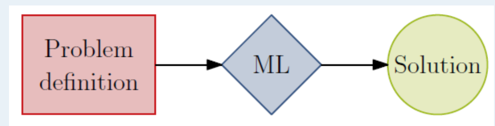
Optimization + Machine Learning

End to end learning

What is it: A first idea to leverage machine learning to solve discrete optimization problems is to train the ML model to output solutions directly from the input instance.



- **Pointer Network:** This approach has been explored recently, especially on Euclidean TSPs. To tackle the problem with deep learning, Vinyals et al. (2015) introduce the pointer network. The authors train the model through supervised learning with precomputed TSP solutions as targets.
  - They address some limitations of supervised (imitation) learning, such as the need to compute optimal (or at least high quality) TSP solutions (the targets), that in turn, may be ill-defined when those solutions are not computed exactly, or when multiple solutions exist.
- **Bello et al. (2017)** use a similar model and train it with reinforcement learning using the tour length as a reward signal.
- **Kool and Welling (2018)** introduce more prior in the model using a GNN instead of an RNN to process the input.
- **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.
  - Their main motivation is that the application needs to make decisions at a tactical level, i.e. under incomplete information, and machine learning is used to address the stochasticity of the problem. The authors use operational solutions, i.e. solutions to the deterministic version of the problem, and aggregate them to provide (tactical) solution targets to the ML model.
    Note that in Larsen et al. (2018) an MLP, i.e. a simple feedforward neural network, is used to process the input instance as a vector, hence integrating little prior knowledge about the problem structure.

Learning meaningful properties of optimization problems

What is it: In many cases, using only machine learning to tackle the problem may not be the most suitable approach. Instead, ML can be applied to provide additional pieces of information to a CO algorithm as illustrated in Figure 8. For example, ML can provide a parametrization of the algorithm (in a very broad sense).

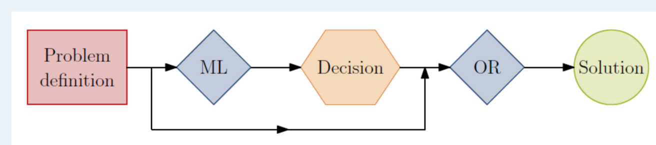


Figure 8: The machine learning model is used to augment an operation research algorithm with valuable pieces of information.

- **Kruber et al. (2017)** use machine learning on MILP instances to estimate beforehand whether or not applying a Dantzig-Wolf decomposition will be effecti e, i.e. will make the solving time faster
- **Bonami et al. (2018)** use machine learning to decide if linearizing the problem will solve faster.
- **Mahmood et al. (2018)** use ML to produce candidate therapies that are afterward refined by a CO algorithm into a deliverable plan.
- a ML model used for learning some representation may in turn use as features pieces of information given by another CO algorithm, such as the decomposition statistics used in **Kruber et al. (2017)**, or the LP information in **Bonami et al. (2018).**

Machine learning alongside optimization algorithms

- **(Lodi and Zarpellon, 2017)**. In this case, the general algorithm remains a branch-and-bound framework, with the same software architecture and the same guarantees on lower and upper bounds, but the branching decisions made at every node are left to be learned.
- **Khalil et al. (2017b)** build a ML model to predict whether or not running a given heuristic will yield a better solution than the best one found so far and then greedily run that heuristic whenever the outcome of the model is positive.
- **Baltean-Lugojan et al. (2018),** the learned model is queried repeatedly to select promising cutting planes. The ML model is used only to select promising cuts, but once selected, cuts are added to the LP relaxation, thus embedding the ML outcome into an exact algorithm. This approach highlights promising directions for this type of algorithms.
- **Learning recurrent algorithmic decisions** is also used in the deep learning community, for instance in the field of meta-learning to decide how to apply gradient updates in stochastic gradient descent (Andrychowicz et al., 2016; Li and Malik, 2017; Wichrowska et al., 2017).
- **Surrogate model: Xijun et. al. 2018** use learning model (offline trained with lots of historical data) in their solution evaluation phase, accelerating process of search.

To generalize the context of the previous section to its full potential, one can build CO algorithms that repeatedly call a ML model throughout their execution, as illustrated in Figure 9. A master algorithm controls the highlevel structur while frequently calling a ML model to assist in lower level decisions.

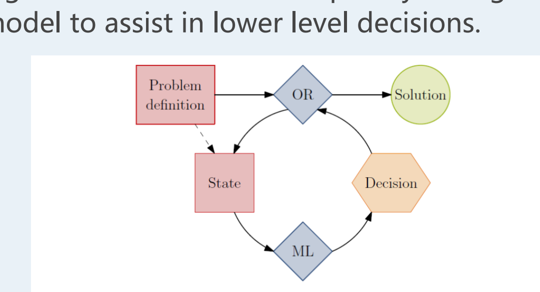


Figure 9: The combinatorial optimization algorithm repeatedly queries the same ML model to make decisions. The ML model takes as input the current state of the algorithm, which may include the problem definition.

Challenges

Feasibility

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. Indeed, we do not know how far the output of the heuristic is from the optimal solution, or if it even respects the constraints of the problem.

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.

Modelling

The problems studied in CO are different from the ones currently being addressed in ML, where most successful applications target natural signals. The architectures used to learn good policies in combinatorial optimization might be very different from what is currently used with deep learning.

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.**

Scaling

Scaling to larger problems can be a challenge. 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".
- Even so, it remains a complex effort to generate problems that capture the essence of real applications. Moreover, CO problems are high dimensional, highly structured, and troublesome to visualize. The sole exercise of generating graphs is already a complicated one.
- 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?