Online Algorithms: going beyond the worst-case

Anupam Gupta (New York University)
analysis of algorithms

reigning paradigm: worst-case analysis of algorithms

*how does the algorithm perform on its worst-possible input?*
analysis of algorithms

reigning paradigm: worst-case analysis of algorithms

+ robustness
+ wide applicability
+ many algorithms with good worst-case bounds
+ often less contentious

Naturally, there are shortcomings as well...

- pessimism, and insensitivity to data model/predictions
Ideally: want to get algorithms that are good for worst-case and "best-case" and .... all cases.

Worst-case: robustness when data is unpredictable

"Best-case": efficiency when data follows anticipated patterns

How to go beyond the worst case?

let’s see glimpse of ideas/techniques in context of online algos
Online Algorithms

Requests arrive over time, must be served immediately/irrevocably

Goal: (say) minimize cost of the decisions taken

**Competitive ratio** of algorithm $A$:

\[
\max \text{ instances } I \quad \frac{\text{cost of algorithm } A \text{ on instance } I}{\text{optimal cost to serve } I}
\]

Want to minimize the competitive ratio.
Online (Steiner) Tree

Metric space. n points arrive over time, maintain a connected tree.

Goal: minimize cost of tree

**Competitive ratio** of algorithm \( A \): 

\[
\text{max instances } I \quad \frac{\text{cost of algorithm } A \text{ on instance } I}{\text{optimal cost to serve } I}
\]

Want to minimize the competitive ratio.
Online Set Cover

Set system. n elements arrive over time, want to maintain a cover.

Goal: minimize cost of sets picked

**Competitive ratio** of algorithm $A$:

$$\max_{\text{instances } I} \frac{\text{cost of algorithm } A \text{ on instance } I}{\text{optimal cost to serve } I}$$

Want to minimize the competitive ratio.
max-K finding

$n$ people arrive over time, each has value $v_i$ -- can pick at most $K$

Goal: (say) **maximize** sum of values of picked people

**Competitive ratio** of algorithm $A$:

$$\frac{\text{max cost of algorithm $A$ on instance $I$}}{\text{optimal cost on instance $I$}}$$

Want to **minimize** the competitive ratio.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steiner tree</td>
<td>$\sim1.3$</td>
<td>$\Omega(\log n)$</td>
</tr>
<tr>
<td>Set Cover</td>
<td>$O(\log n)$</td>
<td>$\Omega(\log^2 n)$</td>
</tr>
<tr>
<td>Max-K-find</td>
<td>1</td>
<td>$O(K/n)$</td>
</tr>
</tbody>
</table>

can we do better in non-worst-case settings?
today’s menu

models to go beyond worst-case: max-find, spanning tree, set cover

but don’t overfit to these models: max-k-finding

and perhaps use predictions...: paging/caching
## price of uncertainty

<table>
<thead>
<tr>
<th></th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steiner tree</td>
<td>~1.3</td>
<td>(\Omega(\log n))</td>
</tr>
<tr>
<td>Set Cover</td>
<td>(O(\log n))</td>
<td>(\Omega(\log^2 n))</td>
</tr>
<tr>
<td>Max-K-find</td>
<td>1</td>
<td>(O(K/n))</td>
</tr>
</tbody>
</table>
max-1 finding

n people arrive over time, each has value $v_i$ -- pick at most one

Goal: maximize value of picked person

worst-case instance:

random guessing is the best option here
$\Rightarrow 1/n$ chance of success
going beyond the worst case

Ways to model non-worst-case instances?

1. values in bounded range

2. draws from some stochastic process? (Say $v_i \sim D_i$)

3. maybe arrival order is not worst-case?

4. train NN to find patterns, give predictions

5. ...
Prophet model

$n$ items arrive online, have value $R_i \sim D_i$ (indep.) Distributions $D_i$ known.

Pick one item. Maximize (expected) value.

**Algorithm:**

Take one sample $S_1, S_2, \ldots, S_n$ from each distribution.

Set threshold $T \leftarrow$ their maximum

Pick first $R_i$ above threshold $T$

**Thm:** $\mathbb{E}[Alg] \geq \frac{1}{4} \mathbb{E}[\max_i R_i]$
prophet model

Can get $\frac{1}{2}$ !!

**Thm:** $\mathbb{E}[Alg] \geq \frac{1}{4} \mathbb{E}[\max_i R_i]$

Samples $S_1, \ldots, S_n$  

Real values $R_1, \ldots, R_n$

sort

$W_1 > W_2 > \cdots > W_{2n}$

Pr[ $W_1$ is real ] = $\frac{1}{2}$  

Pr[ $W_2$ is sample | $W_1$ real ] $\geq \frac{1}{2}$

$\Rightarrow$ Pr[ $W_1 = R_{\max \ chosen}$ ] $\geq \frac{1}{4}$
secretary model

$n$ items have values chosen by adversary. But arrive online in random order.

Pick one item. Maximize (expected) value.

Algorithm:

Ignore first $\frac{1}{2}$ fraction of items.

Set threshold $T \leftarrow$ their maximum

Pick first item among remaining above threshold $T$

Can get $\frac{1}{e}$ !!
algos with predictions

Train a classifier to predict if current item is maximum among remaining

Model: like sec’y, but each prediction correct w.p. $p \geq \frac{1}{2}$ independently

Algo: ignore some fraction of elements
- then (for some fraction) pick any item that is best so far, and predictor = “Yes”
- then (for remaining fraction) pick any item that is best so far (ignore predictor)

Theorem: optimal performance for this model.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline</th>
<th>Online</th>
<th>BWC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steiner tree</td>
<td>~1.3</td>
<td>$\Omega(\log n)$</td>
<td></td>
</tr>
<tr>
<td>Set Cover</td>
<td>$O(\log n)$</td>
<td>$\Omega(\log^2 n)$</td>
<td></td>
</tr>
<tr>
<td>Max-K-find</td>
<td>1</td>
<td>$O(K/n)$</td>
<td>$\Omega(1)$</td>
</tr>
</tbody>
</table>

prophet, RO

BWC
Models to go beyond worst-case: spanning tree and set cover

But don’t overfit to these models: max-k-finding

And perhaps use predictions...: paging/caching
online (steiner) tree

Suppose $n$ requests

Connect each request on arrival

Worst-case comp. ratio: $\Theta(\log n)$

Goal: minimize total cost of edges
prophet (steiner) tree

Suppose n requests: vertex $R_i \sim \mathcal{D}_i$

Connect each request on arrival

**Algorithm:**

For all i, take one sample $S_i \sim \mathcal{D}_i$ each

Build MST on $S_1, \ldots, S_n$

When actual requests $R_i \sim \mathcal{D}_i$ arrive:
connect to closest previous point

**Goal:** minimize total cost of edges
Suppose n requests: vertex $R_i \sim \mathcal{D}_i$

Connect each request on arrival

**Algorithm:**

- For all i, take one sample $S_i \sim \mathcal{D}_i$ each
- Build MST on $S_1, \ldots, S_n$
- When actual requests $R_i \sim \mathcal{D}_i$ arrive: connect to closest previous point

**Theorem:** $\mathbb{E}[\text{Algo}] \leq 2 \mathbb{E}[\text{MST}(R_1, \ldots, R_n)]$

**Proof:**

$$\mathbb{E}[\text{MST}(S_1, \ldots, S_n)] = \mathbb{E}[\text{MST}(R_1, \ldots, R_n)]$$

$$\mathbb{E}[\text{cost}(R_i)] \leq \mathbb{E}[	ext{dist}(R_i, S)]$$

$$\leq \mathbb{E}[	ext{dist}(R_i, S_{-i})]$$

$$= \mathbb{E}[	ext{dist}(S_i, S_{-i})]$$

$$\Rightarrow \Sigma_i \mathbb{E}[\text{cost}(R_i)] \leq \Sigma_i \mathbb{E}[	ext{dist}(S_i, S_{-i})] \leq \mathbb{E}[\text{MST}(S)]$$

[Garg G. Leonardi Sankowski 08]
<table>
<thead>
<tr>
<th></th>
<th>Offline</th>
<th>Online</th>
<th>BWC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steiner tree</td>
<td>~1.3</td>
<td>Ω(\log n)</td>
<td>2</td>
</tr>
<tr>
<td>Set Cover</td>
<td>(O(\log n))</td>
<td>Ω(\log^2 n)</td>
<td>prophet</td>
</tr>
<tr>
<td>Max-K-find</td>
<td>1</td>
<td>(O(K/n))</td>
<td>Ω(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>prophet, RO</td>
</tr>
</tbody>
</table>
Online Set Cover

Goal: pick smallest # sets to cover all elements.

$|\mathcal{U}| = n = \# \text{ elements}$

$|\mathcal{S}| = m = \# \text{ sets}$

[Alon Awerbuch Azar Buchbinder Naor 03]
Online Set Cover

\[ \mathcal{F} \]

\( m \) sets

\[ \mathcal{U} \]

\( n \) elements

[Alon Awerbuch Azar Buchbinder Naor 03]
Online Set Cover

Algorithm:
$O(\log n \log m)$
competitive

CR: $\Omega(\log n \log m)$ for deterministic algos and for poly-time algos

Q: What happens beyond the worst case?
Random Order (RO)

\[ \mathcal{F} \]

$m$ sets

\[ \mathcal{U} \]

$n$ elements
LearnOrCover
(Unit cost, exp time)

Assume we know \( k = \text{OPT} \)

when random element \( \nu \) arrives
  if \( \nu \) not already covered, in parallel:
  1. select random remaining candidate
     pick random set from it
  2. remove candidates that don’t cover \( \nu \)
     pick any set covering \( \nu \)

Q: do \( \frac{1}{2} \) of remaining candidates cover \( \frac{1}{2} \) of uncovered elements?

Yes: random set covers many uncovered elements!
No: random element removes many candidates!!

Sol R:
Case 1: $\geq 1/2$ of $P \in \mathcal{P}$ cover $\geq 1/2$ of $\mathcal{U}$.

$R$ covers $\frac{|\mathcal{U}|}{4k}$ in expectation.

$\mathcal{U}$ shrinks by $\left(1 - \frac{1}{4k}\right)$ in expectation.

$|\mathcal{U}|$ initially $n$

$\Rightarrow O(k \log n)$ COVER steps suffice.

Case 2: $> 1/2$ of $P \in \mathcal{P}$ cover $< 1/2$ of $\mathcal{U}$.

$\geq 1/2$ of $P \in \mathcal{P}$ pruned w.p. $1/2$.

$\mathcal{P}$ shrinks by $3/4$ in expectation.

$|\mathcal{P}|$ initially $\binom{m}{k} \approx m^k$

$\Rightarrow O(k \log m)$ LEARN steps suffice.

$\Rightarrow O(k \log mn)$ steps suffice.
LearnOrCover

(Unit cost)

Init. $x \leftarrow 1/m$.

@ time $t$, element $v$ arrives:

If $v$ covered, do nothing
Else:

(I) Buy random $R \sim x$.
(II) $\forall S \ni v$, set $x_S \leftarrow e \cdot x_S$
Renormalize $x \leftarrow x / \|x\|_1$
Buy arbitrary set to cover $v$

If $E_v[x_v] > \frac{1}{4}$ ⇒ $E_R[k \Delta \log |U^t|]$ drops by $\Omega(1)$.
Else $E_v[k \Delta KL]$ drops by $\Omega(1)$.

[Recall $k = |OPT|$]

\[
\Phi(t) = c_1 KL(x^* || x^t) + c_2 \log |U^t|
\]

Idea: Measure convergence with potential function

$U^t :=$ uncovered elements @ time $t$
$x^* :=$ uniform distribution on OPT

Claim 1: $\Phi(0) = O(\log mn)$, and $\Phi(t) \geq 0$.

Claim 2: If $v$ uncovered, then $E[\Delta \Phi] \leq -\frac{1}{k}$.

[Gupta Kehne Levin FOCS 21]
LearnOrCover
(Some philosophy)

Perspective 1:

Perspective 2:

Define
\[ f(x) = \sum_v \max(0,1 - \sum_{S \ni v} x_S) \]
(Goal is to minimize \( f \) in smallest # of steps)

\[ \nabla f \big|_S(x) = \# \text{ uncovered elements in } S \]
\[ \propto E[\mathbf{1}\{v \in S \mid v \text{ uncovered}\}] \]

RO reveals stochastic gradient...
### price of uncertainty

<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline</th>
<th>Online</th>
<th>BWC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steiner tree</td>
<td>~1.3</td>
<td>Ω(log n)</td>
<td>0(1) prophet</td>
</tr>
<tr>
<td>Set Cover</td>
<td>O(log n)</td>
<td>Ω(log^2 n)</td>
<td>O(log n) RO</td>
</tr>
<tr>
<td>Max-K-find</td>
<td>1</td>
<td>O(K/n)</td>
<td>Ω(1) prophet, RO</td>
</tr>
</tbody>
</table>
today's menu

models to go beyond worst-case: max-finding, spanning tree, set cover

but don't overfit to these models...: max-k-finding

and perhaps use predictions...: paging/caching
robustness vs efficiency

define data model, then give algorithms for data from that model

danger: may overfit to the model

get best of both worlds?
semi-random models

Input first drawn from some (stochastic) data model

Then adversary corrupts in some (bounded) way

E.g., max-finding (secretary setting)

$G$ “green” items appear according to the model

but adversary can inject $R$ red items in worst-case ways

get (at least) pre-corruption value?

[Bradac G. Singla Zuzic 19]
[Garg Kale Rohwedder Svensson 19]
[Molinaro Kesselhiem 19]
byzantine max-K-finding

Adversary chooses values for $G$ green and $R$ red items

Adversary chooses times in $[0,1]$ for each red item

green items appear at random times in $[0,1]$

we don’t see colors, want value $\approx$ sum of top $K$ green items
Adversary chooses values for $G$ green and $R$ red items

Adversary chooses times in $[0,1]$ for each red item

Green items appear at random times in $[0,1]$.

We don’t see colors, want value $\approx$ sum of top $K$ green items.

Recall algorithm without corruptions?
byzantine max-K-finding

Adversary chooses values for \( G \) green and \( R \) red items

Adversary chooses times in \([0,1]\) for each red item

green items appear at random times in \([0,1]\)

we don’t see colors, want value \( \approx \) sum of top \( K \) green items

fails with corruptions!!
but we can still do something...

**Informal Robustness Theorem:**

If $K$ is at least $\approx \log n$ and we have estimate of $OPT$ to within $\text{poly}(n)$, then can achieve value $\Omega(OPT)$ even with corruptions.

Good news: extends to higher-dimensional allocation problems.
robust algorithmic thinking

1. Show “robust” single-parameter algorithm:
   if parameter chosen right ⇒ get good value even after corruption

2. Learn right parameter setting “robustly”

which single-parameter algorithm?
what threshold? idea #1

Suppose green values are $g_1 > g_2 > \ldots > g_n$

**Idea #1:** pick items at least threshold $T^* = g_k$

is this robust to injecting bad items??

Imagine $g_1 = \ldots = g_{k-1} = M$

$g_k = 1$

inject reds of value 1
Idea #2: a robust threshold

Suppose green values are \( g_1 > g_2 > \ldots > g_n \)

Idea #2: pick items at least threshold \( T^* = \frac{\text{OPT}}{2k} \)

Adding red items does not hurt...

\[ \exists \text{ good solution: } \frac{\text{OPT}}{2k} \cdot k \leq \frac{\text{OPT}}{2} \]

each picked red item also gives \( \frac{\text{OPT}}{2k} \)...
1. Show robust “single-parameter” algorithm: right threshold ⇒ get good value even after corruption

2. Learn this parameter robustly
step 2: learn threshold robustly

a. Estimate of OPT to within $\text{poly}(n)$
   $\Rightarrow 0(\log n)$ different guesses for OPT, need to choose right guess

b. Use online learning (“experts” algorithm) to do almost as well as best one

Break time $[0,1]$ into $T$ intervals

Use feedback from each interval to choose guess for next interval

Payoff $\geq \Omega(OPT) - \sqrt{T \times \log \#\text{experts} \times \left(\frac{OPT}{T}\right)}$

small if $T$ is large. But want measure concentration, so $T$ not too large!
our result...

Adversary chooses values for $G$ green and $R$ red items.

Green items appear at random times in $[0,1]$.

Informal Robustness Theorem:

If $K \geq \Theta(\log n \log \log n)$ and we have estimate of $OPT$ to within $\text{poly}(n)$

then we can achieve value $\Omega(OPT)$ even with corruptions.

Good news: extends to higher-dimensional allocation problems.
Online Allocation

Value
25 + 13 + 7

Budget
K

[Agrawal Wang Ye 09]
Online Allocation

\[
\begin{align*}
\max_{x \in \{0,1\}^n} & \quad v \cdot x \\
Ax & \leq K1
\end{align*}
\]

columns of A appear online

Assume: \( A \in [0,1]^{d \times n} \) and \( K \gg 1 \)

Want smallest \( K \) to get \((1 + \varepsilon)\)-apx for value

[Agrawal Wang Ye 09]
packing with corruptions

Random order

Sample

Estimate OPT

Maintain good dual prices via low-regret learning

Greedily assign primal

Byzantine

Low-regret learning algo

[Argue G. Singla Molinaro 22]
rest of today’s menu...

models to go beyond worst-case: max-finding, spanning tree, set cover

but don’t overfit to these models...: max-k-finding

and perhaps use predictions...: paging/caching
(ML-based) predictions...

Use predictions to get better algorithms?

E.g., for caching in memory systems, suppose predict furthest-in-future page

+ If predictions perfect, then get optimal \#page faults (a.k.a. Belady’s rule)

- what if predictions are correct only 10% of the time?
caching with predictions

Informal Theorem:

If predict furthest-in-future page with constant probability
(and no other page predicted too often)
then get constant-competitive paging.

Q: “right” prediction model? Sample complexity of learning?
today we saw...

models to go beyond worst-case: max-finding, spanning tree, set cover

but don't overfit to these models...: max-k-finding

and perhaps use predictions...: paging/caching
to summarize

the worst-case analysis of algorithms has served us well

but we should also look beyond these robust/pessimistic guarantees

+ when do our algorithms outperform these worst-case bounds?

+ what if the input is stochastic?

+ are we over-fitting to the stochastic model?

+ can we train some model and then use its predictions?

+ ...

Thanks!