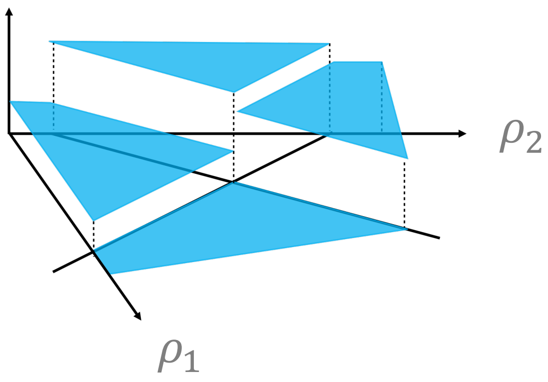
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

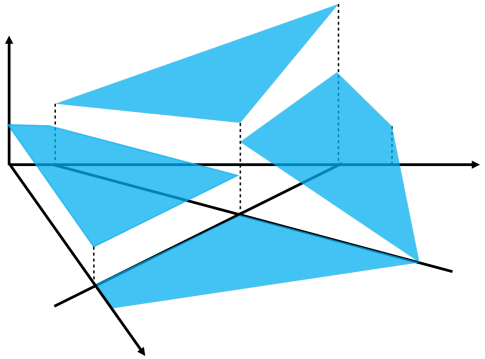
This section: Main result

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

Piecewise constant, linear, quadratic, …

Algorithmic performance on fixed input





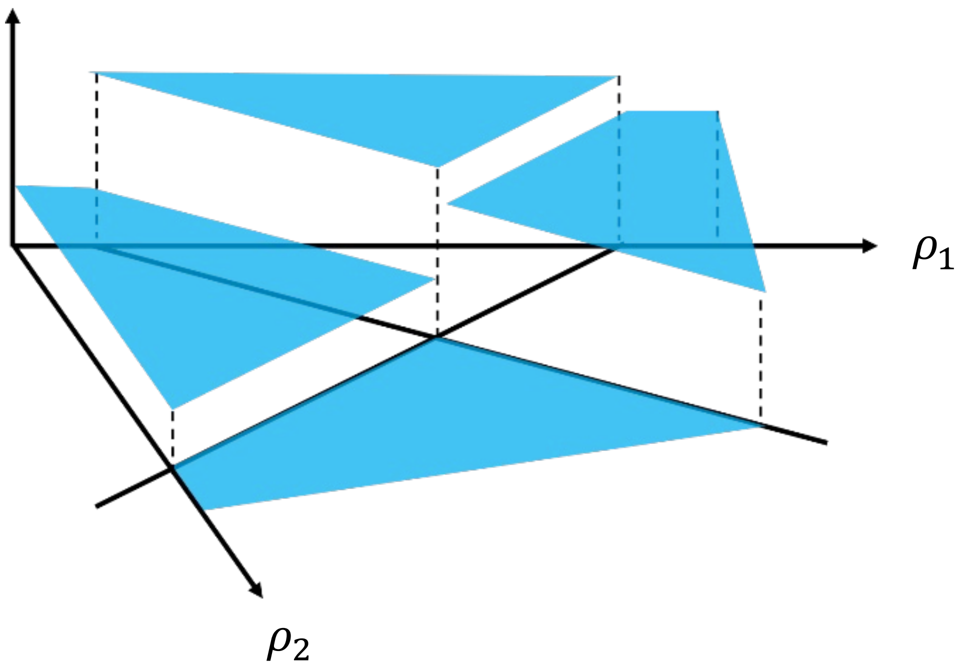
• • •

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

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

Example: Sequence alignment

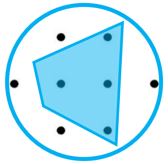
Distance between **algorithm's output** given S, S'
and **ground-truth** alignment is p-wise constant



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

Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:



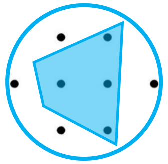
Integer programming

Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22
Balcan, Dick, Sandholm, **V**, JACM'24

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

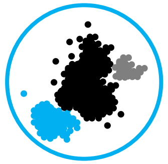
Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:



Integer programming

Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22
Balcan, Dick, Sandholm, **V**, JACM'24



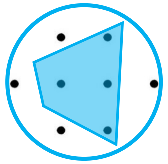
Clustering

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

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

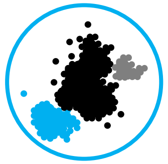
Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:



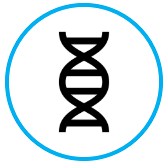
Integer programming

Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22
Balcan, Dick, Sandholm, **V**, JACM'24



Clustering

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



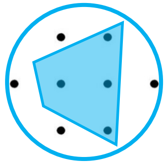
Computational biology

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

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

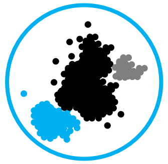
Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:



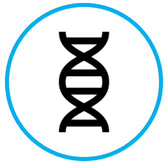
Integer programming

Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22
Balcan, Dick, Sandholm, **V**, JACM'24



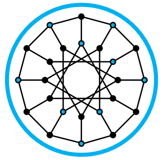
Clustering

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

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

Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:

Integer programming
Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22
Balcan, Dick, Sandholm, **V**, JACM'24

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**, OR'24

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

Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:

Integer programming

Balcan, Prasad, Sandholm, **V**, NeurIPS'21
Balcan, Prasad, Sandholm, **V**, NeurIPS'22
Balcan, Dick, Sandholm, **V**, JACM'24

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**, OR'24

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

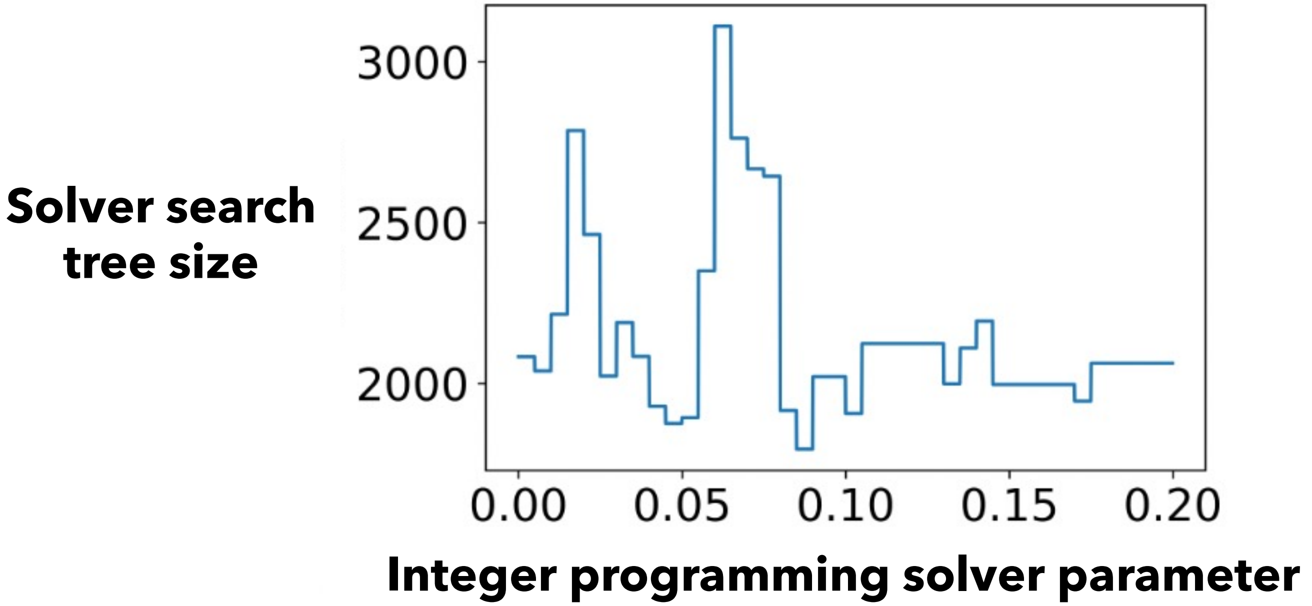
Primary challenge

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

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

Primary challenge

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

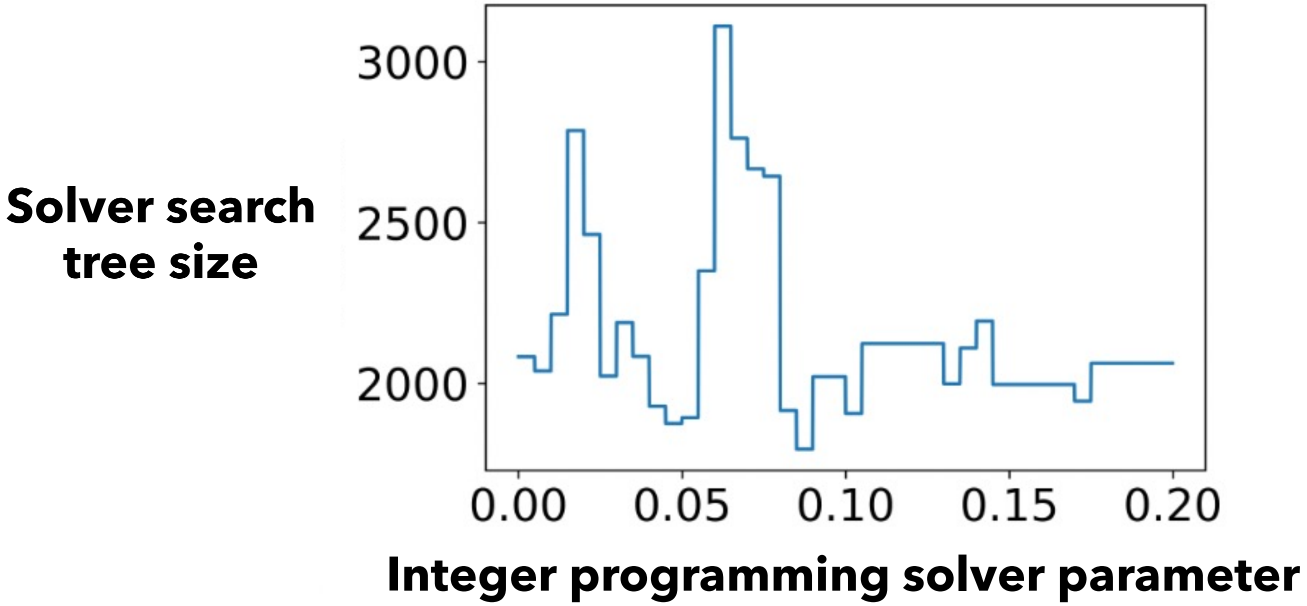


Balcan, Prasad, Sandholm, **Vitercik**, NeurIPS'21

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. Application: Sequence alignment
   v. Online algorithm configuration
2. Algorithms with predictions

Model

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

Model

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

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

Example: Sequence alignment

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

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

Example: Sequence alignment

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

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

$$\begin{array}{cccccc} S & = & A & C & T & G \\ S' & = & G & T & C & A \end{array}$$

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

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, …

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

Model

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

Distribution models specific application domain at hand

Model

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

Distribution models specific application domain at hand

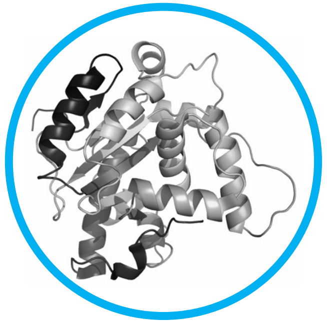
E.g., distribution over pairs of DNA strands

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 $\boldsymbol{\rho}$,

Generalization bounds

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

Generalization bounds

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

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 $\boldsymbol{\rho}$,

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 $\boldsymbol{\rho}$,

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

Generalization bounds

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

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

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

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. Application: Sequence alignment
   v. Online algorithm configuration
2. Algorithms with predictions

Sequence alignment algorithms

Lemma:

For any pair S, S', there's a partition of $\mathbb{R}^3$ s.t. in any region,

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

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

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

$$\begin{array}{lcccc} S = & A & C & T & G \\ S' = & G & T & C & A \end{array}$$

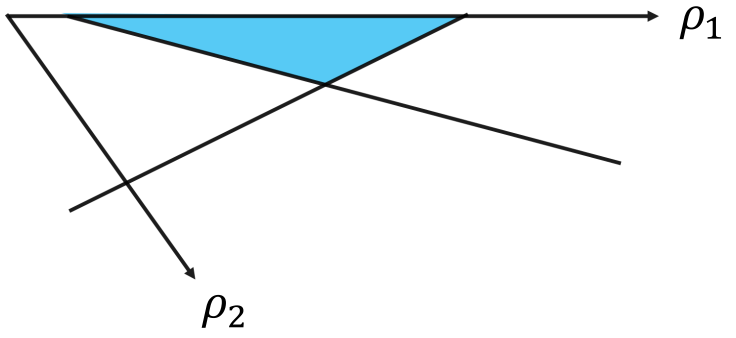
Gusfield et al., Algorithmica '94; Fernández-Baca et al., J. of Discrete Alg. '04

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

$$\begin{array}{l} S \;=\; \mathrm{A\;\;C\;\;T\;\;G} \\ S' = \mathrm{G\;\;T\;\;C\;\;A} \end{array}$$



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

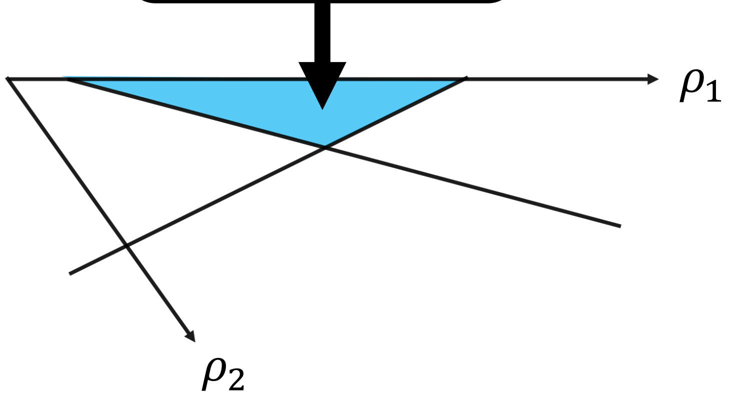
Sequence alignment algorithms

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

A – – C T G
– G T C A –



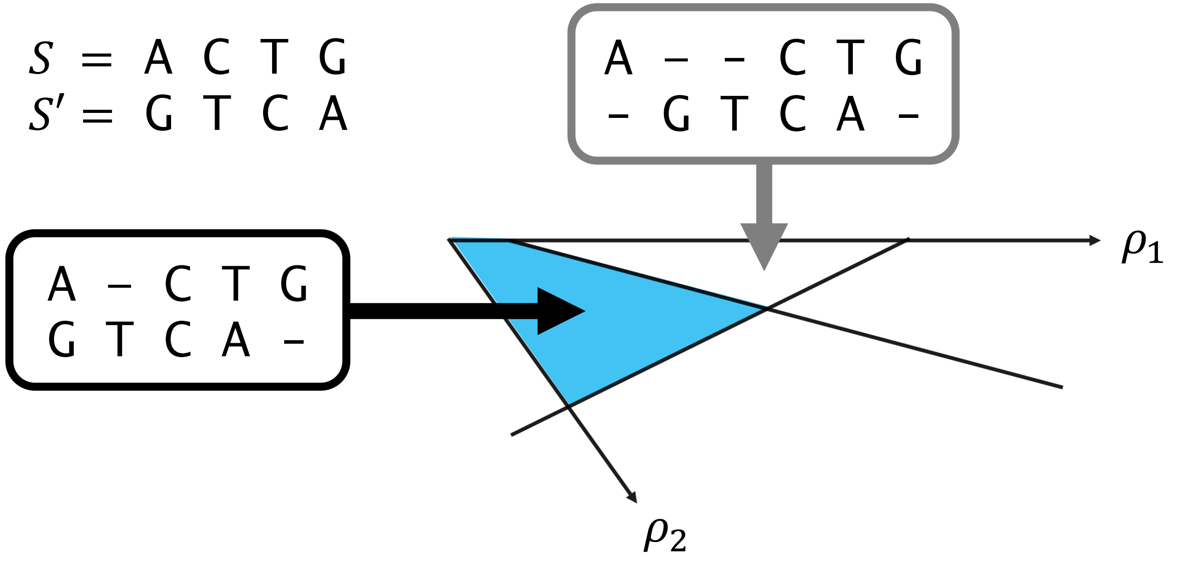
Gusfield et al., Algorithmica '94; Fernández-Baca et al., J. of Discrete Alg. '04

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

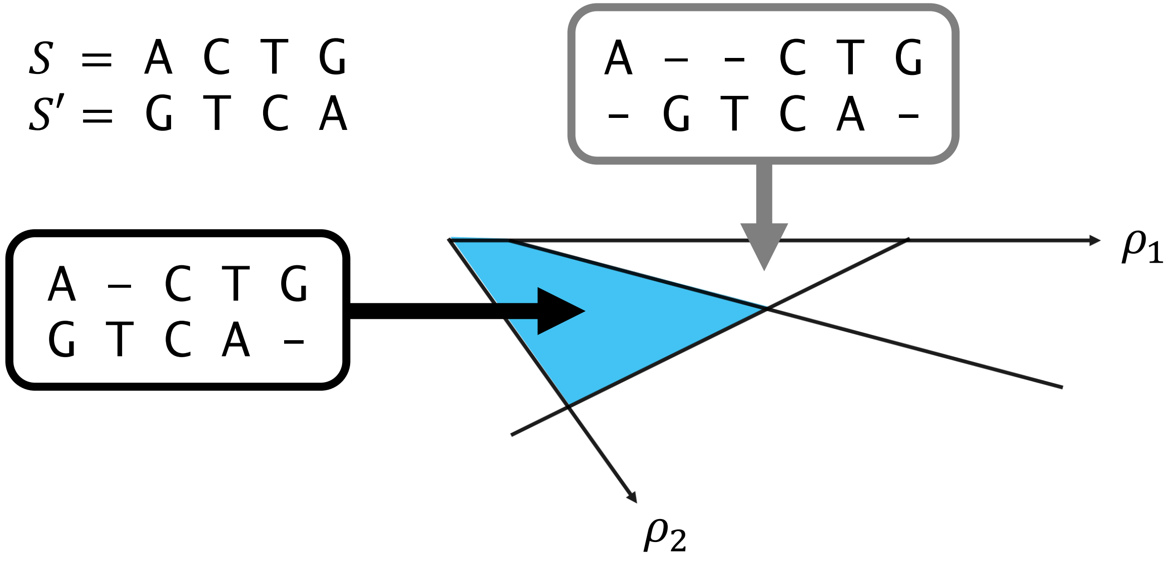
Sequence alignment algorithms

Lemma:

$$\underbrace{\text{partition of } \mathbb{R}^3}_{\text{Defined by } (\max\{|S|, |S'|\})^3 \text{ hyperplanes}}$$

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



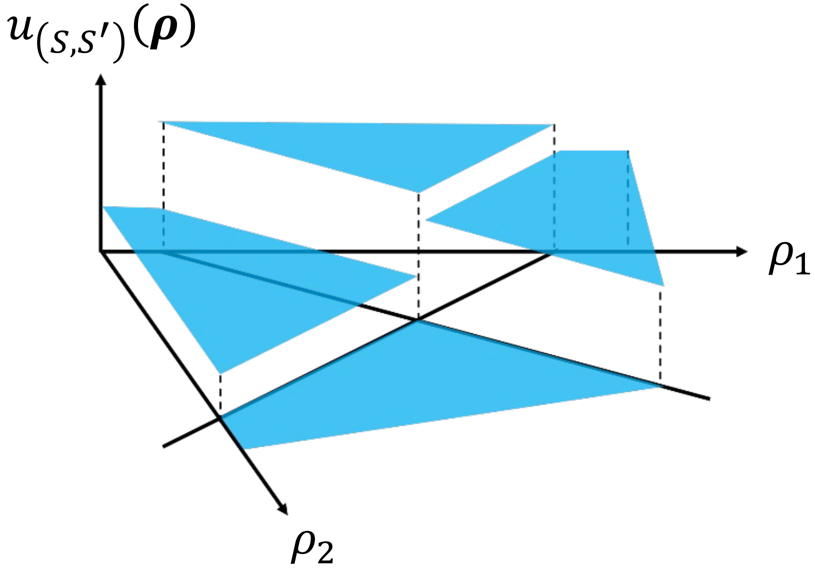
Gusfield et al., Algorithmica '94; Fernández-Baca et al., J. of Discrete Alg. '04

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. Application: Sequence alignment
   v. Online algorithm configuration
2. Algorithms with predictions

Primal & dual classes

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

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

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

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

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

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

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

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

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

Challenge: $\mathcal{U}$ is gnarly

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

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

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

Challenge: $\mathcal{U}$ is gnarly

E.g., in sequence alignment:

- Each domain element is a pair of sequences

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

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

Challenge: $\mathcal{U}$ is gnarly

E.g., in sequence alignment:
- Each domain element is a pair of sequences
- Unclear how to plot or visualize functions $u_{\boldsymbol{\rho}}$

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

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

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

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

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

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

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

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

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

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

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

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

- Dual functions have simple, Euclidean domain

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

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

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

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

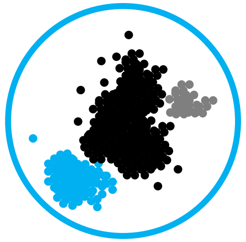
$\mathcal{U}^* = \left\{u_x^*: \mathbb{R}^d \to \mathbb{R} \;\middle|\; x \in \mathcal{X}\right\}$ "Dual" function class

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

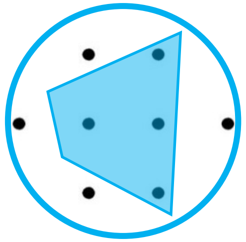
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Piecewise-structured functions

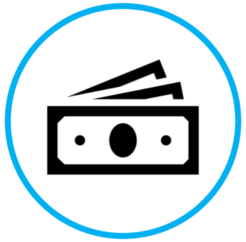
Dual functions $u_x^*: \mathbb{R}^d \to \mathbb{R}$ are **piecewise-structured**



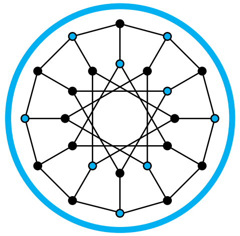
Clustering algorithm configuration



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. Application: Sequence alignment
   v. Online algorithm configuration
2. Algorithms with predictions

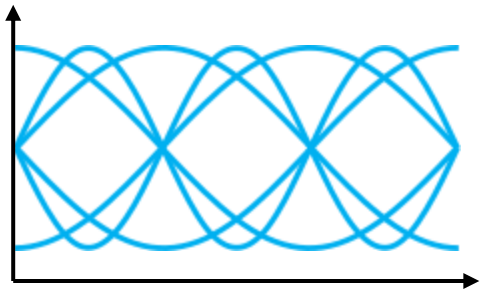
Intrinsic complexity

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

Intrinsic complexity

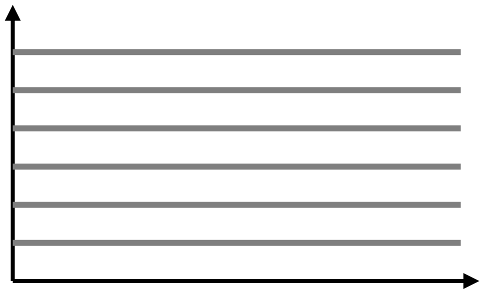
"Intrinsic complexity" of function class $\mathcal{G}$

- Measures how well functions in $\mathcal{G}$ fit complex patterns



More complex

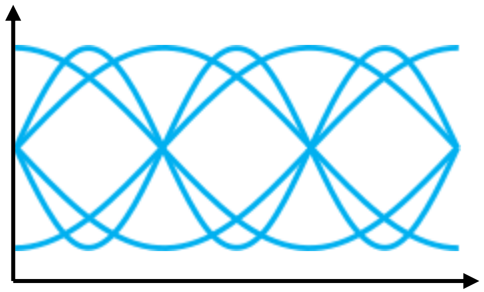
Less complex



Intrinsic complexity

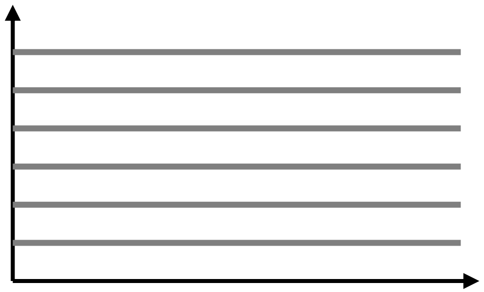
"Intrinsic complexity" of function class $\mathcal{G}$

- 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 $\mathcal{F}$

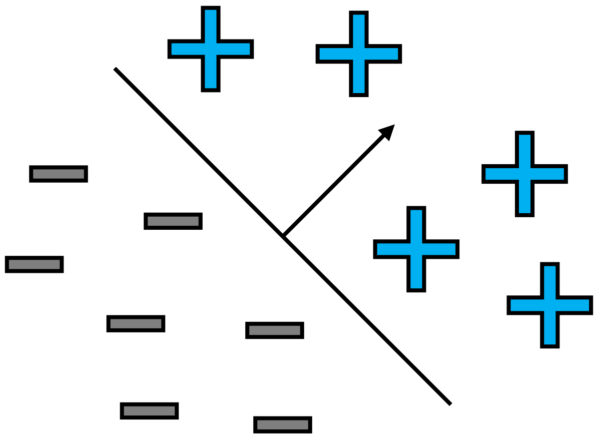
(Classes of functions $f: \mathcal{Y} \to \{-1,1\}$)

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 $\mathcal{S} \subseteq \mathcal{Y}$

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 $\mathcal{F}$

VC dimension

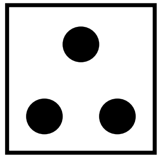
Size of the largest set $\mathcal{S} \subseteq \mathcal{Y}$
that can be labeled in all $2^{|\mathcal{S}|}$ ways by functions in $\mathcal{F}$

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 $\mathcal{F}$

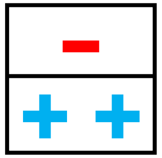
Example: $\mathcal{F}$ = Linear separators in $\mathbb{R}^2$ $\quad$ $\text{VCdim}(\mathcal{F}) \geq 3$



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$(\mathcal{F}) \geq 3$

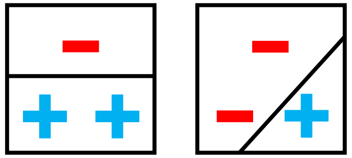


VC dimension

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

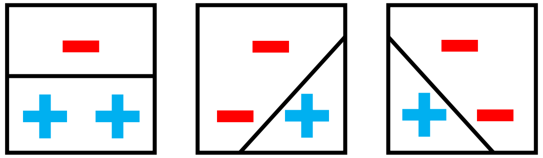


VC dimension

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 $\mathbb{R}^2$

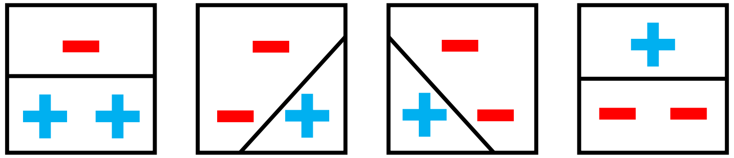
VCdim($\mathcal{F}$) ≥ 3



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 $\mathcal{F}$

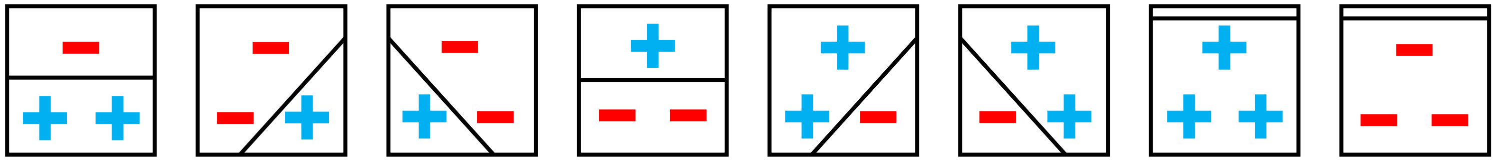
Example: $\mathcal{F}$ = Linear separators in $\mathbb{R}^2$ VCdim($\mathcal{F}$) ≥ 3



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

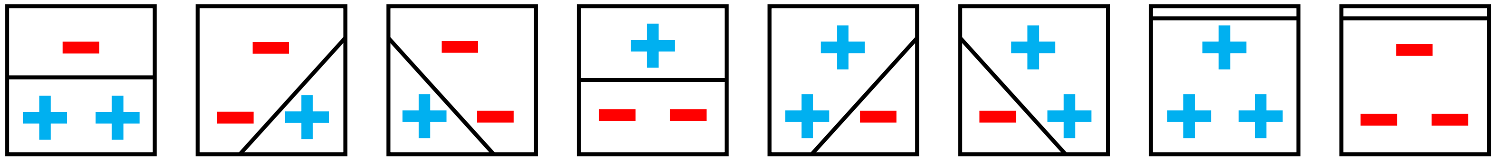


VC dimension

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

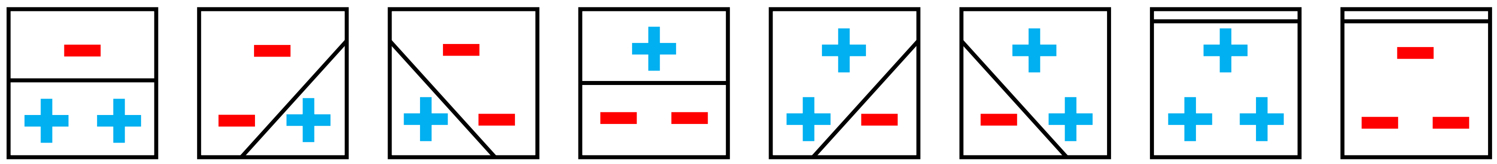


VCdim($\mathcal{F}$) ≤ 3

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 $\mathbb{R}^d$}) $= d + 1$

Pseudo-dimension

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

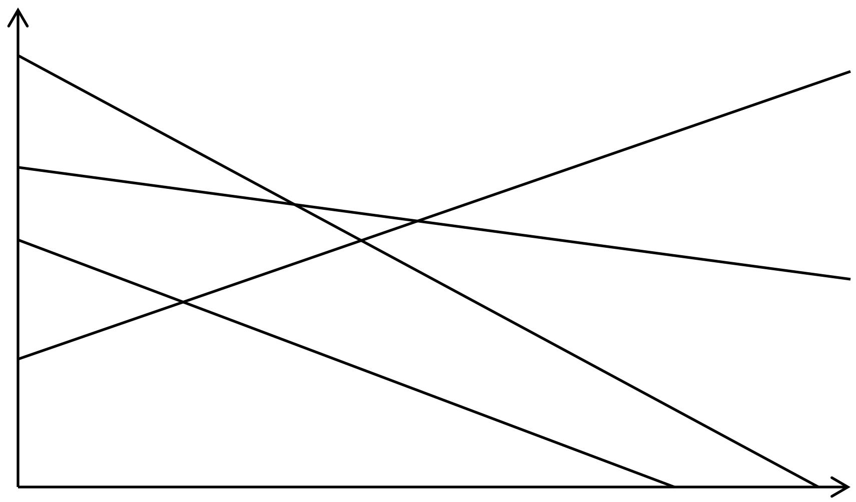
(Classes of functions $g: \mathcal{Y} \to [-1,1]$)

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 $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ s.t.:

Pseudo-dimension of $\mathcal{G}$

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

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

Pseudo-dimension of $\mathcal{G}$

Size of the largest set $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ s.t.:
for some *targets* $z_1, \ldots, z_N \in \mathbb{R}$,
all 2^N above/below patterns achieved by functions in $\mathcal{G}$

Pseudo-dimension of $\mathcal{G}$

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

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

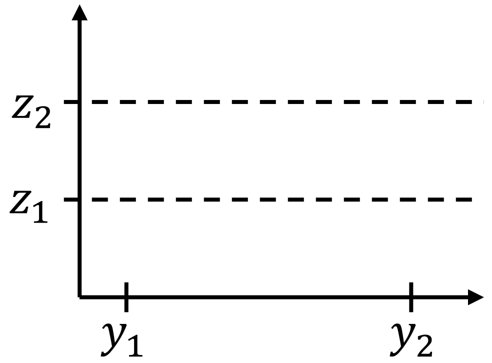
    all 2^N above/below patterns achieved by functions in $\mathcal{G}$

Example: $\mathcal{G}$ = Affine functions in $\mathbb{R}$ Pdim($\mathcal{G}$) ≥ 2

Pseudo-dimension of $\mathcal{G}$

Size of the largest set $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ s.t.:
for some *targets* $z_1, \ldots, z_N \in \mathbb{R}$,
all 2^N above/below patterns achieved by functions in $\mathcal{G}$

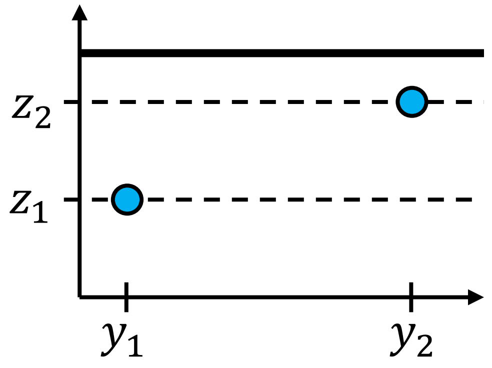
Example: $\mathcal{G}$ = Affine functions in $\mathbb{R}$ $\quad$ Pdim$(\mathcal{G}) \geq 2$



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 $\mathcal{G}$

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



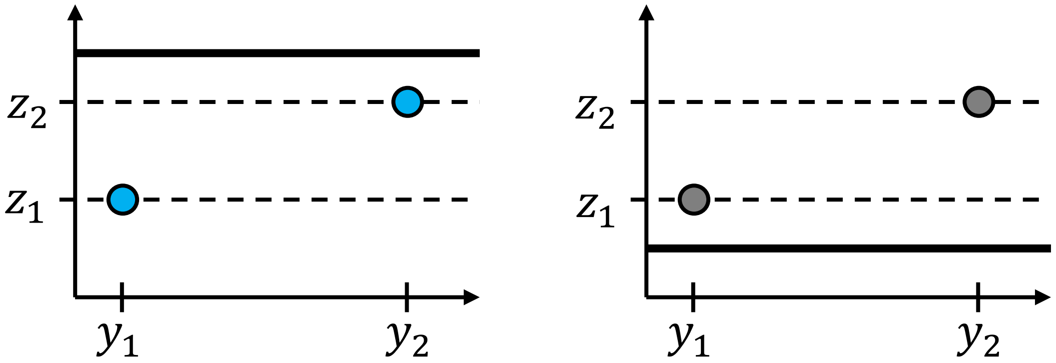
Pseudo-dimension of $\mathcal{G}$

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

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

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

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



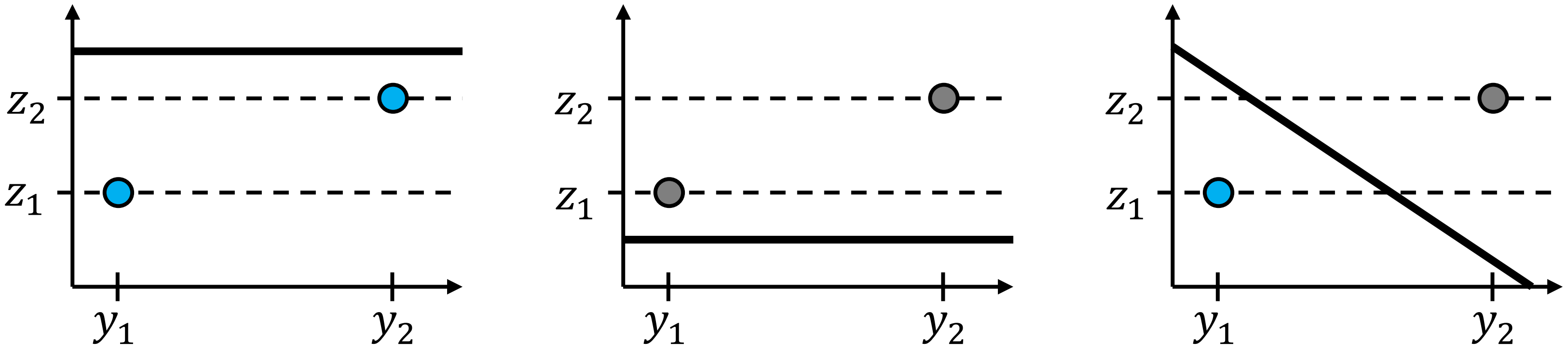
Pseudo-dimension of $\mathcal{G}$

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 $\mathcal{G}$

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



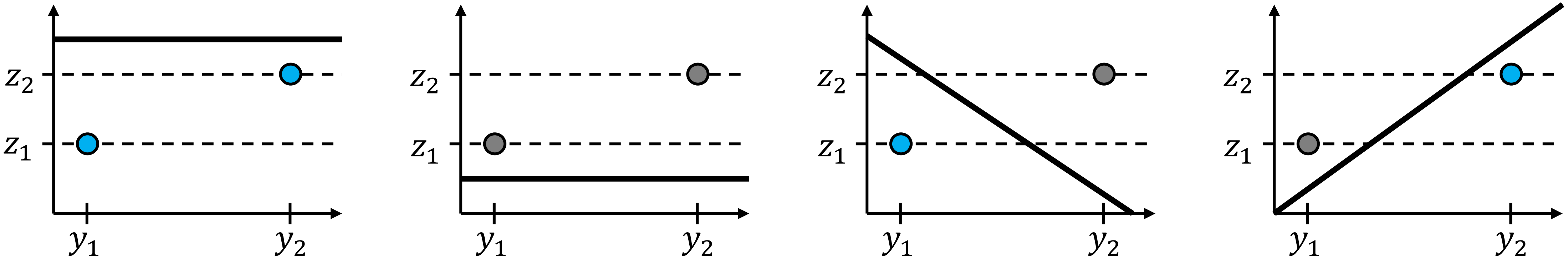
Pseudo-dimension of $\mathcal{G}$

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 $\mathcal{G}$

Example: $\mathcal{G}$ = Affine functions in $\mathbb{R}$ Pdim($\mathcal{G}$) ≥ 2



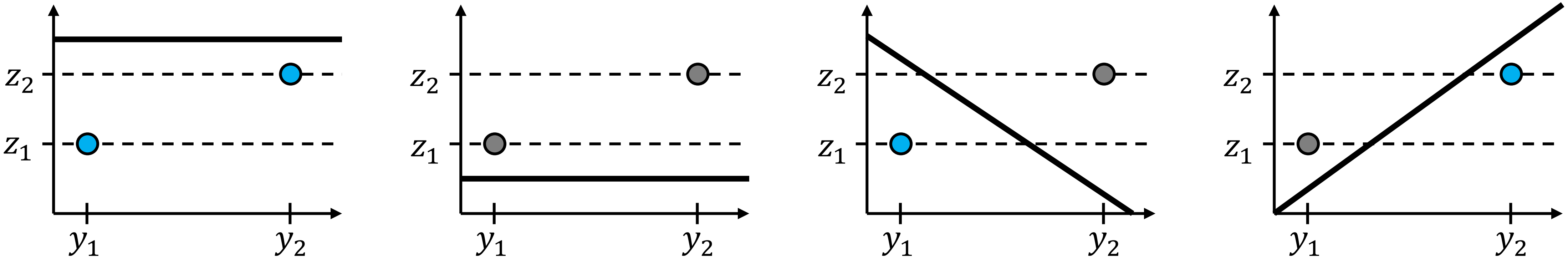
Pseudo-dimension of $\mathcal{G}$

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

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

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

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



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

Sample complexity using pseudo-dim

In the context of **algorithm configuration**:

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**

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 $N = O\left(\frac{\mathrm{Pdim}(\mathcal{U})}{\epsilon^2} \log \frac{1}{\delta}\right)$

Sample complexity using pseudo-dim

In the context of **algorithm configuration**:

- $\mathcal{U} = \left\{u_{\boldsymbol{\rho}}: \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ 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 $x_1, \ldots, x_N \sim \mathcal{D}, \forall \boldsymbol{\rho} \in \mathbb{R}^d$,

Sample complexity using pseudo-dim

In the context of **algorithm configuration**:

- $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ 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 $x_1, \ldots, x_N \sim \mathcal{D}, \forall \boldsymbol{\rho} \in \mathbb{R}^d$,

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

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 $N = O\left(\frac{\text{Pdim}(\mathcal{U})}{\epsilon^2} \log \frac{1}{\delta}\right)$
- With probability at least $1 - \delta$ over $x_1, \dots, x_N \sim \mathcal{D}, \forall \boldsymbol{\rho} \in \mathbb{R}^d$,

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

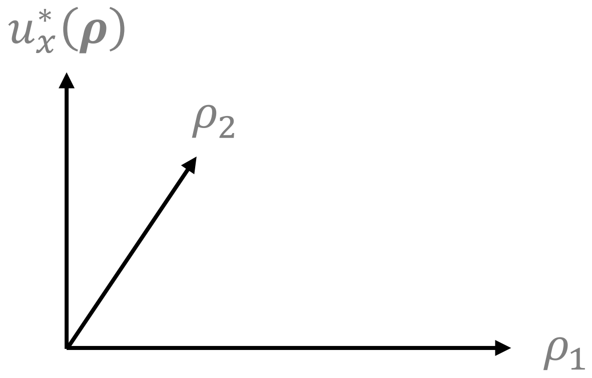
Sample complexity using pseudo-dim

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 $N = O\left(\frac{\text{Pdim}(\mathcal{U})}{\epsilon^2} \log \frac{1}{\delta}\right)$
- With probability at least $1 - \delta$ over $x_1, \ldots, x_N \sim \mathcal{D}, \forall \boldsymbol{\rho} \in \mathbb{R}^d$,

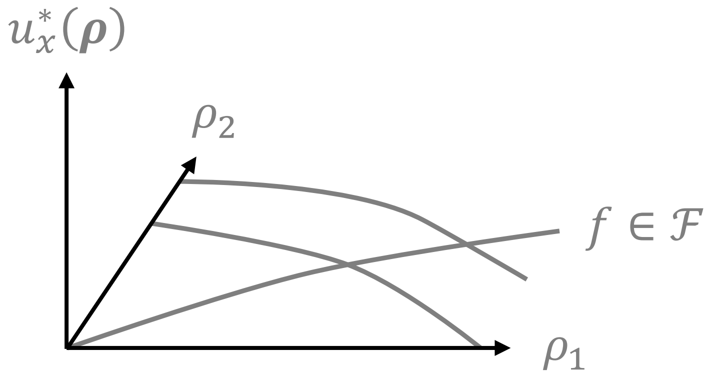
$$\left| \underbrace{\frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(x_i)}_{\text{Empirical average utility}} - \underbrace{\mathbb{E}_{x \sim \mathcal{D}}\left[u_{\boldsymbol{\rho}}(x)\right]}_{\text{Expected utility}} \right| \leq \epsilon$$

Main result (informal)



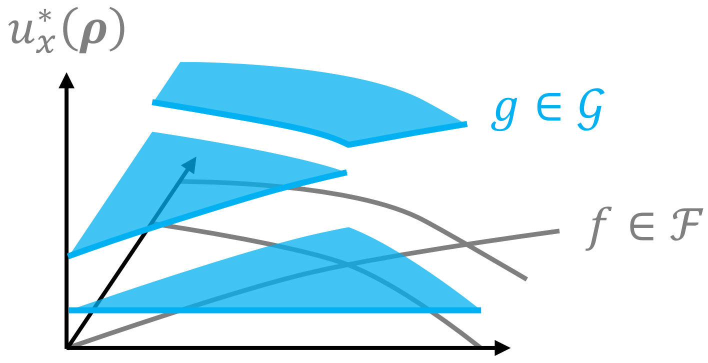
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Main result (informal)



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

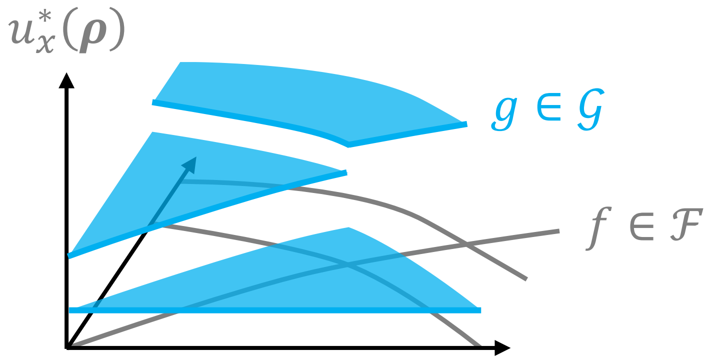
Main result (informal)



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,

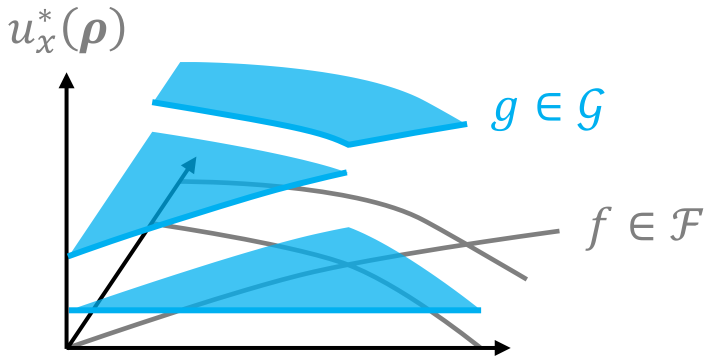


Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

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 $g \in \mathcal{G}$.

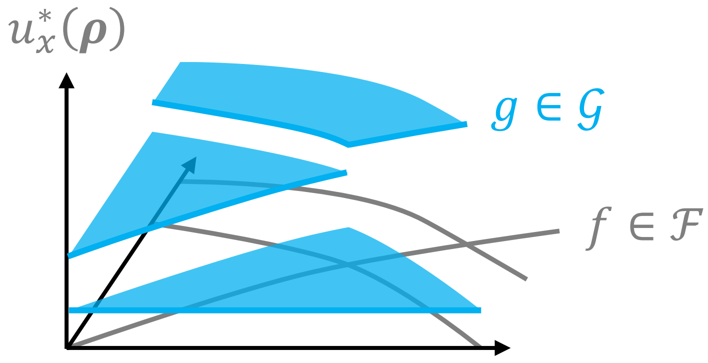


Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

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 $g \in \mathcal{G}$.

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

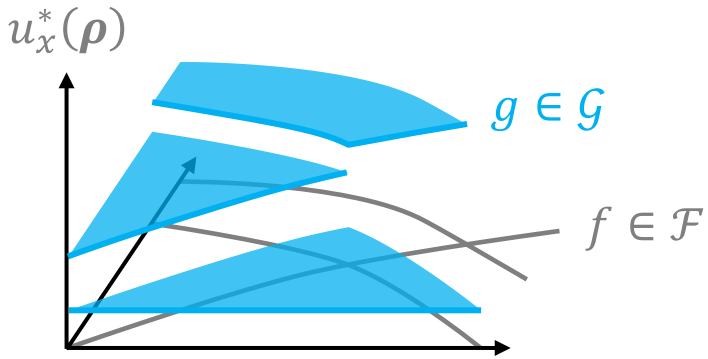


Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

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 $g \in \mathcal{G}$.

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 $f_1, \dots, f_k \in \mathcal{F}$ 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 $\tilde{O}\left(\frac{\text{Pdim}(\mathcal{G}^*)+\text{VCdim}(\mathcal{F}^*)\log k}{\epsilon^2}\right)$ implies
WHP $\forall \boldsymbol{\rho}$, |**avg** utility over training set – **exp** utility| $\leq \epsilon$

$\mathcal{F}, \mathcal{G}$ are typically very well structured

- $\mathcal{G}$ = set of all constant functions $\Rightarrow$ Pdim$(\mathcal{G}^*) = O(1)$
- $\mathcal{G}$ = set of all linear functions in $\mathbb{R}^d$ $\Rightarrow$ Pdim$(\mathcal{G}^*) = O(d)$

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

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

Main result (informal)

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

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

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

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

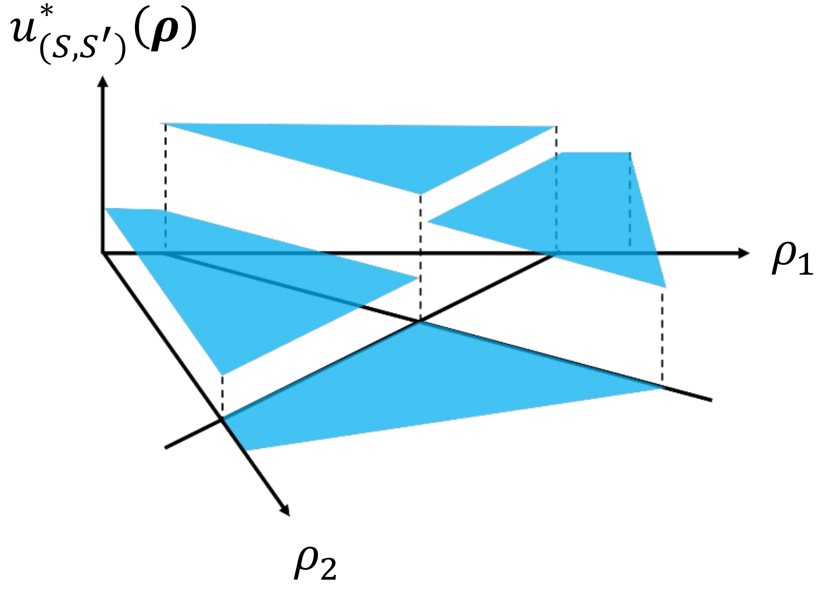
Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
   i. Model
   ii. Piecewise-structured algorithmic performance
   iii. Main result
   iv. **Application: Sequence alignment**
   v. Online algorithm configuration
2. Algorithms with predictions

Piecewise constant dual functions

Lemma:

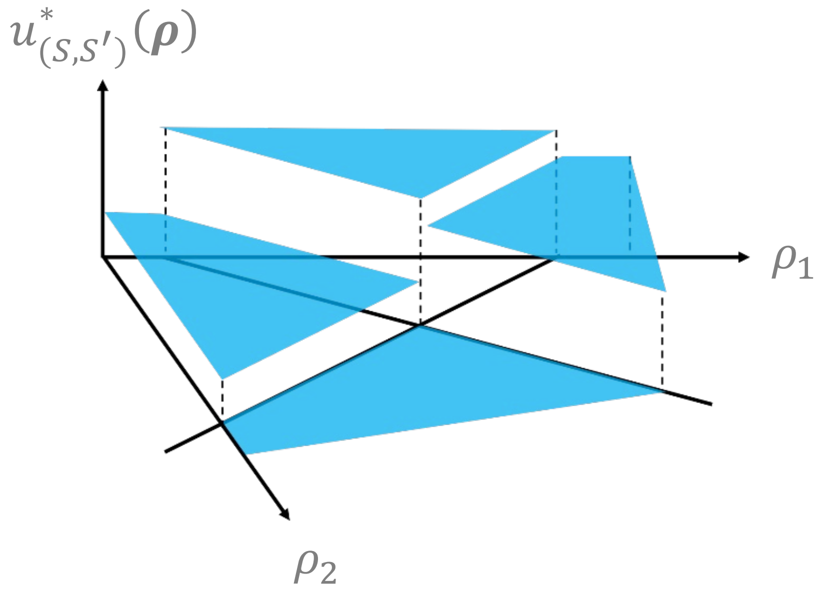
Utility is piecewise constant function of parameters



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

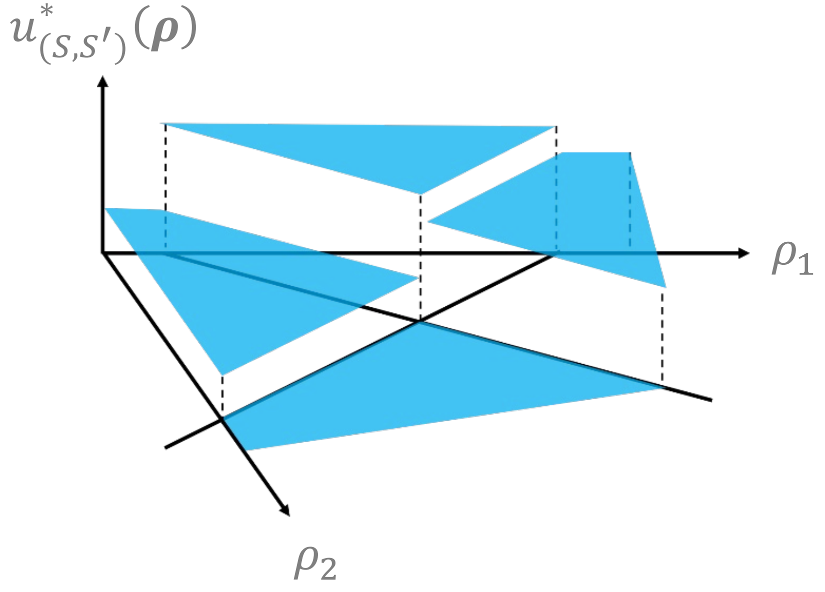
Sequence alignment guarantees

Theorem: Training set of size $\tilde{O}\left(\frac{\log(\text{seq. length})}{\epsilon^2}\right)$ implies WHP $\forall \boldsymbol{\rho}$,



Sequence alignment guarantees

Theorem: Training set of size $\tilde{O}\left(\frac{\log(\text{seq. length})}{\epsilon^2}\right)$ implies WHP $\forall \boldsymbol{\rho}$,
|**avg** utility over training set – **exp** utility| $\leq \epsilon$

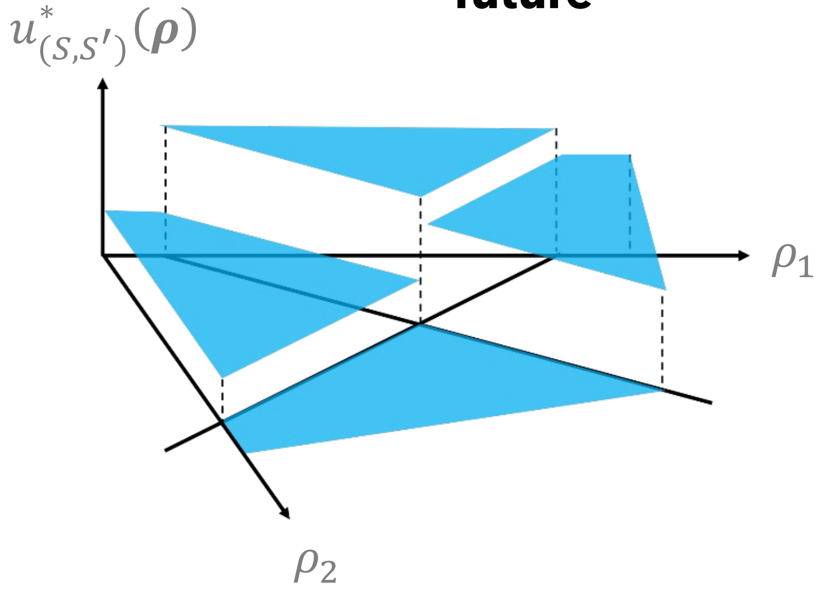


Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

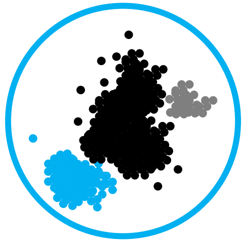
Sequence alignment guarantees

Theorem: Training set of size $\tilde{O}\left(\frac{\log(\text{seq. length})}{\epsilon^2}\right)$ implies WHP $\forall \boldsymbol{\rho}$,

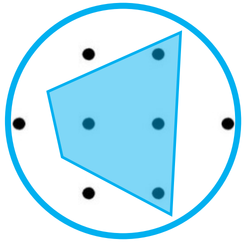
$|$**avg** utility over training set – $\underbrace{\textbf{exp}}_{\textbf{future}}$ utility$| \leq \epsilon$



Many more applications

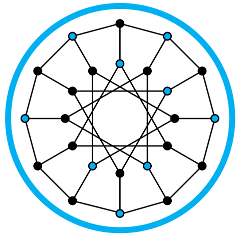


Clustering algorithm configuration

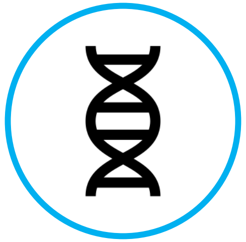


Integer programming algorithm configuration

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. Application: Sequence alignment
   v. Online algorithm configuration
2. Algorithms with predictions

Online algorithm configuration

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

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

Online algorithm configuration

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

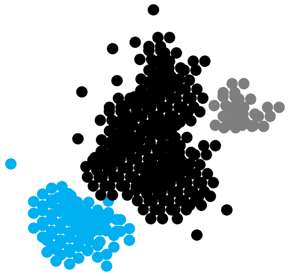
Day 1: $\boldsymbol{\rho}_1$

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

Online algorithm configuration

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

Day 1: $\boldsymbol{\rho}_1$

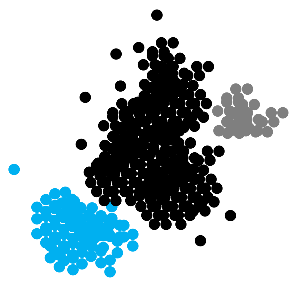


Gupta and Roughgarden, ITCS'16; Cohen-Addad, Kanade AISTATS'17; Balcan, Dick, **Vitercik**, FOCS'18; …

Online algorithm configuration

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

Day 1: $\boldsymbol{\rho}_1$ Day 2: $\boldsymbol{\rho}_2$

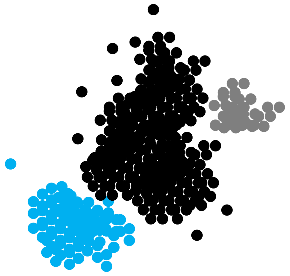


Gupta and Roughgarden, ITCS'16; Cohen-Addad, Kanade AISTATS'17; Balcan, Dick, **Vitercik**, FOCS'18; …

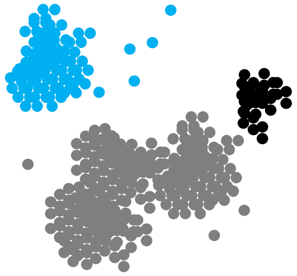
Online algorithm configuration

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

Day 1: $\boldsymbol{\rho}_1$



Day 2: $\boldsymbol{\rho}_2$

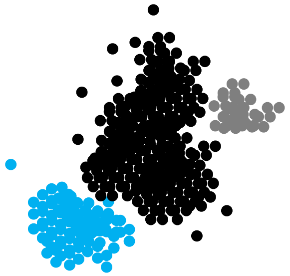


Gupta and Roughgarden, ITCS'16; Cohen-Addad, Kanade AISTATS'17; Balcan, Dick, **Vitercik**, FOCS'18; …

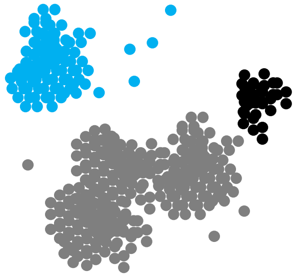
Online algorithm configuration

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

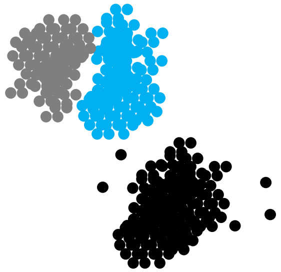
Day 1: $\boldsymbol{\rho}_1$



Day 2: $\boldsymbol{\rho}_2$



Day 3: $\boldsymbol{\rho}_3$

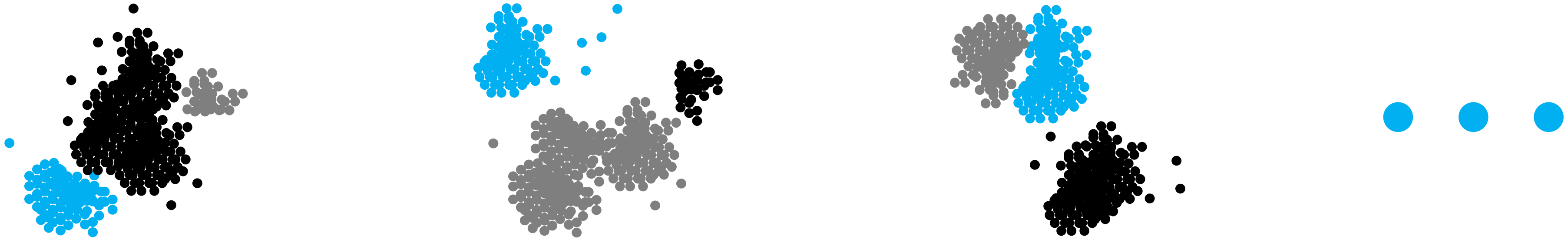


Gupta and Roughgarden, ITCS'16; Cohen-Addad, Kanade AISTATS'17; Balcan, Dick, **Vitercik**, FOCS'18; …

Online algorithm configuration

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

Day 1: $\boldsymbol{\rho}_1$ Day 2: $\boldsymbol{\rho}_2$ Day 3: $\boldsymbol{\rho}_3$



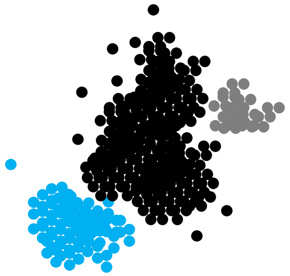
Goal: Compete with best parameter setting in hindsight

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

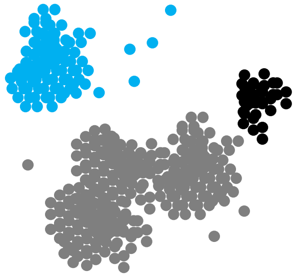
Online algorithm configuration

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

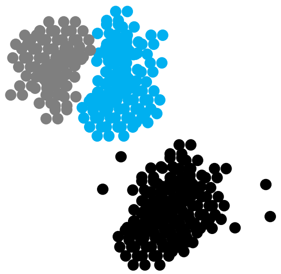
Day 1: $\boldsymbol{\rho}_1$



Day 2: $\boldsymbol{\rho}_2$



Day 3: $\boldsymbol{\rho}_3$



...

Goal: Compete with best parameter setting in hindsight

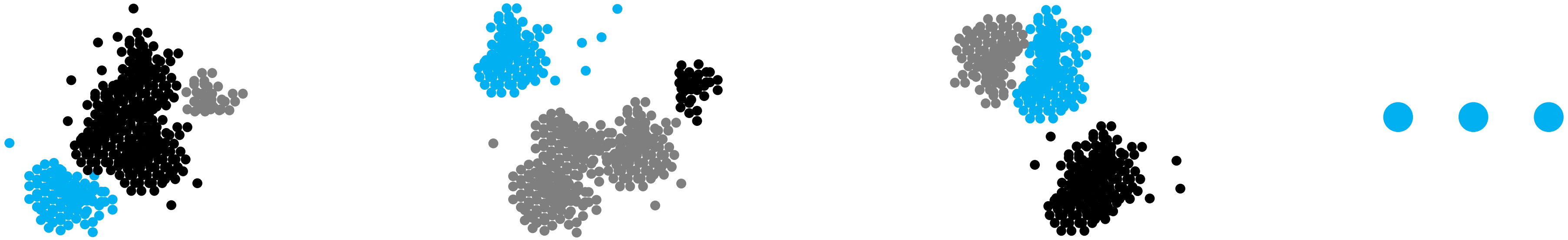
- Impossible in the worst case

Gupta and Roughgarden, ITCS'16; Cohen-Addad, Kanade AISTATS'17; Balcan, Dick, **Vitercik**, FOCS'18; ...

Online algorithm configuration

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

Day 1: $\boldsymbol{\rho}_1$ Day 2: $\boldsymbol{\rho}_2$ Day 3: $\boldsymbol{\rho}_3$



Goal: Compete with best parameter setting in hindsight

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

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

Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. **Algorithms with predictions**

Algorithms with predictions

Assume you have some **predictions** about your problem, e.g.:

Algorithms with predictions

Assume you have some **predictions** about your problem, e.g.:

Probability any given element is in a huge database

Kraska et al., SIGMOD'18; Mitzenmacher, NeurIPS'18

Algorithms with predictions

Assume you have some **predictions** about your problem, e.g.:

Probability any given element is in a huge database

Kraska et al., SIGMOD'18; Mitzenmacher, NeurIPS'18

In caching, the next time you'll see an element

Lykouris, Vassilvitskii, ICML'18

Algorithms with predictions

Assume you have some **predictions** about your problem, e.g.:

Probability any given element is in a huge database

Kraska et al., SIGMOD'18; Mitzenmacher, NeurIPS'18

In caching, the next time you'll see an element

Lykouris, Vassilvitskii, ICML'18

Main question:

How to use predictions to improve algorithmic performance?

Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Algorithms with predictions
    a. **Searching a sorted array**
    b. Online algorithms
    c. Additional research

Example: Searching in a sorted array

- **Goal:** Given query q & sorted array A, find q's index (if q in A)

Book chapter by Mitzenmacher, Vassilvitskii, '20

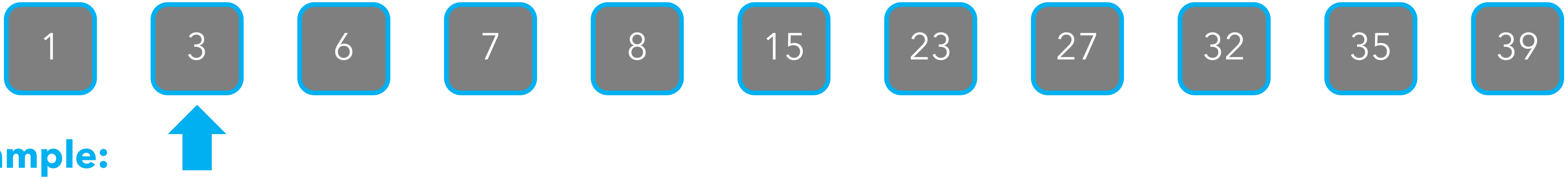
Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

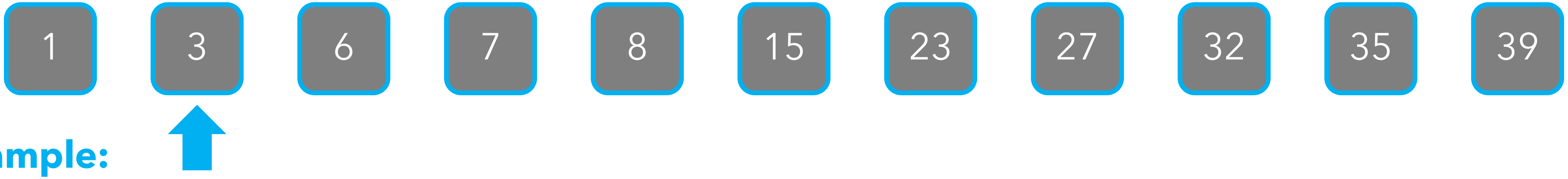


Example:
- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array



Example:

- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index
- **Algorithm:** Check $A[h(q)]$. If q is there, return $h(q)$. Else:

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

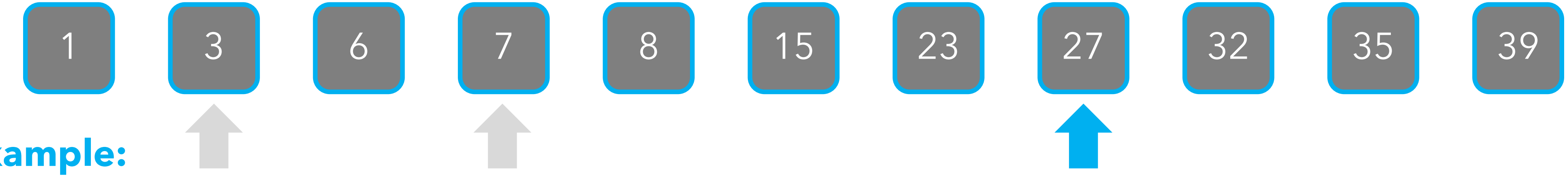
Example:

- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index
- **Algorithm:** Check $A[h(q)]$. If q is there, return $h(q)$. Else:
  - If $q > A[h(q)]$, check $A\left[h(q) + 2^i\right]$ for $i > 1$ until find something larger

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

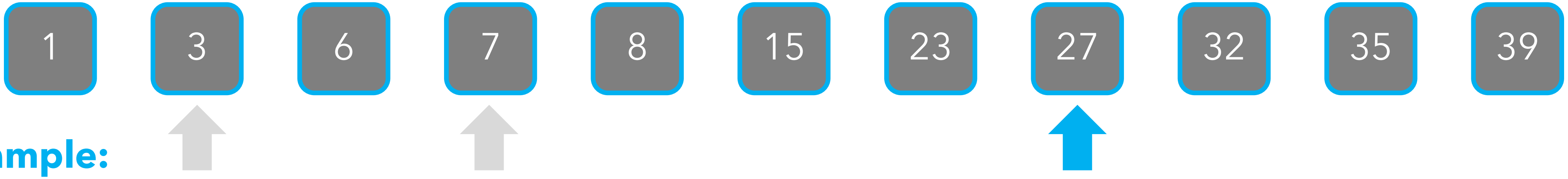


Example:
- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index
- **Algorithm:** Check $A[h(q)]$. If q is there, return $h(q)$. Else:
  - If $q > A[h(q)]$, check $A\left[h(q) + 2^i\right]$ for $i > 1$ until find something larger

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

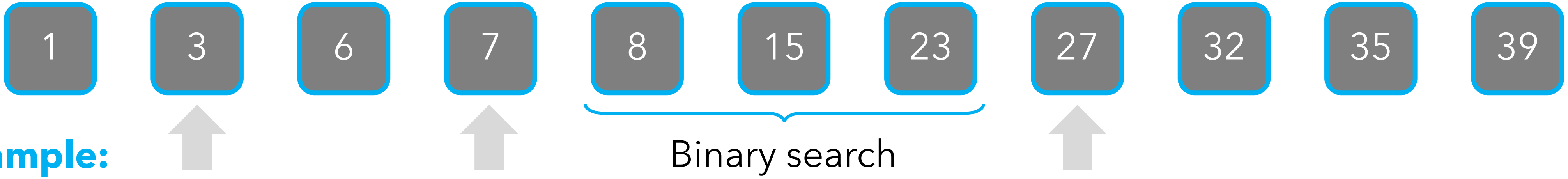


Example:

- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index
- **Algorithm:** Check $A[h(q)]$. If q is there, return $h(q)$. Else:
  - If $q > A[h(q)]$, check $A\left[h(q) + 2^i\right]$ for $i > 1$ until find something larger
    - Do binary search on interval $\left(h(q) + 2^{i-1}, h(q) + 2^i\right)$

Example: Searching in a sorted array



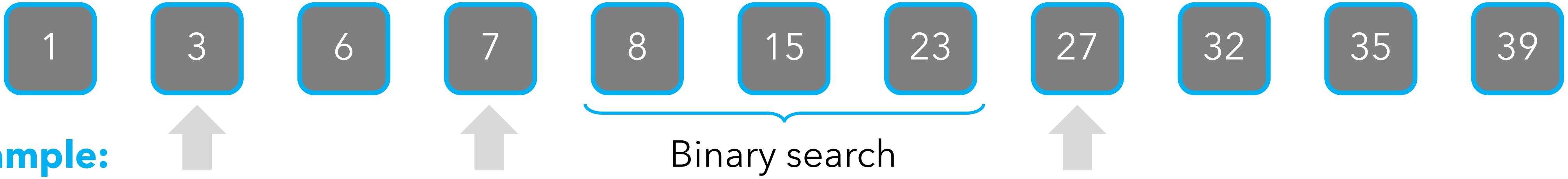
Example:

- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index
- **Algorithm:** Check $A[h(q)]$. If q is there, return $h(q)$. Else:
  - If $q > A[h(q)]$, check $A\left[h(q) + 2^i\right]$ for $i > 1$ until find something larger
    - Do binary search on interval $\left(h(q) + 2^{i-1}, h(q) + 2^i\right)$

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array



Example:

- $q = 8$
- $h(q) = 2$

- **Goal:** Given query q & sorted array A, find q's index (if q in A)
- **Predictor:** $h(q)$ = guess of q's index
- **Algorithm:** Check $A[h(q)]$. If q is there, return $h(q)$. Else:
  - If $q > A[h(q)]$, check $A\left[h(q) + 2^i\right]$ for $i > 1$ until find something larger
    - Do binary search on interval $\left(h(q) + 2^{i-1}, h(q) + 2^i\right)$
  - If $q < A[h(q)]$, symmetric

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

Analysis:

- Let $t(q)$ be index of q in A or of smallest element larger than q

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

Analysis:

- Let $t(q)$ be index of q in A or of smallest element larger than q
- Runtime is $O(\log \underbrace{|t(q) - h(q)|}_{\text{Prediction error}})$:

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

Analysis:

- Let $t(q)$ be index of q in A or of smallest element larger than q
- Runtime is $O(\log\underbrace{|t(q) - h(q)|}_{\text{Prediction error}})$:
  - Finding larger/smaller element takes $O(\log|t(q) - h(q)|)$ steps

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

Analysis:

- Let $t(q)$ be index of q in A or of smallest element larger than q
- Runtime is $O(\log|\underbrace{t(q) - h(q)}_{\text{Prediction error}}|)$:
  - Finding larger/smaller element takes $O(\log|t(q) - h(q)|)$ steps
  - Binary search takes $O(\log|t(q) - h(q)|)$ steps

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

1 | 3 | 6 | 7 | 8 | 15 | 23 | 27 | 32 | 35 | 39

Analysis:

- Let $t(q)$ be index of q in A or of smallest element larger than q
- Runtime is $O(\log\underbrace{|t(q) - h(q)|}_{\text{Prediction error}})$:
  - Finding larger/smaller element takes $O(\log|t(q) - h(q)|)$ steps
  - Binary search takes $O(\log|t(q) - h(q)|)$ steps
- Better predictions lead to **better runtime**

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

Analysis:

- Let $t(q)$ be index of q in A or of smallest element larger than q
- Runtime is $O(\log|\underbrace{t(q) - h(q)}_{\text{Prediction error}}|)$:
  - Finding larger/smaller element takes $O(\log|t(q) - h(q)|)$ steps
  - Binary search takes $O(\log|t(q) - h(q)|)$ steps
- Better predictions lead to **better runtime**
- Runtime **never worse than worst-case** $O(\log|A|)$

Book chapter by Mitzenmacher, Vassilvitskii, '20

Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Algorithms with predictions
    a. Searching a sorted array
    b. Online algorithms
    c. Additional research

Purohit, Svitkina, Kumar, NeurIPS'18

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

> **Matching:** nodes of a graph arrive over time

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

- **Matching:** nodes of a graph arrive over time
  - Must irrevocably decide whether to match a node when it arrives

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

- **Matching:** nodes of a graph arrive over time
  - Must irrevocably decide whether to match a node when it arrives
- **Caching:** memory access requests arrive over time

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

- **Matching:** nodes of a graph arrive over time
  - Must irrevocably decide whether to match a node when it arrives
- **Caching:** memory access requests arrive over time
  - Must decide what to keep in cache

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

- **Matching:** nodes of a graph arrive over time
  - Must irrevocably decide whether to match a node when it arrives
- **Caching:** memory access requests arrive over time
  - Must decide what to keep in cache
- **Scheduling:** job lengths not revealed until they terminate

Online algorithms

Full input not revealed upfront, but at some later stage, e.g.:

Matching: nodes of a graph arrive over time

Must irrevocably decide whether to match a node when it arrives

Caching: memory access requests arrive over time

Must decide what to keep in cache

Scheduling: job lengths not revealed until they terminate

Must decide which jobs to schedule when

Competitive ratio (CR)

Standard measure of online algorithm's performance:

$$\mathrm{CR} = \frac{\mathrm{ALG}}{\mathrm{OPT}}$$

Offline optimal solution that knows the entire input

Competitive ratio (CR)

Standard measure of online algorithm's performance:

$$\mathrm{CR} = \frac{\mathrm{ALG}}{\mathrm{OPT}}$$

Offline optimal solution that knows the entire input

E.g., in matching:

$$\mathrm{CR} = \frac{\text{weight of algorithm's matching}}{\text{maximum weight matching}}$$

Online algorithms

Full input not revealed upfront, but at some later stage

What if algorithm receives some **predictions** about input?

Online algorithms

Full input not revealed upfront, but at some later stage

What if algorithm receives some **predictions** about input?

- **Online advertising**

  e.g., Mahdian et al. [EC'07]; Devanur, Hayes [EC'09]; Muñoz Medina, Vassilvitskii [NeurIPS'17]

Online algorithms

Full input not revealed upfront, but at some later stage

What if algorithm receives some **predictions** about input?

- **Online advertising**

  e.g., Mahdian et al. [EC'07]; Devanur, Hayes [EC'09]; Muñoz Medina, Vassilvitskii [NeurIPS'17]

- **Caching**

  e.g., Lykouris, Vassilvitskii [ICML'18]

Online algorithms

Full input not revealed upfront, but at some later stage

What if algorithm receives some **predictions** about input?

- **Online advertising**

  e.g., Mahdian et al. [EC'07]; Devanur, Hayes [EC'09]; Muñoz Medina, Vassilvitskii [NeurIPS'17]
- **Caching**

  e.g., Lykouris, Vassilvitskii [ICML'18]
- **Data structures**

  e.g., Mitzenmacher [NeurIPS'18]
- …

Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Algorithms with predictions
    a. Searching a sorted array
    b. Online algorithms
        i. Overview
        ii. Ski rental problem
        iii. Job scheduling
    c. Additional research

Ski rental problem

Family of problems that revolve around a decision:

Ski rental problem

Family of problems that revolve around a decision:

- Incur a **recurring expense**, or

Ski rental problem

Family of problems that revolve around a decision:

- Incur a **recurring expense**, or
- Pay a **single fee** that eliminates the ongoing cost

Ski rental problem

Problem: Skier will ski for unknown number of days

Ski rental problem

Problem: Skier will ski for unknown number of days

- Can either **rent each day** for \$1/day or **buy** for \$$b$

Ski rental problem

Problem: Skier will ski for unknown number of days

- Can either **rent each day** for \$1/day or **buy** for $\$b$
- E.g., if ski for 5 days and then buy, total price is $5 + b$

Ski rental problem

Problem: Skier will ski for unknown number of days
- Can either **rent each day** for \$1/day or **buy** for \$$b$
- E.g., if ski for 5 days and then buy, total price is $5 + b$

If ski x days, **optimal clairvoyant** strategy pays $\mathrm{OPT} = \min\{x, b\}$

Ski rental problem

Problem: Skier will ski for unknown number of days

- Can either **rent each day** for \$1/day or **buy** for \$$b$
- E.g., if ski for 5 days and then buy, total price is $5 + b$

If ski x days, **optimal clairvoyant** strategy pays $\mathrm{OPT} = \min\{x, b\}$

Breakeven strategy: Rent for $b - 1$ days, then buy

Ski rental problem

Problem: Skier will ski for unknown number of days

- Can either **rent each day** for \$1/day or **buy** for \$$b$
- E.g., if ski for 5 days and then buy, total price is $5 + b$

If ski x days, **optimal clairvoyant** strategy pays $\mathrm{OPT} = \min\{x, b\}$

Breakeven strategy: Rent for $b - 1$ days, then buy

- CR $= \frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{x\mathbf{1}_{\{x<b\}} + (b-1+b)\mathbf{1}_{\{x\geq b\}}}{\min\{x,b\}} < 2$ (best deterministic)

Ski rental problem

Problem: Skier will ski for unknown number of days
- Can either **rent each day** for \$1/day or **buy** for $\$b$
- E.g., if ski for 5 days and then buy, total price is $5 + b$

If ski x days, **optimal clairvoyant** strategy pays $\mathrm{OPT} = \min\{x, b\}$

Breakeven strategy: Rent for $b - 1$ days, then buy
- CR $= \frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{x\mathbf{1}_{\{x<b\}}+(b-1+b)\mathbf{1}_{\{x\geq b\}}}{\min\{x,b\}} < 2$ (best deterministic)
- Randomized alg. CR $= \frac{e}{e-1}$ [Karlin et al., Algorithmica '94]

Ski rental problem

Prediction y of number of skiing days, error $\eta = |x - y|$

Purohit, Svitkina, Kumar, NeurIPS'18

Ski rental problem

Prediction y of number of skiing days, error $\eta = |x - y|$

Baseline: Buy at beginning if $y > b$, else rent all days

Purohit, Svitkina, Kumar, NeurIPS'18

Ski rental problem

Prediction y of number of skiing days, error $\eta = |x - y|$

Baseline: Buy at beginning if $y > b$, else rent all days

Theorem: $\mathrm{ALG} \leq \mathrm{OPT} + \eta$

Purohit, Svitkina, Kumar, NeurIPS'18

Ski rental problem

Prediction y of number of skiing days, error $\eta = |x - y|$

Baseline: Buy at beginning if $y > b$, else rent all days

Theorem: $\mathrm{ALG} \leq \mathrm{OPT} + \eta$

If y small but $x \gg b$, CR can be unbounded

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

- If **really trust** predictions: set $\lambda = 0$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

- If **really trust** predictions: set $\lambda = 0$

  Equivalent to blindly following predictions

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

- If **really trust** predictions: set $\lambda = 0$
  Equivalent to blindly following predictions
- If **don't trust** predictions: set $\lambda = 1$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

- If **really trust** predictions: set $\lambda = 0$
  Equivalent to blindly following predictions
- If **don't trust** predictions: set $\lambda = 1$
  Equivalent to running the worst-case algorithm

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

- If predictor is perfect ($\eta = 0$), **CR is small** ($\leq 1 + \lambda$)

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Prediction y of number of skiing days, error $\eta = |x - y|$

Algorithm (with parameter $\lambda \in [0,1]$):

If $y \geq b$, buy on start of day $\lceil \lambda b \rceil$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

- If predictor is perfect ($\eta = 0$), **CR is small** ($\leq 1 + \lambda$)
- No matter how big η is, setting $\lambda = 1$ **recovers baseline** CR $= 2$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Proof sketch: If $y \geq b$, buys on start of day $\lceil \lambda b \rceil$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Proof sketch: If $y \geq b$, buys on start of day $\lceil \lambda b \rceil$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \begin{cases} \frac{x}{x} & \text{if } x < \lceil \lambda b \rceil \\ & \end{cases}$$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Proof sketch: If $y \geq b$, buys on start of day $\lceil \lambda b \rceil$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \begin{cases} \dfrac{x}{x} & \text{if } x < \lceil \lambda b \rceil \\ \dfrac{\lceil \lambda b \rceil - 1 + b}{x} & \text{if } \lceil \lambda b \rceil \leq x \leq b \\ & \end{cases}$$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Proof sketch: If $y \geq b$, buys on start of day $\lceil \lambda b \rceil$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \begin{cases} \frac{x}{x} & \text{if } x < \lceil \lambda b \rceil \\ \frac{\lceil \lambda b \rceil - 1 + b}{x} & \text{if } \lceil \lambda b \rceil \leq x \leq b \\ \frac{\lceil \lambda b \rceil - 1 + b}{b} & \text{if } x \geq b \end{cases}$$

Purohit, Svitkina, Kumar, NeurIPS'18

Deterministic algorithm

Theorem: Algorithm has CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Proof sketch: If $y \geq b$, buys on start of day $\lceil \lambda b \rceil$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \begin{cases} \dfrac{x}{x} & \text{if } x < \lceil \lambda b \rceil \\ \dfrac{\lceil \lambda b \rceil - 1 + b}{x} & \text{if } \lceil \lambda b \rceil \leq x \leq b \\ \dfrac{\lceil \lambda b \rceil - 1 + b}{b} & \text{if } x \geq b \end{cases}$$

Worst when $x = \lceil \lambda b \rceil$ and CR = $\frac{b+\lceil \lambda b \rceil - 1}{\lceil \lambda b \rceil} \leq \frac{1+\lambda}{\lambda}$; similarly for $y < b$

Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal
- Algorithm is α-consistent if CR $\rightarrow \alpha$ as error $\eta \rightarrow 0$

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal
- Algorithm is α-consistent if CR → α as error $\eta \to 0$

Robustness:

- Predictions are terrible ⇒ no worse than worst-case

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal
- Algorithm is α-consistent if CR → α as error $\eta \to 0$

Robustness:

- Predictions are terrible ⇒ no worse than worst-case
- Algorithm is β-consistent if CR ≤ β for all η

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal
- Algorithm is α-consistent if CR → α as error $\eta \to 0$

Robustness:

- Predictions are terrible ⇒ no worse than worst-case
- Algorithm is β-consistent if CR ≤ β for all η

E.g., ski rental: CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal
- Algorithm is α-consistent if CR → α as error $\eta \to 0$

Robustness:

- Predictions are terrible ⇒ no worse than worst-case
- Algorithm is β-consistent if CR ≤ β for all η

E.g., ski rental: CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

$(1+\lambda)$-consistent, $\left(\frac{1+\lambda}{\lambda}\right)$-robust

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Design principals

Consistency:

- Predictions are perfect ⇒ recover offline optimal
- Algorithm is α-consistent if CR → α as error $\eta \to 0$

Robustness:

- Predictions are terrible ⇒ no worse than worst-case
- Algorithm is β-consistent if CR ≤ β for all η

E.g., ski rental: CR $\leq \min\left\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)\mathrm{OPT}}\right\}$

$(1+\lambda)$-consistent, $\left(\frac{1+\lambda}{\lambda}\right)$-robust

Bounds are tight [Gollapudi, Panigrahi, ICML'19; Angelopoulos et al., ITCS'20]

Lykouris, Vassilvitskii, ICML'18; Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm

if $y \geq b$:

Let $k \leftarrow \lfloor \lambda b \rfloor$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm

if $y \geq b$:

Let $k \leftarrow \lfloor \lambda b \rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{k-i} \frac{1}{b\left(1-\left(1-{}^{1}/_{b}\right)^{k}\right)}$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm

if $y \geq b$:

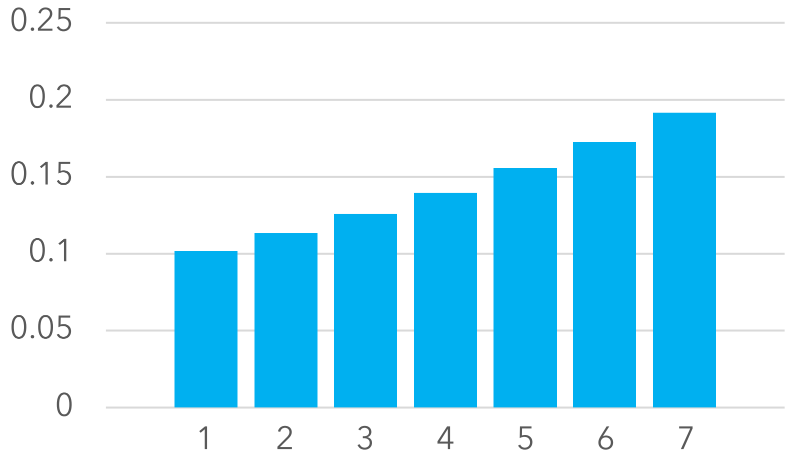
Let $k \leftarrow \lfloor \lambda b \rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{k-i} \frac{1}{b\left(1-\left(1-1/_b\right)^k\right)}$

Buy on day $j \in [k]$ sampled from distribution defined by $q_1, \ldots, q_k$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm



q_i when $b = 10, \lambda = 0.7, y \geq b$

if $y \geq b$:

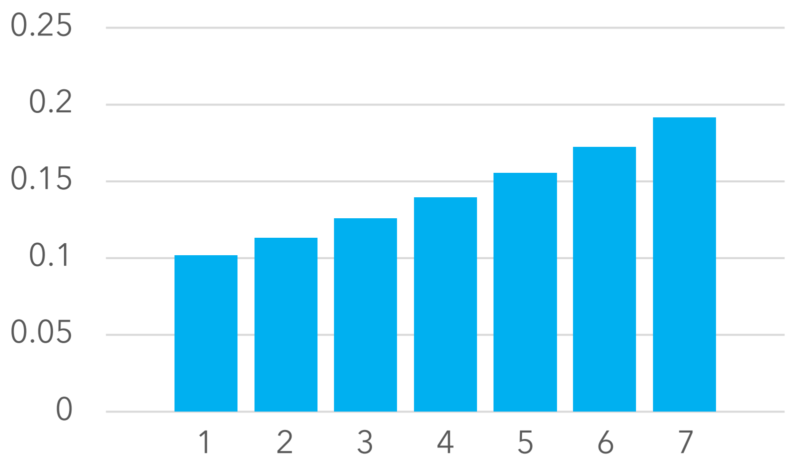
Let $k \leftarrow \lfloor \lambda b \rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{k-i} \frac{1}{b\left(1-\left(1-{}^{1}/_{b}\right)^{k}\right)}$

Buy on day $j \in [k]$ sampled from distribution defined by $q_1, \ldots, q_k$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm



q_i when $b = 10, \lambda = 0.7, y \geq b$

if $y \geq b$:

Let $k \leftarrow \lfloor \lambda b \rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{k-i} \frac{1}{b\left(1-\left(1-{}^{1}/_{b}\right)^{k}\right)}$

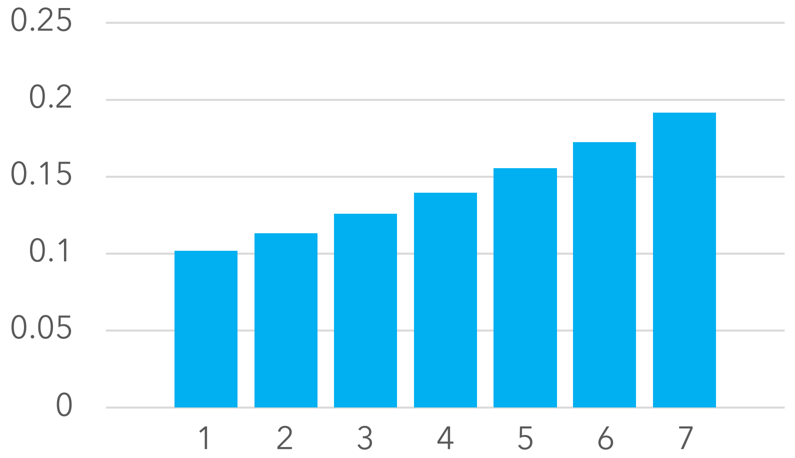
Buy on day $j \in [k]$ sampled from distribution defined by $q_1, \ldots, q_k$

else

Let $\ell \leftarrow \left\lfloor \frac{b}{\lambda} \right\rfloor$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm



q_i when $b = 10, \lambda = 0.7, y \geq b$

if $y \geq b$:

Let $k \leftarrow \lfloor \lambda b \rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{k-i} \frac{1}{b\left(1-\left(1-1/b\right)^k\right)}$

Buy on day $j \in [k]$ sampled from distribution defined by $q_1, \ldots, q_k$

else

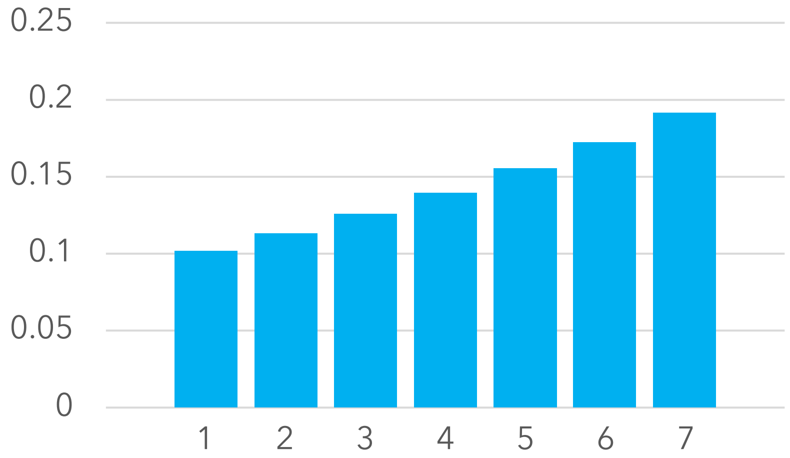
Let $\ell \leftarrow \left\lfloor \frac{b}{\lambda} \right\rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{\ell-i} \frac{1}{b\left(1-\left(1-1/b\right)^\ell\right)}$

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm



q_i when $b = 10$, $\lambda = 0.7$, $y \geq b$

if $y \geq b$:

Let $k \leftarrow \lfloor \lambda b \rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{k-i} \frac{1}{b\left(1-\left(1-{}^{1}/_{b}\right)^{k}\right)}$

Buy on day $j \in [k]$ sampled from distribution defined by $q_1, \ldots, q_k$

else

Let $\ell \leftarrow \left\lfloor \frac{b}{\lambda} \right\rfloor$

For $i \in [k]$, define $q_i \leftarrow \left(\frac{b-1}{b}\right)^{\ell-i} \frac{1}{b\left(1-\left(1-{}^{1}/_{b}\right)^{\ell}\right)}$

Buy on day $j \in [\ell]$ sampled from distribution defined by $q_1, \ldots, q_\ell$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm

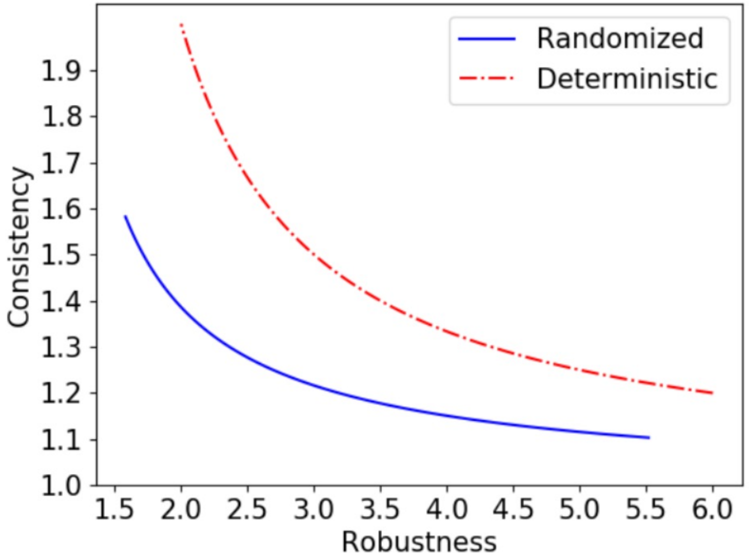
Theorem: $\mathrm{CR} \leq \min\left\{\frac{1}{1-\exp\left(-(\lambda-1/b)\right)}, \frac{\lambda}{1-\exp(-\lambda)}\left(1+\frac{\eta}{\mathrm{OPT}}\right)\right\}$

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm

Theorem: CR $\leq \min\left\{\frac{1}{1-\exp\left(-\left(\lambda-{}^{1}/_{b}\right)\right)}, \frac{\lambda}{1-\exp(-\lambda)}\left(1+\frac{\eta}{\mathrm{OPT}}\right)\right\}$

- $\left(\frac{\lambda}{1-\exp(-\lambda)}\right)$-consistent, $\left(\frac{1}{1-\exp\left(-\left(\lambda-{}^{1}/_{b}\right)\right)}\right)$-robust



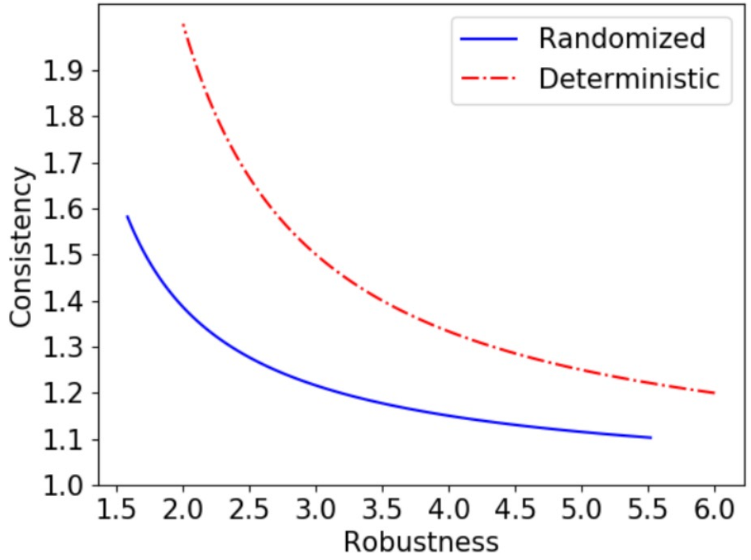
Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Purohit, Svitkina, Kumar, NeurIPS'18

Randomized algorithm

Theorem: $\text{CR} \leq \min\left\{\frac{1}{1-\exp\left(-\left(\lambda-{}^{1}/_{b}\right)\right)}, \frac{\lambda}{1-\exp(-\lambda)}\left(1+\frac{\eta}{\text{OPT}}\right)\right\}$

- $\left(\frac{\lambda}{1-\exp(-\lambda)}\right)$-consistent, $\left(\frac{1}{1-\exp\left(-\left(\lambda-{}^{1}/_{b}\right)\right)}\right)$-robust
- Bounds are **tight** [Wei, Zhang, NeurIPS'20]



Purohit, Svitkina, Kumar, NeurIPS'18

Outline (theoretical guarantees)

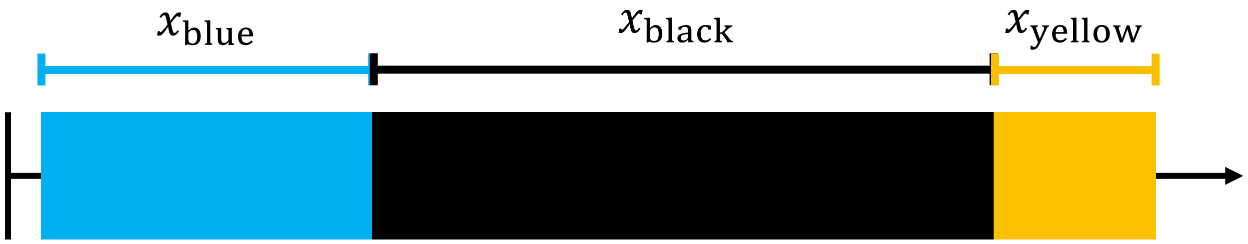
1. Statistical guarantees for algorithm configuration
2. Algorithms with predictions
    a. Searching a sorted array
    b. Online algorithms
        i. Overview
        ii. Ski rental problem
        iii. Job scheduling
    c. Additional research

Job scheduling

- Task: schedule n jobs on a single machine

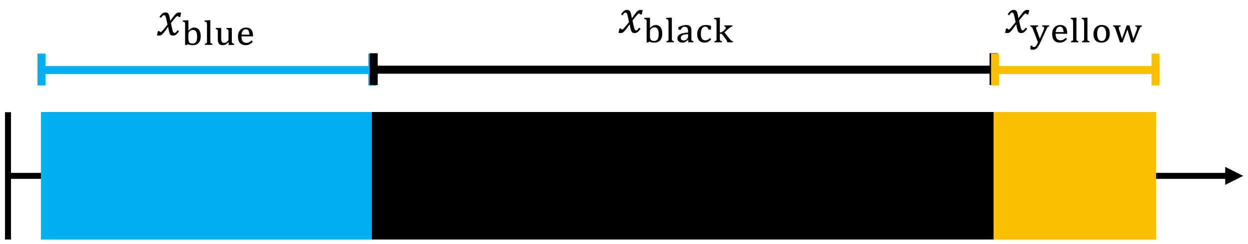
Job scheduling

- Task: schedule n jobs on a single machine
- Job j has **unknown** processing time x_j



Job scheduling

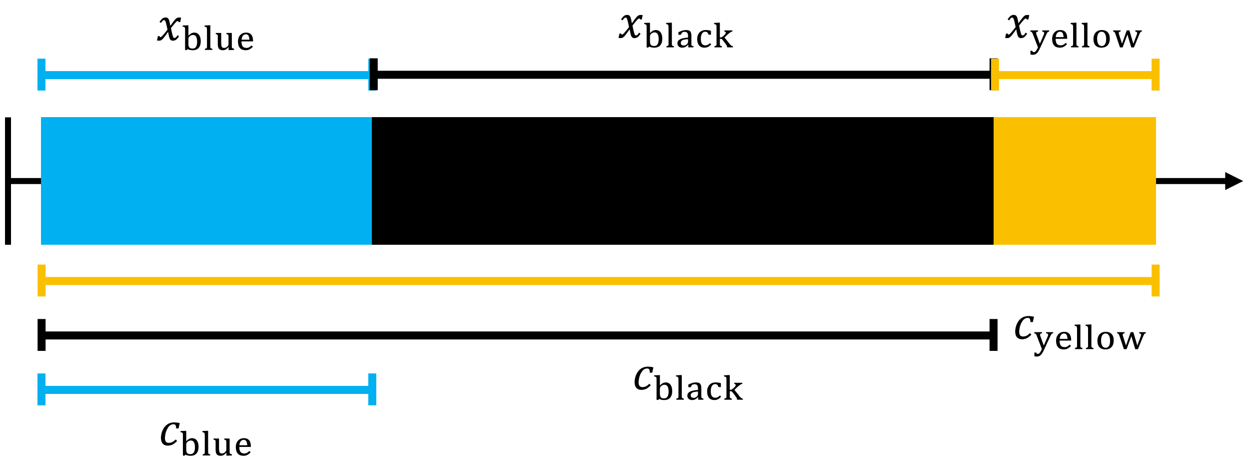
- Task: schedule n jobs on a single machine
- Job j has **unknown** processing time x_j
- Goal: minimize **sum of completion times** of the jobs



Job scheduling

- Task: schedule n jobs on a single machine
- Job j has **unknown** processing time x_j
- Goal: minimize **sum of completion times** of the jobs

  i.e., if job j completes at time c_j, goal is to minimize $\sum c_j$



Job scheduling

- Task: schedule n jobs on a single machine
- Job j has **unknown** processing time x_j
- Goal: minimize **sum of completion times** of the jobs

  i.e., if job j completes at time c_j, goal is to minimize $\sum c_j$
- Can switch between jobs

Job scheduling

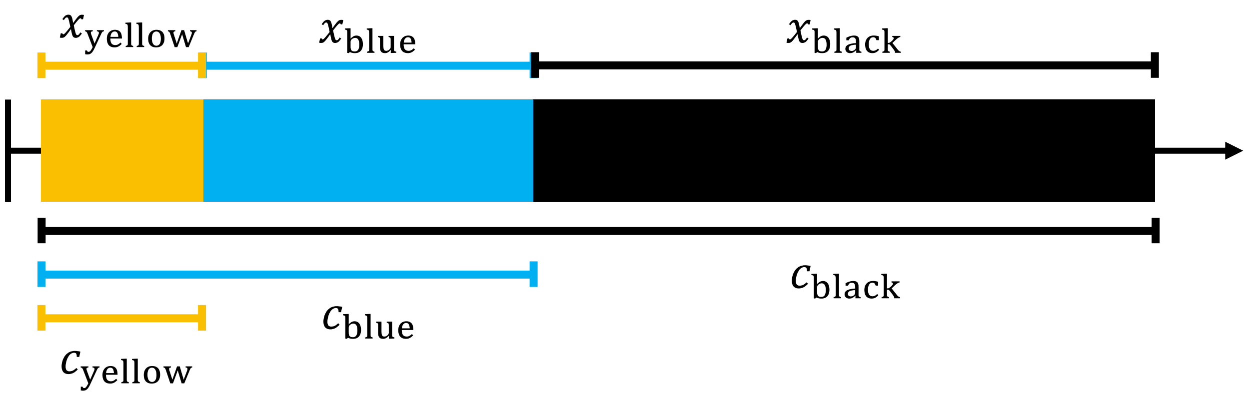
Optimal solution if processing times x_j's are known:

Job scheduling

Optimal solution if processing times x_j's are known:
schedule jobs in increasing order of x_j

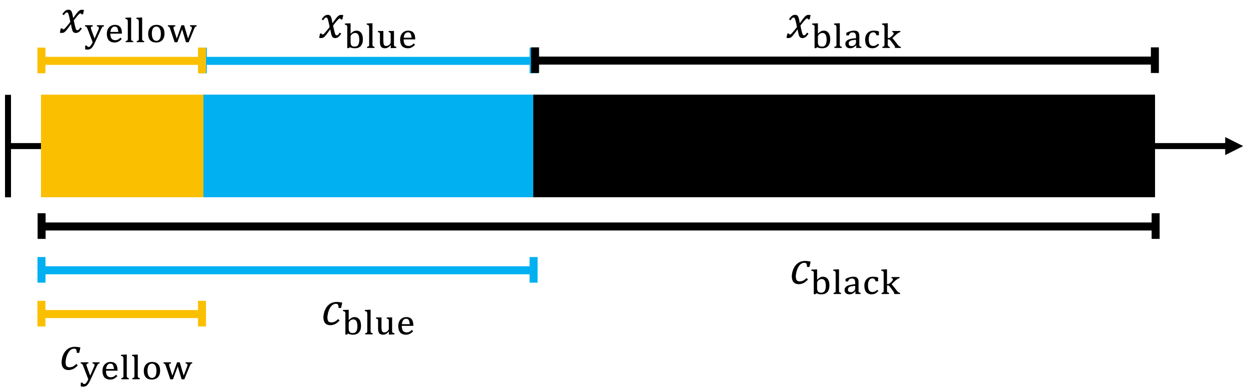
Job scheduling

Optimal solution if processing times x_j's are known:
  schedule jobs in increasing order of x_j



Job scheduling

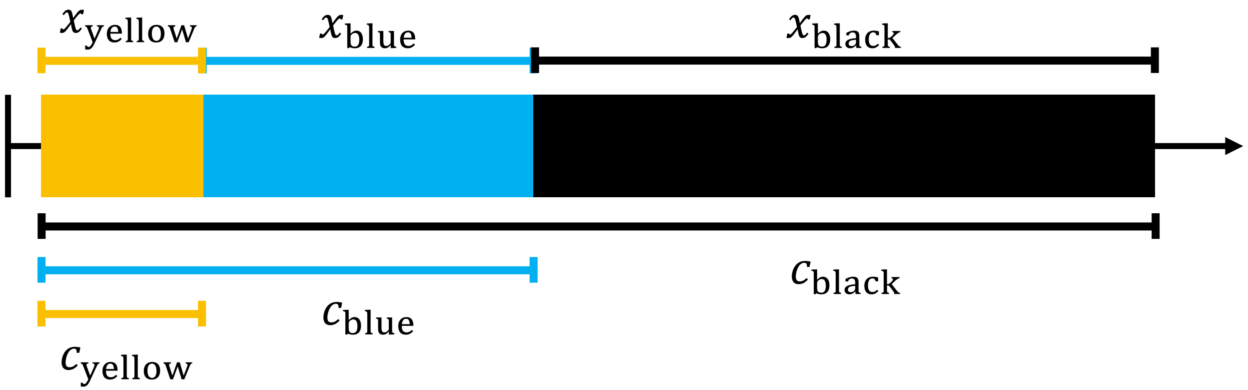
Optimal solution if processing times x_j's are known:
schedule jobs in increasing order of x_j



If $x_1 \leq \cdots \leq x_n$,

Job scheduling

Optimal solution if processing times x_j's are known:
schedule jobs in increasing order of x_j



If $x_1 \leq \cdots \leq x_n$,

$$\text{OPT} = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j$$

Round robin

Algorithm with a competitive ratio of 2: **round robin**

Round robin

Algorithm with a competitive ratio of 2: **round robin**

- Schedule 1 unit of time per remaining job, round-robin

Round robin

Algorithm with a competitive ratio of 2: **round robin**

Schedule 1 unit of time per remaining job, round-robin

Round robin:

Round robin

Algorithm with a competitive ratio of 2: **round robin**

Schedule 1 unit of time per remaining job, round-robin

Round robin:

Round-robin over k jobs $\equiv$ run jobs simultaneously at rate of $\frac{1}{k}$

Round robin

Algorithm with a competitive ratio of 2: **round robin**

Schedule 1 unit of time per remaining job, round-robin

Round robin:

Round-robin over k jobs $\equiv$ run jobs simultaneously at rate of $\frac{1}{k}$

Simultaneous round robin:

Algorithms-with-predictions approach

Predictions $y_1, \dots, y_n$ of $x_1, \dots, x_n$ with $\eta = \sum_{i=1}^{n} |y_i - x_i|$

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Predictions $y_1, \dots, y_n$ of $x_1, \dots, x_n$ with $\eta = \sum_{i=1}^{n} |y_i - x_i|$

If **really trust** predictions: schedule in increasing order of y_i

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Predictions $y_1, \ldots, y_n$ of $x_1, \ldots, x_n$ with $\eta = \sum_{i=1}^{n} |y_i - x_i|$

If **really trust** predictions: schedule in increasing order of y_i

"Shortest predicted job first (SPJF)"

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Predictions $y_1, \dots, y_n$ of $x_1, \dots, x_n$ with $\eta = \sum_{i=1}^{n} |y_i - x_i|$

If **really trust** predictions: schedule in increasing order of y_i

"Shortest predicted job first (SPJF)"

If **don't trust** predictions: round-robin (RR)

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)
Run SPJF and RR **simultaneously**

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Example: $\lambda = \frac{1}{2}$, 3 jobs, shortest predicted job is **blue** job

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Example: $\lambda = \frac{1}{2}$, 3 jobs, shortest predicted job is **blue** job

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Example: $\lambda = \frac{1}{2}$, 3 jobs, shortest predicted job is **blue** job

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Example: $\lambda = \frac{1}{2}$, 3 jobs, shortest predicted job is **blue** job

- **Blue** job at a rate of $\lambda + (1 - \lambda) \cdot \frac{1}{3} = \frac{2}{3}$

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Example: $\lambda = \frac{1}{2}$, 3 jobs, shortest predicted job is **blue** job

- **Blue** job at a rate of $\lambda + (1 - \lambda) \cdot \frac{1}{3} = \frac{2}{3}$
- **Yellow** job at a rate of $(1 - \lambda) \cdot \frac{1}{3} = \frac{1}{6}$

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Purohit, Svitkina, Kumar, NeurIPS'18

Algorithms-with-predictions approach

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Example: $\lambda = \frac{1}{2}$, 3 jobs, shortest predicted job is **blue** job

- **Blue** job at a rate of $\lambda + (1 - \lambda) \cdot \frac{1}{3} = \frac{2}{3}$
- **Yellow** job at a rate of $(1 - \lambda) \cdot \frac{1}{3} = \frac{1}{6}$
- **Black** job at a rate of $(1 - \lambda) \cdot \frac{1}{3} = \frac{1}{6}$

Purohit, Svitkina, Kumar, NeurIPS'18

Preferential round-robin

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Theorem: Algorithm's competitive ratio is

Purohit, Svitkina, Kumar, NeurIPS'18

Preferential round-robin

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Theorem: Algorithm's competitive ratio is

$$\mathrm{CR} \leq \min\left\{\frac{1}{\lambda}\left(1 + \frac{2\eta}{n}\right), \frac{1}{1-\lambda} \cdot 2\right\}$$

Purohit, Svitkina, Kumar, NeurIPS'18

Preferential round-robin

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Theorem: Algorithm's competitive ratio is

$$\mathrm{CR} \leq \min\left\{\underbrace{\frac{1}{\lambda}\left(1 + \frac{2\eta}{n}\right)}_{\text{CR of SPJF}}, \frac{1}{1-\lambda} \cdot \underbrace{2}_{\text{CR of RR}}\right\}$$

Purohit, Svitkina, Kumar, NeurIPS'18

Preferential round-robin

Algorithm: Preferential round-robin (with parameter $\lambda \in (0,1)$)

Run SPJF and RR **simultaneously**

- SPJF at a rate λ
- RR at a rate $1 - \lambda$

Theorem: Algorithm's competitive ratio is

$$\mathrm{CR} \le \min\left\{\underbrace{\frac{1}{\lambda}\left(1 + \frac{2\eta}{n}\right)}_{\text{CR of SPJF}}, \frac{1}{1-\lambda} \cdot \underbrace{2}_{\text{CR of RR}}\right\}$$

So it's $\frac{1}{\lambda}$-consistent, $\frac{2}{1-\lambda}$-robust

Purohit, Svitkina, Kumar, NeurIPS'18

Outline (theoretical guarantees)

1. Statistical guarantees for algorithm configuration
2. Algorithms with predictions
   a. Searching a sorted array
   b. Online algorithms
   c. **Additional research**

Just scratched the surface

Online advertising
Mahdian, Nazerzadeh, Saberi, EC'07; Devanur, Hayes, EC'09; Medina, Vassilvitskii, NeurIPS'17; …

Caching
Lykouris, Vassilvitskii, ICML'18; Rohatgi, SODA'19; Wei, APPROX-RANDOM'20; …

Frequency estimation
Hsu, Indyk, Katabi, Vakilian, ICLR'19; …

Learning low-rank approximations
Indyk, Vakilian, Yuan, NeurIPS'19; …

Scheduling
Mitzenmacher, ITCS'20; Moseley, Vassilvitskii, Lattanzi, Lavastida, SODA'20; …

algorithms-with-predictions.github.io

Just scratched the surface

Online advertising
Mahdian, Nazerzadeh, Saberi, EC'07; Devanur, Hayes, EC'09; Medina, Vassilvitskii, NeurIPS'17; …

Caching
Lykouris, Vassilvitskii, ICML'18; Rohatgi, SODA'19; Wei, APPROX-RANDOM'20; …

Frequency estimation
Hsu, Indyk, Katabi, Vakilian, ICLR'19; …

Learning low-rank approximations
Indyk, Vakilian, Yuan, NeurIPS'19; …

Scheduling
Mitzenmacher, ITCS'20; Moseley, Vassilvitskii, Lattanzi, Lavastida, SODA'20; …

algorithms-with-predictions.github.io

Closely related: the "predict-then-optimize" framework
Elmachtoub, Grigas, Management Science '22; Elmachtoub et al., ICML'20; …

Summary

1 Applied techniques

a. Graph neural networks
   a. Neural algorithmic alignment
   b. Variable selection for integer programming
b. Learning greedy heuristics with RL

2 Theoretical guarantees

a. Statistical guarantees for algorithm configuration
b. Algorithms with predictions

3 Future directions

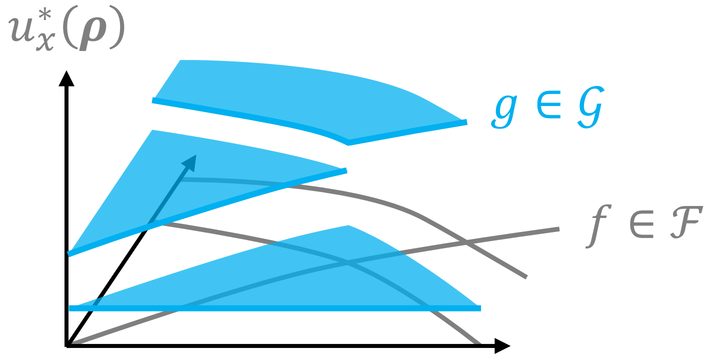
Outline (future directions)

1. **Tighter statistical bounds**
2. Multi-task algorithm design: Knowledge transfer
3. Size generalization
4. ML as a toolkit for theory

Future work: Tighter statistical bounds

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

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



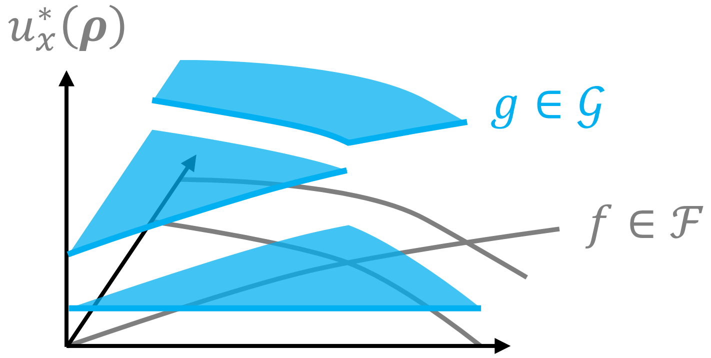
Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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



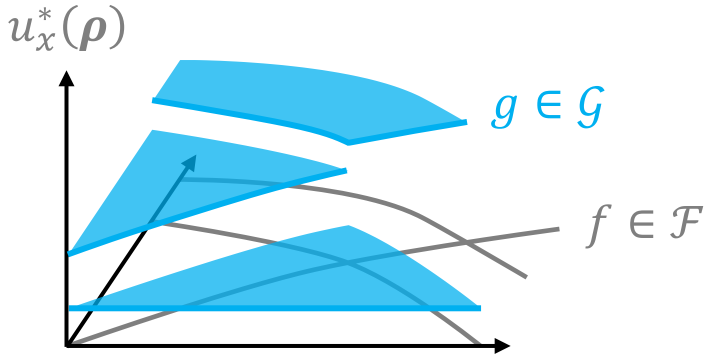
Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

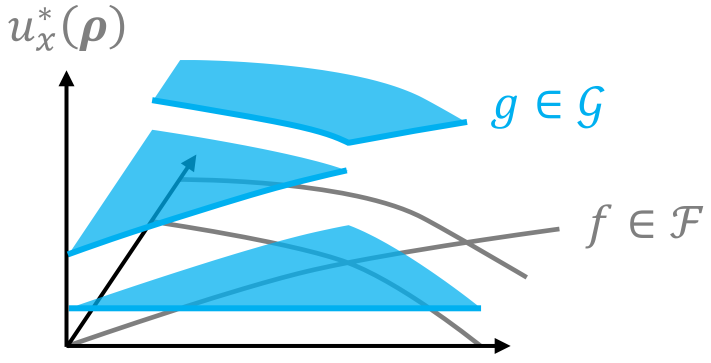
Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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

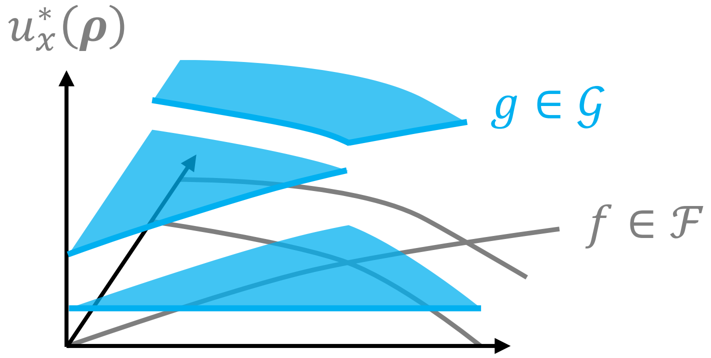
I expect this can sometimes be avoided!

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

Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

Outline (future directions)

1. Tighter statistical bounds
2. **Multi-task algorithm design: Knowledge transfer**
3. Size generalization
4. ML as a toolkit for theory

Future work: Knowledge transfer

- Training a GNN to solve multiple related problems…

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

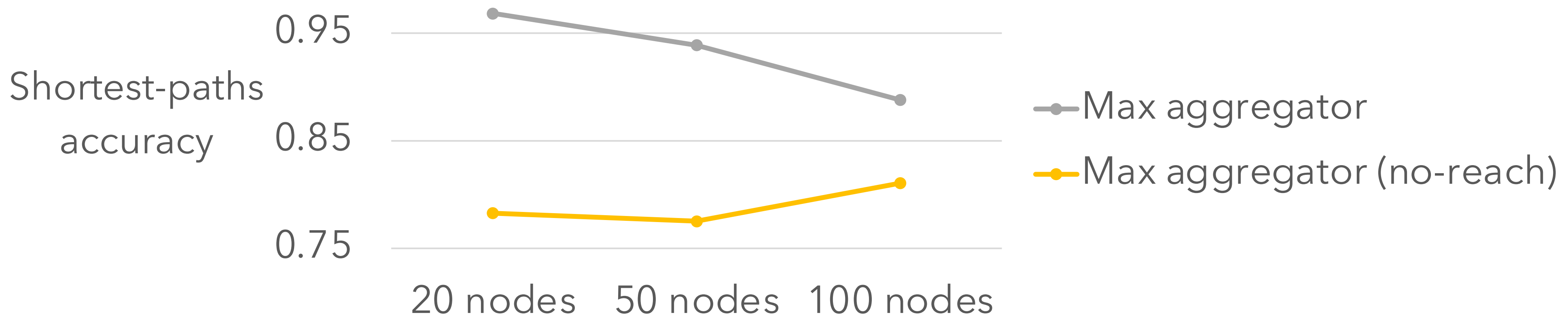
Future work: Knowledge transfer

- Training a GNN to solve multiple related problems…
  can sometimes lead to better **single-task** performance

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

Future work: Knowledge transfer

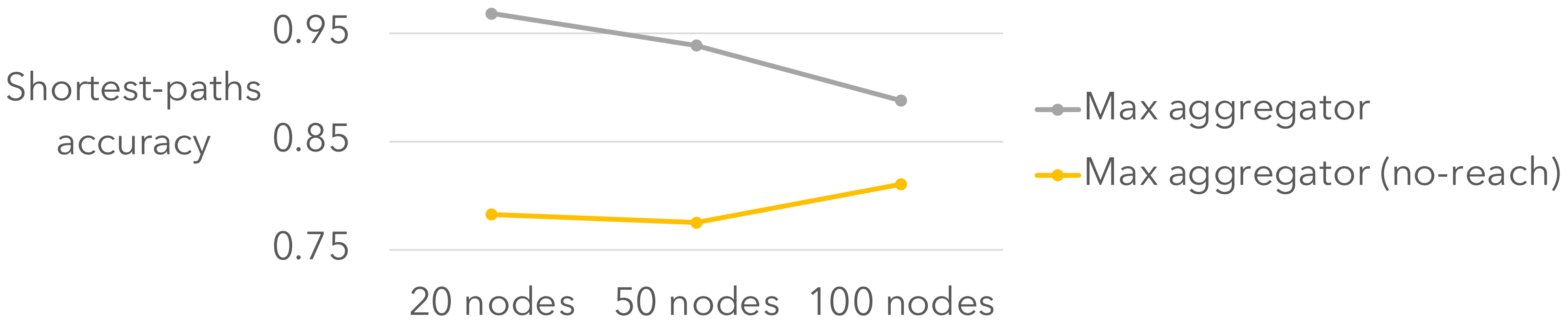
- Training a GNN to solve multiple related problems…
  can sometimes lead to better **single-task** performance
- E.g., training reachability and shortest-paths (grey line)



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
- E.g., training reachability and shortest-paths (grey line)
  v.s. just training shortest-paths (yellow line)



Ellen Vitercik, AAAI'24 tutorial
vitercik.github.io/slides/AAAI24.pdf

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?

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**?

Outline (future directions)

1. Tighter statistical bounds
2. Multi-task algorithm design: Knowledge transfer
3. **Size generalization**
4. ML as a toolkit for theory

Future work: Size generalization

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

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

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**

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:

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**

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?

Future work: Size generalization

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

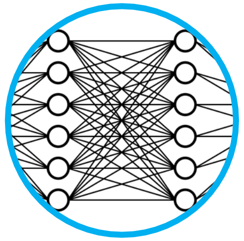
Applied research: Dai et al., NeurIPS'17; Veličković, et al., ICLR'20; …

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**

Future work: Size generalization

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

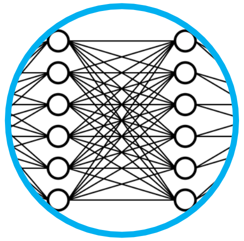
Applied research: Dai et al., NeurIPS'17; Veličković, et al., ICLR'20; …

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**

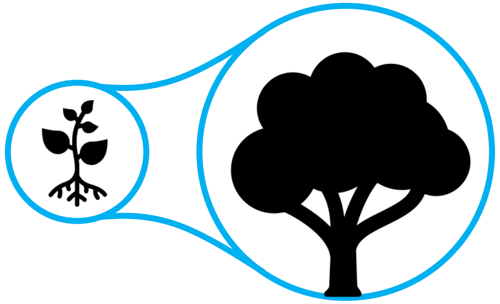
However, size generalization is not immediate! It depends on:

- The **machine-learned algorithm**

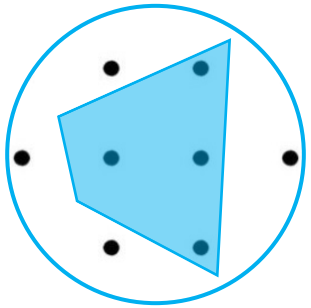
  Is the algorithm scale sensitive?

- The **problem instances**

  As size scales, what features must be preserved?



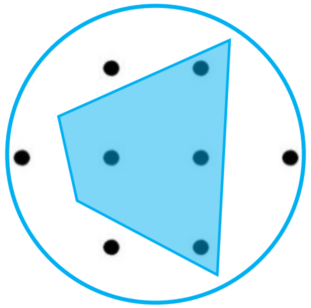
Future work: Size generalization



Can you:

1. **Shrink** a set of big integer programs

Future work: Size generalization

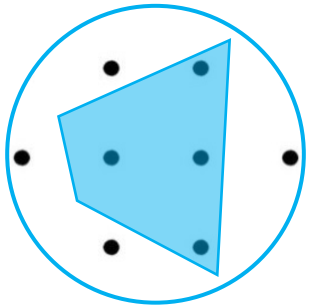


Can you:

1. **Shrink** a set of big integer programs

graphs

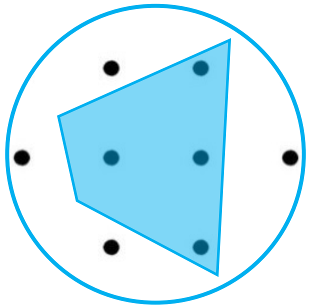
Future work: Size generalization



Can you:

1. **Shrink** a set of big integer programs
   graphs
   …

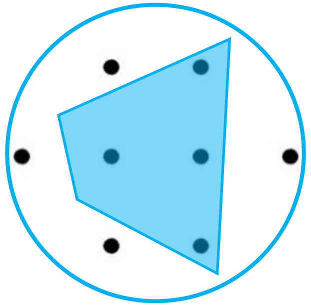
Future work: Size generalization



Can you:

1. **Shrink** a set of big integer programs
   graphs
   …
2. **Learn** a good algorithm on the **small** instances

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?

Outline (future directions)

1. Tighter statistical bounds
2. Multi-task algorithm design: Knowledge transfer
3. Size generalization
4. **ML as a toolkit for theory**

Future work: ML as a toolkit for theory

Which algorithm classes to optimize over?

Classical algorithm design & analysis

Data-driven algorithm design

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?

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:

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

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,"

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."

Thank you! Questions?