Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization

Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, Andre Cire
Finding an **optimal solution** from a **discrete set** of solutions is **hard**

and for some industrial problems it needs to be done **often**!
Search-based approaches

How to exploit this fact in order to explore the solution more efficiently

Complete methods
1. Integer Programming
2. Constraint Programming
3. SAT Solving

Pros
1. Optimality guarantees
2. General-purpose solvers

Cons
1. Prohibitive execution time
2. Do not leverage past resolution of similar problems
End-to-end learning-based approaches

How to leverage valuable knowledge from past experiments in order to learn how to solve the problem

Pros
1. Execution time is super-fast
2. Problem structure is learned
3. Easy to use (when trained)

Cons
1. No optimality guarantees
2. Non trivial to represent a problem
3. Hard to handle constraints (which are vastly present in industrial settings)

[Bengio et al., 2018]
Solving COPs by searching and learning

Taking the best of the two worlds

Searching (OR): controlling the execution and getting guarantees
Learning (ML): leveraging past knowledge for speeding-up the search

[Bengio et al., 2018]
Yes, but how do we do that?

Still an open question that interests many research groups

**Recent works**

- Learning to search in branch and bound algorithms
  - [He et al., 2014, NeurIPS]
- Learning to branch in mixed integer programming
  - [Khalil et al., 2016, AAAI]
- Exact combinatorial optimization with graph convolutional neural networks
  - [Gasse et al., 2019, NeurIPS]
- Improving Optimization Bounds using Machine Learning: Decision Diagrams meet Deep Reinforcement Learning
  - [Cappart et al., 2019, AAAI]
- Solving Mixed Integer Programs Using Neural Networks
  - [Nair et al., 2020, ArXiv]

**Main limitations of these works**

Training often done with imitation learning *(requires labeled data)*
We would like to tackle some of these difficulties

1. Able to prove optimality
2. Not restricted to integer programs
3. Efficiently learn from previous decisions
4. No need of ground truth (labeled data)

Dynamic programming (DP) as an unifying representation between constraint programming and reinforcement learning
Given a generic combinatorial optimization problem
\[ \text{COP } Q : \{ \max f(x) : x \in X \subseteq \mathbb{Z}^n \} \]

In DP, the problem would be define using:

- A set *decision* or *actions* \( x_i \) taking values from *domains* \( \text{Dom}(x_i) \)
- That enforce a *transition* \( T : S \times X \to S \) from a *state* \( s_i \) to the next \( s_{i+1} \)
- A initial state \( s_1 \) and a transition to perform at every *stage* \( i \in \{1, \ldots, n\} \)
- A *reward* \( R : S \times X \to \mathbb{R} \) is induced after each transition
- A set of conditions (*validity* and *dominance*) restrict the possible transitions

Which can be solved recursively using the *Bellman Equation*

\[ g_i(s_i) = \max \left\{ R(s_i, x_i) + g_{i+1}(T(s_i, x_i)) \right\} \quad \forall i \in \{1..n\} \quad s.t. \quad T(s_i, x_i) \neq \bot \]
In CP, a combinatorial optimization problem is defined using:

- A set of variables $X$ taking values in their domains $D(X)$, subject to a set of constraints $C(X)$ and objective function $O$.
- We use an encoding that uses two types of variables: decision and auxiliary variables.
- Auxiliary variables $x_i^s$ represent the current state at stage $i$.
- Decision variables $x_i^a$ denote the action that will be taken at stage $i$.

The previous DP can be expressed in CP in the following manner:

$$
\max_{x^a} \left( \sum_{i=1}^{n} R(x_i^s, x_i^a) \right)
$$

subject to:

- $x_1^s = \epsilon$ (Setting initial state)
- $x_{i+1}^s = T(x_i^s, x_i^a)$, $\forall i \in \{1, \ldots, n\}$ (Enforcing transitions)
- $\text{validityCondition}(x_i^s, x_i^a)$, $\forall i \in \{1, \ldots, n\}$ (Keeping valid transitions)
- $\text{dominanceCondition}(x_i^s, x_i^a)$, $\forall i \in \{1, \ldots, n\}$ (Pruning dominated states)
Main assumption: we have a DP model of the problem

Proposed Framework

Learning phase
- Training instances (randomly generated)

Unifying representation
- Combinatorial optimization problem
- Dynamic programming Model

Solving phase
- Evaluated instances

Reinforcement learning
- Environment
- Agent

Value-selection heuristic

Constraint programming
- Model
- Search
- Solution

What we propose
- Pytorch (python)

What exists already
- Gecode (C++)
The DL architecture selected need to:
1. handle instances of the same COPs, but different number of variables
2. be invariant to input permutation

We have experimented with
Graph Attention Network (Veličković et al., ICML 2018)
Set Transformers (Lee et al., ICML 2019)

Which then provide input to a feed forward network

The RL agents tested are either
DQN: one value for each action (expected value of action)
PPO: policy gradient (a probability we should select each action)

Embedding the RL agents into the CP Search
BaB: Depth-First Branch and Bound Search (using DQN)
ILDs: Iterative Limited Discrepancy Search (using DQN)
RBS: Restart-based Search (using PPO)
Illustration on TSP

Dynamic programming model

Require to define: states, actions, transition function, reward function

State
- Last customer visited
- Remaining customers to visit

Action
- Visit a new customer
- Subject to some validity conditions

Transition
- Update the state

Reward
- Travelling distance (negative reward)

Exemple

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Action</th>
<th>Cost</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, {1,2,3,4})</td>
<td>4</td>
<td>(d(0 \to 4))</td>
<td>(4, {1,2,3})</td>
</tr>
</tbody>
</table>
Exploiting *again* similarities with dynamic programming

<table>
<thead>
<tr>
<th>DP Model</th>
<th>RL Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td><strong>State</strong></td>
</tr>
<tr>
<td>• Last customer visited</td>
<td>• Last customer visited</td>
</tr>
<tr>
<td>• Remaining customers to visit</td>
<td>• Remaining customers to visit</td>
</tr>
<tr>
<td>• Information about the instance</td>
<td><strong>Action</strong></td>
</tr>
<tr>
<td>• Visit a new customer</td>
<td>• Visit a new customer</td>
</tr>
<tr>
<td><strong>Transition</strong></td>
<td><strong>Transition</strong></td>
</tr>
<tr>
<td>• Update the state</td>
<td>• Same update</td>
</tr>
<tr>
<td><strong>Reward</strong></td>
<td><strong>Reward</strong></td>
</tr>
<tr>
<td>• Travelling distance</td>
<td>• <strong>FIRST</strong>, find a feasible solution</td>
</tr>
<tr>
<td></td>
<td>• <strong>THEN</strong>, minimize the distance</td>
</tr>
</tbody>
</table>
Two possibilities:
• Call RL agent again
• Reuse previous DQN/PPO (caching)
Adding Constraints

Let’s consider the Traveling Salesman Problem with Time Windows

- 8 to 10 am
- 9:30 to 11 am
- 11 am to 2 pm
- 4 pm to 6 pm
TSPTW: A DP model

The model is index in \( i \in S \), and each stage corresponds a customer visit

The state definition

\( s_i \in S \) includes:

- \( v_i \) last node visited
- \( t_i \) is the time of visit
- \( m_i \in \mathcal{P}([2, \ldots, n]) \) set of remaining customers

The problem data

- \( l_i \) & \( u_i \) are bounds on the time window
- \( d_{ij} \) is the distance between two customers

Taking action \( a_i \in \{1, \ldots, n\} \) corresponds to the customer to visit at stage \( i \)

State transition functions as follows

\[
\begin{align*}
    s_1 &= \{m_1 = \{2..n\}, v_1 = 1, t_1 = 0\} \\
    m_{i+1} &= m_i \setminus a_i \\
    v_{i+1} &= a_i \\
    t_{i+1} &= \max\left(t_i + d_{v_i, a_i}, l_{a_i}\right) \\
    V_1 &: a_i \in m_i \\
    V_2 &: u_{a_i} \geq t_i + d_{v_i, a_i} \\
    P &: (t_i \geq u_j) \Rightarrow (j \notin m_i)
\end{align*}
\]

(Initial state definition)

\( \forall i \in \{1..n\} \) (Transition function for \( m_i \))

\( \forall i \in \{1..n\} \) (Transition function for \( v_i \))

\( \forall i \in \{1..n\} \) (Transition function for \( t_i \))

\( \forall i \in \{1..n\} \) (First validity condition)

\( \forall i \in \{1..n\} \) (Second validity condition)

\( \forall i, j \in \{1..n\} \) (Dominance pruning)
## TSPTW: Results

<table>
<thead>
<tr>
<th>Approaches</th>
<th>20 cities</th>
<th>50 cities</th>
<th>100 cities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td><strong>Name</strong></td>
<td><strong>Success</strong></td>
<td><strong>Opt.</strong></td>
</tr>
<tr>
<td>Constraint programming</td>
<td>OR-Tools</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CP-model</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>CP-nearest</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Reinforcement learning</td>
<td>DQN</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>PPO Beam search w=64</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Hybrid (no cache)</td>
<td>BaB-DQN</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>ILDS-DQN</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>RBS-PPO</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Hybrid (with cache)</td>
<td>BaB-DQN*</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>ILDS-DQN*</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>RBS-PPO*</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

**Observation:**
- PPO dominates DQN in end-to-end ML
- Reverse is observed when used inside CP

**Average time to make a decision**
- BaB-DQN – 34 ms
- Bab-DQQ: caching – 0.16ms
- CP-nearest – 0.004 ms
4-Moments Portfolio Optimization

Given a set of $n$ investments, each with a specific cost ($a_i$), an expected return ($\mu_i$), a standard deviation ($\sigma_i$), a skewness ($\gamma_i$), and a kurtosis ($\kappa_i$).

Each investor attributes an importance ($\lambda_{\{1...4\}}$) to each moment and must decide ($x_i \in \{0,1\}$) whether he makes each investment or not, subject to a budget $B$. The objective is to select large positive expected return and skewness, with large negative variance and kurtosis.

Math. Programming model

$$\text{maximize } \left( \lambda_1 \sum_{i=1}^{n} \mu_i x_i - \lambda_2 \sum_{i=1}^{n} \sigma_i^2 x_i + \lambda_3 \sum_{i=1}^{n} \gamma_i^3 x_i - \lambda_4 \sum_{i=1}^{n} \kappa_i^4 x_i \right)$$

subject to

$$\sum_{i=1}^{n} a_i x_i \leq B \quad \forall i \in \{1..n\}$$

$$x_i \in \{0,1\} \quad \forall i \in \{1..n\}$$

Dynamic Programming model

$$s_1 = 0$$

$$s_{i+1} = s_i + x_i a_i \quad \forall i \in \{1..n\}$$

$$V_i: s_i + x_i a_i \leq B. \quad \forall i \in \{1..n\}$$

Discrete Non-Convex Programming Problem
PORT: Results

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Continuous coefficients</th>
<th>Discrete coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 items</td>
<td>50 items</td>
</tr>
<tr>
<td>Non-linear solver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNITRO</td>
<td>343.79</td>
<td>0</td>
</tr>
<tr>
<td>APQPT</td>
<td>342.62</td>
<td>0</td>
</tr>
<tr>
<td>Constraint programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP-model</td>
<td><strong>356.49</strong></td>
<td>98</td>
</tr>
<tr>
<td>DQN</td>
<td>306.71</td>
<td>0</td>
</tr>
<tr>
<td>PPO</td>
<td>344.95</td>
<td>0</td>
</tr>
<tr>
<td>Reinforcement learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BaB-DQN*</td>
<td><strong>356.49</strong></td>
<td><strong>100</strong></td>
</tr>
<tr>
<td>ILDS-DQN*</td>
<td><strong>356.49</strong></td>
<td>1</td>
</tr>
<tr>
<td>RBS-PPO*</td>
<td>356.35</td>
<td>0</td>
</tr>
</tbody>
</table>

**Observations:**
- Discrete coefficients break the continuity of the objective function making the problem harder of NL solver, but not for hybrid.
- CP/Hybrid can prove optimality on smaller problems.
Conclusion and perspectives

Contributions and results:
1. A **generic approach** based on RL and CP for solving COP (modelled as DPs)
2. **Promising results on challenging problems** (TSPTW, and portfolio optimization)
3. Open-source release of our code (A Julia version, *SeaPearl* (CPRL) is coming soon...)

Perspectives and future work:
1. **Testing on more combinatorial optimization problems**
2. **Speeding-up the prediction time**
Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization

louis-martin.rousseau@polymtl.ca

arxiv.org/abs/2006.01610 (this talk)
arxiv.org/abs/2102.09193 (SeaPearl.jl)
github.com/qcappart/hybrid-cp-rl-solver