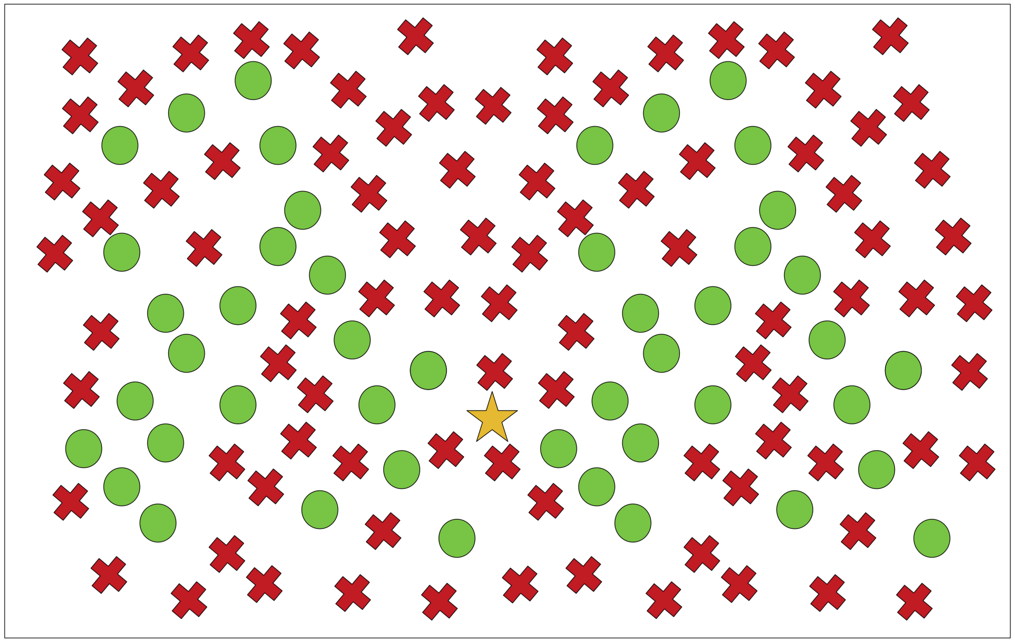
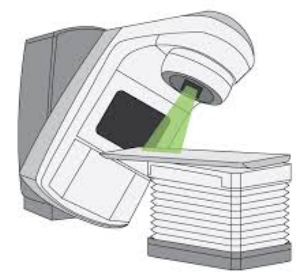
Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization

*Quentin Cappart, Thierry Moisan, **Louis-Martin Rousseau**,*
Isabeau Prémont-Schwarz, Andre Cire

Combinatorial optimization

Finding an optimal solution from a discrete set of solutions is hard





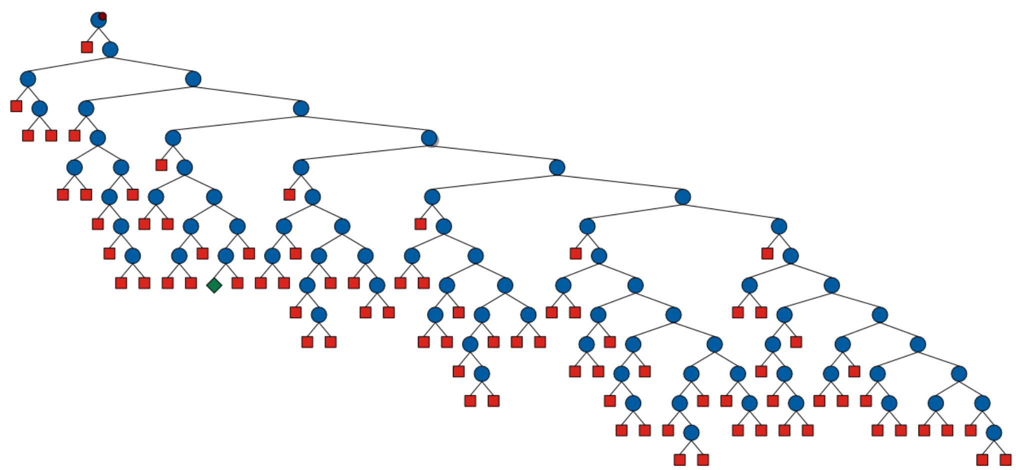
and for some industrial problems it needs to be done often!

Search-based approaches

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

Complete methods

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



Pros

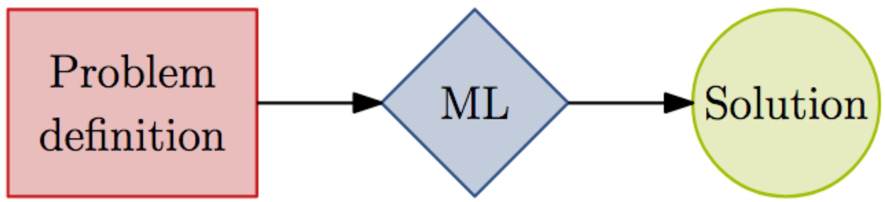
1. Optimality guarantees
2. General-purpose solvers

Cons

1. Prohibitive execution time
2. Do not leverage past resolution of similar problems problems

End-to-end learning-based approaches

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



[Bengio et al., 2018]

Pros

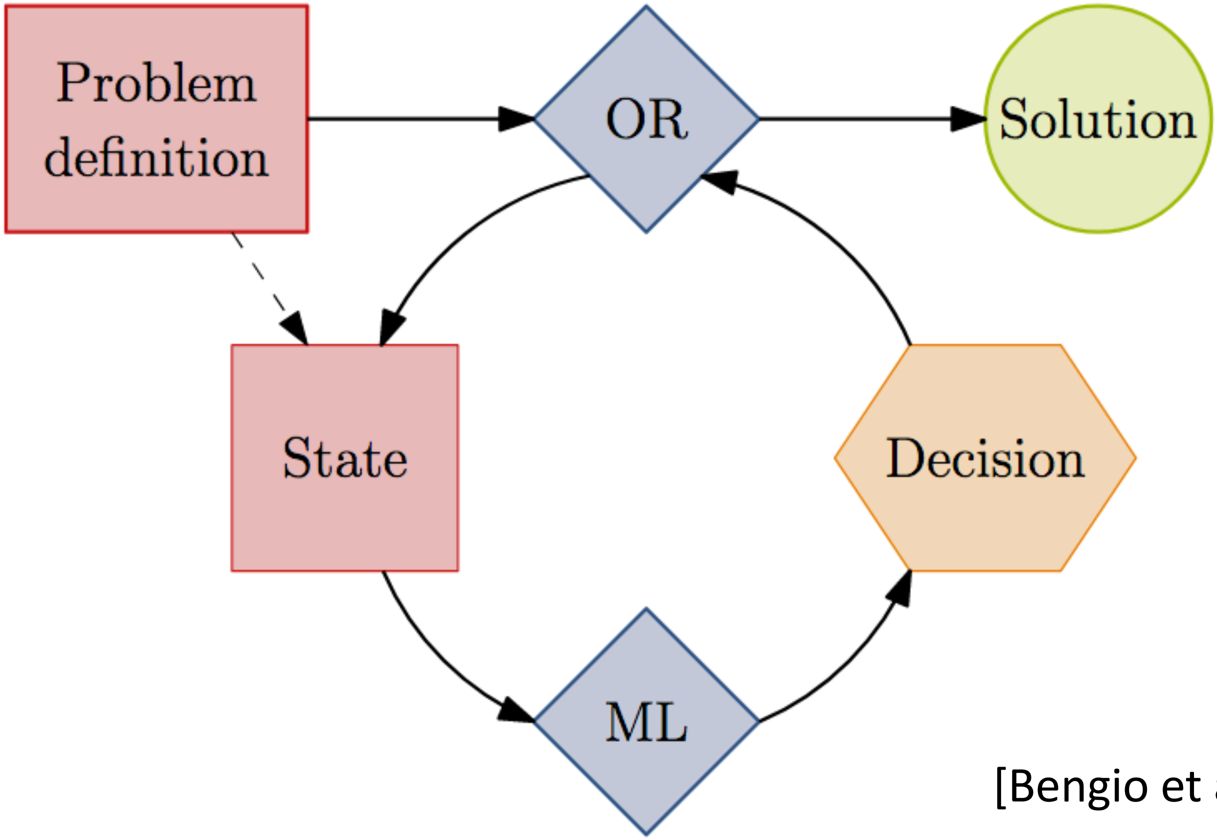
1. Execution time is super-fast
2. Problem structure is learned
3. Easy to use (when trained)

Cons

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

Solving COPs by searching and learning

Taking the best of the two worlds



[Bengio et al., 2018]

Searching (OR): controlling the execution and getting guarantees

Learning (ML): leveraging past knowledge for speeding-up the search

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)

Proposed approach

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)

Search-based tools

- Integer Programming
- Constraint Programming (CP)
- SAT solving
- Local Search

Learning-based tools

- Supervised Learning
- Reinforcement Learning (RL)
- Unsupervised Learning

Dynamic programming (DP) as an unifying representation between constraint programming and reinforcement learning

DP notation

Given a generic combinatorial optimization problem

$$\text{COP } \mathcal{Q} : \{\max f(x) : x \in X \subseteq \mathbb{Z}^n\}$$

In DP, the problem would be define using:

- A set *decision* or *actions* (x_i) taking values from *domains* ($Dom(x_i)$)
- That enforce a *transition* ($T: S \times X \rightarrow S$) from a *state* ($s_i$) to the next ($s_{i+1}$)
- A initial state (s_1) and a transition to perform at every *stage* ($i \in \{1, \dots, n\}$)
- A *reward* ($R: S \times X \rightarrow \mathbb{R}$) 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*

$$g_i(s_i) = \max \left\{ R(s_i, x_i) + g_{i+1}\big(T(s_i, x_i)\big) \right\} \quad \forall i \in \{1..n\} \;\; s.t. \;\; T(s_i, x_i) \neq \perp$$

From DP to CP

In CP, a combinatorial optimizatoin problem is define using:

- A set variables `X` taking value in their *domains* `D(X)`, subject to a set of constraints C(X) and objective funciton O.
- We use an encoding that uses two types of variables: decision and auxiliary
- Auxiliary variables $\mathtt{x}_i^s$ represent the current state at stage i
- Decision variables $\mathtt{x}_i^a$ denotes the action that will be taken at stage i.

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

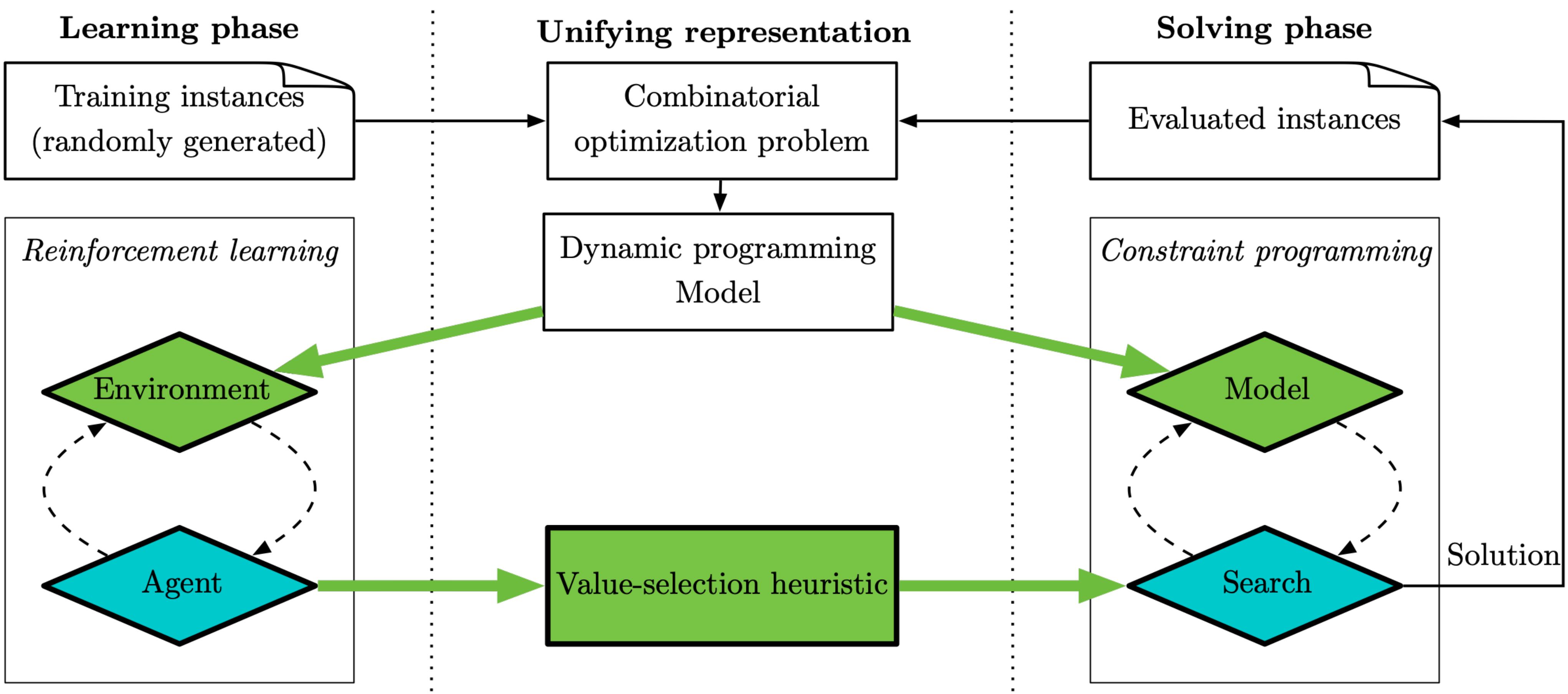
Easily implemented
with element and table constraints

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

$$\begin{aligned}
\text{s.t.} \quad & \mathtt{x}_1^s = \epsilon & & \text{(Setting initial state)} \\
& \mathtt{x}_{i+1}^s = T(\mathtt{x}_i^s, \mathtt{x}_i^a) & \forall i \in \{1, \dots, n\} & \quad \text{(Enforcing transitions)} \\
& \mathtt{validityCondition}(\mathtt{x}_i^s, \mathtt{x}_i^a) & \forall i \in \{1, \dots, n\} & \quad \text{(Keeping valid transitions)} \\
& \mathtt{dominanceCondition}(\mathtt{x}_i^s, \mathtt{x}_i^a) & \forall i \in \{1, \dots, n\} & \quad \text{(Pruning dominated states)}
\end{aligned}$$

Proposed Framework

Main assumption: we have a DP model of the problem



Pytorch (python)

Gecode (C++)

What we propose **What exists already**

DL, RL and Search Architecture

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 experiemented 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)

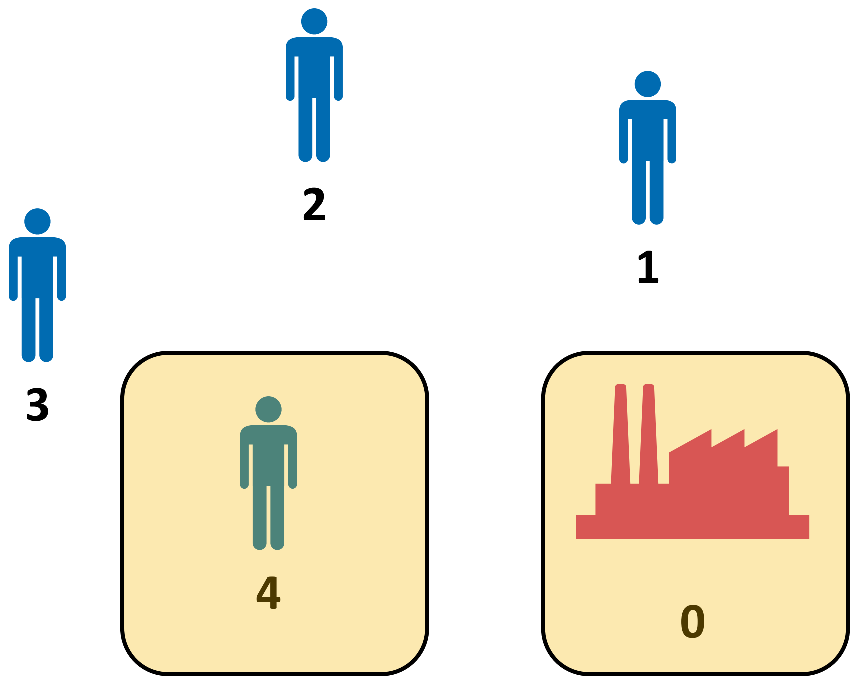
ILDS: Iterative Limited Discrepency Search (using DQN)

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

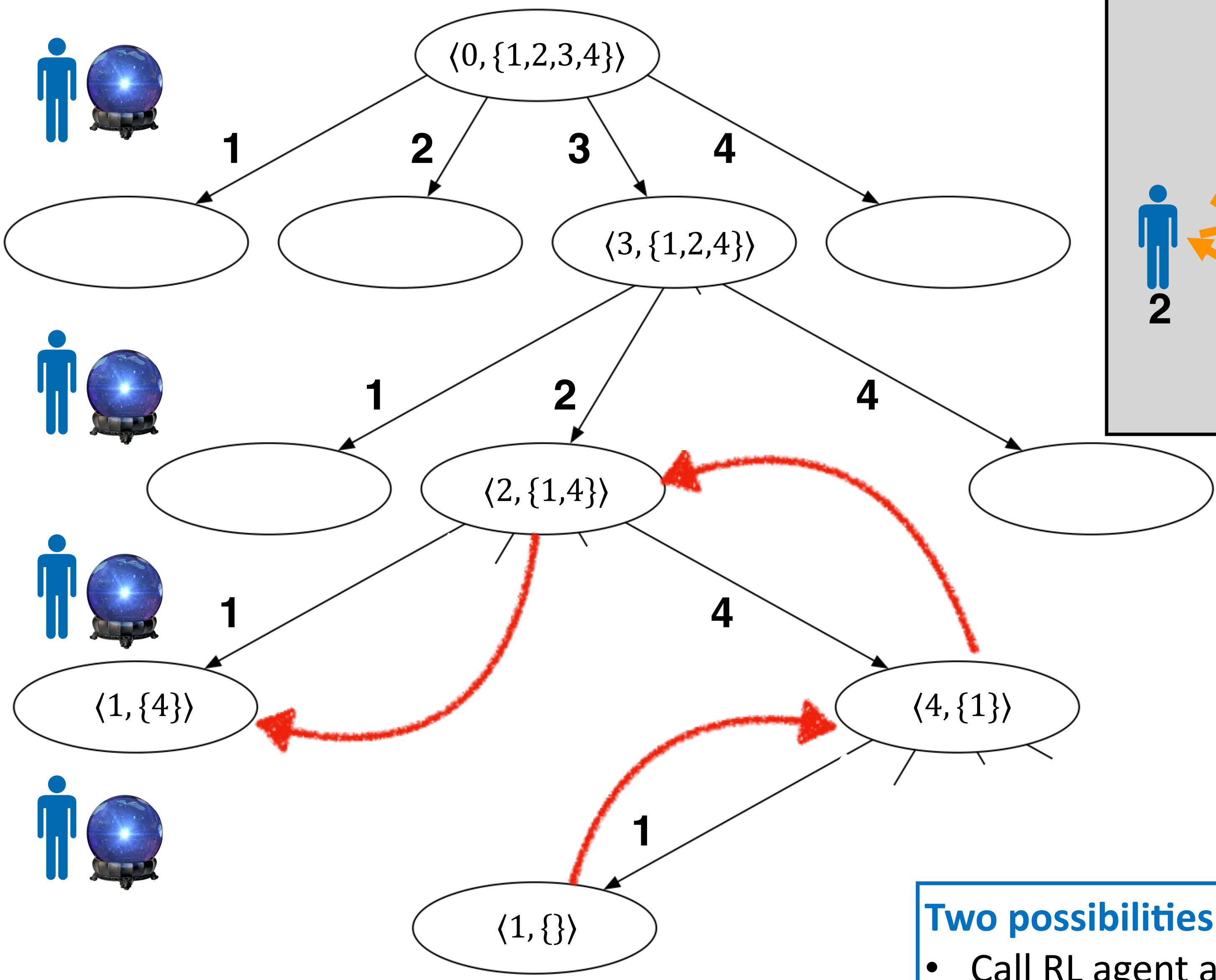
Initial state	**Action**	**Cost**	**Next state**
$\langle 0, \{1,2,3,4\}\rangle$	4	$d(0 \rightarrow 4)$	$\langle 4, \{1,2,3\}\rangle$

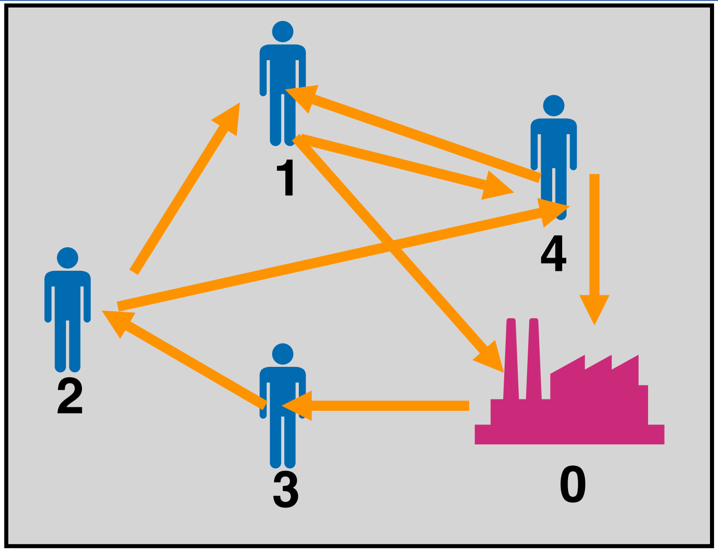
Link To RL environment

Exploiting **again** similarities with dynamic programming

	DP Model	RL Environment
State	• Last customer visited • Remaining customers to visit	• Last customer visited • Remaining customers to visit • **Information about the instance**
Action	• Visit a new customer	• Visit a new customer
Transition	• Update the state	• Same update
Reward	• Travelling distance	• **FIRST**, find a feasible solution • **THEN**, minimize the distance

Constraint programming search

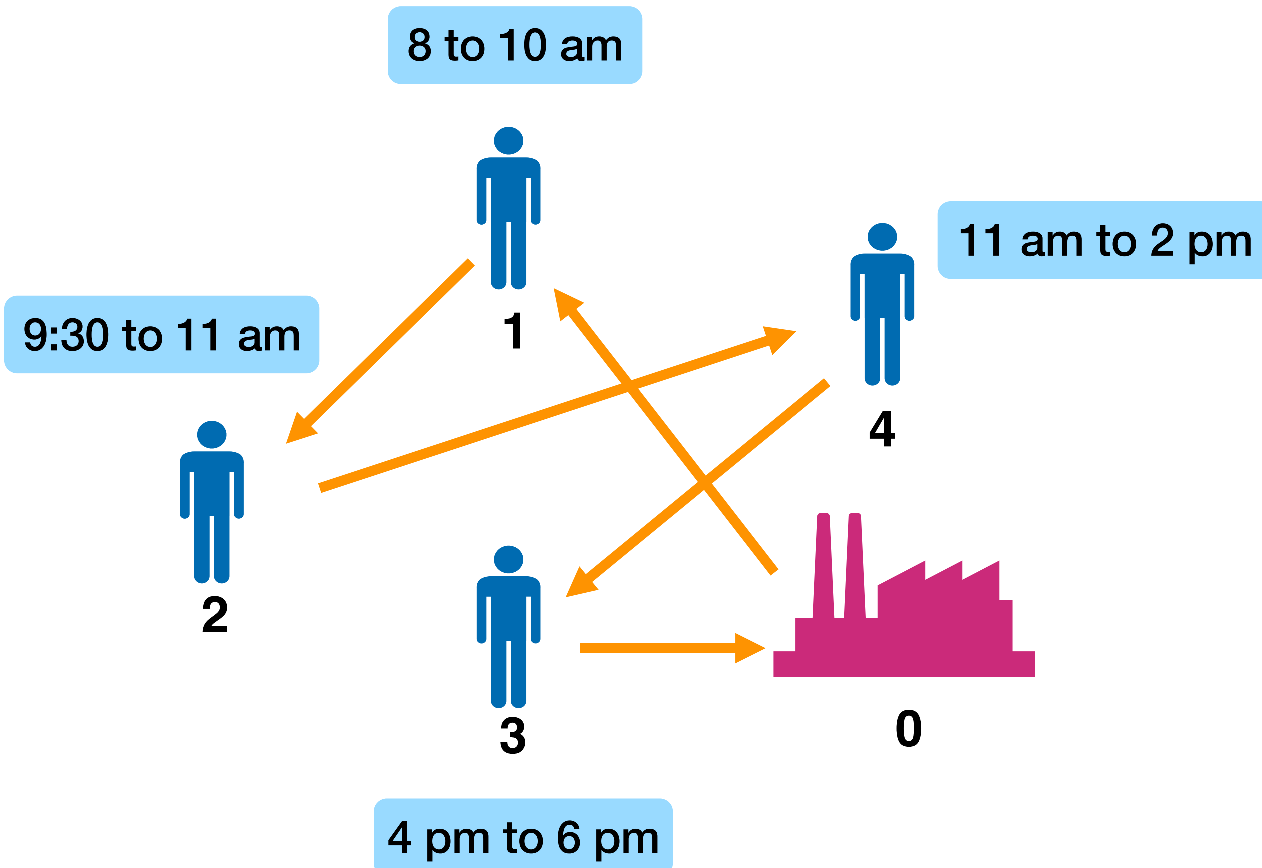




Two possibilities:
- Call RL agent again
- Reuse previous DQN/PPO (caching)

Adding Constraints

Let's consider the Traveling Salesman Problem **with Time Windows**



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
- $d_{\cdot,\cdot}$ 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

$$s_1 = \left\{m_1 = \{2..n\},\ v_1 = 1,\ t_1 = 0\right\} \qquad \text{(Initial state definition)}$$

$$m_{i+1} = m_i \setminus a_i \qquad \forall i \in \{1..n\} \quad \text{(Transition function for } m_i\text{)}$$

$$v_{i+1} = a_i \qquad \forall i \in \{1..n\} \quad \text{(Transition function for } v_i\text{)}$$

$$t_{i+1} = \max\left(t_i + d_{v_i,a_i}, l_{a_i}\right) \qquad \forall i \in \{1..n\} \quad \text{(Transition function for } t_i\text{)}$$

$$V_1 : a_i \in m_i \qquad \forall i \in \{1..n\} \quad \text{(First validity condition)}$$

$$V_2 : u_{a_i} \geq t_i + d_{v_i,a_i} \qquad \forall i \in \{1..n\} \quad \text{(Second validity condition)}$$

$$P : \left(t_i \geq u_j\right) \Rightarrow \left(j \notin m_i\right) \qquad \forall i, j \in \{1..n\} \quad \text{(Dominance pruning)}$$

TSPTW: Results

Approaches		20 cities			50 cities			100 cities		
Type	Name	Success	Opt.	Time	Success	Opt.	Time	Success	Opt.	Time (in min.)
Constraint programming	OR-Tools	100	0	< 1	0	0	t.o.	0	0	t.o.
	CP-model	100	100	< 1	0	0	t.o.	0	0	t.o.
	CP-nearest	100	100	< 1	99	99	6	0	0	t.o.
Reinforcement learning	DQN	100	0	< 1	0	0	< 1	0	0	< 1
	PPO Beam wearch w=64	100	0	< 1	100	0	5	21	0	46
Hybrid (no cache)	BaB-DQN	100	100	< 1	100	99	2	100	52	20
	ILDS-DQN	100	100	< 1	100	100	2	100	53	39
	RBS-PPO	100	100	< 1	100	80	12	100	0	t.o.
Hybrid (with cache)	BaB-DQN*	100	100	< 1	100	100	< 1	100	91	15
	ILDS-DQN*	100	100	< 1	100	100	1	100	90	15
	RBS-PPO*	100	100	< 1	100	99	2	100	11	32

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* (μ_i), a *standard deviation* (σ_i), a *skewness* (γ_i), and a *kurtosis* (κ_i).

Each investors 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

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

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

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

Dynamic Programming model

$$s_1 = 0$$

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

$$V_1 \colon s_i + x_i a_i \leq B. \ \ \forall i \in \{1..n\}$$

Discrete Non-Convex Programming Problem

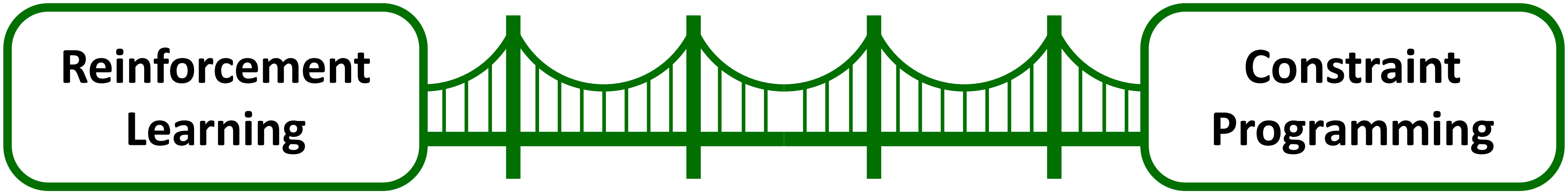
PORT: Results

Approaches		Continuous coefficients						Discrete coefficients					
		20 items		50 items		100 items		20 items		50 items		100 items	
Type	Name	Sol.	Opt.	Sol.	Opt.	Sol.	Opt.	Sol.	Opt.	Sol.	Opt.	Sol.	Opt.
Non-linear solver	KNITRO	343.79	0	**1128.92**	0	**2683.55**	0	211.60	0	1039.25	0	2635.15	0
	APOPT	342.62	0	1127.71	0	2678.48	0	-	-	-	-	-	-
Constraint programming	CP-model	**356.49**	98	1028.82	0	2562.59	0	**359.81**	**100**	1040.30	0	2575.64	0
Reinforcement learning	DQN	306.71	0	879.68	0	2568.31	0	309.17	0	882.17	0	2570.81	0
	PPO Beam 64	344.95	0	1123.18	0	2662.88	0	347.85	0	1126.06	0	2665.68	0
Hybrid (with cache)	BaB-DQN*	**356.49**	**100**	1047.13	0	2634.33	0	**359.81**	**100**	1067.37	0	2641.22	0
	ILDS-DQN*	**356.49**	1	1067.20	0	2639.18	0	**359.81**	**100**	1084.21	0	2652.53	0
	RBS-PPO*	356.35	0	1126.09	0	2674.96	0	359.69	0	**1129.53**	0	**2679.57**	0

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



Dynamic Programming

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

POLYTECHNIQUE MONTRÉAL

UNIVERSITY OF TORONTO SCARBOROUGH