Machine learning for discrete optimization: Theoretical guarantees and applied frontiers

Ellen Vitercik
Stanford University
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
IP solvers (CPLEX, Gurobi) have a ton of parameters

- CPLEX has a 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 of parameters

- CPLEX has a 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

Ellen Vitercik, AAAI’24 tutorial
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

K-means  Mean shift  Ward  Agglomerative  Birch

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

Ellen Vitercik, AAAI'24 tutorial
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

💡 Conceive

👨‍💻 Prototype

 mù 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:

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
In practice, we have data about the application domain.
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
- Many different ways to incorporate learning into solving
- Solving is very difficult, so ML can make a huge difference

Ellen Vitercik, AAAI’24 tutorial
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

Ellen Vitercik, AAAI’24 tutorial
Leyton-Brown et al., PNAS’17; Leyton-Brown and Hutter, ICML’19 tutorial
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

Algorithm selection
[Rice ‘76]

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

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

RL for discrete optimization
[e.g., Zhang, Dietterich, ’95]
A bit of history

2017: Around the start of the material in this tutorial
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
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

- Graph-level prediction
- Node level
- Community (subgraph) level
- Edge level

Figure by Leskovec
Prediction with graphs: Examples

Graph-level tasks:
E.g., for a molecule represented as a graph, could predict:
- What the molecule smells like
- Whether it will bind to a receptor implicated in a disease
Prediction with graphs: Examples

Node-level tasks:
E.g., political affiliations of users in a social network
Prediction with graphs: Examples

**Edge-level tasks:** E.g.:
- Suggesting new friends
- Recommendations on Amazon, Netflix, …
Example: Traffic routing

E.g., Google maps
deeplearning.org/blog/article/traffic-prediction-with-advanced-graph-neural-networks

Ellen Vitercik, AAAI’24 tutorial
Example: Learning to simulate physics

**Nodes:** Particles

**Edges:** Interaction between particles

**Goal:** Predict how a graph will evolve over time
Example: Combinatorial optimization

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

\[
\begin{align*}
\text{maximize} & \quad c \cdot z \\
\text{subject to} & \quad Az \leq b \\
& \quad z \in \mathbb{Z}^n
\end{align*}
\]
Graph neural networks: First step

• Design features for nodes/links/graphs
• Obtain features for all training data
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
Traffic prediction with advanced graph neural networks

Ellen Vitercik, AAAI’24 tutorial

Traffic prediction with advanced graph neural networks

Ellen Vitercik, AAAI’24 tutorial

Ellen Vitercik, AAAI’24 tutorial

Traffic prediction with advanced graph neural networks

Ellen Vitercik, AAAI’24 tutorial

Encoding neighborhoods: General form

\[ h_v^{(0)} = x_v \] (feature representation for node \( v \))

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

1. **Aggregate** over neighbors

   \[ m_{N(v)}^{(k)} = \text{AGGREGATE}^{(k)} \left( \{ h_u^{(k-1)} : u \in N(v) \} \right) \]
Encoding neighborhoods: General form

\[ h_v^{(0)} = x_v \] (feature representation for node \( v \))

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

1. **Aggregate** over neighbors
   \[ m_{N(v)}^{(k)} = \text{AGGREGATE}^{(k)} \left( \{ h_u^{(k-1)} : u \in N(v) \} \right) \]

2. **Update** current node representation
   \[ h_v^{(k)} = \text{COMBINE}^{(k)} \left( h_v^{(k-1)}, m_{N(v)}^{(k)} \right) \]
The basic GNN

\[ m_{N(v)} = \text{AGGREGATE}(\{h_u : u \in N(v)\}) = \sum_{u \in N(v)} h_u \]

\[ \text{COMBINE}(h_v, m_{N(v)}) = \sigma(W_{\text{self}}h_v + W_{\text{neigh}}m_{N(v)} + b) \]

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

Trainable parameters

Ellen Vitercik, AAAI’24 tutorial

Figure by Jegelka
Aggregation functions

\[ m_{N(v)} = \text{AGGREGATE}([h_u: u \in N(v)]) = \bigoplus_{u \in N(v)} h_u \]

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

Grey boxes: aggregation functions that we learn
Node embeddings unrolled

Grey boxes: aggregation functions that we learn

Ellen Vitercik, AAAI’24 tutorial
Figures by Leskovec
Node embeddings unrolled

Grey boxes: aggregation functions that we learn

Ellen Vitercik, AAAI’24 tutorial

Figures by Leskovec
Node embeddings unrolled

Use the same aggregation functions for all nodes

Can generate encodings for previously unseen nodes & graphs!
Outline \textbf{(applied techniques)}

1. GNNs overview
2. \textbf{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)**

\[
\begin{align*}
\text{max} & \quad c \cdot z \\
\text{s.t.} & \quad Az \leq b \\
& \quad z \in \mathbb{Z}^n
\end{align*}
\]

**Linear program (LP)**

\[
\begin{align*}
\text{max} & \quad c \cdot z \\
\text{s.t.} & \quad Az \leq b \\
& \quad z \in \mathbb{Z}^n
\end{align*}
\]

LP provides valuable guidance in B&B

Ellen Vitercik, AAAI’24 tutorial
\[
\text{max } (40, 60, 10, 10, 3, 20, 60) \cdot z \\
\text{s.t. } (40, 50, 30, 10, 10, 40, 30) \cdot z \leq 100 \\
z \in \{0, 1\}^7
\]
\[
\begin{align*}
\text{max} & \quad (40,60,10,10,3,20,60) \cdot z \\
\text{s.t.} & \quad (40,50,30,10,10,40,30) \cdot z \leq 100 \\
& \quad z \in \{0,1\}^7
\end{align*}
\]

This section: Variable selection

Ellen Vitercik, AAAI’24 tutorial
Variable selection policies (VSPs)

**Score-based variable selection policies:**
At leaf \( Q \), branch on variable \( z_i \) maximizing \( \text{score}(Q, i) \in \mathbb{R} \)

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

\[
(z(j) - z_i^+(j))(z(j) - z_i^-(j))
\]

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 $z(j) - z_i^-(j)$

Ellen Vitercik, AAAI’24 tutorial
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
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
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 $\pi_\theta$ with training set $S$
State encoding

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

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

1. Pass from variables $\rightarrow$ constraints

$$c_i \leftarrow f_C \left( c_i, \sum_{j:(i,j)\in E} g_C(c_i, v_j, e_{ij}) \right)$$

- **Constraint features**
- **2-layer MLP with relu activations**
- **Edge features**
- **Variable features**

Constraints $\Rightarrow$ Variables

Ellen Vitercik, AAAI’24 tutorial

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

1. Pass from variables $\rightarrow$ constraints
   $$c_i \leftarrow f_C \left(c_i, \sum_{j: (i,j) \in E} g_C(c_i, v_j, e_{ij}) \right)$$

2. Pass from constraints $\rightarrow$ variables
   $$v_j \leftarrow f_V \left(v_j, \sum_{i: (i,j) \in E} g_V(c_i, v_j, e_{ij}) \right)$$
GNN structure

3. Compute distribution over variables
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
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):

$$\Delta_i^+ (j) \cdot \Delta_i^- (j)$$

Estimate of $z(j) - z_i^+(j)$  
Estimate of $z(j) - z_i^-(j)$

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

Ellen Vitercik, AAAI’24 tutorial

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 **SVMrank** [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**
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
## Set covering instances

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

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

- **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

Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS’19
Set covering instances

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

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

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

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

Ellen Vitercik, AAAI’24 tutorial

Gasse, Chételat, Ferroni, Charlin, Lodi; NeurIPS’19
## Set covering instances

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Time</th>
<th>Wins</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full strong branching</td>
<td>Timed out</td>
<td>0/0</td>
<td>N/A</td>
</tr>
<tr>
<td>Reliability pseudo-cost</td>
<td>1677.98±3.0%</td>
<td>4/65</td>
<td>47299±4.9%</td>
</tr>
<tr>
<td>Regression trees</td>
<td>2869.21±3.2%</td>
<td>0/35</td>
<td>59013±9.3%</td>
</tr>
<tr>
<td>SVMrank</td>
<td>2389.92±2.3%</td>
<td>0/47</td>
<td>42120±5.4%</td>
</tr>
<tr>
<td>lambdaMART</td>
<td>2165.96±2.0%</td>
<td>0/54</td>
<td>45319±3.4%</td>
</tr>
<tr>
<td>GNN</td>
<td>1489.91±3.3%</td>
<td>66/70</td>
<td>29981±4.9%</td>
</tr>
</tbody>
</table>
## Set covering instances

Performance generalizes to **larger instances**

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

Similar results for auction design & facility location problems
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

**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

+ 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; ...

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**
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

Ellen Vitercik, AAAI’24 tutorial

Veličković, Ying, Padovano, Hadsell, Blundell, ICLR’20
Neural algorithmic pipeline

Encoder network $f$
- E.g., makes sure input is in correct dimension for next step
Neural algorithmic pipeline

Processor network $P$
- Graph neural network
- Run multiple times (termination determined by a NN)
Neural algorithmic pipeline

Decoder network $g$
- Transform’s GNNs output into algorithmic output
Neural algorithmic pipeline

Multi-task approach

• Learn a **single** processor network $P$ for related problems
• Learn **task-specific** encoder, decoder functions $f_A, g_A$
Neural algorithmic pipeline

Figure by Ibarz et al., LoG’22
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
    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
  ...but algorithm only admits abstract inputs
• Could try **manually** converting from one input to another

---

Ellen Vitercik, AAAI’24 tutorial

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**
  • 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!*

Ellen Vitercik, AAAI’24 tutorial

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
  *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

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

Abstract inputs $\bar{x}$

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

Figure by Cappart, Chételat, Khalil, Lodi, Morris, Veličković, JMLR’23
Neural algorithmic pipeline

After training on abstract inputs, processor $P$:
1. Admits useful gradients
2. Operates over high-dim latent space (better use of data)
Neural algorithmic pipeline

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

\[ \tilde{y} \approx g(P(f(x))) \]
Neural algorithmic pipeline

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

Ellen Vitercik, AAAI’24 tutorial
Breadth-first search

• Source node $s$

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

• Node is reachable from $s$ if any of its neighbors are reachable:

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

• Algorithm output at round $t$: $y_i^{(t)} = x_i^{(t+1)}$
Bellman-Ford (shortest path)

- Source node $s$

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

- Node is reachable from $s$ if any of its neighbors are reachable

  Update distance to node as minimal way to reach neighbors

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

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

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

Improvement of max-aggregator increases with size. It aligns better with underlying algorithm [Xu et al., ICLR’20]

Ellen Vitercik, AAAI’24 tutorial
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 $x_i^{(1)}$
Shortest-path predecessor prediction

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

Max aggregator
Max aggregator (no-reach)
Max aggregator (no-algo)

- Trained on 20-node graphs
- Variety of graph types in train/test set: Erdős-Reyni, Barabási-Albert, …
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
Why does this difference grow as we extrapolate to bigger graphs?
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:** $\mathbb{E}_{x \sim \mathcal{P}}[\ell(f(x), g(x))]$
Aggregation functions

\[
\mathbf{h}_\mathbf{A}^{(t)} = \sum_{j \in \mathcal{N}(\mathbf{A})} \text{MLP}^{(t)} \left( \mathbf{h}_\mathbf{A}^{(t-1)}, \mathbf{h}_j^{(t-1)} \right)
\]

Ellen Vitercik, AAAI’24 tutorial
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 \( \mathbf{v} \in \mathbb{R}^d \), \( f \) approaches a linear function
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 \( v \in \mathbb{R}^d \), \( f \) approaches a linear function
- More formally, let \( x = tv \)
  - Then \( f(x + hv) - f(x) = f(tv + hv) - f(tv) \to \beta_v h \)
  at a rate \( O \left( \frac{1}{t} \right) \)
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: \[ h_i^{(t)} = \sum_{j \in N(i)} \text{MLP} \left( h_i^{(t-1)}, h_j^{(t-1)} \right) \]

MLP must learn a **non-linearity**
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: \[ h_i^{(t)} = \sum_{j \in N(i)} \text{MLP}(h_i^{(t-1)}, h_j^{(t-1)}) \]

GNN 2: \[ h_i^{(t)} = \max_{j \in N(i)} \text{MLP}(h_i^{(t-1)}, h_j^{(t-1)}) \]

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

\[ \mathcal{P}_{6789} = \mathcal{P}_{6} ; \neq \mathcal{P}_{6} ; \]

\[ \text{error} \]

\[ \mathcal{P}_{\text{train}} = \mathcal{P}_{\text{test}} \quad \mathcal{P}_{\text{train}} \neq \mathcal{P}_{\text{test}} \]

\( \text{5x larger graphs} \)

Ellen Vitercik, AAAI’24 tutorial

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.:

• 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

- Reinforcement learning refresher
  - Markov decision processes
- Reinforcement learning
  - Overview: RL for combinatorial optimization
  - Examples: Min vertex cover and max cut
  - RL formulation

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

State $s$ → Reward $r$ → Action $a$ → Environment

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

**Goal:** Policy $\pi: S \rightarrow A$ that maximizes total (discounted) reward
Policies and value functions

**Value function for a policy:**

Expected sum of discounted rewards

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

\[
= R(s) + \gamma \sum_{s' \in S} P(s' | s, \pi(s)) V^\pi(s')
\]

(Bellman equation)

Discount factor
Optimal policy and value function

**Optimal policy** $\pi^*$ achieves the highest value for every state $V^{\pi^*}(s) = \max_{\pi} V^\pi(s)$

Several different ways to find $\pi^*$
- 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** \((S, A, P, R)\):

- \(S\): 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 \(R: S \rightarrow \mathbb{R}\)

**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

\[
Q^\pi(s, a) = R(s) + \gamma \sum_{s' \in S} P(s' | s, a) Q^\pi(s', \pi(s'))
\]

I.e., Q function is the value of:
  1. Starting in state \(s\)
  2. Taking action \(a\)
  3. Then acting according to \(\pi\)
Q-learning

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

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

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

$Q^*$ is the value of:

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

Ellen Vitercik, AAAI’24 tutorial
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 \text{(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. \textbf{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

**Travelling salesman**
- Bello et al., ICLR’17; Dai et al., NeurIPS’17; Nazari et al., NeurIPS’18; …

**Bin packing**
- Hu et al., ‘17; Laterre et al., ‘18; Cai et al., DRL4KDD’19; Li et al., ‘20; …

**Maximum cut**
- Dai et al., NeurIPS’17; Cappart et al., AAAI’19; Barrett et al., AAAI’20; …

**Minimum vertex cover**
- Dai et al., NeurIPS’17; Song et al., UAI’19; …

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

Goal: use RL to learn new greedy strategies for graph problems
Feasible solution constructed by successively adding nodes to solution

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

RL state representation: Graph embedding
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
Minimum vertex cover

Find smallest vertex subset such that each edge is covered
Minimum vertex cover

Find smallest vertex subset such that each edge is covered

2-approximation:
Greedily add vertices of edge with maximum degree sum
Minimum vertex cover

Find smallest vertex subset such that each edge is covered

2-approximation:
Greedily add vertices of edge with maximum degree sum

Scoring function that guides greedy algorithm
Maximum cut

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

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

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

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

\[
\sum_{(u,v) \in C} w(u,v) = 5
\]
Maximum cut

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

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

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

**Greedy:** move node from one side of cut to the other
Move node that results in the largest improvement in cut weight
Maximum cut

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

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

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

**Greedy:** move node from one side of cut to the other
Move node that results in the largest improvement in cut weight

**Scoring function** that guides greedy algorithm
Outline (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

**Goal:** learn a scoring function to guide greedy algorithm

<table>
<thead>
<tr>
<th>Problem</th>
<th>Greedy operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min vertex cover</td>
<td>Insert node into cover</td>
</tr>
<tr>
<td>Max cut</td>
<td>Insert node into subset</td>
</tr>
<tr>
<td>Traveling salesman</td>
<td>Insert node into sub-tour</td>
</tr>
</tbody>
</table>

Ellen Vitercik, AAAI’24 tutorial

Dai, Khalil, Zhang, Dilkina, Song; NeurIPS’17
## RL for combinatorial optimization

<table>
<thead>
<tr>
<th>Greedy algorithm</th>
<th>Reinforcement learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial solution</td>
<td>State</td>
</tr>
<tr>
<td>Scoring function</td>
<td>Q-function</td>
</tr>
<tr>
<td>Select best node</td>
<td>Greedy policy</td>
</tr>
</tbody>
</table>

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
Reinforcement learning formulation

**State:**

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

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

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

  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 $x_v = 1$
Reinforcement learning formulation

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

**Policy** (deterministic): $\pi(\nu|S) = \begin{cases} 1 & \text{if } \nu = \arg\max_{\nu' \notin S} \hat{Q}(\mu, \nu') \\
0 & \text{else} \end{cases}$
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
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
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

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]
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

Graph showing runtime comparisons for CPLEX and other algorithms on the MVC Barabasi-Albert model.
Min vertex cover visualization

Nodes seem to be selected to balance between:

• Degree
• Connectivity of the remaining graph
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
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?

Best configuration for routing problems likely not suited for scheduling

Ellen Vitercik, AAAI’24 tutorial
Running example: Sequence alignment

Goal: Line up pairs of strings

Applications: Biology, natural language processing, etc.

Ellen Vitercik, AAAI'24 tutorial
Sequence alignment algorithms

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

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

$S = A C T G$

$S' = G T C A$

Insertion/deletion (indel)

Match

Mismatch

Gap
Sequence alignment algorithms

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

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

$$S = \text{AC} \text{TG}$$
$$S' = \text{G} \text{TCA}$$
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]

Ellen Vitercik, AAAI’24 tutorial
Sequence alignment algorithms

-GRTCPKPDDLPFSTVVP-LKTFYEPEGEEITYSCKPGYVSRGGMRKFICPLTGLWPINTLKCTP
E-VKCPFPSRPDNFGVNYPAKPTLYYKDKATFGCHDGYSLDGP-EEIECTKLGNWSAMPSC-KA

Ground-truth alignment of protein sequences
Sequence alignment algorithms

Ground-truth alignment of protein sequences

Alignment by algorithm with poorly-tuned parameters
Sequence alignment algorithms

Ground-truth alignment of protein sequences

Alignment by algorithm with poorly-tuned parameters

Alignment by algorithm with well-tuned parameters
Automated parameter tuning procedure

1. Fix parameterized algorithm
2. Receive training set $T$ of “typical” inputs
3. Find parameter setting w/ good avg performance over $T$

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

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

Runtime, solution quality, etc.

Ellen Vitercik, AAAI’24 tutorial
Automated parameter tuning procedure

1. Fix parameterized algorithm
2. Receive training set $T$ of “typical” inputs
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
3. Find parameter setting w/ good avg performance over $T$

Key question:
How to find parameter setting with good avg performance?

Ellen Vitercik, AAAI'24 tutorial
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

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

**Key question (focus of this section):**
Will that parameter setting have good future performance?
Automated parameter tuning procedure

Key question (focus of this section): Will that parameter setting have good future performance?

Ellen Vitercik, AAAI’24 tutorial
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...

Ellen Vitercik, AAAI’24 tutorial
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, ...
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 ...*
Example: Sequence alignment

Distance between algorithm’s output given $S, S'$ and ground-truth alignment is $p$-wise constant.
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; ...
Primary challenge

Algorithmic performance is a volatile function of parameters

**Complex** connection between parameters and performance
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
\( \mathcal{X} \): Set of all inputs
Example: Sequence alignment

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

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

$S = A\ C\ T\ G$
$S' = G\ T\ C\ A$
Algorithmic performance

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

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

Assume \( u_\rho(x) \in [-1,1] \)

Can be generalized to \( u_\rho(x) \in [-H,H] \)
Model

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

*Distribution models specific application domain at hand*

- E.g., distribution over pairs of DNA strands
- E.g., distribution over pairs of protein sequences
Generalization bounds

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

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

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

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

Empirical average utility    Expected utility

Good average empirical utility $\Rightarrow$ Good expected utility

Ellen Vitercik, AAAI’24 tutorial
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, 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:**

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.

\[
S = \text{A C T G} \\
S' = \text{G T C A}
\]

\[
A - - C T \quad G \\
- G T \quad C \quad A -
\]

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

Ellen Vitercik, AAAI'24 tutorial

Gusfield et al., Algorithmica ‘94; Fernández-Baca et al., J. of Discrete Alg. ‘04
Piecwise-constant utility function

**Corollary:**
Utility is piecewise constant function of parameters

Distance between algorithm’s output and ground-truth alignment
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

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

\[ \mathcal{U} = \{ u_\rho : \mathcal{X} \to \mathbb{R} \mid \rho \in \mathbb{R}^d \} \text{ “Primal” function class} \]

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 \( u_\rho \)
- No obvious notions of Lipschitz continuity or smoothness to rely on
Primal & dual classes

\[ u_\rho(x) = \text{utility of algorithm parameterized by } \rho \in \mathbb{R}^d \text{ on input } x \]
\[ \mathcal{U} = \{ u_\rho : \mathcal{X} \to \mathbb{R} \mid \rho \in \mathbb{R}^d \} \]  
“Primal” function class

\[ u_x^*(\rho) = \text{utility as function of parameters} \]
\[ u_x^*(\rho) = u_\rho(x) \]
\[ \mathcal{U}^* = \{ u_x^* : \mathbb{R}^d \to \mathbb{R} \mid x \in \mathcal{X} \} \]  
“Dual” function class

• Dual functions have simple, Euclidean domain
• Often have ample structure can use to bound complexity of \( \mathcal{U} \)
Piecewise-structured functions

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

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

Complexity measure for binary-valued function classes $\mathcal{F}$
(Classes of functions $f: \mathcal{Y} \to \{-1,1\}$)

E.g., linear separators
VC dimension

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

Example: $\mathcal{F} =$ Linear separators in $\mathbb{R}^2$ \hspace{1cm} $\text{VCdim}(\mathcal{F}) \geq 3$
VC dimension

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

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

\[
\begin{array}{cccc}
- & + & - & + \\
+ & - & - & - \\
+ & + & + & + \\
\end{array}
\]

\[
\begin{array}{cc}
+ & + \\
+ & + \\
\end{array}
\]

\[
\begin{array}{ccc}
- & - & - \\
+ & - & + \\
+ & + & - \\
\end{array}
\]

\[
\begin{array}{cc}
+ & + \\
- & + \\
\end{array}
\]

VCdim(\{\text{Linear separators in } \mathbb{R}^d\}) = d + 1
Pseudo-dimension

Complexity measure for real-valued function classes $\mathcal{G}$
(Classes of functions $g: \mathcal{Y} \rightarrow [-1,1]$)

E.g., affine functions
Pseudo-dimension of $\mathcal{G}$

Size of the largest set $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ s.t.: 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 $\mathbb{R}$  \[ \text{Pdim}(\mathcal{G}) \geq 2 \]

Can also show that $\text{Pdim}(\mathcal{G}) \leq 2$
Sample complexity using pseudo-dim

In the context of **algorithm configuration**:

- \( \mathcal{U} = \{u_\rho : \rho \in \mathbb{R}^d\} \) 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 \( x_1, \ldots, x_N \sim \mathcal{D}, \forall \rho \in \mathbb{R}^d \),
  \[
  \left| \frac{1}{N} \sum_{i=1}^{N} u_\rho(x_i) - \mathbb{E}_{x \sim \mathcal{D}}[u_\rho(x)] \right| \leq \epsilon
  \]

  *Empirical average utility* \hspace{1cm} *Expected utility*

Ellen Vitercik, AAAI’24 tutorial
Main result \textbf{(informal)}

Boundary functions $f_1, \ldots, f_k \in \mathcal{F}$ partition $\mathbb{R}^d$ s.t. in each region, $u^*_x(\rho) = g(\rho)$ 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 $\forall \rho$, $|\text{avg} \text{ utility over training set - exp utility}| \leq \epsilon$.
Main result (informal)

Boundary functions $f_1, \ldots, f_k \in \mathcal{F}$ partition $\mathbb{R}^d$ s.t. in each region, $u^*_x(\rho) = g(\rho)$ 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 $\forall \rho$, $|\text{avg utility over training set} - \text{exp utility}| \leq \epsilon$

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

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

Boundary functions $f_1, \ldots, f_k \in \mathcal{F}$ partition $\mathbb{R}^d$ s.t. in each region, $u^*_x(\rho) = g(\rho)$ 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 $\mathcal{U} = \{u_\rho \mid \rho \in \mathbb{R}^d\}$
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
**Lemma:**
Utility is piecewise constant function of parameters

\[ u^*_{(s,s')}(\rho) \]
Sequence alignment guarantees

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

Ellen Vitercik, AAAI’24 tutorial
Many more applications

- Clustering algorithm configuration
- Integer programming algorithm configuration
- Selling mechanism configuration
- Greedy algorithm configuration
- Computational biology algorithm configuration
- Voting mechanism configuration
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?

Day 1: $\rho_1$

Day 2: $\rho_2$

Day 3: $\rho_3$

Goal: Compete with best parameter setting in hindsight

• Impossible in the worst case
• Under what conditions is online configuration possible?
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.:

Probability any given element is in a huge database
Kraska et al., SIGMOD’18; Mitzenmacher, NeurlPS’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 $A$)

Predictor: $h(q) =$ guess of $q$'s index

Algorithm: Check $A[h(q)]$. If $q$ is there, return $h(q)$. Else:

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

Ellen Vitercik, AAAI’24 tutorial

Book chapter by Mitzenmacher, Vassilvitskii, ‘20
Example: Searching in a sorted array

- **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 $A[h(q) + 2^i]$ for $i > 1$ until find something larger

---

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

Ellen Vitercik, AAAI’24 tutorial

Book chapter by Mitzenmacher, Vassilvitskii, ’20
Example: Searching in a sorted array

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

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

Ellen Vitercik, AAAI’24 tutorial

Book chapter by Mitzenmacher, Vassilvitskii, ‘20
Example: Searching in a sorted array

• **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 $A[h(q) + 2^i]$ for $i > 1$ until find something larger
    • Do binary search on interval $(h(q) + 2^{i-1}, h(q) + 2^i)$
  • If $q < A[h(q)]$, symmetric

Example:
• $q = 8$
• $h(q) = 2$

Ellen Vitercik, AAAI’24 tutorial
Example: Searching in a sorted array

Analysis:
• Let $t(q)$ be index of $q$ in $A$ or of smallest element larger than $q$
• Runtime is $O(\log|t(q) - h(q)|)$:
  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 $O(\log|A|)$
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.:

**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:

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

E.g., in matching:

$$\text{CR} = \frac{\text{weight of algorithm’s matching}}{\text{maximum weight matching}}$$

Offline optimal solution that knows the entire 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]

• 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:

• 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
  • 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 $\text{OPT} = \min\{x, b\}$

**Breakeven strategy:** Rent for $b - 1$ days, then buy
  • $\text{CR} = \frac{\text{ALG}}{\text{OPT}} = \frac{x 1_{x < b} + (b - 1 + b) 1_{x \geq b}}{\min\{x, b\}} < 2$ (best deterministic)
  • Randomized alg. $\text{CR} = \frac{e}{e - 1}$ [Karlin et al., Algorithmica ‘94]
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:** $\text{ALG} \leq \text{OPT} + \eta$

*If $y$ small but $x \gg b$, CR can be unbounded*

Ellen Vitercik, AAAI'24 tutorial

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 $[\lambda b]$; else buy on start of day $\left\lfloor \frac{b}{\lambda} \right\rfloor$

• 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

Ellen Vitercik, AAAI’24 tutorial

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 $[\lambda b]$; else buy on start of day $\left\lceil \frac{b}{\lambda} \right\rceil$

**Theorem:** Algorithm has $CR \leq \min \left\{ \frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)OPT} \right\}$

- If predictor is perfect ($\eta = 0$), **CR is small** ($\leq 1 + \lambda$)
- No matter how big $\eta$ is, setting $\lambda = 1$ **recovers baseline** $CR = 2$

Ellen Vitercik, AAAI’24 tutorial
Deterministic algorithm

**Theorem:** Algorithm has \( CR \leq \min \left\{ \frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)OPT} \right\} \)

**Proof sketch:** If \( y \geq b \), buys on start of day \([\lambda b]\)

\[
\text{alg} \quad \text{opt} = \begin{cases} 
\frac{x}{x} & \text{if } x < [\lambda b] \\
\frac{x}{x} & \text{if } x \geq [\lambda b]
\end{cases}
\]
Deterministic algorithm

**Theorem:** Algorithm has CR $\leq \min \left\{ \frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)OPT} \right\}$

**Proof sketch:** If $y \geq b$, buys on start of day $[\lambda b]$

$$\frac{ALG}{OPT} = \begin{cases} \frac{x}{x} & \text{if } x < [\lambda b] \\ \frac{x}{[\lambda b] - 1 + b} & \text{if } [\lambda b] \leq x \leq b \end{cases}$$
Deterministic algorithm

**Theorem:** Algorithm has CR $\leq \min \left\{ \frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)OPT} \right\}$

**Proof sketch:** If $y \geq b$, buys on start of day $[\lambda b]$

$$\frac{ALG}{OPT} = \begin{cases} 
\frac{x}{x} & \text{if } x < [\lambda b] \\
\frac{[\lambda b] - 1 + b}{x} & \text{if } [\lambda b] \leq x \leq b \\
\frac{[\lambda b] - 1 + b}{b} & \text{if } x \geq b
\end{cases}$$

Worst when $x = [\lambda b]$ and $CR = \frac{b + [\lambda b] - 1}{[\lambda b]} \leq \frac{1+\lambda}{\lambda}$; similarly for $y < b$
Design principals

**Consistency:**
- Predictions are perfect ⇒ recover offline optimal
- Algorithm is $\alpha$-consistent if $\text{CR} \to \alpha$ as error $\eta \to 0$

**Robustness:**
- Predictions are terrible ⇒ no worse than worst-case
- Algorithm is $\beta$-consistent if $\text{CR} \leq \beta$ for all $\eta$

E.g., ski rental: $\text{CR} \leq \min \left\{ \frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda)\text{OPT}} \right\}$

$(1 + \lambda)$-consistent, $\left( \frac{1+\lambda}{\lambda} \right)$-robust

Bounds are tight [Gollapudi, Panigrahi, ICML’19; Angelopoulos et al., ITCS’20]

Ellen Vitercik, AAAI’24 tutorial

Lykouris, Vassilvitskii, ICML’18; Purohit, Svitkina, Kumar, NeurIPS’18
Randomized algorithm

**if** \( y \geq b \):

Let \( k \leftarrow \lfloor \lambda b \rfloor \)

For \( i \in [k] \), define \( q_i \leftarrow \left( \frac{b-1}{b} \right)^{k-i} \frac{1}{b(1-(1-1/b)^k)} \)

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 \( q_i \leftarrow \left( \frac{b-1}{b} \right)^{\ell-i} \frac{1}{b(1-(1-1/b)^\ell)} \)

Buy on day \( j \in [\ell] \) sampled from distribution defined by \( q_1, \ldots, q_\ell \)
Randomized algorithm

**Theorem:** \( CR \leq \min \left\{ \frac{1}{1-\exp\left(-\left(\frac{1}{b}\right)\right)}, \frac{\lambda}{1-\exp(-\lambda)} \left(1 + \frac{\eta}{\text{OPT}}\right) \right\} \)

- \( \left(\frac{\lambda}{1-\exp(-\lambda)}\right) \)-consistent, \( \left(\frac{1}{1-\exp\left(-\left(\frac{1}{b}\right)\right)}\right) \)-robust
- Bounds are **tight** [Wei, Zhang, NeurIPS’20]
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

Ellen Vitercik, AAAI’24 tutorial
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: 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**
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 $y_1, ..., y_n$ of $x_1, ..., x_n$ 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)

Ellen Vitercik, AAAI’24 tutorial

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 $\lambda$
- 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 $(1 - \lambda) \cdot \frac{1}{3} = \frac{1}{6}$
Preferential round-robin

**Algorithm:** Preferential round-robin (with parameter $\lambda \in (0,1)$)
Run SPJF and RR **simultaneously**
- SPJF at a rate $\lambda$
- RR at a rate $1 - \lambda$

**Theorem:** Algorithm’s competitive ratio is

$$CR \leq \min \left\{ \frac{1}{\lambda} \left(1 + \frac{2\eta}{n}\right), \frac{1}{1 - \lambda} \cdot 2 \right\}$$

So it’s $\frac{1}{\lambda}$-consistent, $\frac{2}{1 - \lambda}$-robust
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; ...

Closely related: the “predict-then-optimize” framework
Elmachtoub, Grigas, Management Science ’22; Elmachtoub et al., ICML’20; ...

algorithms-with-predictions.github.io

Ellen Vitercik, AAAI’24 tutorial
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 \( \forall \rho \), |avg utility over training set - exp utility| \( \leq \epsilon \)

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

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
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…
  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)
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; …

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

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

- The **machine-learned algorithm**
  *Is the algorithm scale sensitive?*

**Example** [Xu et al., ICLR’21]:

- Algorithms represents by GNNs **do generalize**
- Algs represented by MLPs **don’t generalize** across size
Future work: Size generalization

Machine-learned algorithms can **scale to larger instances**
Applied research: Dai et al., NeurIPS’17; Veličković, et al., ICLR’20; …
**Goal:** eventually, solve problems **no one’s ever been able to solve**

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

- **The machine-learned algorithm**
  *Is the algorithm scale sensitive?*
- **The problem instances**
  *As size scales, what features must be preserved?*
Future work: Size generalization

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

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?