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

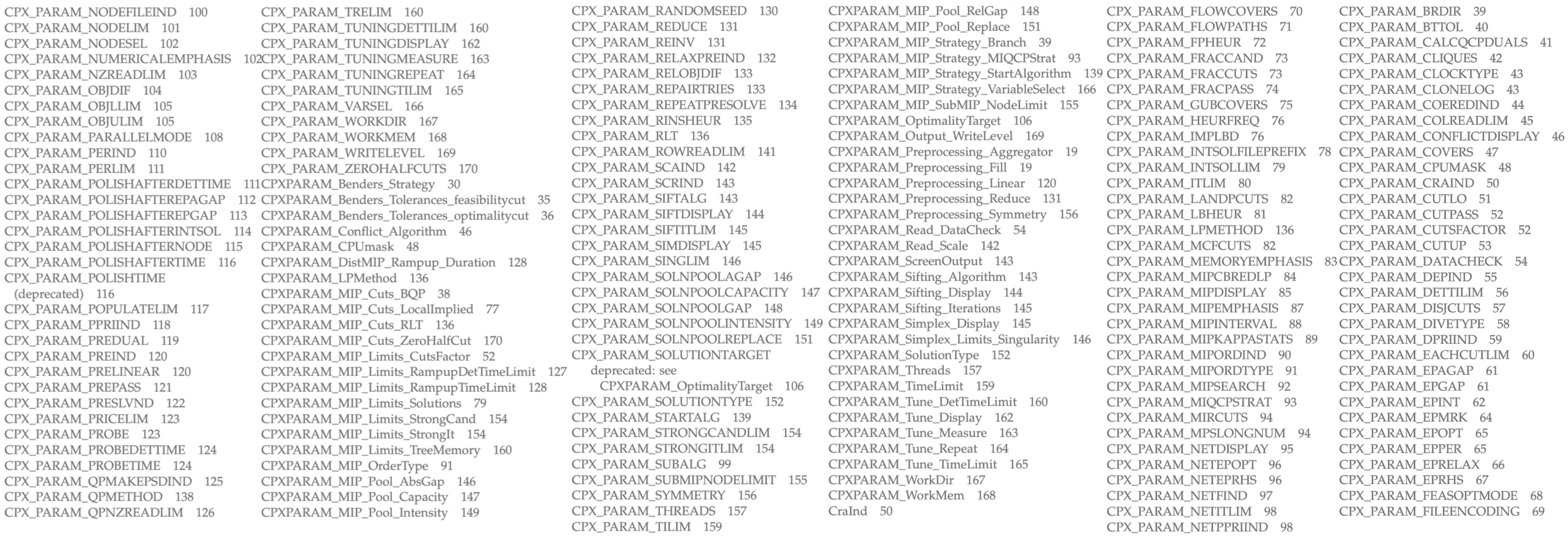
…but they can have **unsatisfactory** default performance

Slow runtime, poor solutions quality, …

Example: Integer programming (IP)

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

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



CPX_PARAM_NODEFILEIND 100
CPX_PARAM_NODELIM 101
CPX_PARAM_NODESEL 102
CPX_PARAM_NUMERICALEMPHASIS 102
CPX_PARAM_NZREADLIM 103
CPX_PARAM_OBJDIF 104
CPX_PARAM_OBJLLIM 105
CPX_PARAM_OBJULIM 105
CPX_PARAM_PARALLELMODE 108
CPX_PARAM_PERIND 110
CPX_PARAM_PERLIM 111
CPX_PARAM_POLISHAFTERDETTIME 111
CPX_PARAM_POLISHAFTEREPAGAP 112
CPX_PARAM_POLISHAFTEREPGAP 113
CPX_PARAM_POLISHAFTERINTSOL 114
CPX_PARAM_POLISHAFTERNODE 115
CPX_PARAM_POLISHAFTERTIME 116
CPX_PARAM_POLISHTIME
(deprecated) 116
CPX_PARAM_POPULATELIM 117
CPX_PARAM_PPRIIND 118
CPX_PARAM_PREDUAL 119
CPX_PARAM_PREIND 120
CPX_PARAM_PRELINEAR 120
CPX_PARAM_PREPASS 121
CPX_PARAM_PRESLVND 122
CPX_PARAM_PRICELIM 123
CPX_PARAM_PROBE 123
CPX_PARAM_PROBEDETTIME 124
CPX_PARAM_PROBETIME 124
CPX_PARAM_QPMAKEPSDIND 125
CPX_PARAM_QPMETHOD 138
CPX_PARAM_QPNZREADLIM 126

CPX_PARAM_TRELIM 160
CPX_PARAM_TUNINGDETTILIM 160
CPX_PARAM_TUNINGDISPLAY 162
CPX_PARAM_TUNINGMEASURE 163
CPX_PARAM_TUNINGREPEAT 164
CPX_PARAM_TUNINGTILIM 165
CPX_PARAM_VARSEL 166
CPX_PARAM_WORKDIR 167
CPX_PARAM_WORKMEM 168
CPX_PARAM_WRITELEVEL 169
CPX_PARAM_ZEROHALFCUTS 170
CPXPARAM_Benders_Strategy 30
CPXPARAM_Benders_Tolerances_feasibilitycut 35
CPXPARAM_Benders_Tolerances_optimalitycut 36
CPXPARAM_Conflict_Algorithm 46
CPXPARAM_CPUmask 48
CPXPARAM_DistMIP_Rampup_Duration 128
CPXPARAM_LPMethod 136
CPXPARAM_MIP_Cuts_BQP 38
CPXPARAM_MIP_Cuts_LocalImplied 77
CPXPARAM_MIP_Cuts_RLT 136
CPXPARAM_MIP_Cuts_ZeroHalfCut 170
CPXPARAM_MIP_Limits_CutsFactor 52
CPXPARAM_MIP_Limits_RampupDetTimeLimit 127
CPXPARAM_MIP_Limits_RampupTimeLimit 128
CPXPARAM_MIP_Limits_Solutions 79
CPXPARAM_MIP_Limits_StrongCand 154
CPXPARAM_MIP_Limits_StrongIt 154
CPXPARAM_MIP_Limits_TreeMemory 160
CPXPARAM_MIP_OrderType 91
CPXPARAM_MIP_Pool_AbsGap 146
CPXPARAM_MIP_Pool_Capacity 147
CPXPARAM_MIP_Pool_Intensity 149

CPX_PARAM_RANDOMSEED 130
CPX_PARAM_REDUCE 131
CPX_PARAM_REINV 131
CPX_PARAM_RELAXPREIND 132
CPX_PARAM_RELOBJDIF 133
CPX_PARAM_REPAIRTRIES 133
CPX_PARAM_REPEATPRESOLVE 134
CPX_PARAM_RINSHEUR 135
CPX_PARAM_RLT 136
CPX_PARAM_ROWREADLIM 141
CPX_PARAM_SCAIND 142
CPX_PARAM_SCRIND 143
CPX_PARAM_SIFTALG 143
CPX_PARAM_SIFTDISPLAY 144
CPX_PARAM_SIFTITLIM 145
CPX_PARAM_SIMDISPLAY 145
CPX_PARAM_SINGLIM 146
CPX_PARAM_SOLNPOOLAGAP 146
CPX_PARAM_SOLNPOOLCAPACITY 147
CPX_PARAM_SOLNPOOLGAP 148
CPX_PARAM_SOLNPOOLINTENSITY 149
CPX_PARAM_SOLNPOOLREPLACE 151
CPX_PARAM_SOLUTIONTARGET
deprecated: see
CPXPARAM_OptimalityTarget 106
CPX_PARAM_SOLUTIONTYPE 152
CPX_PARAM_STARTALG 139
CPX_PARAM_STRONGCANDLIM 154
CPX_PARAM_STRONGITLIM 154
CPX_PARAM_SUBALG 99
CPX_PARAM_SUBMIPNODELIMIT 155
CPX_PARAM_SYMMETRY 156
CPX_PARAM_THREADS 157
CPX_PARAM_TILIM 159

CPXPARAM_MIP_Pool_RelGap 148
CPXPARAM_MIP_Pool_Replace 151
CPXPARAM_MIP_Strategy_Branch 39
CPXPARAM_MIP_Strategy_MIQCPStrat 93
CPXPARAM_MIP_Strategy_StartAlgorithm 139
CPXPARAM_MIP_Strategy_VariableSelect 166
CPXPARAM_MIP_SubMIP_NodeLimit 155
CPXPARAM_OptimalityTarget 106
CPXPARAM_Output_WriteLevel 169
CPXPARAM_Preprocessing_Aggregator 19
CPXPARAM_Preprocessing_Fill 19
CPXPARAM_Preprocessing_Linear 120
CPXPARAM_Preprocessing_Reduce 131
CPXPARAM_Preprocessing_Symmetry 156
CPXPARAM_Read_DataCheck 54
CPXPARAM_Read_Scale 142
CPXPARAM_ScreenOutput 143
CPXPARAM_Sifting_Algorithm 143
CPXPARAM_Sifting_Display 144
CPXPARAM_Sifting_Iterations 145
CPXPARAM_Simplex_Display 145
CPXPARAM_Simplex_Limits_Singularity 146
CPXPARAM_SolutionType 152
CPXPARAM_Threads 157
CPXPARAM_TimeLimit 159
CPXPARAM_Tune_DetTimeLimit 160
CPXPARAM_Tune_Display 162
CPXPARAM_Tune_Measure 163
CPXPARAM_Tune_Repeat 164
CPXPARAM_Tune_TimeLimit 165
CPXPARAM_WorkDir 167
CPXPARAM_WorkMem 168
CraInd 50

CPX_PARAM_FLOWCOVERS 70
CPX_PARAM_FLOWPATHS 71
CPX_PARAM_FPHEUR 72
CPX_PARAM_FRACCAND 73
CPX_PARAM_FRACCUTS 73
CPX_PARAM_FRACPASS 74
CPX_PARAM_GUBCOVERS 75
CPX_PARAM_HEURFREQ 76
CPX_PARAM_IMPLBD 76
CPX_PARAM_INTSOLFILEPREFIX 78
CPX_PARAM_INTSOLLIM 79
CPX_PARAM_ITLIM 80
CPX_PARAM_LANDPCUTS 82
CPX_PARAM_LBHEUR 81
CPX_PARAM_LPMETHOD 136
CPX_PARAM_MCFCUTS 82
CPX_PARAM_MEMORYEMPHASIS 83
CPX_PARAM_MIPCBREDLP 84
CPX_PARAM_MIPDISPLAY 85
CPX_PARAM_MIPEMPHASIS 87
CPX_PARAM_MIPINTERVAL 88
CPX_PARAM_MIPKAPPASTATS 89
CPX_PARAM_MIPORDIND 90
CPX_PARAM_MIPORDTYPE 91
CPX_PARAM_MIPSEARCH 92
CPX_PARAM_MIQCPSTRAT 93
CPX_PARAM_MIRCUTS 94
CPX_PARAM_MPSLONGNUM 94
CPX_PARAM_NETDISPLAY 95
CPX_PARAM_NETEPOPT 96
CPX_PARAM_NETEPRHS 96
CPX_PARAM_NETFIND 97
CPX_PARAM_NETITLIM 98
CPX_PARAM_NETPPRIIND 98

CPX_PARAM_BRDIR 39
CPX_PARAM_BTTOL 40
CPX_PARAM_CALCQCPDUALS 41
CPX_PARAM_CLIQUES 42
CPX_PARAM_CLOCKTYPE 43
CPX_PARAM_CLONELOG 43
CPX_PARAM_COEREDIND 44
CPX_PARAM_COLREADLIM 45
CPX_PARAM_CONFLICTDISPLAY 46
CPX_PARAM_COVERS 47
CPX_PARAM_CPUMASK 48
CPX_PARAM_CRAIND 50
CPX_PARAM_CUTLO 51
CPX_PARAM_CUTPASS 52
CPX_PARAM_CUTSFACTOR 52
CPX_PARAM_CUTUP 53
CPX_PARAM_DATACHECK 54
CPX_PARAM_DEPIND 55
CPX_PARAM_DETTILIM 56
CPX_PARAM_DISJCUTS 57
CPX_PARAM_DIVETYPE 58
CPX_PARAM_DPRIIND 59
CPX_PARAM_EACHCUTLIM 60
CPX_PARAM_EPAGAP 61
CPX_PARAM_EPGAP 61
CPX_PARAM_EPINT 62
CPX_PARAM_EPMRK 64
CPX_PARAM_EPOPT 65
CPX_PARAM_EPPER 65
CPX_PARAM_EPRELAX 66
CPX_PARAM_EPRHS 67
CPX_PARAM_FEASOPTMODE 68
CPX_PARAM_FILEENCODING 69

Example: Integer programming (IP)

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**

Example: Sequence alignment

Goal: Line up pairs of strings

Applications: Biology, natural language processing, etc.

vitterchik

Did you mean: **vitercik**

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

A – – C T G

– G T C A –

Insertion/deletion (*indel*)

Match

Mismatch

Example: Sequence alignment

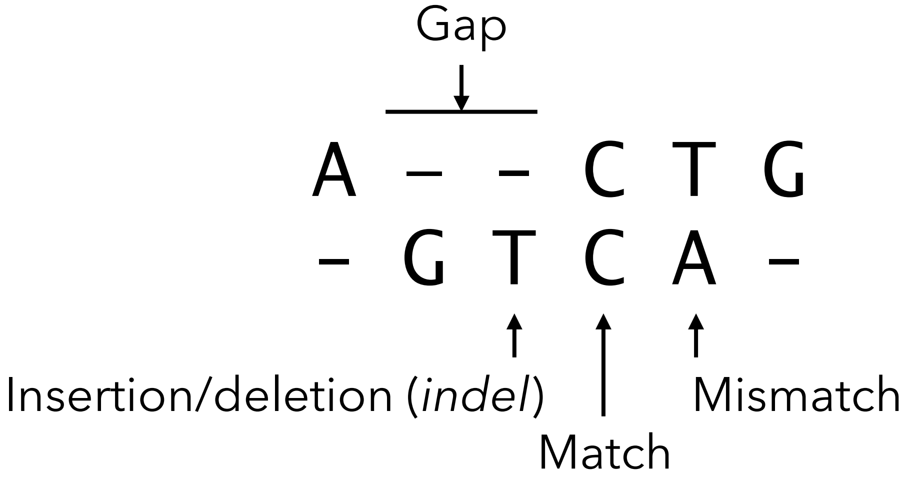
Standard algorithm with parameters $\rho_1, \rho_2, \rho_3 \geq 0$:

Return alignment maximizing:

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

$S = \mathrm{A\ C\ T\ G}$

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



Gap

A – – C T G

– G T C A –

Insertion/deletion (*indel*)

Match

Mismatch

Example: Sequence alignment

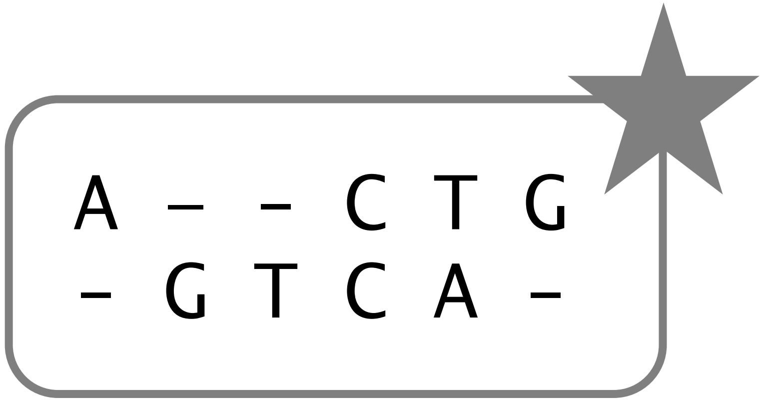
Can sometimes access **ground-truth, reference** alignment

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]



A – – C T G
– G T C A –

Example: Sequence alignment

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

Ground-truth alignment of protein sequences

Example: Sequence alignment

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

Ground-truth alignment of protein sequences

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

Alignment by algorithm with poorly-tuned parameters

Example: Sequence alignment

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

Ground-truth alignment of protein sequences

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

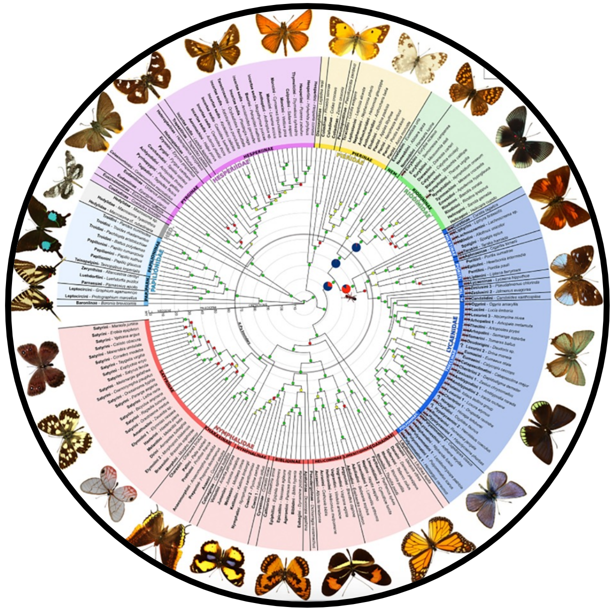
Alignment by algorithm with poorly-tuned parameters

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

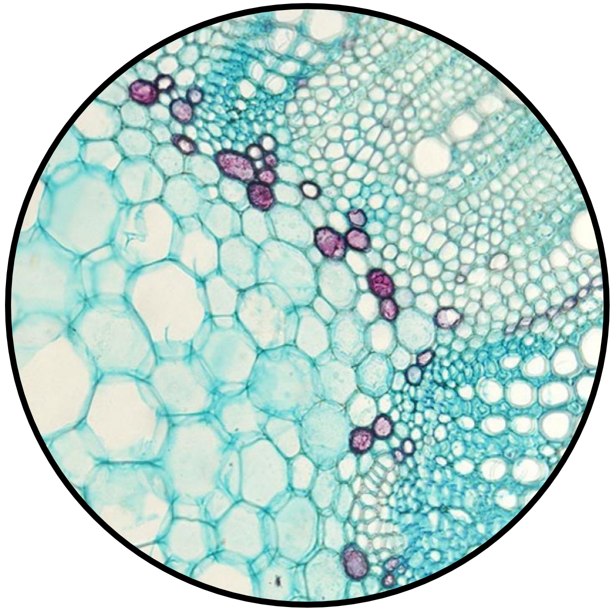
Alignment by algorithm with well-tuned parameters

Example: Clustering

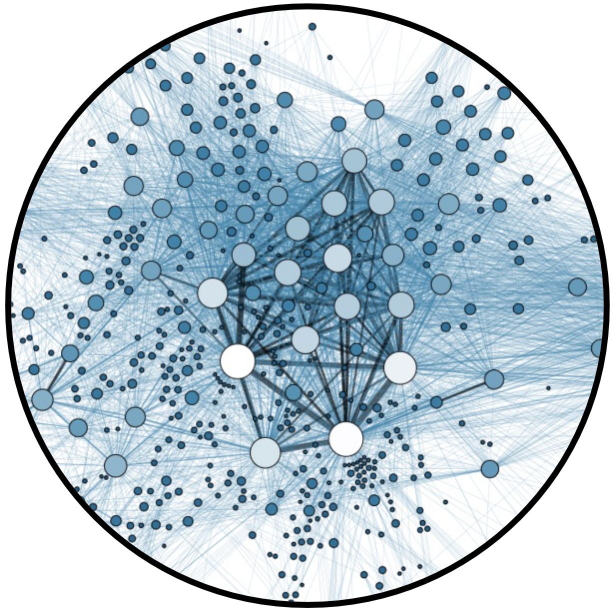
Diverse applications, including:



Ecology



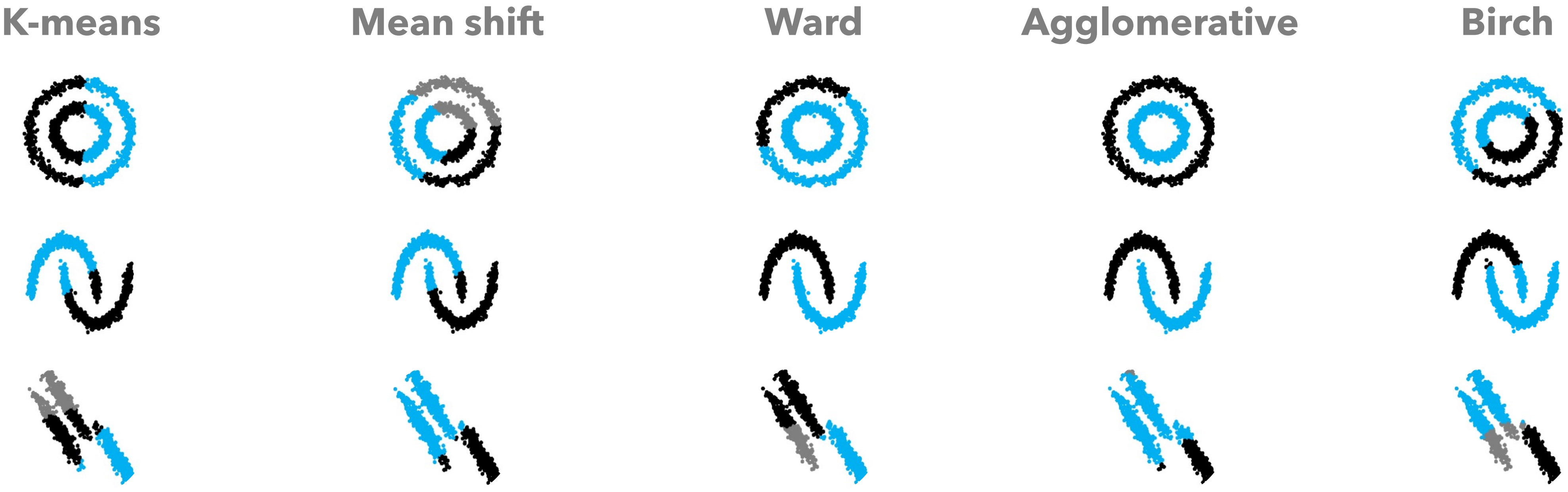
Biology



Network analysis

Example: Clustering

Many different algorithms



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

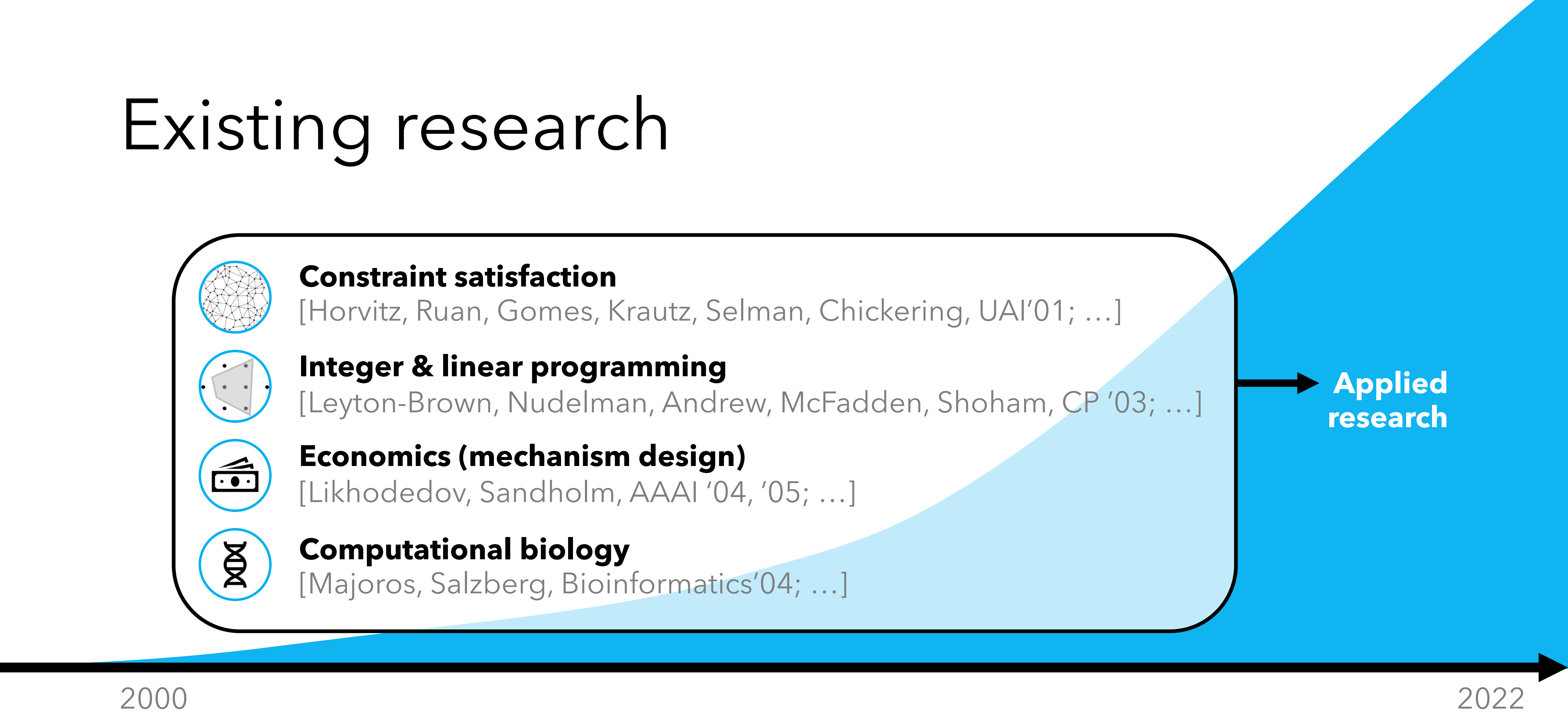
In practice, we have data about the application domain

Clustering problems a biology lab solves

In practice, we have data about the application domain

Scheduling problems an airline solves

Existing research



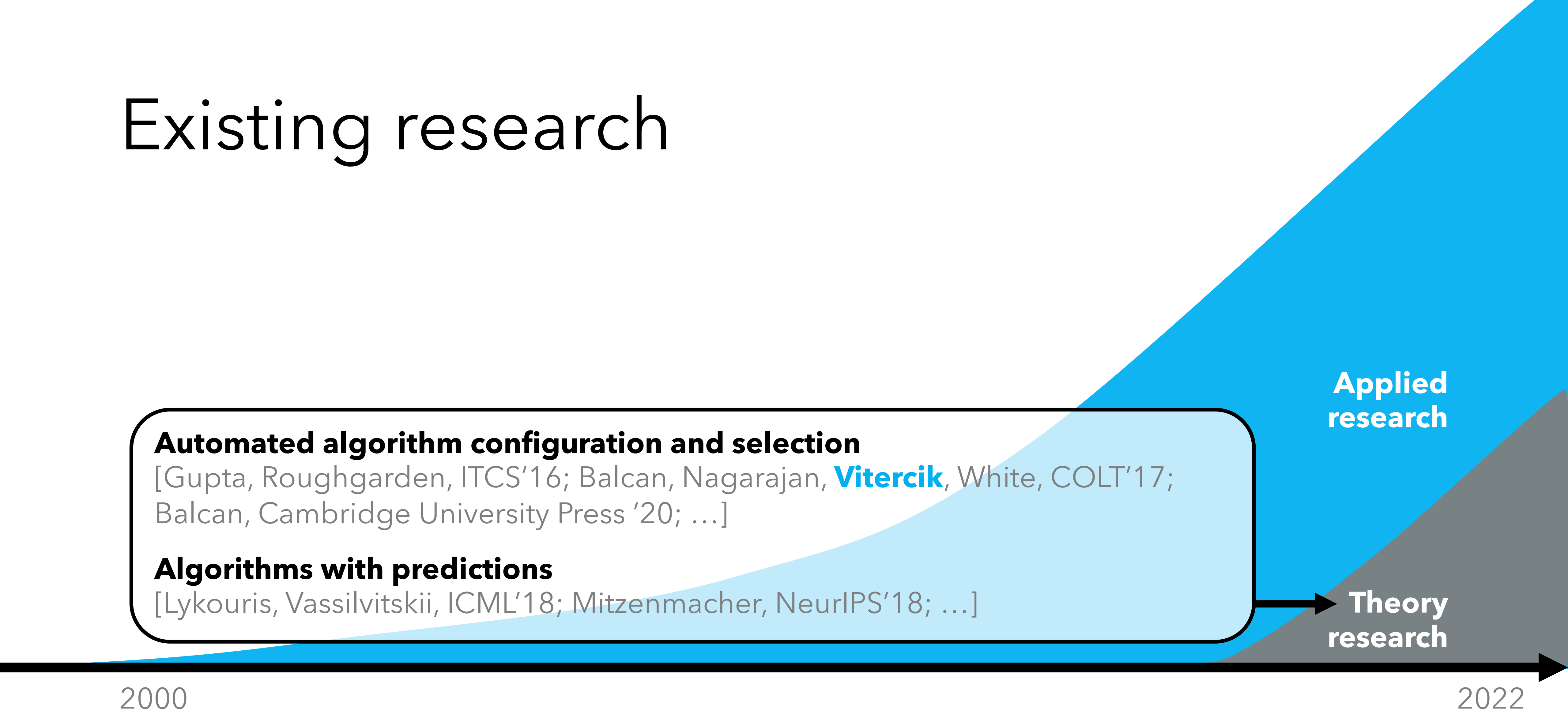
Constraint satisfaction
[Horvitz, Ruan, Gomes, Krautz, Selman, Chickering, UAI’01; …]

Integer & 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 $\mathcal{D}$

Sequence S_1
Sequence S_1'
Reference alignment A_1

Sequence S_2
Sequence S_2'
Reference alignment A_2

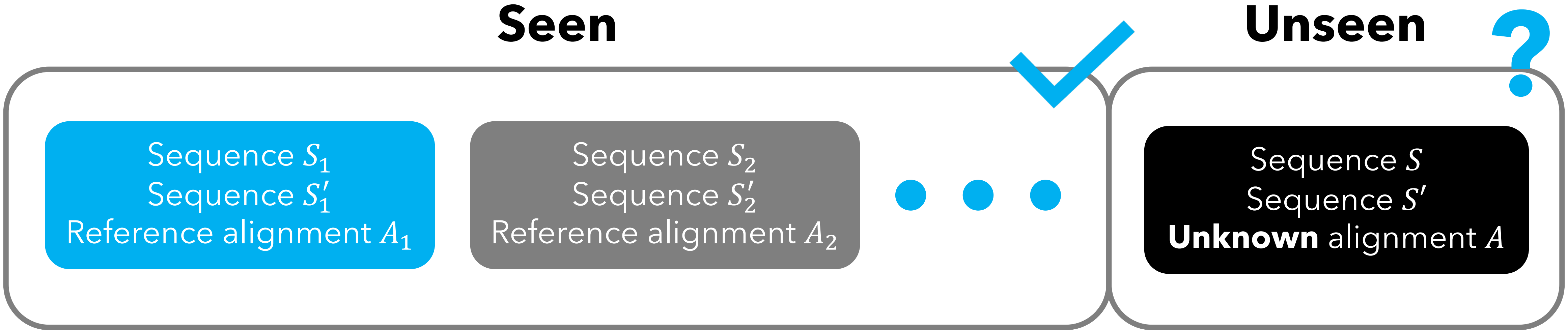
3. Return parameter setting $\hat{\boldsymbol{\rho}}$ with good avg performance

Runtime, solution quality, etc.

Key question: How to find $\hat{\boldsymbol{\rho}}$ 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

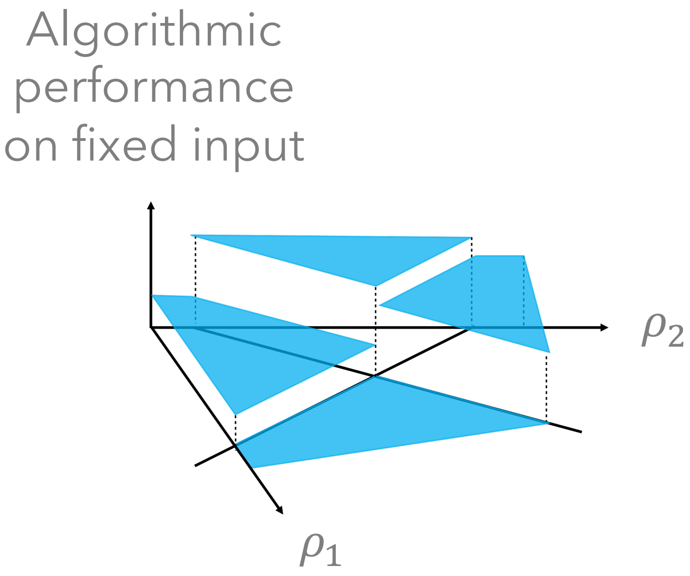
Piecewise constant, linear, quadratic, …

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

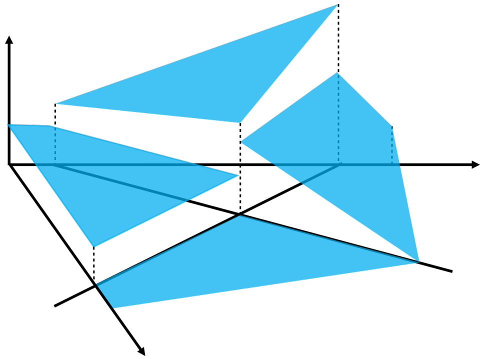
Results overview

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

Piecewise constant, linear, quadratic, …



Piecewise constant



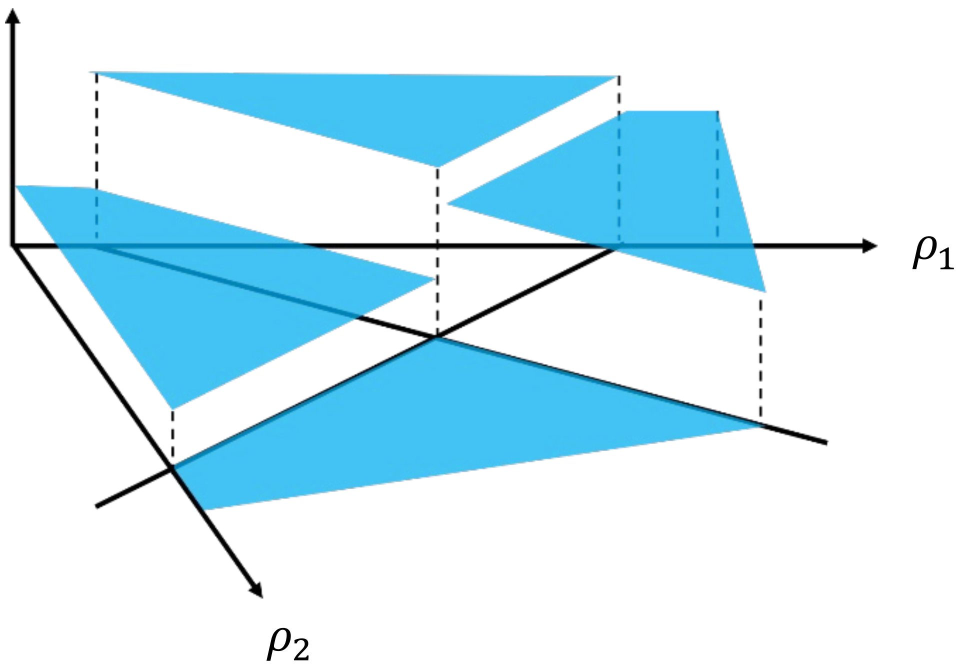
Piecewise linear

Piecewise …

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

Example: Sequence alignment

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



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

Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:

Integer programming
Balcan, Dick, Sandholm, **V**, ICML'18
Balcan, 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]

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

Piecewise structure

Piecewise structure unifies **seemingly disparate** problems:

Integer programming
Balcan, Dick, Sandholm, **V**, ICML’18
Balcan, 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

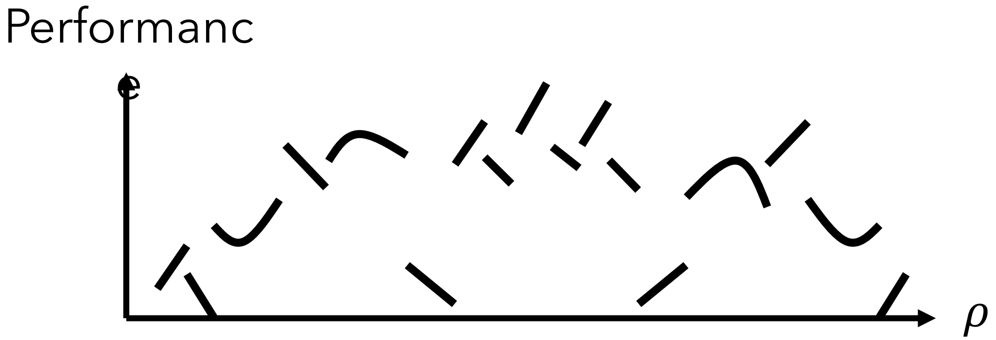
Mechanism configuration
Balcan, Sandholm, **V**, EC’18

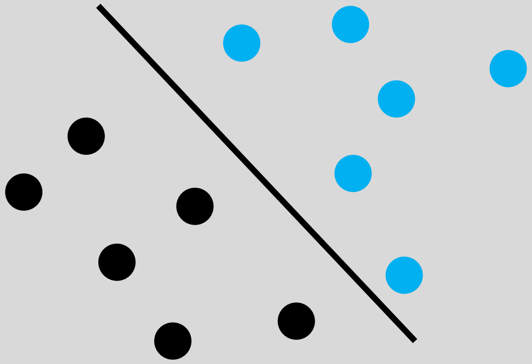
Ties to a long line of research on machine learning for **revenue maximization**
Likhodedov, Sandholm, AAAI'04, ’05; Balcan, Blum, Hartline, Mansour, FOCS’05; Elkind, SODA’07; Cole, Roughgarden, STOC’14; Mohri, Medina, ICML’14; Devanur, Huang, Psomas, STOC’16; …

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

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
   1. Example of piecewise-structured utility function
   2. Piecewise-structured functions more formally
   3. Main theorem
   4. Application: Sequence alignment
   5. Online algorithm configuration

Model

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

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

Example: Sequence alignment

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

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

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

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

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

Algorithmic performance

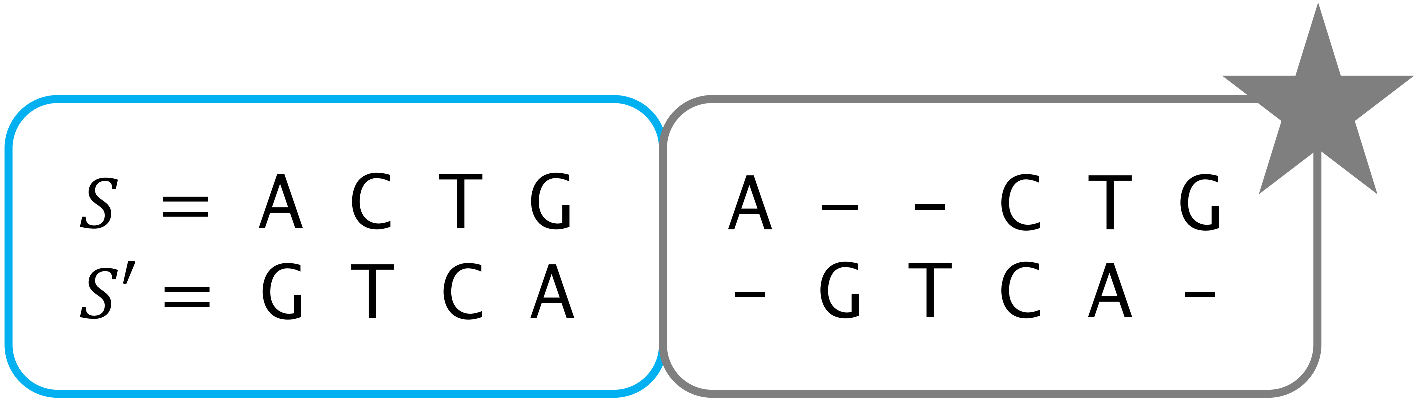
$u_{\boldsymbol{\rho}}(x)$ = utility of algorithm parameterized by $\boldsymbol{\rho} \in \mathbb{R}^d$ on input x

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

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

Algorithmic performance

$u_{\boldsymbol{\rho}}(x)$ = distance between algorithm’s output and ground-truth



$S = \mathrm{A\ C\ T\ G}$

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

A – – C T G

– G T C A –

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

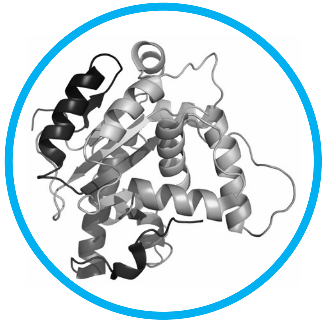
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC’21

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

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

Generalization bounds

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

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

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

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

Generalization bounds

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

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

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

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

Generalization bounds

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

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

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

Good **average empirical** utility ➡ 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

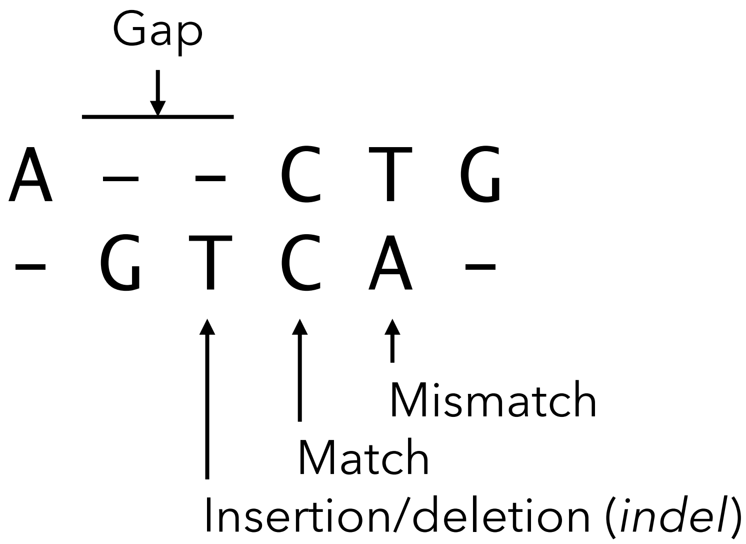
Sequence alignment algorithms

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

Return alignment maximizing:

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

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



Gap

A – – C T G

– G T C A –

Mismatch

Match

Insertion/deletion (*indel*)

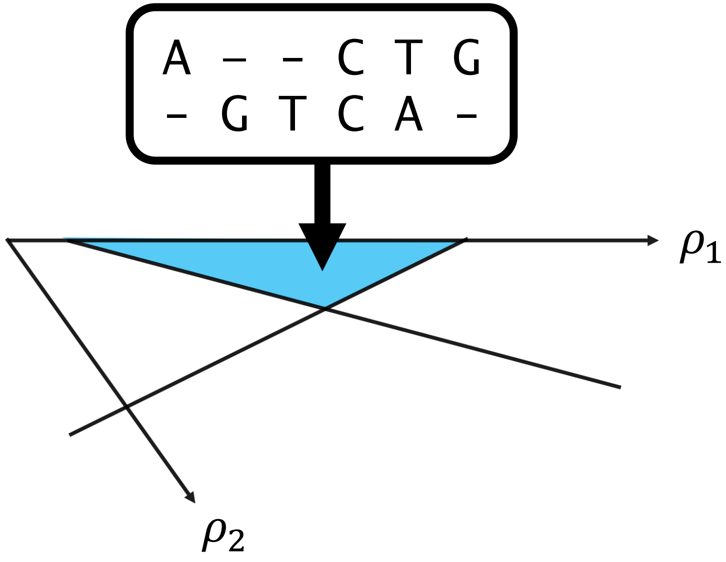
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC’21

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

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

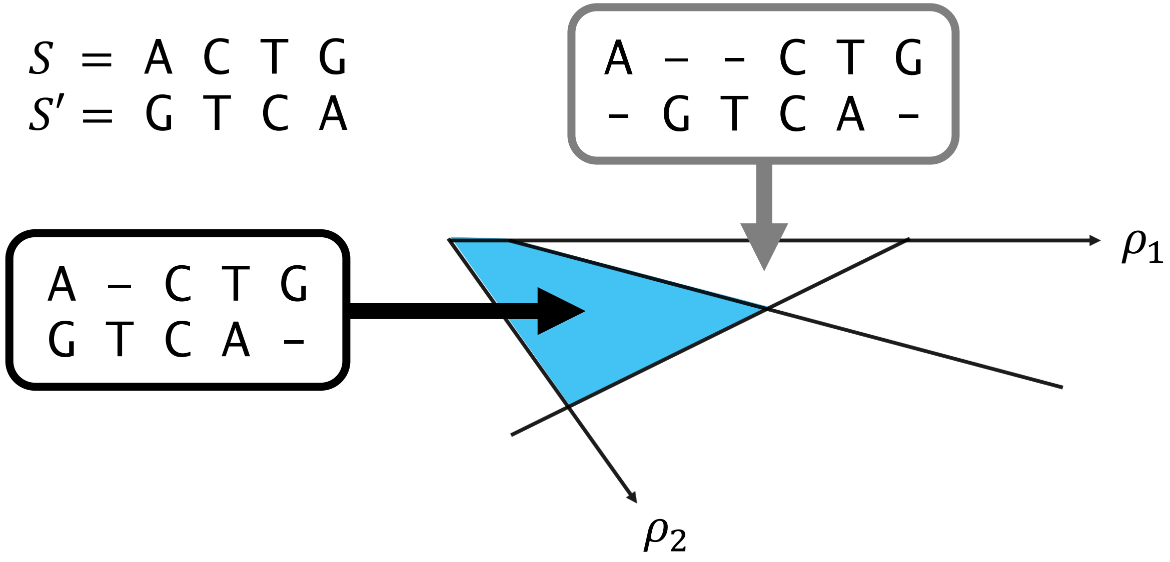

A – – C T G
– G T C A –

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

Sequence alignment algorithms

Lemma:

For any pair S, S', there's a 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

A – – C T G
– G T C A –

A – C T G
G T C A –

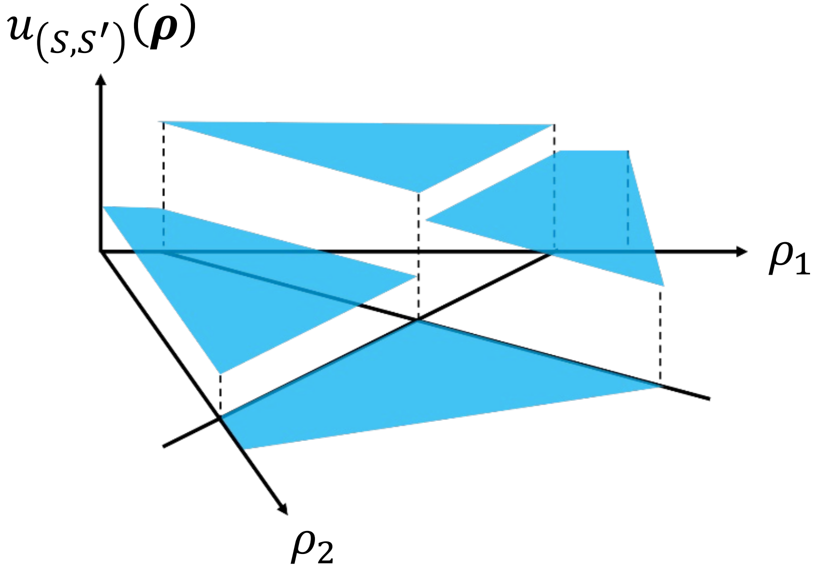
Gusfield et al., Algorithmica '94; Fernández-Baca et al., J. of Discrete Alg. '04

Piecewise-constant utility function

Corollary:

Utility is piecewise constant function of parameters

Distance between algorithm's output and ground-truth alignment



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

Outline: 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

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

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

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

VC dimension, pseudo-dimension, Rademacher complexity, …

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

Primal & dual classes

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

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

Challenge: $\mathcal{U}$ is gnarly

E.g., in sequence alignment:

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

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

Primal & dual classes

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

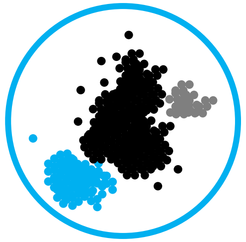
$u_x^*(\boldsymbol{\rho})$ = utility as function of parameters
$u_x^*(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(x)$
$\mathcal{U}^* = \left\{u_x^*: \mathbb{R}^d \to \mathbb{R} \,\middle|\, x \in \mathcal{X}\right\}$ "Dual" function class

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

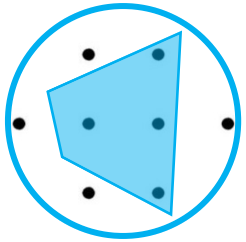
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Piecewise-structured functions

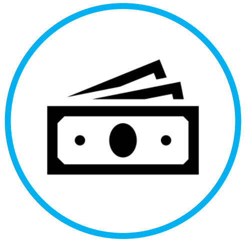
Dual functions $u_x^*: \mathbb{R}^d \rightarrow \mathbb{R}$ are piecewise-structured



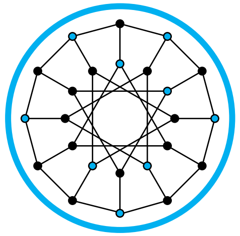
Clustering algorithm configuration



Integer programming algorithm configuration



Selling mechanism configuration



Greedy algorithm configuration

Computational biology algorithm configuration

Voting mechanism configuration

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

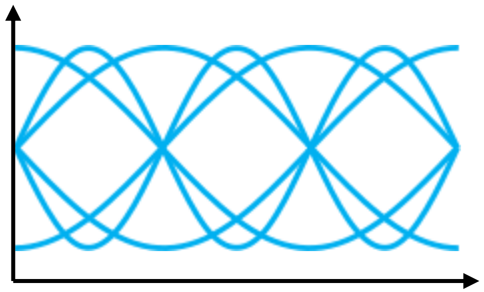
Outline: Algorithm configuration

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

Intrinsic complexity

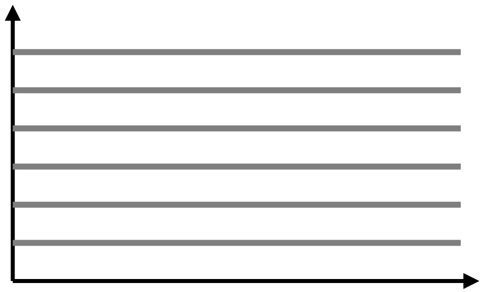
"Intrinsic complexity" of function class $\mathcal{G}$

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



More complex

Less complex



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

Generalization to future inputs

With high probability, for all $\boldsymbol{\rho}$:

|**Avg** utility on training set – **expected** utility|

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

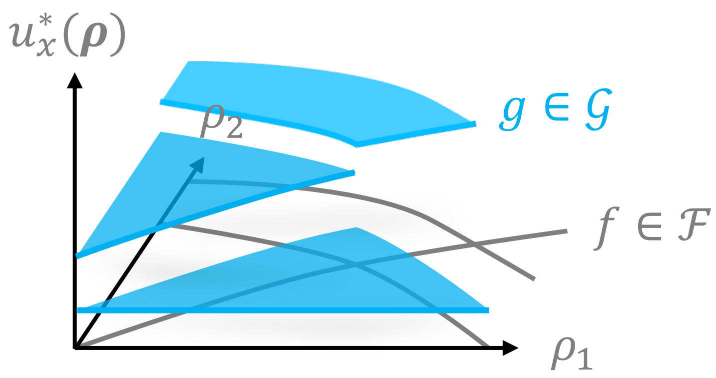
\# boundary functions

$$= \tilde{O}\left(H\sqrt{\frac{\mathrm{Pdim}(\mathcal{G}^*)+\mathrm{VC}(\mathcal{F}^*)\ln k}{N}}\right)$$

Upper bound on utility

Training set size

Revenue, runtime, solution quality, …



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

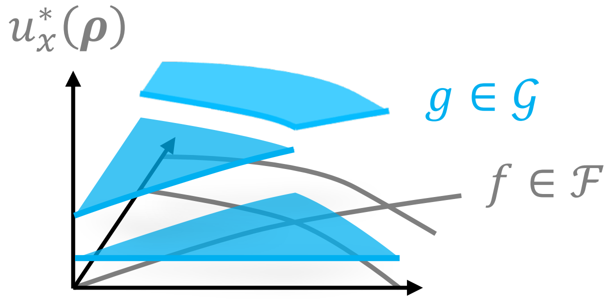
Proof sketch

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

Proof sketch: Fix any set $S \subseteq \mathcal{X}$ of inputs

- Count regions induced by the $|S|k$ boundaries
  - Depends not on $\mathrm{VC}(\mathcal{F})$, but rather $\mathrm{VC}(\mathcal{F}^*)$
- In each region, $\{u_x^* : x \in S\}$ are simultaneously structured
  - Count # parameters in region w/ "significantly different" performance
  - Use $\mathrm{Pdim}(\mathcal{G}^*)$
- Aggregate bounds over all regions to get:

$$\mathrm{Pdim}(\mathcal{U}) = O(\mathrm{Pdim}(\mathcal{G}^*) + \mathrm{VC}(\mathcal{F}^*)\ln k)$$



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

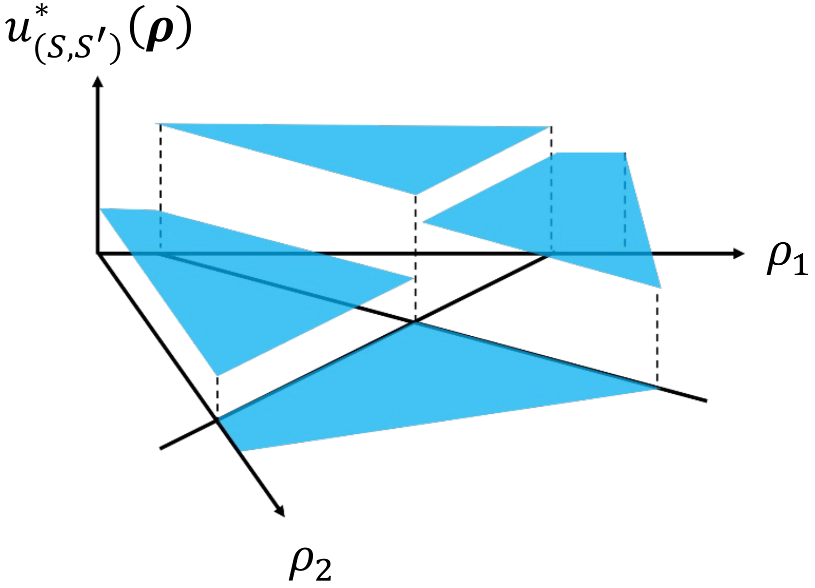
Outline: Algorithm configuration

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

Piecewise constant dual functions

Lemma:

Utility is piecewise constant function of parameters

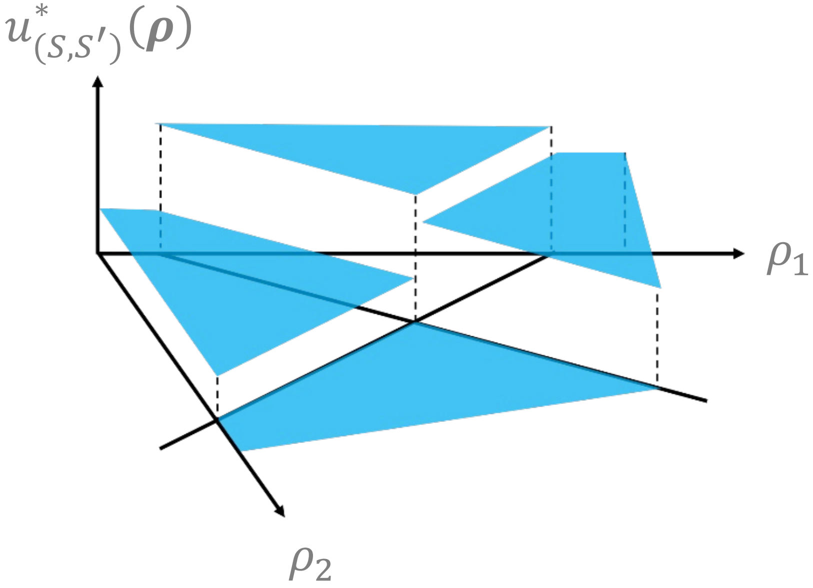


Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

Sequence alignment guarantees

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

|**avg** utility over training set – **exp** utility| $\leq \epsilon$



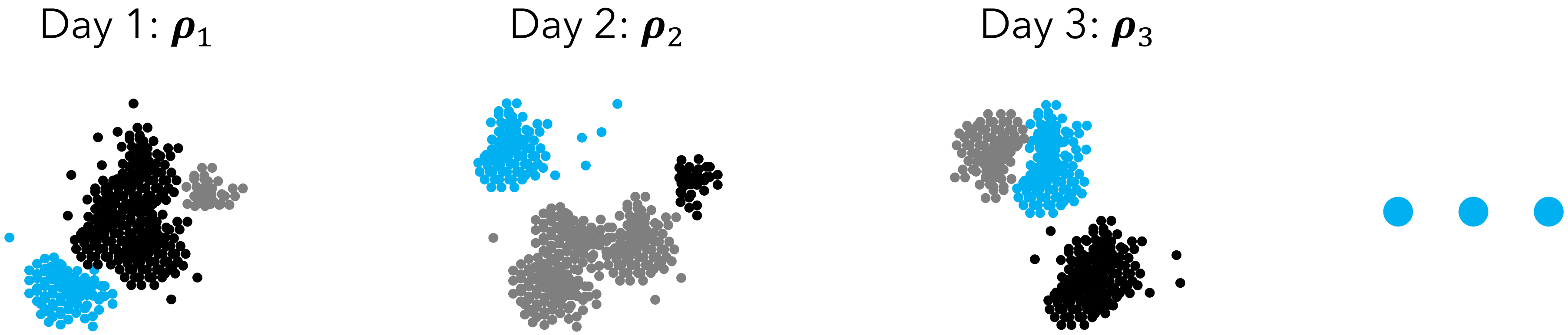
Balcan, DeBlasio, Dick, Kingsford, Sandholm, **Vitercik**, STOC'21

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

Goal: Compete with best parameter setting in hindsight

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

Gupta, 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?

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

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

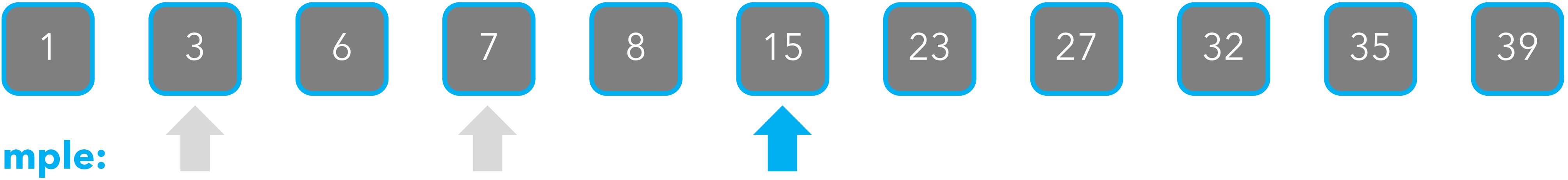
Example:

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

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

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array



Example:

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

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

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

Example:

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

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

Book chapter by Mitzenmacher, Vassilvitskii, '20

Example: Searching in a sorted array

1	3	6	7	8	15	23	27	32	35	39

Analysis:

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

Book chapter by Mitzenmacher, Vassilvitskii, '20

Outline

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 $\mathrm{OPT} = \min\{x, b\}$

Breakeven strategy: Rent for $b - 1$ days, then buy

- $\mathrm{CR} = \frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{x\mathbf{1}_{\{x<b\}} + (b-1+b)\mathbf{1}_{\{x\geq b\}}}{\min\{x,b\}} < 2$ (best deterministic)
- Randomized alg. $\mathrm{CR} = \frac{e}{e-1}$ [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: $\mathrm{ALG} \leq \mathrm{OPT} + \eta$

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

Purohit, Svitkina, Kumar, NeurIPS'18

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

Don't jump the gun… *…but don't wait too long*

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

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

Purohit, Svitkina, Kumar, NeurIPS'18

Example: Ski rental problem

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

Proof sketch: If $y \geq b$, buys on start of day $\lceil \lambda b \rceil$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \begin{cases} \dfrac{x}{x} & \text{if } x < \lceil \lambda b \rceil \\ \dfrac{\lceil \lambda b \rceil - 1 + b}{x} & \text{if } \lceil \lambda b \rceil \leq x \leq b \\ \dfrac{\lceil \lambda b \rceil - 1 + b}{b} & \text{if } x \geq b \end{cases}$$

Worst when $x = \lceil \lambda b \rceil$ and CR = $\frac{b+\lceil \lambda b \rceil - 1}{\lceil \lambda b \rceil} \leq \frac{1+\lambda}{\lambda}$; similarly for $y < b$

Purohit, Svitkina, Kumar, NeurIPS'18

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 α-consistent if CR → α as error $\eta \to 0$

Robustness:

- Predictions are terrible ⇒ no worse than worst-case
- Algorithm is β-consistent if CR ≤ β for all η

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

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

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

Lykouris, Vassilvitskii, ICML'18

Design principals

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

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

Also give **randomized algorithm**:

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

Bounds are **tight** [Wei, Zhang, NeurIPS'20]

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

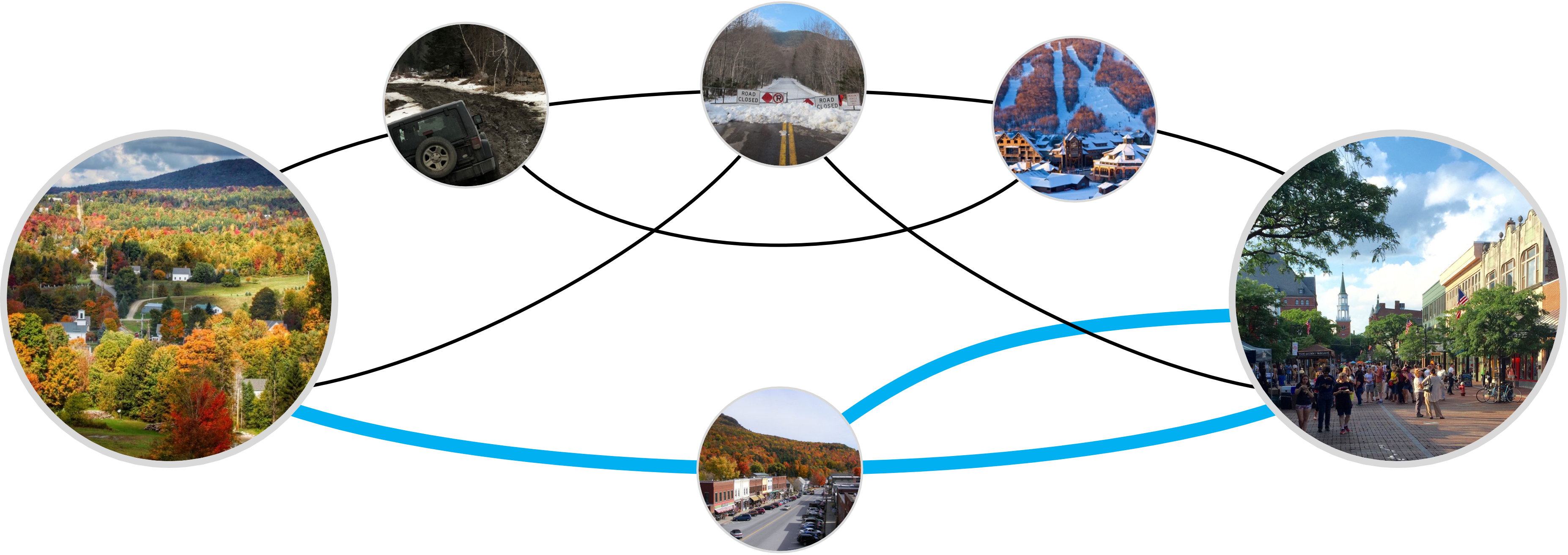
1. Introduction
2. Algorithm configuration
3. Algorithms with predictions
4. **Learning to prune**
5. Conclusion and future directions

Alabi, Kalai, Ligett, Musco, Tzamos, **Vitercik**, COLT'19

Lincoln, Vermont

Burlington, Vermont

Traffic varies daily, but only a few different routes we'd take



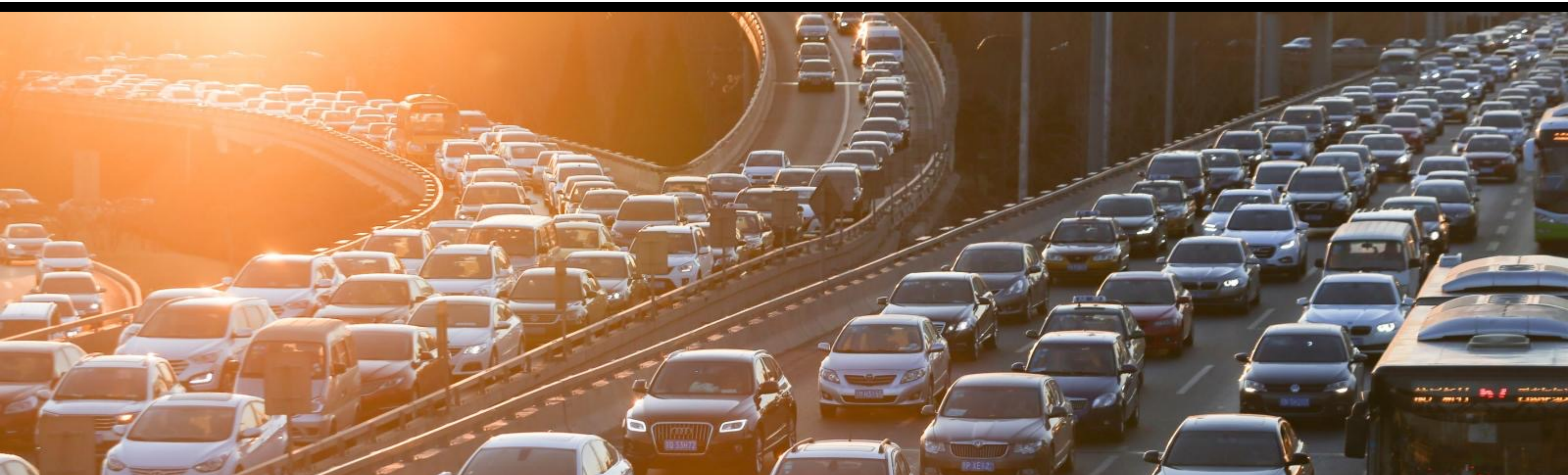
Dijkstra's algorithm wastes time searching muddy dirt roads

Alabi, Kalai, Ligett, Musco, Tzamos, **Vitercik**, COLT'19

Goal

Quickly solve sequences of similar problems

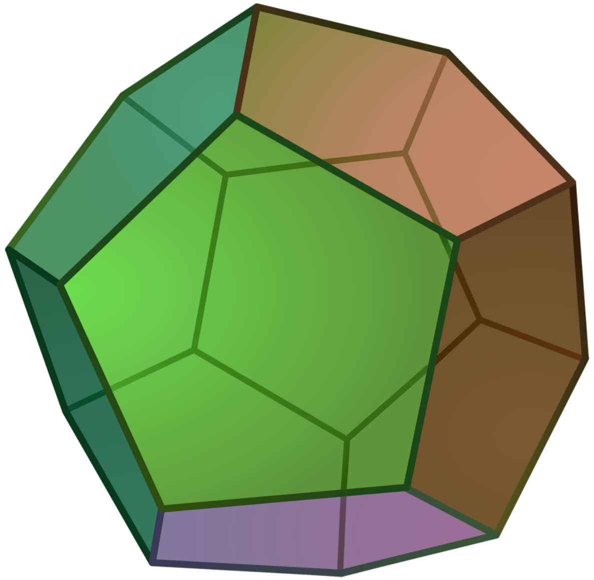
- Exploiting common structures



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 $x_1, \ldots, x_T \in X$

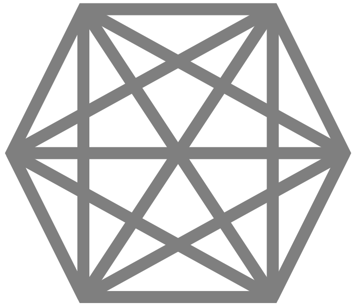
E.g., each $x_i \in \mathbb{R}^{|E|}$ equals edge weights for fixed road network

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 $X \to Y$

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

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

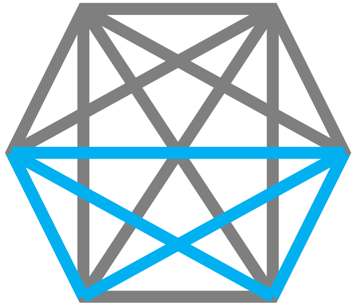
Alabi, Kalai, Ligett, Musco, Tzamos, **Vitercik**, COLT'19

Algorithm

```
1. Initialize pruned set $\bar{S}_1 \leftarrow \emptyset$
2. For each round $j \in \{1, \ldots, T\}$:
    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: $\bar{S}_{j+1} \leftarrow \bar{S}_j \cup S^*(x_j)$
    c. Otherwise (with probability $1 - 1/\sqrt{j}$), exploit:
        i. Output $f_{\bar{S}_j}(x_j)$
        ii. Don't update pruned set: $\bar{S}_{j+1} \leftarrow \bar{S}_j$
```

Alabi, Kalai, Ligett, Musco, Tzamos, **Vitercik**, COLT'19

Guarantees



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

S_j depends on $x_{1:j}$.

Goal 1: Minimize $|S_j|$

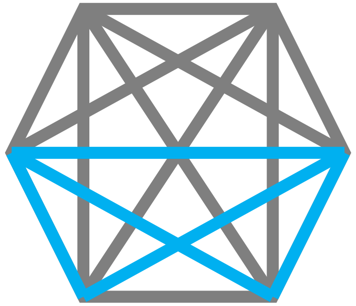
In our applications, time it takes to compute $f_{S_j}(x_j)$ grows with $|S_j|$

Theorem: Let $S^* = \bigcup_{j=1}^{T} S^*(x_j)$

Then $\mathbb{E}\left[\frac{1}{T}\sum_{j=1}^{T}|S_j|\right] \leq |S^*| + \frac{|\mathcal{U}|-|S^*|}{\sqrt{T}}$

Alabi, Kalai, Ligett, Musco, Tzamos, **Vitercik**, COLT'19

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: $\mathbb{E}$[# 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

Alabi, Kalai, Ligett, Musco, Tzamos, **Vitercik**, COLT'19

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; …

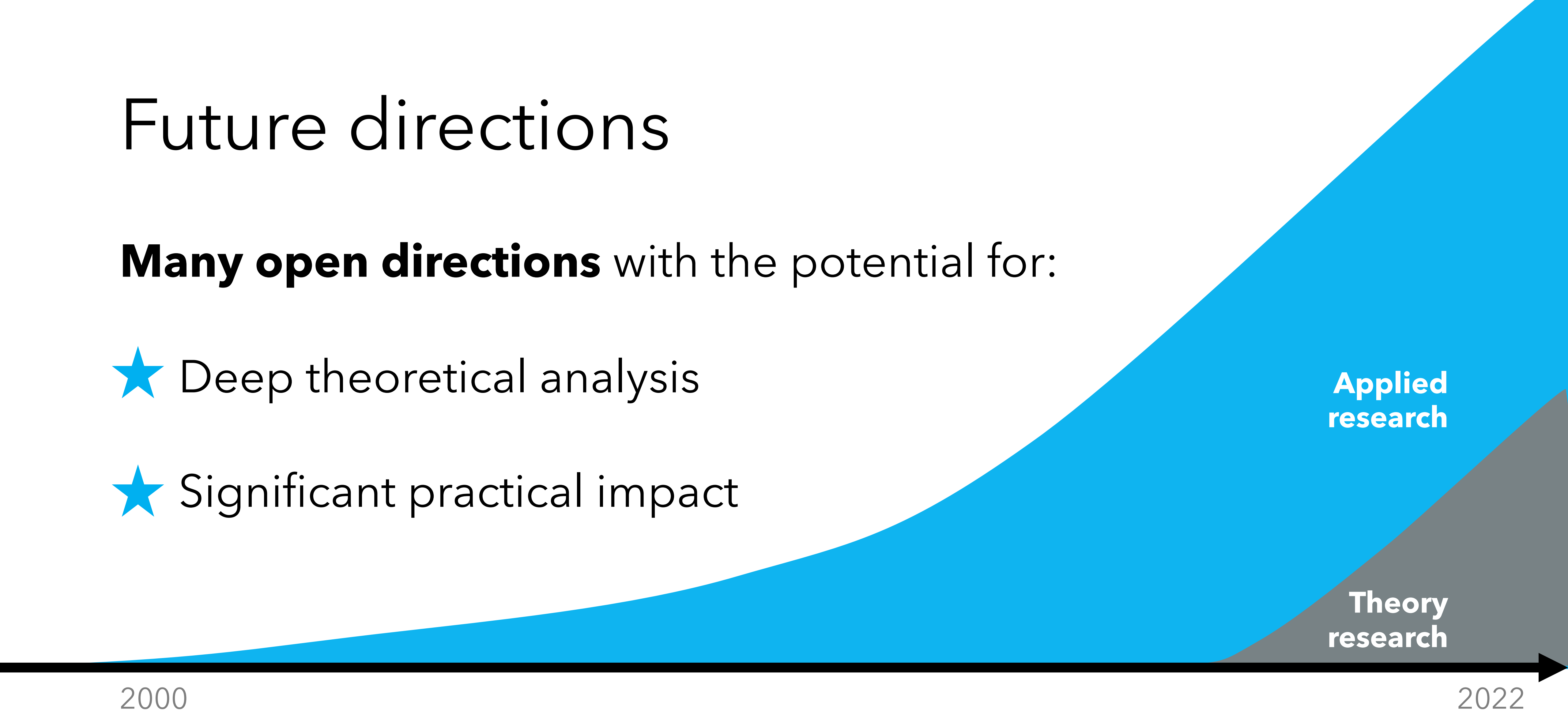
Learning to prune

Alabi, Kalai, Ligett, Musco, Tzamos, **V**, COLT'19

Future directions

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?

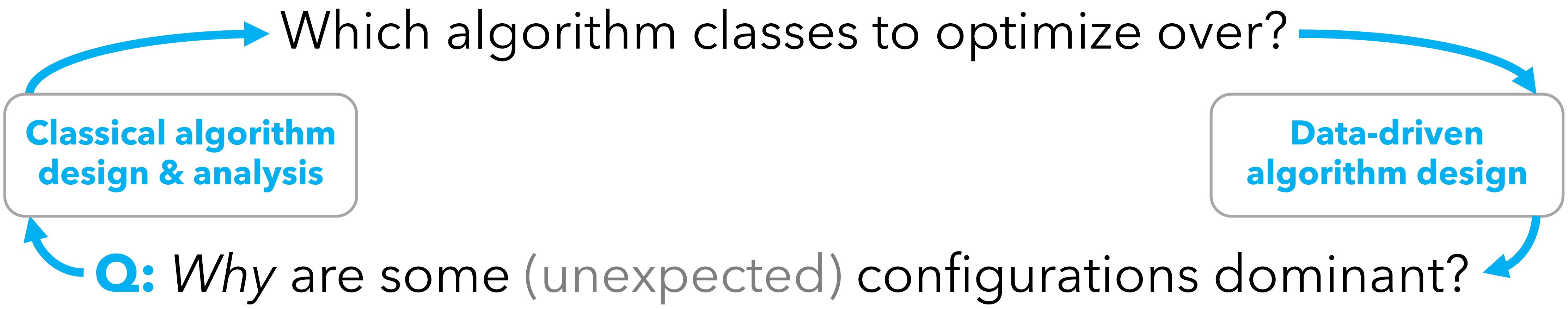
E.g., a shipping company starting out with just one routing IP

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

Data-driven algorithm design

Q: *Why* are some (unexpected) configurations 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.”

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.”

“extremely muscular teapot”

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

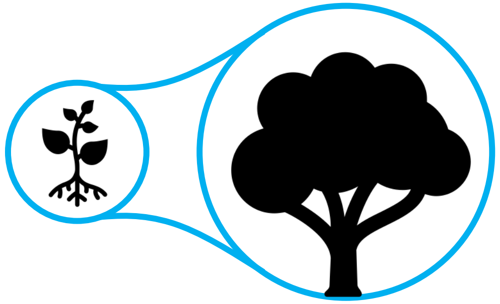
Future directions

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?



Machine learning for **algorithm design**

Ellen Vitercik

Stanford University