Machine Learning for Optimization

Ben Moseley
Operations Research
Tepper School of Business, Carnegie Mellon University
Relational-AI
Collaborators

T. Lavastida

C. Xu

M. Dinitz S. Im S. Lattanzi R. Ravi S. Vassilvitskii

Online Scheduling via Learned Weights. SODA 2020.
Learnable and Instance-Robust Predictions for Matchings, Flows and Load Balancing. ESA 2021
Using Predicted Weights for Ad Delivery. ACDA 2021
Faster Matching via Learned Duals. NeurIPS 2021 (oral)
Machine Learning is Transforming Society

- Has not fundamentally changed combinatorial optimization
- However, could it?
Optimization Augmented with Machine Learning
Motivating Example
[Kraska et al. SIGMOD 2018]

- Array of $n$ integers $A$
- Over time queries arrive asking if $q$ is in $A$

![Diagram of a sorted array with a query element]

2 4 7 11 16 22 37 38 44 88 89 93 94 95 96 97 98
Motivating Example

• Array of $n$ integers $A$

• Over time queries arrive asking if $q$ is in $A$

O(\log n) lookup time
Motivating Example

- Train a predictor $h(q)$ to predict where $q$ is in the array
  - Estimates where the integer is based on prior queries
  - Could be wrong, but hopefully not too far off

- Use *doubling* binary search from prediction
Motivating Example

- Analysis
  - Let $\eta$ be the value of $|h(q) - OPT(q)|$, the error in the prediction
  - Run time is $O(\log \eta)$
  - Need to be careful about overhead of the prediction
  - Can make this work in practice
Learning Augmented Algorithm

• Run time binary search $O(\log n)$

• Run time prediction $O(\log \eta)$

• Perfect predictions give constant lookup

• Worst case same as the best classical algorithm
  • Gracefully degrades to the worst case
Learning Augmented Algorithms

• Punchline:
  • Machine learning can be combined with classical algorithms to obtain better results
  • Gives us new widely applicable models for beyond worst-case analysis
Learning Augmented Algorithms

Distribution over Typical Instances

Training Sample → Learning Algorithm → Prediction → Learning Augmented Algorithm → New Instance

Evaluate average performance!
Learning Augmented Algorithms

What parameter should be predicted?

Algorithmically how should we use the prediction?

Distribution over Typical Instances

Training Sample

Learning Algorithm

Prediction

New Instance

Learning Augmented Algorithm

Sample

What parameter should be predicted?

Algorithmically how should we use the prediction?
Learning Augmented Algorithms

Can the parameter be learned?

What parameter should be predicted?

Algorithmically how should we use the prediction?

Distribution over Typical Instances

Learning Algorithm

New Instance

Training Sample

Prediction

Learning Augmented Algorithm

Can the parameter be learned?

What parameter should be predicted?

Algorithmically how should we use the prediction?
Current Status
**ERL: Desirable Analysis Framework**

- **Existence:** Predictions should allow the algorithm to go beyond worst-case bounds
  - Good example: Location in the array
  - What to predict is often the main question

- **Robustness:** Algorithms are robust to minor changes in the problem input
  - Good example: Algorithm is robust to incorrect location in the array
  - Bad example: yes/no if an item is in the array

- **Learnability:** Predictions should be learnable if data is coming from a distribution
  - Example: PAC-Learning
Beyond Worst-Case Analysis Frameworks

• Online algorithm design
  • Competitive ratio parameterized by error in the predictions

• Running time
  • Worst case run time parameterized by error in the predictions
Online Restricted Assignment
Makespan Minimization

• Client Server Scheduling
  • Processed in \( m \) machines in the \textit{restricted assignment} setting (some results hold for \textit{unrelated} machines)
  • Jobs arrive over time in the \textit{online-list} model
    • All arrive at time 0
    • Jobs revealed one at a time
  • Assign jobs to the machines to minimize \textit{makespan}

\[
\begin{align*}
\text{Given:} & \\
& m \text{ machines, } n \text{ non-preemptive jobs with weights } w_j \text{ and machine-dependent processing times } p_{ij}:
\end{align*}
\]
\[
C_j := \text{completion time job } j \text{ in schedule; minimize } \sum w_j C_j
\]

\[\text{Known: problem is APX-hard [Hoogeveen et al., 2002]}\]
\[\text{Also: (3/2)-approximation [Bansal et al. 2016, Li, 2018]}\]
Restricted Assignment
Makespan Minimization

- $m$ machines
- $n$ jobs
  - Online list: a job must be immediately assigned before the next job arrives
  - $N(j)$: feasible machines for job $j$
  - $p(j)$: size of job $j$ (complexity essentially the same if *unit sized*)
- Minimize the maximum makespan
  - Optimal makespan is $T$
Online Competitive Analysis Model

• $c$-competitive
  \[ \frac{ALG(I)}{OPT(I)} \leq c \]

• Worst case relative performance on each input $I$

• Problem well understood:
  • A $\Omega(\log m)$ lower bound on any online algorithm
  • Greedy is a $O(\log m)$ competitive algorithm [Azar, Naor, and Rom 1995]
Beyond Worst Case via Predictions

• Reasonable assumption:
  • Access to last week’s job sequence

• What should be predicted?
• How can it be used?
Existence

• First show natural predictions that fail
• Next give a good parameter to predict
What (not) to Predict?

- **Number of jobs assigned to machines** in the optimal solution?
- Perhaps we can identify the contentious machines?

![Bar chart showing machine assignments.](chart.png)
What (not) to Predict?

• **Load** of the machines in the optimal solution?

• Perhaps we can identify the contentious machines? **No**

![Bar chart showing load distribution across machines]

- New instance padded with dummy jobs loads the **same**
What (not) to Predict?

- Predict dual variables

- Known to be useful for matching in the random order model [Devanur and Hayes, Vee et al.]
  
  - Read a portion of the input
  
  - Compute the duals
  
  - Prove a primal assignment can be (approximately) constructed from the duals online
  
  - Use duals to make assignments on remaining input
What (not) to Predict?

- Predict **dual variables** for makespan scheduling
  - Can derive primal based on dual
  - Sensitive to small error (e.g. changing a variable by a factor of \(1 + 1/poly(n)\) has the potential to drastically change the schedule)
What to Predict?

• Idea: capture **contentiousness** of a machine

  • Seems like the most important quantity besides types of jobs
Prediction: Machine Weights

- Predict a **weight** for each machine
  - **Single** number (compact)
  - Lower weight means more restrictive machine
  - Higher weight less restrictive

- Framework:
  - Predict machine **weights**
  - Using to construct **fractional** assignments online
  - **Round** to an **integral** solution online
Fractional Assignments via Weights

- Each machine $i$ has a weight $w_i$

- Job $j$ is assigned to machine $i$ fractionally as follows:

$$x_{i,j} = \frac{w_i}{\sum_{i' \in N(j)} w_{i'}}$$
Existence

- **Theorem (existence of weights):** Let $T$ be optimal max load. For any $\epsilon > 0$, there exists machine weights such that the resulting fractional max load is at most $(1+\epsilon)T$.

- **Theorem (rounding assignments) [Li, Xian ICML 2021]:** There exists an online algorithm that takes as input fractional assignments and outputs integer assignments for which the maximum load is bounded by $O((\log\log(m))T')$, where $T'$ is maximum fractional load of the input. The algorithm is randomized and succeeds with probability at least $1 - 1/m$.

- **Theorem (tightness of rounding):** Any randomized online rounding algorithm has worst case load at least $\Omega(T' \log \log m)$.

- **Large makespan case:** [fractional makespan larger than $\log(m)$]
  
  - Randomized rounding gives gives a $(1+\epsilon)T'$ where $T'$ is maximum fractional load of the input with probability at least $1 - 1/m$. 

Results on Robustness

• **Theorem:** Given predictions of the machine weights with maximum relative error $\eta > 1$, there exists an online algorithm yielding fractional assignments for which the fractional max load is bounded by $O(T \min\{\log(\eta), \log(m)\})$.

• **Corollary:** There exists an $O(\min\{\log\log(m)\log(\eta), \log m\})$ competitive algorithm for restricted assignment in the online algorithms with learning setting
Learnability

- Unknown distribution model $\mathcal{D}$
  - Instance drawn from unknown distribution
  - Best prediction $y^* := \arg\max_y \mathbb{E}_{I \sim \mathcal{D}}[ALG(I, y)]$
  - How many samples $s$ to compute $\hat{y}$ giving the following performance with high probability

\[
\mathbb{E}_{I \sim \mathcal{D}}[ALG(I, \hat{y})] \geq (1 - \epsilon) \mathbb{E}_{I \sim \mathcal{D}}[ALG(I, y^*)]
\]
Learnability

• Similar to
  • PAC learning
  • Data-driven algorithm design
• Alternative: competitive analysis
  • Show a small number of samples needed for the following performance with good probability
    \[
    \mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[ALG(\mathcal{I}, \hat{y})] \geq (1 - \epsilon)\mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[OPT(\mathcal{I})]
    \]
Learnability

- **Theorem:** Let $\mathcal{D}$ be a product distribution such that $\mathbb{E}_{S \sim \mathcal{D}}[OPT(S)] \geq \Omega(\log m)$. There exists an algorithm that constructs **nearly optimal** weights using a polynomial number of samples in $m$. 
Summary for Restricted Assignment

• Existence
  • Weights

• Robustness
  • Near optimal with perfect predictions
  • Bounded by the best worst case performance

• Learnability
  • Low sample complexity
Predictions for Online Algorithms

- Lots of success for online algorithm design
  - Matching
  - Caching
  - Ski-rental
  - Scheduling
  - Online learning
  - Heavy hitters

- What about the original question of speeding up algorithms offline?
Warm-Start

• Many problems are solved repeatedly on ‘similar’ instances
  • e.g. scheduling yesterday versus today

• We solve from scratch
Pitfalls

• Robustness

  • Feasibility: The warm start may not be feasible

  • Optimization: The warm start may not be useful

• Learnability: The starting solution may not be learnable
Weighted Bipartite Matching

- Input a bipartite graph $G = (L \cup R, E)$ with edge costs $c_{i,j}$

- Output the minimum cost perfect matching
Existence

What to Predict?

• Idea 1: Edges in optimal solution
  • Brittle

• Idea 2: LP duality
Existence

Primal

\[
\min \sum_{e \in E} c_e x_e \\
\text{subject to: } \sum_{e \in N(i)} x_e = 1 \quad \forall i \in V \\
x_e \geq 0 \quad \forall e \in E
\]

Dual

\[
\max \sum_{i \in V} y_i \\
\text{subject to: } y_i + y_j \leq c_{ij} \quad \forall (i, j) \in E
\]

• Dual:
  • Assigns prices to vertices
  • Complementary slackness
  • Edges in the matching have tight dual constraints
Existence

Primal

\[
\begin{align*}
\min \sum_{e \in E} c_e x_E \\
\text{subject to:} \quad & \sum_{e \in N(i)} x_e = 1 \quad \forall i \in V \\
& x_e \geq 0 \quad \forall e \in E
\end{align*}
\]

Dual

\[
\begin{align*}
\max \sum_{i \in V} y_i \\
\text{subject to:} \quad & y_i + y_j \leq c_{ij} \quad \forall (i, j) \in E
\end{align*}
\]

- Hungarian algorithm (popular in practice)
  - Start with dual values at 0
  - Compute max cardinality matching on tight edges
  - If not done, find a set violating Hall’s theorem. Update duals
Existence

Primal

\[
\begin{align*}
\min & \sum_{e \in E} c_e x_e \\
\text{subject to:} & \sum_{e \in N(i)} x_e = 1 \quad \forall i \in V \\
& x_e \geq 0 \quad \forall e \in E
\end{align*}
\]

Dual

\[
\begin{align*}
\max & \sum_{i \in V} y_i \\
\text{subject to:} & y_i + y_j \leq c_{ij} \quad \forall (i, j) \in E
\end{align*}
\]

• Hungarian algorithm (popular in practice)
  • Predict dual values
  • Compute max cardinality matching on tight edges
  • If not done, find a set violating Hall’s theorem. Update duals
Robustness
Main Idea

Idea:
- Predict the dual values, i.e. predict $\hat{y}_i$
- “Warm start” Hungarian algorithm from predicted duals.

Feasibility issue:
- Hungarian algorithm slowly increases duals. Always has a feasible solution
- But, predicted dual may be infeasible
- Have an edge s.t.: $\hat{y}_i + \hat{y}_j > c_{ij}$

Approach:
- Minimally reduce predicted duals to attain feasibility
- Must do it quickly (since speed is of the essence)
Robustness
Making Duals Feasible

• Write LP for the feasibility problem:

\[
\begin{align*}
\min & \quad \sum_{i \in V} \delta_i \\
\text{subject to:} & \quad \delta_i + \delta_j \geq (\hat{y}_i + \hat{y}_j - c_{ij})^+ \quad \forall (i, j) \in E \\
& \quad \delta_i \geq 0 \quad \forall i \in V
\end{align*}
\]

Algorithm (greedy):
– Pick any vertex \( i \). Set its \( \delta_i \) value to the minimum that satisfies all of the constraints
– Remove \( i \) from the graph and repeat.
– **Theorem**: Resulting solution is a 2-approximation for the LP, runs in linear time!
Robustness

Overall approach:

- Obtain (learn) duals: \( \hat{y}_1, \ldots, \hat{y}_n \)
- Given a new matching instance, \( G = (V, E) \) find feasible duals \( y'_1, \ldots, y'_n \)
- Run Hungarian method starting with \( y'_1, \ldots, y'_n \)

**Theorem:** The overall running time is: \( O(\|\hat{y} - y^*\|_1) \cdot m\sqrt{n} \)

- Strictly better when the error is small
- Can prove that it's no worse than vanilla Hungarian algorithm
Does it Work?

Experiment 1(a):

- Start with a bipartite graph with a planted min cost perfect matching
- Generate new instances by adding random noise of increasing magnitude to the edge weights

- When noise is low, learning approach dominates.
Does it Work?

Experiment 1(b):
- Start with a bipartite graph with a planted min cost perfect matching
- Generate new instances by adding random noise of increasing magnitude to the edge weights
- When noise gets high, nothing to be learned, so converge to Hungarian method.
Does it Work?

Experiment 2:

- Perfect matching problems derived from geometric datasets

- Learned gains can be substantial (10x in some cases)
Does it Work?

Experiment 3:

– How many samples do you need to learn?

– Many fewer than the theory predicts
Future Work

• Learnability
  • Are unlearnable predictions more useful than learnable predictions

• How useful is this new paradigm empirically and theoretically
  • Rich area: Online algorithms to cope with uncertainty, running time off-line, other applications?

https://algorithms-with-predictions.github.io/
Thank you!

Questions?