SeaPearl: A Constraint Programming Solver guided by Reinforcement Learning

Authors: Félix Chalumeau, Ilan Coulon, Quentin Cappart and Louis-Martin Rousseau

Article presentation
Experimental Analysis of Algorithms - UAIC

Tirpescu Cosmin Nicu

https://arxiv.org/pdf/2102.09193.pdf

Motivation

- Authors of the article propose a framework SeaPearl that combines CP solver (Flux) with reinforcement learning in order to search the optimal solution in each search node of the CP solver
- In the last years, more and more interests is focus on building heuristics using machine learning algorithms
- This is because the goal is to use the knowledge of the past historical data to approximate solutions to similar instances
- Since CP solvers do not include such machine learning techniques, the authors build this framework on top of Flux (Julia) in order to avoid the overhead of Python calls on C++

Technical background - Reinforcement learning

- Reinforcement learning is a sub-field of machine learning dedicated to train agents to take actions in an environment in order to maximize accumulated reward
  - S represents the set of states that can be encountered in the environment (a representation of a chess board in any point of the game, tic-tac toe, the nodes of a partial tour of TSP, etc.)
  - A represents the set of actions that can be taken by the agent (move a piece, put an X/O in a specific position, visit a node, etc.)
  - T is the transition function leading the agent from a state to another given the action that was previously taken
  - R is the reward function associated to a particular transition
- The behaviour of the agent is driven by a policy denoted as **π**: S → A
- The objective is to maximize the accumulated reward in each episode (sequence of states and actions) using a discount factor (how much of the recent states/action should weight than a long time distanced states/action)

$$G_t = \Sigma_{k=1}^{T} \gamma^{k-t} R(s_k, a_k)$$

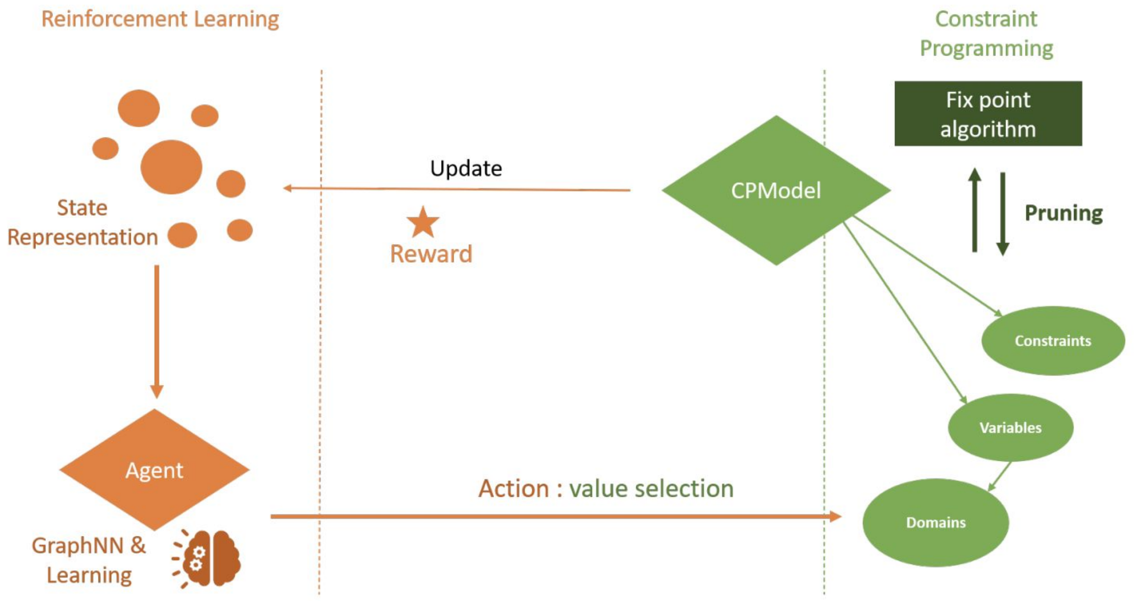
Technical background - Graph Neural Network

- Learning on graph structures is an active field of research due to its many applications such as molecular biology, physics, social sciences, etc. and recently it was considered also in solving combinatorial optimization problems
- Formally:
  - V is the set of vertices
  - E is the set of edges
  - $f_v \in \mathfrak{R}^k$ Is a vector of k features attached to a vertex v from V
  - Similarly, $h_{v,u} \in \mathfrak{R}^q$ is a vector of q features attached to an edge (v, u) from E
- The goal is to learn a p-dimensional representation $\mu_v \in \mathfrak{R}^p$ for each node of G
- The recursive function of this representation is

$$\mu_v^{t+1} = \sigma\left(\theta_1 \mathbf{f}_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^t + \theta_3 \sum_{u \in \mathcal{N}(v)} \sigma\left(\theta_4 \mathbf{h}_{v,u}\right)\right) \quad \forall t \in \{1, \ldots, T\}$$

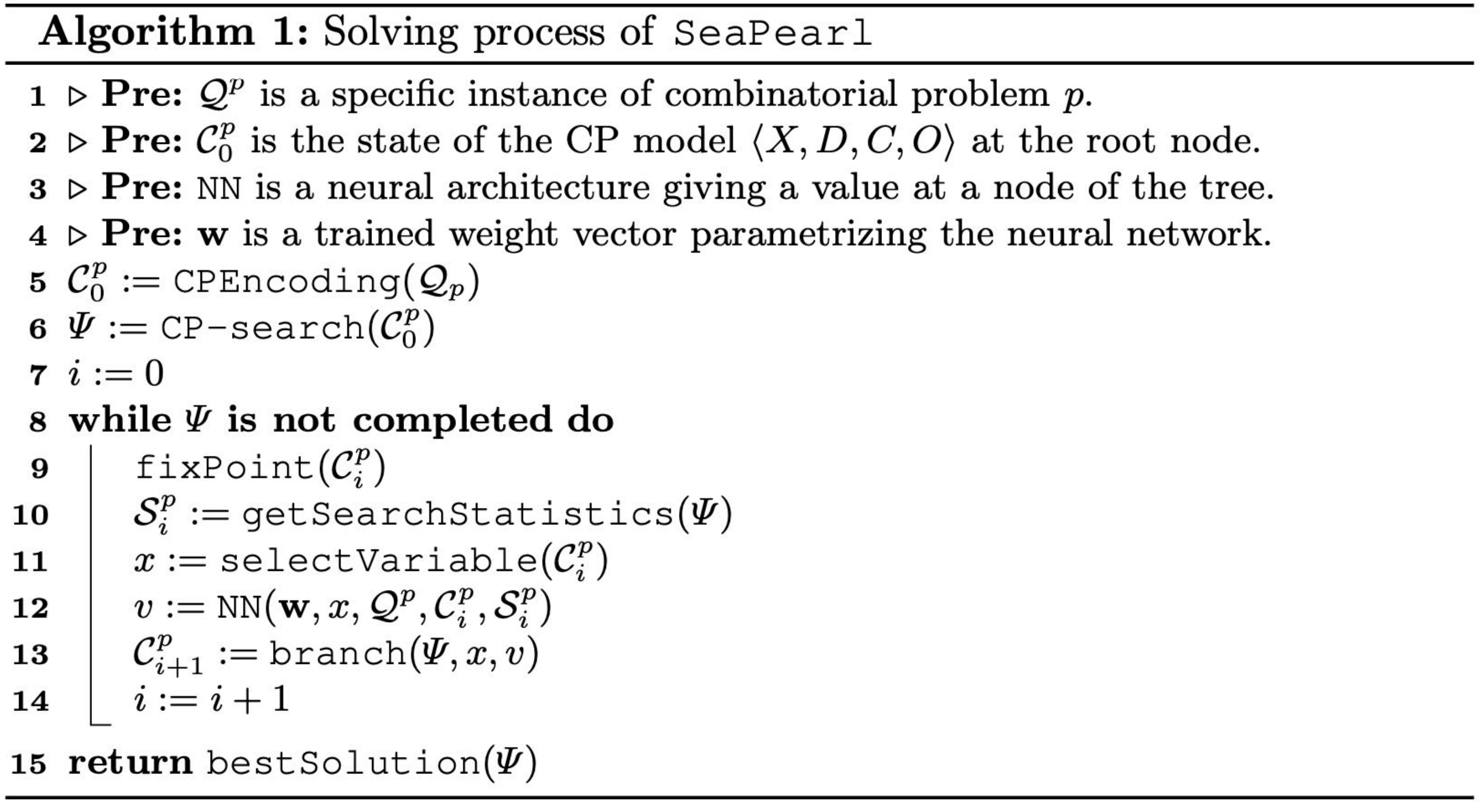
Embedding Learning in CP

- Below, it is a high level overview of the SeaPearl architecture



- The CP model includes:
  - The variables X
  - The domains of the variables D(X)
  - The constraints C
  - The objective O that must respect the constraints C and it is the subject of optimization
  - Tuple <X, D, C, O>
- The RL model has the goal to improve the CP solving process by learning an appropriate value-selection heuristic and using it at each node of the tree search

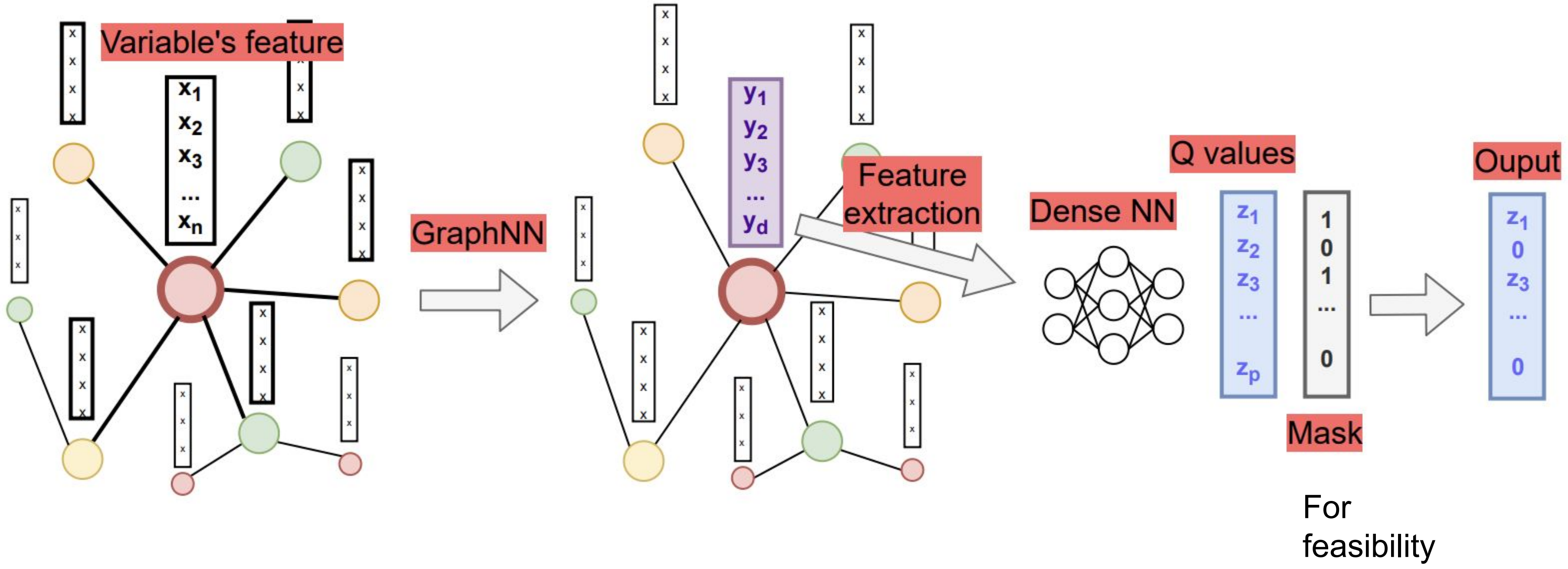
Algorithm



Algorithm 1: Solving process of `SeaPearl`

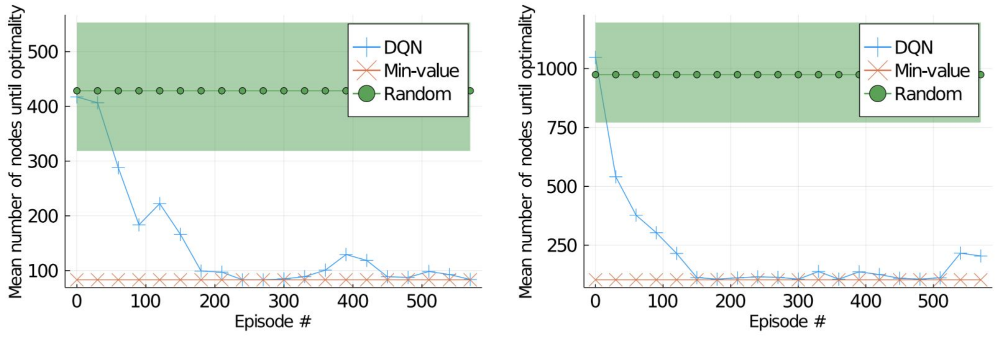
```
▷ Pre: $\mathcal{Q}^p$ is a specific instance of combinatorial problem $p$.
▷ Pre: $\mathcal{C}_0^p$ is the state of the CP model $\langle X, D, C, O \rangle$ at the root node.
▷ Pre: NN is a neural architecture giving a value at a node of the tree.
▷ Pre: $\mathbf{w}$ is a trained weight vector parametrizing the neural network.
$\mathcal{C}_0^p :=$ CPEncoding($\mathcal{Q}_p$)
$\Psi :=$ CP-search($\mathcal{C}_0^p$)
$i := 0$
while $\Psi$ is not completed do
    fixPoint($\mathcal{C}_i^p$)
    $\mathcal{S}_i^p :=$ getSearchStatistics($\Psi$)
    $x :=$ selectVariable($\mathcal{C}_i^p$)
    $v :=$ NN($\mathbf{w}, x, \mathcal{Q}^p, \mathcal{C}_i^p, \mathcal{S}_i^p$)
    $\mathcal{C}_{i+1}^p :=$ branch($\Psi, x, v$)
    $i := i + 1$
return bestSolution($\Psi$)
```

Neural network representation



Experiments

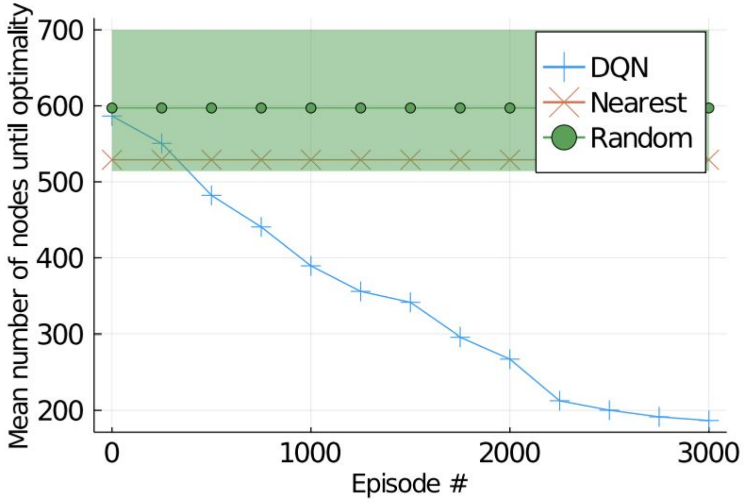
- The authors ran the framework on Graph Coloring Problem and on Traveling Salesman with time window (it is the classic TSP with some additional constraints as: in node v you can only visit it in a given period of time - this is the node feature in GNN)
- AWS2, 1 CPU - Intel Xeon, 32 GB RAM
- On Graph Coloring problem - trained 600 episodes, 13 hours



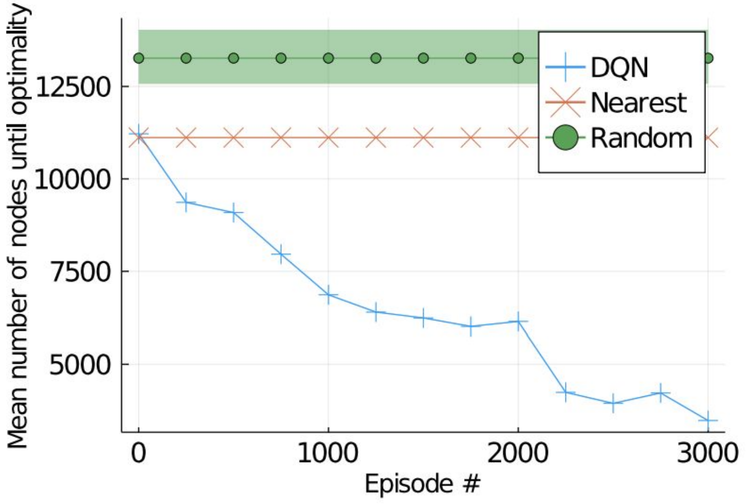
(a) Instances with 20 nodes (b) Instances with 30 nodes

Experiments

- On TSPTW - trained 3000 episodes, 6 hours



(a) Instances with 20 cities



(b) Instances with 50 cities

Conclusions

- Because it’s still an area in development, this article can give good perspective in solving combinatorial problems with neural networks