Machine learning for algorithm design

Ellen Vitercik
Stanford University
An important property of algorithms used in practice is **broad applicability**

Example: **Integer programming solvers**
Most popular tool for solving combinatorial (\& nonconvex) problems

- Routing
- Manufacturing
- Scheduling
- Planning
- Finance

...but they can have **unsatisfactory** default performance
Slow runtime, poor solutions quality, ...
IP solvers (CPLEX, Gurobi) have a ton parameters

- CPLEX has a 170-page manual describing 172 parameters
- Tuning by hand is notoriously slow, tedious, and error-prone

Example: Integer programming (IP)
Example: Integer programming (IP)

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

• CPLEX has **170-page** manual describing **172** parameters
• Tuning by hand is notoriously **slow, tedious, and error-prone**

What’s the best **configuration** for the application at hand?

Best configuration for **routing** problems likely not suited for **scheduling**
Example: Sequence alignment

Goal: Line up pairs of strings

**Applications:** Biology, natural language processing, etc.
Example: Sequence alignment

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

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

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

Gap

Insertion/deletion (indel)

Match

Mismatch
Example: Sequence alignment

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

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

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

$\overline{A - - C T G}$
$\overline{- G T C A -}$

Gap
Insertion/deletion (\textit{indel})
Match
Mismatch
Example: Sequence alignment

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]
Example: Sequence alignment

- GRTCPKPDDLPSTVVP-LKTFYEPEEITYSCKPGYVSRGGMRKIFICPLTGLWIPINTLKCTP
  E-VKCPFPSRPDNGFVNYPAKPTLYYDKATFGCHDGYSLDGP-EEIETKLGNSAMPSC-KA

Ground-truth alignment of protein sequences
Example: Sequence alignment

Ground-truth alignment of protein sequences

Alignment by algorithm with poorly-tuned parameters
Example: Sequence alignment

Ground-truth alignment of protein sequences

Alignment by algorithm with poorly-tuned parameters

Alignment by algorithm with well-tuned parameters
Example: Clustering

Diverse applications, including:

Ecology

Biology

Network analysis
Example: Clustering

Many different algorithms

- K-means
- Mean shift
- Ward
- Agglomerative
- Birch

How to **select** the best algorithm for the application at hand?
In practice, we have data about the application domain

Data we could use in the process of

- **Algorithm selection**
  Given a variety of algorithms, which to use?

- **Algorithm configuration**
  How to tune the algorithm’s parameters?

- **Algorithm design**
In practice, we have data about the application domain.

Routing problems a shipping company solves.
Clustering problems a biology lab solves

In practice, we have data about the application domain
In practice, we have data about the application domain
Existing research

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

- **Integer & linear programming**
  
  [Leyton-Brown, Nudelman, Andrew, McFadden, Shoham, CP ‘03; …]

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

- **Computational biology**
  
  [Majoros, Salzberg, Bioinformatics‘04; …]
Existing research

Automated algorithm configuration and selection
[Gupta, Roughgarden, ITCS’16; Balcan, Nagarajan, Vitercik, White, COLT’17; Balcan, Cambridge University Press ‘20; …]

Algorithms with predictions
[Lykouris, Vassilvitskii, ICML’18; Mitzenmacher, NeurIPS’18; …]
Outline

1. Introduction

2. Algorithm configuration

3. Algorithms with predictions

4. Learning to prune

5. Conclusion and future directions

Gupta, Roughgarden, ITCS’16
Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
Book chapter by Balcan, ’20
Automated configuration procedure

1. Fix parameterized algorithm/mechanism
2. Receive set of “typical” inputs sampled from unknown $\mathcal{D}$
3. Return parameter setting $\hat{\rho}$ with good avg performance

Runtime, solution quality, etc.

Key question: How to find $\hat{\rho}$ with good avg performance?

Hutter et al. [JAIR’09, LION’11], Ansótegui et al. [CP’09], Kleinberg et al. [NeurIPS’19, IJCAI’17], Weisz et al. [ICML’19, NeurIPS’19]; Balcan, Sandholm, V [AAAI’20], …
Automated configuration procedure

Focus of this section: Will $\hat{\rho}$ have good future performance? More formally: Is the expected performance of $\hat{\rho}$ also high?

Gupta, Roughgarden, ITCS’16; Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
Results overview

Key question (focus of 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
Results overview

Performance is \textbf{piecewise-structured} function of parameters

Piecewise constant, linear, quadratic, …

Algorithmic performance on fixed input

\textbf{Piecewise constant} \hspace{2cm} \textbf{Piecewise linear} \hspace{2cm} \textbf{Piecewise …}

Balcan, DeBlasio, Dick, Kingsford, Sandholm, \textit{Vitercik}, STOC’21
Example: Sequence alignment

Distance between \textsf{algorithm’s output} given $S, S'$ and \textsf{ground-truth} alignment is $p$-wise constant

Balcan, DeBlasio, Dick, Kingsford, Sandholm, \textit{Vitercik}, STOC’21
Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:

- **Integer programming**
  - Balcan, Dick, Sandholm, V, ICML’18
  - Balcan, Nagarajan, V, White, COLT’17

- **Clustering**
  - Balcan, Nagarajan, V, White, COLT’17
  - Balcan, Dick, White, NeurIPS’18
  - Balcan, Dick, Lang, ICLR’20

- **Computational biology**
  - Balcan, DeBlasio, Dick, Kingsford, Sandholm, V, STOC’21

- **Greedy algorithms**
  - Gupta, Roughgarden, ITCS’16

- **Mechanism configuration**
  - Balcan, Sandholm, V, EC’18

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

Exploited piecewise-Lipschitz structure to provide regret bounds

[Balcan, Dick, V, FOCS’18; Balcan, Dick, Pegden, UAI’20; Balcan, Dick, Sharma, AISTATS’20]
Piecwise structure

Piecwise structure unifies **seemingly disparate** problems:

- **Integer programming**
  - Balcan, Dick, Sandholm, **V**, ICML’18
  - Balcan, Nagarajan, **V**, White, COLT’17

- **Clustering**
  - Balcan, Nagarajan, **V**, White, COLT’17
  - Balcan, Dick, White, NeurIPS’18
  - Balcan, Dick, Lang, ICLR’20

- **Computational biology**
  - Balcan, DeBlasio, Dick, Kingsford, Sandholm, **V**, STOC’21

- **Greedy algorithms**
  - Gupta, Roughgarden, ITCS’16

- **Mechanism configuration**
  - Balcan, Sandholm, **V**, EC’18

Ties to a long line of research on machine learning for **revenue maximization**:
- Likhodedov, Sandholm, AAAI’04, ’05; Balcan, Blum, Hartline, Mansour, FOCS’05; Elkind, SODA’07; Cole, Roughgarden, STOC’14; Mohri, Medina, ICML’14; Devanur, Huang, Psomas, STOC’16; ...
Primary challenge:
Algorithmic performance is a **volatile** function of parameters

*Complex* connection between parameters and performance

For well-understood functions in machine learning theory:

*Simple* connection between function parameters and value

Balcan, DeBlasio, Dick, Kingsford, Sandholm, *Vitercik*, STOC’21
Outline: Algorithm configuration

1. Overview

2. Model and problem formulation

3. Our guarantees
   a. Example of piecewise-structured utility function
   b. Piecewise-structured functions more formally
   c. Main theorem
   d. Application: Sequence alignment
   e. Online algorithm configuration
Model

$\mathbb{R}^d$: Set of all parameters
$\mathcal{X}$: Set of all inputs
Example: Sequence alignment

\( \mathbb{R}^3 \): Set of alignment algorithm parameters
\( \mathcal{X} \): Set of sequence pairs

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

\[
S = A C T G \\
S' = G T C A
\]
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, …
Algorithmic performance

\[ u_\rho(x) = \text{distance between algorithm’s output and ground-truth} \]

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

Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
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?

**Formally:** Given samples $x_1, ..., x_N \sim D$, 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

Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC'21
Generalization bounds

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

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

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

**Expected utility**

Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
Generalization bounds

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

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

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

Good **average empirical** utility $\Rightarrow$ Good **expected** utility

Balcan, DeBlasio, Dick, Kingsford, Sandholm, *Vitercik*, STOC’21
Outline: Algorithm configuration

1. Overview
2. Model and problem formulation
3. Our guarantees
   a. Example of piecewise-structured utility function
   b. Piecewise-structured functions more formally
   c. Main theorem
   d. Application: Sequence alignment
   e. Online algorithm configuration
Sequence alignment algorithms

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

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

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

Insertion/deletion (indel)

Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
Sequence alignment algorithms

Lemma:
For any pair $S, S'$, there's a small partition of $\mathbb{R}^3$ s.t. in any region, algorithm's output is fixed across all parameters in region $A - - C T G - G T C A -$

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

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 small 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
Outline: Algorithm configuration

1. Overview
2. Model and problem formulation
3. Our guarantees
   a. Example of piecewise-structured utility function
   b. Piecewise-structured functions more formally
   c. Main theorem
   d. Application: Sequence alignment
   e. Online algorithm configuration
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

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

VC dimension, pseudo-dimension, Rademacher complexity, …
Primal & dual classes

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

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

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

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

- Dual functions have simple, Euclidean domain
- Often have ample structure can use to bound complexity of \( \mathcal{U} \)
Dual functions $u^*: \mathbb{R}^d \rightarrow \mathbb{R}$ are piecewise-structured.
Outline: Algorithm configuration

1. Overview
2. Model and problem formulation
3. Our guarantees
   a. Example of piecewise-structured utility function
   b. Piecewise-structured functions more formally
   c. **Main theorem**
   d. Application: Sequence alignment
   e. Online algorithm configuration
Intrinsic complexity

“Intrinsic complexity” of function class $\mathcal{G}$

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

Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
Generalization to future inputs

With high probability, for all $\rho$:

$$\left| \text{Avg utility on training set} - \text{expected utility} \right| = \tilde{O}\left( H \sqrt{\text{Pdim}(\mathcal{G}^*) + \text{VC}(\mathcal{F}^*) \ln k} \right)$$

$\rho_1$, $\rho_2$, $g \in \mathcal{G}$, $f \in \mathcal{F}$

Revenue, runtime, solution quality, ...

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

# boundary functions

Upper bound on utility

Training set size

Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik, STOC’21
Proof sketch

**Theorem:** \(|\text{Avg utility} - \text{expected utility}| = \tilde{O}\left(H \sqrt{\text{Pdim}(\mathcal{G}^*) + \text{VC}(\mathcal{F}^*) \ln k}\right)\)

**Proof sketch:** Fix any set \(S \subseteq X\) of inputs

- Count regions induced by the \(|S|k\) boundaries
  - Depends not on \(\text{VC}(\mathcal{F})\), but rather \(\text{VC}(\mathcal{F}^*)\)
- In each region, \(\{u_x^* : x \in S\}\) are simultaneously structured
  - Count # parameters in region w/ “significantly different” performance
  - Use \(\text{Pdim}(\mathcal{G}^*)\)
- Aggregate bounds over all regions to get:
  \[\text{Pdim}(\mathcal{U}) = O\left(\text{Pdim}(\mathcal{G}^*) + \text{VC}(\mathcal{F}^*) \ln k\right)\]
Outline: Algorithm configuration

1. Overview
2. Model and problem formulation
3. Our guarantees
   a. Example of piecewise-structured utility function
   b. Piecewise-structured functions more formally
   c. Main theorem
   d. Application: Sequence alignment
   e. Online algorithm configuration
Piecewise constant dual functions

**Lemma:**
Utility is piecewise constant function of parameters
Sequence alignment guarantees

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

Balcan, DeBlasio, Dick, Kingsford, Sandholm, *Vitercik*, STOC’21
Outline: Algorithm configuration

1. Overview
2. Model and problem formulation
3. Our guarantees
   a. Example of piecewise-structured utility function
   b. Piecewise-structured functions more formally
   c. Main theorem
   d. Application: Sequence alignment
   e. Online algorithm configuration
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?

Gupta, Roughgarden, ITCS’16; Cohen-Addad, Kanade, AISTATS’17; Balcan, Dick, Vitercik, FOCS’18; Balcan, Dick, Pegden, UAI’20; …
Outline

1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
4. Learning to prune
5. Conclusion and future directions

Book chapter by Mitzenmacher, Vassilvitskii, ‘20
Purohit, Svitkina, Kumar, 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

Main question:
How to use predictions to improve algorithmic performance?
1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
   a. **Searching a sorted array**
   b. Ski rental problem
   c. Design principals and additional research
4. Learning to prune
5. Conclusion and future directions
Example: Searching in a sorted array

Goal: Given query $q$ & sorted array $A$, find $q$’s index (if $q$ in $A$)

Predictor: $h(q) = \text{guess of } q\text{’s index}$

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

Example:

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

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:
- \( q = 8 \)
- \( h(q) = 2 \)
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 \( A[h(q) + 2^i] \) 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: 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 $(h(q) + 2^{i-1}, h(q) + 2^i)$
  - If $q < A[h(q)]$, symmetric

---

Example:
- $q = 8$
- $h(q) = 2$
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 $O(\log|t(q) - h(q)|)$:
  - 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

1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
   a. Searching a sorted array
   b. Ski rental problem
   c. Design principals and additional research
4. Learning to prune
5. Conclusion and future directions
Example: 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
- $\text{CR} = \frac{\text{ALG}}{\text{OPT}} = \frac{x1_{x<b} + (b-1+b)1_{x\geq b}}{\min\{x, b\}} < 2$ (best deterministic)
- Randomized alg. $\text{CR} = \frac{e}{e-1}$ [*Karlin et al., Algorithmica ’94*]
Example: 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
Example: Ski rental problem

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 $\left\lfloor \frac{b}{\lambda} \right\rfloor$

*Don’t jump the gun...* ...*but don’t wait too long*

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

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

Purohit, Svitkina, Kumar, NeurIPS’18
Example: Ski rental problem

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

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

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

Worst when \( x = [\lambda b] \) and \( \text{CR} = \frac{b + [\lambda b] - 1}{[\lambda b]} \leq \frac{1+\lambda}{\lambda} \); similarly for \( y < b \)
Outline

1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
   a. Searching a sorted array
   b. Ski rental problem
   c. Design principals and additional research
4. Learning to prune
5. Conclusion and future directions
Design principals

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

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

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

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

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

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

Also give **randomized algorithm**:

$(\frac{\lambda}{1 - \exp(-\lambda)})$-consistent, $\left( \frac{1}{1 - \exp\left(-\left(\frac{1}{\lambda - 1/b}\right)\right)} \right)$-robust

Bounds are **tight** [Wei, Zhang, NeurIPS’20]
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; …

**Matching**
Antoniadis, Gouleakis, Kleer, Kolev, NeurIPS’20; …

**Queuing**
Mitzenmacher, ACDA’21; …

**Covering problems**
Bamas, Maggiori, Svensson, NeurIPS’20; …

algorithms-with-predictions.github.io
Outline

1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
4. Learning to prune
5. Conclusion and future directions

Alabi, Kalai, Ligett, Musco, Tzamos, Vitercik, COLT’19
Traffic varies daily, but only a few different routes we’d take

Dijkstra’s algorithm wastes time searching muddy dirt roads

Alabi, Kalai, Ligett, Musco, Tzamos, Vitercik, COLT’19
Goal

Quickly solve sequences of similar problems
Exploiting common structures
Speeding up repeated computations

Often, large swaths of search space never contain solutions…

Learn to ignore them!

Only handful of LP constraints ever bind

Large portions of DNA strings never contain patterns of interest

Alabi, Kalai, Ligett, Musco, Tzamos, Vitercik, COLT’19
Model

Function \( f : X \to Y \) maps problem instances \( x \) to solutions \( y \)

Learning algorithm receives sequence \( x_1, \ldots, x_T \in X \)

E.g., each \( x_i \in \mathbb{R}^{|E|} \) equals edge weights for fixed road network
Model

**Goal:** Correctly compute $f$ on most rounds, minimize runtime

*Worst-case algorithm would compute and return $f(x_i)$ for each $x_i$*

Assume access to other functions mapping $X \rightarrow Y$

- Faster to compute
- Defined by subsets (prunings) $S$ of universe $\mathcal{U}$
  - Universe $\mathcal{U}$ represents entire search space
  - Denote corresponding function $f_S : X \rightarrow Y$
  - $f_\mathcal{U} = f$

Example:

- $\mathcal{U} =$ all edges in fixed graph
- $S =$ subset of edges

Alabi, Kalai, Ligett, Musco, Tzamos, Vitercik, COLT’19
Model

**Goal:** Correctly compute \( f \) on most rounds, minimize runtime

*Worst-case algorithm would compute and return \( f(x_i) \) for each \( x_i \)*

Assume access to other functions mapping \( X \rightarrow Y \)

- Faster to compute
- Defined by subsets (prunings) \( S \) of universe \( \mathcal{U} \)
  - Universe \( \mathcal{U} \) represents entire search space
  - Denote corresponding function \( f_S : X \rightarrow Y \)
  - \( f_\mathcal{U} = f \)

Assume exists set \( S^*(x) \subseteq \mathcal{U} \) where \( f_S(x) = f(x) \) iff \( S^*(x) \subseteq S \)

- “Minimally pruned set”
- E.g., the shortest path
Algorithm

1. Initialize pruned set $\tilde{S}_1 \leftarrow \emptyset$
2. For each round $j \in \{1, ..., T\}$:
   a. Receive problem instance $x_j$
   b. With probability $1/\sqrt{j}$, explore:
      i. Output $f(x_j)$
      ii. Compute minimally pruned set $S^*(x_j)$
      iii. Update pruned set: $\tilde{S}_{j+1} \leftarrow \tilde{S}_j \cup S^*(x_j)$
   c. Otherwise (with probability $1 - 1/\sqrt{j}$), exploit:
      i. Output $f_{\tilde{S}_j}(x_j)$
      ii. Don’t update pruned set: $\tilde{S}_{j+1} \leftarrow \tilde{S}_j$
Guarantees

Recap: At round $j$, algorithm outputs $f_{S_j}(x_j)$. $S_j$ depends on $x_{1:j}$.

Goal 1: Minimize $|S_j|$ 
In our applications, time it takes to compute $f_{S_j}(x_j)$ grows with $|S_j|$

Theorem: Let $S^* = \bigcup_{j=1}^{T} S^*(x_j)$
Then $\mathbb{E}\left[\frac{1}{T} \sum_{j=1}^{T} |S_j|\right] \leq |S^*| + \frac{|U| - |S^*|}{\sqrt{T}}$
Guarantees

**Recap:** At round $j$, algorithm outputs $f_{S_j}(x_j)$. $S_j$ depends on $x_{1:j}$.

**Goal 2:** Minimize # of mistakes
Rounds where $f_{S_j}(x_j) \neq f(x_j)$

**Theorem:** $\mathbb{E}[\# ~\text{of mistakes}] \leq \frac{|S^*|}{\sqrt{T}}$, where $S^* = \bigcup_{j=1}^{T} S^*(x_j)$

Alabi, Kalai, Ligett, Musco, Tzamos, Vitercik, COLT'19
Goal: Reach right star from left star

Grey nodes: Nodes $A^*$ explores over 30 rounds

Black nodes: Nodes in the pruned subgraph

Fraction of mistakes: 0.06 over 5000 runs of the algorithm, 30 rounds each
Outline

1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
4. Learning to prune

5. Conclusion and future directions
Conclusions

**Automated configuration**

Applied research dating back several decades

Horvitz et al., UAI’01; Leyton-Brown et al., CP ’03; Likhodedov, Sandholm, AAAI ’04, ’05; …

Learning-theoretic guarantees

Gupta, Roughgarden, ITCS’16; Balcan, DeBlasio, Dick, Kingsford, Sandholm, V, STOC’21; …

**Algorithms with predictions**

Lykouris, Vassilvitskii, ICML’18; Mitzenmacher, NeurIPS’18; Purohit et al., NeurIPS’18; Hsu, Indyk, Katabi, Vakilian, ICLR’19; …

**Learning to prune**

Alabi, Kalai, Ligett, Musco, Tzamos, V, COLT’19
Many open directions with the potential for:

- Deep theoretical analysis
- Significant practical impact
Future directions

What about when you don’t have enough data to learn?

E.g., a shipping company starting out with just one routing IP
Could CPLEX still use ML to optimize performance?

Could similar problems provide guidance?
What does it mean for, say, IPs to be “similar enough”?
Future directions

Which algorithm classes to optimize over?

Classical algorithm design & analysis

Q: Why are some (unexpected) configurations dominant?

Data-driven algorithm design

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.”

Future directions

E.g., Dai et al. [NeurIPS’17] write that their RL alg discovered:
“New and interesting” greedy strategies for MAXCUT and MVC
“which intuitively make sense but have not been analyzed before,”
thus could be a “good assistive tool for discovering new algorithms.”

Similar to how DALL-E will (ideally) serve as an assistive tool for artists

“extremely muscular teapot”
Future directions

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

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

Eventually, solve problems **no one’s ever been able to solve**

Can theory provide guidance about how/when algs generalize?