CO+ML
Bengio Survey 2018

Demonstration

what is it: supervised learning, using expert knowledge

- **Baltean-Lugojan et al. (2018),** the authors use a neural network to approximate the lower bound improvement generated by tightening the current relaxation via cutting planes (cuts, for short)
- **Branching Policies in B&B Trees of MILPs**
  - **Strong branching (Applegate et al., 2007):** Namely, for every branching decision to be made, strong branching performs a one step look-ahead by tentatively branching on many candidate variables, computes the LP relaxations to get the potential lower bound improvements, and eventually branches on the variable providing the best improvement
  - **Marcos Alvarez et al. (2014, 2017)** use a special type of decision tree (a classical model in supervised learning) to approximate strong branching decisions using supervised learning.
  - **Khalil et al. (2016)** propose a similar approach, where a linear model is learned on the fly for every instance by using strong branching at the top of the tree, and eventually replacing it by its ML approximation.
  - **Marcos Alvarez et al. (2016)** is learned in an active fashion: when the estimator is deemed unreliable, the algorithm falls back to true strong branching and the results are then used for both branching and learning
  - **Interaction between ML and CO**: in the case of MILPs, this includes branching among fractional variables of the LP relaxation, selecting the node to explore among open branching nodes (He et al., 2014), deciding on the frequency to run heuristics on the B&B nodes (Khalil et al., 2017b), selecting cutting planes among valid inequalities (Baltean-Lugojan et al., 2018), removing previous cutting planes if they are not original constraints or branching decision, etc.
  - **He et al. (2014)** learn a policy to select among the open branching nodes the one that contains the optimal solution in its sub-tree. The training algorithm is an online learning method collecting expert behaviors throughout the entire learning phase.
  - **Lodi and Zarpellon (2017)** for an extended survey on learning and branching in MILPs.

Drawback:
1. In the demonstration setting, the performance of the learned policy is bounded by the expert, which is a limitation when the expert is not optimal. More precisely, without a reward signal, the imitation policy can only hope to marginally outperform the expert (for example because the learner can reduce the variance of the answers of the expert).
2. Furthermore, the performance of the learned policy may not generalize well to unseen examples and small variations of the task.

Experience

what is it: learning a policy by experience is the framework of reinforcement learning, where an agent maximizes the return (defined in Section 2.2). By matching the reward signal with the optimization objective, the goal of the learning agent becomes to solve the problem, without assuming any expert knowledge.

- **Khalil et al. (2017a)** suggest learning the criterion for this selection. They build a greedy heuristic framework, where the node selection policy is learned using a GNN (Dai et al., 2016), a type of neural network able to process input graphs of any size by a mechanism of message passing (Gilmer et al., 2017).

drawback:
1. with a reward, the algorithm learns to optimize for that signal and can potentially outperform any expert, at the cost of a much longer training time.
2. In the case where policies are approximated (e.g. with a neural network), the learning process may get stuck around poor solutions if exploration is not sufficient or solutions which do not generalize well are found.
3. Furthermore, it may not always be straightforward to define a reward signal.

Challenges

- **Feasibility:** Rather than learning the solution, it would be more precise to say that the algorithm is learning a heuristic. As already repeatedly noted, the learned algorithm does not give any guarantee in terms of optimality, but it is even more critical that feasibility is not guaranteed either. Finding feasible solutions is not an easy problem (theoretically NP-hard for MILPs), but it is even more challenging in ML, especially by using neural networks. Indeed, trained with gradient descent, neural architectures must be designed carefully in order not to break differentiability. For instance, both **pointer networks (Vinyals et al., 2015) and the Sinkhorn layer (Emami and Ranka, 2018)** are complex architectures used to make a network output a permutation, a constraint easy to satisfy when writing a classical CO heuristic.
- **Modeling:** In ML, in general, and in deep learning, in particular, we know some good prior for some given problems. The architectures used to learn good policies in combinatorial optimization might be very different from what is currently used with deep learning. Current deep learning already provides many techniques and architectures
  for tackling problems of interest in CO. As pointed out in section 2.2, techniques such as parameter sharing made it possible for neural networks to process sequences of variable size with RNNs or, more recently, to process graph structured data through GNNs. Processing graph data is of uttermost importance in CO because many problems are formulated (represented) on graphs. For a very general example, Selsam et al. (2018) represent a satisfiability problem using a bipartite graph on variables and clauses. This can generalize to MILPs, where the constraint matrix can be represented as the adjacency matrix of a bipartite graph on variables and constraints.
- **Scaling:** Indeed, all of the papers tackling TSP through ML and attempting to solve larger instances see degrading performance as size increases (Vinyals et al., 2015; Bello et al., 2017; Khalil et al., 2017a; Kool and Welling, 2018). To tackle this issue, one may try to learn on larger instances, but this may turn out to be a computational and generalization issue.
- **Data generation:** Collecting data (for example instances of optimization problems) is a subtle task. Larsen et al. (2018) claim that "sampling from historical data is appropriate when attempting to mimic a behavior reflected in such data". In other words, given an external process on which we observe instances of an optimization problem, we can collect data to train some policy needed for optimization, and expect the policy to generalize on future instances of this application.
  Deciding how to represent the data is also not an easy task, but can have a dramatic impact on learning. For instance, how does one properly represent a B&B node, or even the whole B&B tree? These representations need to be expressive enough for learning, but at the same time, concise enough to be used frequently without excessive computations.

Mahmood et al. (2018) use ML to produce candidate therapies that are afterward refined by a CO algorithm into a deliverable plan.

Generality: A compromise between instance-specific learning and learning a generic policy is what we typically have in multi-task learning: some parameters are shared across tasks and some are specific to each task. A common way to do that (in the transfer learning scenario) is to start from a generic policy and then adapt it to the particular instance by a form of fine-tuning procedure: training proceeds in two stages, first training the generic policy across many instances from the same distribution, and then continuing training on the examples associated with a given instance on which we are hoping to get more specialized and accurate predictions.

Meta learning and transfer learning: Machine learning advances in the areas of meta-learning and transfer learning are particularly interesting. This can be a successful strategy for generalizing from very few examples if we have access to many such training tasks. It is related to transfer learning, where we want that what has been learned in one or many tasks helps improve generalization on another. These approaches can help rapidly adapt to a new problem, which would be useful in the context of solving many MILPs instances, seen as many related tasks.

To stay with the branching example on MILPs, one may not want the policy to perform well out of the box on new instances (from the given distribution). Instead, one may want to learn a policy offline that can be adapted to a new instance in a few training steps, every time it is given one.

Larsen et al. (2018) train a neural network to predict the solution of a stochastic load planning problem for which a deterministic MILP formulation exists.

Kruber et al. (2017) use machine learning on MILP instances to estima
e beforehand whether or not applying a Dantzig-Wolf decompositio
will be effective, i.e. will make the solving time faster.