ML4CO

Methodology of Machine Learning for Combinatorial Optimization

Antoine Prouvost, Andrea Lodi

antoine.prouvost@polymtl.ca

Polytechnique Montréal

28th of June 2018

Motivation

IBM Deep Blue

V.S.

Deep Mind AlphaGo

Motivation

Traditional OR algorithms

- 👍 Hard rules & constraints
- 👎 Can have big complexity

Focuses on all possible cases

Deep Learning

- 👍 Perception: fast, intuitive approximate
- 👎 Integers & rules

Focuses on the cases shown

Objectives

🤷‍♀️ Sometimes in OR, we just don't know.

$\longrightarrow$ Build mixed algorithm where **decisions** are ~~arbitrary~~ learned.

😴 Sometimes in OR, the complexity is just too big.

$\longrightarrow$ Build a fast **approximator** using deep learning.

In this talk, we refer to both cases as **estimation**.

A 3-class taxonomy

End-to-End learning 🧠

Build a machine learning model to directly output solutions.

Offline estimation 🧠➡️⚙️

Build a machine learning model to parametrize your OR algorithm.

Online estimation 🧠🔄⚙️

The machine learning model is used repeatedly by the OR algorithm to assist it in its solving process.

End-to-end learned algorithms 🧠

- Given a problem, learn the **solution**.
  E.g. Given a weighted graph, learn a solution to the TSP.[1]
- Solutions may not be fully **detailed**.
  E.g. Predict only aggregation of MILP solution[2]*.*
- `Runtime`: input the problem to the ML model, get the solution.

[1] Bello et al. 2016; Dai et al. 2017; Emami and Ranka 2018; Kool and Welling 2018; Nowak et al. 2017; Vinyals, Fortunato, and Jaitly 2015.
[2] Larsen et al. n.d.

End-to-end learned algorithms 🧠

Advantages 🙂

- 👍 Fast inference (Polynomial complexity)
- 👍 Based on data appropriate to the task
- 👍 Repurposed by retraining
- 👍 Good default solution when nothing else is available

Disadvantages 🙁

- 👎 Needs data (eventually labels)
- 👎 May need (very) expensive training
- 👎 **Very** heuristic:
  - No optimality guarantee
  - Little feasibility guarantee
    $\rightarrow$ Dedicated architectures
    $\rightarrow$ Must not break differentiability
- 👎 Generalization with size?

Offline estimation 🧠➡️⚙️

- Predict a **property** of a problem instance that can get **exploited** by an OR algorithm.
  E.g. Learn when to linearize MIQP[3]*, when to apply DW decomposition*[4]*.*
- May be seen as a more general case of the previous.
- `Runtime`: Run the prediction, then run the OR algorithm accordingly.

[3] Bonami, Lodi, and Zarpellon 2018.
[4] Kruber, Lübbecke, and Parmentier 2017.

Online estimation 🧠🔄⚙️

- Throughout it **progress**, the OR algorithm **repeatedly** calls the ML model.
  E.g. Learning to branch[5]*, where to run heuristics*[6] *in MILP.*
  Learning to apply SGD updates[7]*.*
  Learning to select cutting plane in convex SDP relaxation of QP[8]*.*
- Can also build heuristics[9].
- `Runtime`: The OR & the ML algorithms need to be deployed and run together → more specific code.

[5] Lodi and Zarpellon 2017.
[6] Shao et al. 2017.
[7] Andrychowicz et al. 2016; Li and Malik 2016, 2017; Wichrowska et al. 2017.
[8] Baltean-Lugojan and Misener n.d.
[9] Dai et al. 2017.

Representation challenge

Deep learning works well for natural signals (images, speech, etc).
Will it work well for OR problems?

- Are common architectures the best prior?
- Additional constraints: large sparse structured inputs/ outputs of variable sizes.

- Partial solutions: graph NN, attention, CNN, RNN and other parameter reuse.

Supervised vs reinforcement

Supervised

- 👍 Straightforward
- 👎 Expensive targets
- 👎 Not well suited when multiples targets are corrects (E.g. multiple solutions)

Reinforcement

- 👍 Natural formulation to the *online* setting
- 👍 Naturally **maximizes** sum of expected **future rewards**
- 👎 More complex to define and train

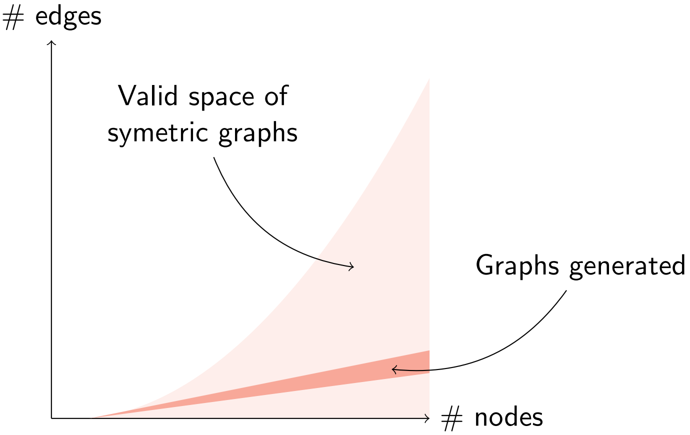
Supervised in the online setting?

- No apparent link between performance of ML and performance of overall algorithm.
- When estimating a *known* quantity, it's possible to link the performance fo ML to the overall performance[10]

[10] Baltean-Lugojan and Misener n.d.; Shao et al. 2017.

Data distributions

Easily fooled 🤡



Especially if the test set is generated from the same distribution as the training set 🤦🚑

Data distributions

Historical data?

"[S]ampling from historical data is appropriate when attempting to mimic a behavior reflected in such data."[11]

[11] Larsen et al. n.d.

Other motivations

- ML to model uncertainty[12].
- Extract knowledge out of learned models[13]

[12] Larsen et al. n.d.

[13] Bonami, Lodi, and Zarpellon 2018; Dai et al. 2017.

Questions

🙋‍♀️ 🔍 🙈

References I

- Andrychowicz, Marcin et al. (2016). "Learning to Learn by Gradient Descent by Gradient Descent". In: arXiv: 1606.04474 [cs].
- Baltean-Lugojan, Radu and Ruth Misener (n.d.). "Learning-Based Cutting Plane Approximation of QP Convex (SDP) Relaxations". In: p. 12.
- Bello, Irwan et al. (2016). "Neural Combinatorial Optimization with Reinforcement Learning". In: arXiv: 1611.09940 [cs, stat].
- Bonami, Pierre, Andrea Lodi, and Giulia Zarpellon (2018). "Learning a Classification of Mixed-Integer Quadratic Programming Problems". In: Integration of Constraint Programming, Artificial Intelligence, and Operations Resea International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Lecture Notes in Computer Science. Springer, Cham, pp. 595–604.
- Dai, Hanjun et al. (2017). "Learning Combinatorial Optimization Algorithms over Graphs". In: arXiv: 1704.01665 [cs, stat].
- Emami, Patrick and Sanjay Ranka (2018). "Learning Permutations with Sinkhorn Policy Gradient". In: arXiv: 1805.07010 [cs, stat].
- Kool, W. W. M. and M. Welling (2018). "Attention Solves Your TSP". In: arXiv: 1803.08475 [cs, stat].

References II

Kruber, Markus, Marco E. Lübbecke, and Axel Parmentier (2017). "Learning When to Use a Decomposition". In: Integration of AI and OR Techniques in Constraint Programming. International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science. Springer, Cham, pp. 202–210.

Larsen, Eric et al. (n.d.). "A Machine Learning Approximation Algorithm for Fast Prediction of Solutions to Discrete Optimization Problems". In: p. 22.

Li, Ke and Jitendra Malik (2016). "Learning to Optimize". In: arXiv: `1606.01885 [cs, math, stat]`.

– (2017). "Learning to Optimize Neural Nets". In: arXiv: `1703.00441 [cs, math, stat]`.

Lodi, Andrea and Giulia Zarpellon (2017). "On Learning and Branching: A Survey". In: TOP 25.2, pp. 207–236.

Nowak, Alex et al. (2017). "A Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks". In: arXiv: `1706.07450 [cs, stat]`.

Shao, Yufen et al. (2017). "Learning to Run Heuristics in Tree Search". In: pp. 659–666.

References III

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly (2015). "Pointer Networks". In: Advances in Neural Information Processing Systems 28. Ed. by C. Cortes et al. Curran Associates, Inc., pp. 2692–2700.

Wichrowska, Olga et al. (2017). "Learned Optimizers That Scale and Generalize". In: