Structured learning for vehicle routing problems

Léo Baty[1], Guillaume Dalle[1], Louis Bouvier[1], Axel Parmentier[1], Kai Jungel[2], Patrick Klein[2], Maximilian Schiffer[2]

[1]CERMICS, École des Ponts, [2]Technical University of Munich

November 30, 2022

Learning to solve hard combinatorial problems

We consider a hard combinatorial problem

$$(H)\colon \min_{y \in \mathcal{Y}(x)} c(y)$$

- x: input instance
- $\mathcal{Y}$: finite combinatorial constraints set
- $c\colon \mathcal{Y} \to \mathbb{R}$: objective function

Learning to solve hard combinatorial problems

We consider a hard combinatorial problem

$$(H): \min_{y \in \mathcal{Y}(x)} c(y)$$

- x: input instance
- $\mathcal{Y}$: finite combinatorial constraints set
- $c \colon \mathcal{Y} \to \mathbb{R}$: objective function

Usual multiclass classification end-to-end learning

$$\xrightarrow[x]{\text{Input}} \boxed{\text{ML predictor } \varphi_w} \xrightarrow[y \in [0,1]^{|\mathcal{Y}|}]{\text{Output}}$$

Problem: too many classes!

Machine Learning with combinatorial layers

We want to use a Combinatorial Optimization (CO) oracle

$$f\colon \theta \longmapsto \operatorname*{argmax}_{y\in\mathcal{Y}} \theta^\top y$$

where $\mathcal{Y}$ is a finite set, inside the pipeline

$$\xrightarrow[x]{\text{Instance}} \boxed{\text{ML predictor } \varphi_w} \xrightarrow[\theta\,=\,\varphi_w(x)]{\text{Objective}} \boxed{\text{CO oracle } f} \xrightarrow[y\,=\,f(\theta)]{\text{Solution}}$$

Flat derivatives everywhere

$$f : \theta \longmapsto \operatorname*{argmax}_{y \in \mathcal{Y}} \theta^\top y$$

When we apply Automatic Differentiation (AD) to a CO oracle:

- It usually doesn't work (lack of compatibility with solver)

Flat derivatives everywhere

$$f: \theta \longmapsto \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \, \theta^\top y$$

When we apply Automatic Differentiation (AD) to a CO oracle:

- It usually doesn't work (lack of compatibility with solver)
- Even when it does, the Jacobian is either zero or undefined (because f is piecewise constant on $\mathcal{Y}$)

Regularized CO oracle

We replace our CO oracle

$$f: \theta \longmapsto \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, \theta^\top y$$

by using a probability distribution $p(\cdot|\theta)$ on $\mathcal{Y}$

$$\widehat{f}: \theta \longmapsto \mathbb{E}_{p(\cdot|\theta)}[Y] = \sum_{y \in \mathcal{Y}} y\, p(y|\theta)$$

New pipeline:

$$\xrightarrow[x]{\text{Instance}} \boxed{\text{ML predictor } \varphi_w} \xrightarrow{\text{Objective } \theta} \boxed{\text{Probabilistic CO layer } \widehat{f}} \xrightarrow{\text{Solution } y}$$

Building the distribution

$$\widehat{f} \colon \theta \longmapsto \mathbb{E}_{p(\cdot|\theta)}[Y] = \sum_{y \in \mathcal{Y}} y \, p(y|\theta)$$

We want a distribution $p(\cdot|\theta)$ such that:

- $\theta \mapsto p(\cdot|\theta)$ is differentiable
- $\widehat{f}$ approximates f
- Computing $\widehat{f}$ is easy (only requires the oracle f for example)

Additive perturbation

Perturb the objective with an additive noise [Berthet et al., 2020]:

$$\widehat{f}^{+}_{\varepsilon} \colon \theta \longmapsto \mathbb{E}\left[\underset{y\in\mathcal{Y}}{\operatorname{argmax}} (\theta + \varepsilon Z)^\top y\right] = \mathbb{E}[f(\theta + \varepsilon Z)]$$

with $Z \sim \mathcal{N}(0,1)$, and $\varepsilon \in \mathbb{R}_+$.

Intractable expectation $\Rightarrow$ Monte-Carlo sampling approximation

Other distributions

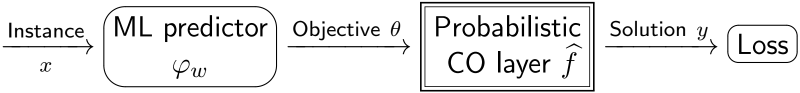
- Multiplicative perturbations
- Convex regularization
- ...

see our paper [Dalle et al., 2022]

Learn by imitation or by experience ?

$$\xrightarrow[x]{\text{Instance}} \boxed{\begin{array}{c}\text{ML predictor}\\ \varphi_w\end{array}} \xrightarrow{\text{Objective } \theta} \boxed{\begin{array}{c}\text{Probabilistic}\\ \text{CO layer } \widehat{f}\end{array}} \xrightarrow{\text{Solution } y} \boxed{\text{Loss}}$$

Learn by imitation or by experience ?



1. Learning by imitation:
   - Instance/solutions pairs: $\mathcal{D} = \{(x^1, \overline{y}^1), \dots, (x^n, \overline{y}^n)\}$
   - Goal: imitate target solutions $\overline{y}$

Learn by imitation or by experience ?

$\xrightarrow[x]{\text{Instance}}$ $\boxed{\text{ML predictor } \varphi_w}$ $\xrightarrow{\text{Objective } \theta}$ $\boxed{\text{Probabilistic CO layer } \widehat{f}}$ $\xrightarrow{\text{Solution } y}$ $\boxed{\text{Loss}}$

1. Learning by imitation:
   - Instance/solutions pairs: $\mathcal{D} = \{(x^1, \overline{y}^1), \dots, (x^n, \overline{y}^n)\}$
   - Goal: imitate target solutions $\overline{y}$
2. Learning by experience:
   - Instances only: $\mathcal{D} = \{x^1, \dots, x^n\}$
   - Goal: minimize $c(y)$

Loss functions

$$\xrightarrow[x]{\text{Instance}} \boxed{\begin{array}{c}\text{ML predictor}\\ \varphi_w\end{array}} \xrightarrow{\text{Objective } \theta} \boxed{\begin{array}{c}\text{Probabilistic}\\ \text{CO layer } \widehat{f}\end{array}} \xrightarrow{\text{Solution } y} \boxed{\text{Loss}}$$

1. Learning by imitation:

$$\mathcal{L}_\varepsilon^{\mathsf{FY}}(\theta, \overline{y}) = \mathbb{E}\left[\max_{y \in \mathcal{Y}} (\theta + \varepsilon Z)^\top y\right] - \theta^\top \overline{y}$$

$$\widehat{f}_\varepsilon(\theta) - \overline{y} \in \partial_\theta \mathcal{L}_\varepsilon^{\mathsf{FY}}(\theta, \overline{y})$$

2. Learning by experience:

$$\mathcal{L}_p^c(\theta) = \mathbb{E}_{p(\cdot|\theta)}[c(Y)]$$

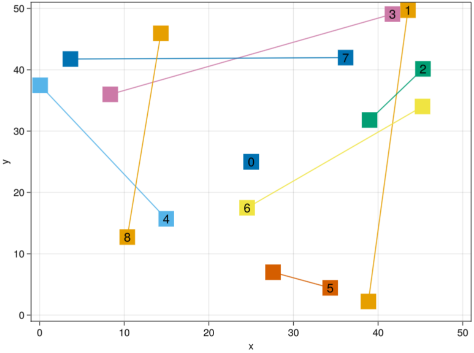
How to implement these pipelines ?

Our package `InferOpt.jl` [Dalle et al., 2022], written in Julia:

- Open source: `https://github.com/axelparmentier/InferOpt.jl`
- Easy to use
- Works with any CO oracle, independent of the implementation
- Compatible with Julia ML and AD ecosystem (through `ChainRules.jl`)

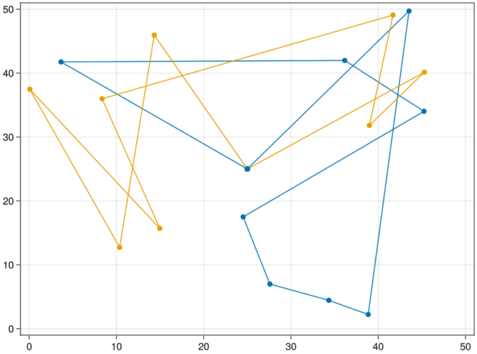
(Deterministic) Vehicle Scheduling Problem (VSP)

- Set of tasks v to complete



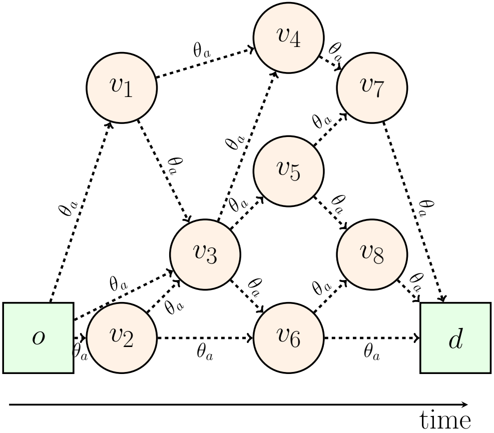
(Deterministic) Vehicle Scheduling Problem (VSP)

- Set of tasks v to complete
- Objective: build routes to minimize total distance cost



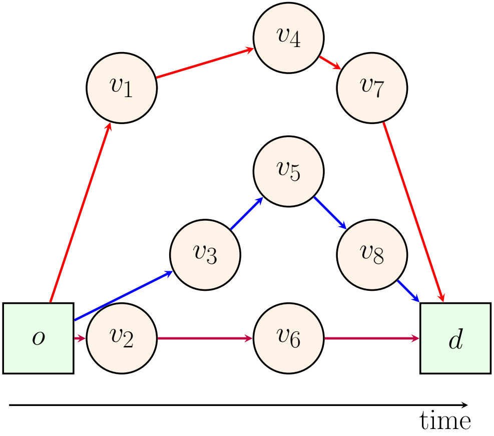
(Deterministic) Vehicle Scheduling Problem (VSP)

- Set of tasks v to complete
- Objective: build routes to minimize total distance cost
- Easy problem $\Rightarrow$ flow formulation, linear program



(Deterministic) Vehicle Scheduling Problem (VSP)

- Set of tasks v to complete
- Objective: build routes to minimize total distance cost
- Easy problem $\Rightarrow$ flow formulation, linear program



Stochastic Vehicle Scheduling (StoVSP)

- After routes are scheduled, we observe random delays
  $\Rightarrow$ delay propagation along vehicle routes
  - set of scenarios $s \in S$
  - intrinsic delay: γ_v^s
  - slack: $\Delta_{u,v}^s$
  - delay propagation along (u, v):

$$d_v^s = \gamma_v^s + \underbrace{\max(d_u^s - \Delta_{u,v}^s, 0)}_{\text{propagated delay}}$$

Stochastic Vehicle Scheduling (StoVSP)

- After routes are scheduled, we observe random delays
  $\Rightarrow$ delay propagation along vehicle routes
  - set of scenarios $s \in S$
  - intrinsic delay: γ_v^s
  - slack: $\Delta_{u,v}^s$
  - delay propagation along (u, v):

$$d_v^s = \gamma_v^s + \underbrace{\max(d_u^s - \Delta_{u,v}^s, 0)}_{\text{propagated delay}}$$

- Objective: minimize vehicle costs and expected delay costs.

Stochastic Vehicle Scheduling (StoVSP)

- After routes are scheduled, we observe random delays
  ⇒ delay propagation along vehicle routes
  - set of scenarios $s \in S$
  - intrinsic delay: γ_v^s
  - slack: $\Delta_{u,v}^s$
  - delay propagation along (u, v):

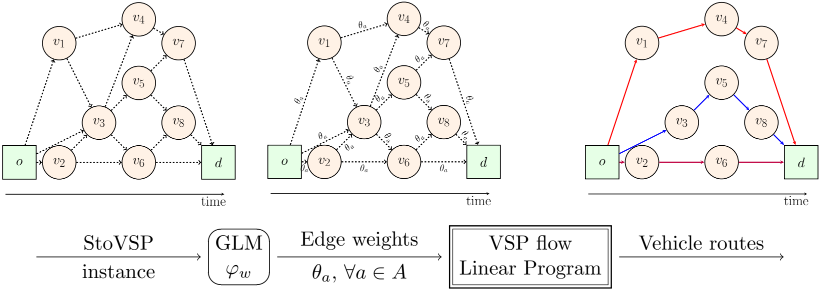
$$d_v^s = \gamma_v^s + \underbrace{\max(d_u^s - \Delta_{u,v}^s, 0)}_{\text{propagated delay}}$$

- Objective: minimize vehicle costs and expected delay costs.
- More difficult to solve, two OR options
  1. Quadratic constraints ⇒ linearize with Mc Cormick
  2. Column generation with constrained shortest path subproblem

  ⇒ does not scale on large instances

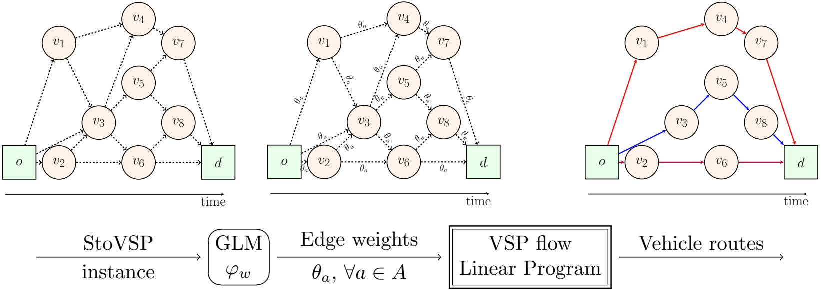
Learning pipeline

- $\theta = \varphi_w(x)$.
- $\theta_a = w^\top \phi(x, a)$, with $\phi(x, a)$ feature vector.



Learning pipeline

- $\theta = \varphi_w(x)$.
- $\theta_a = w^\top \phi(x, a)$, with $\phi(x, a)$ feature vector.



Training datasets (50 instances each):

- 25 tasks and 10 scenarios $\Rightarrow$ label with optimal solution
- 50 tasks and 50 scenarios $\Rightarrow$ label with heuristic solution
- 100 tasks and 50 scenarios $\Rightarrow$ label with heuristic solution

Learning by imitation: gap to target solution

Train dataset	**Test dataset**					
	25 tasks		50 tasks		100 tasks	
	mean	max	mean	max	mean	max
25 tasks	0.68%	9.46%	-0.41%	4.26%	-1.02%	2.4%
50 tasks	0.49%	3.01%	-0.46%	2.34%	-1.6%	0.62%
100 tasks	0.62%	3.36%	-0.14%	9.9%	-1.2%	0.11%

$\Rightarrow$ good imitation

Learning by imitation: average cost per task

Train dataset	**Test dataset** (number of tasks in each instance)							
	25	50	100	200	300	500	750	1000
25 tasks	274.72	225.29	207.14	194.46	186.68	182.56	178.57	177.3
50 tasks	274.27	225.23	205.97	195.78	193.12	194.48	196.99	199.38
100 tasks	274.61	225.87	206.8	197.97	195.53	207.02	219.34	227.14

$\Rightarrow$ good imitation

$\Rightarrow$ poor generalization on large instances when imitating non-optimal solutions

Learning by experience: gap to target solution

Train dataset	Test dataset					
	25 tasks		50 tasks		100 tasks	
	mean	max	mean	max	mean	max
25 tasks	0.45%	4.2%	-0.77%	0.63%	-2.11%	-0.14%
50 tasks	0.43%	3.04%	-0.78%	0.74%	-2.06%	-0.22%
100 tasks	0.43%	3.28%	-0.83%	0.97%	-2.06%	-0.29%

$\Rightarrow$ better gaps, and lower variance

Learning by experience: average cost per task

Train dataset	**Test dataset** (number of tasks in each instance)							
	25	50	100	200	300	500	750	1000
25 tasks	274.19	224.55	204.9	191.86	184.71	181.29	178.0	177.02
50 tasks	274.12	224.51	205.0	191.85	184.3	180.48	176.96	176.0
100 tasks	274.13	224.41	205.0	191.85	184.63	181.08	177.81	176.74

$\Rightarrow$ better gaps, and lower variance
$\Rightarrow$ better generalization

See `https://github.com/BatyLeo/StochasticVehicleScheduling.jl` for reproducible experiments.

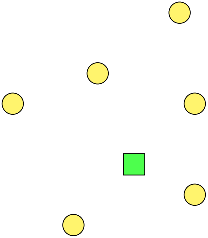
1. Combinatorial Optimization in Machine Learning pipelines

2. Stochastic Vehicle Scheduling

3. Dynamic Vehicle Routing Problem with Time Windows

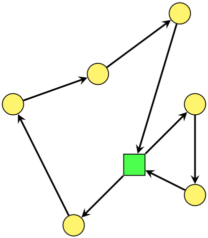
Static Vehicle Routing Problem with Time Windows

- Set of requests to serve: location, time window, demand, service time
- Distance matrix $d_{u,v}$
- **Objective**: serve all requests, minimize total travel distance
- **State-of-the-art**: Hybrid Genetic Search [Vidal, 2021]



Static Vehicle Routing Problem with Time Windows

- Set of requests to serve: location, time window, demand, service time
- Distance matrix $d_{u,v}$
- **Objective**: serve all requests, minimize total travel distance
- **State-of-the-art**: Hybrid Genetic Search [Vidal, 2021]

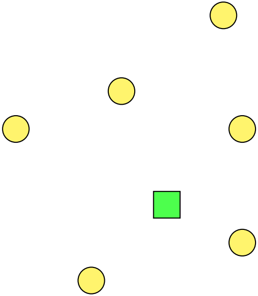


Dynamic VRPTW

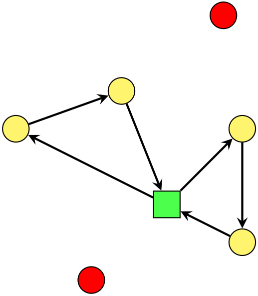
- Time horizon $\{1, \ldots, T\}$, 1-hour epochs
- Requests are not known in advance (only their probability)
- At every epoch t:
  - Decide which request to dispatch
  - Build routes serving them, other requests are postponed
  - Each request must be served before end of its time window $\Rightarrow$ some requests must be dispatched
- State x_t of the system at epoch t: set of requests arrived at t or arrived before but not yet served
- **Objective**: serve all requests, minimize total travel distance

$\Rightarrow$ no state-of-the-art

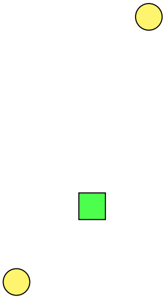
Example: start of epoch 1/2



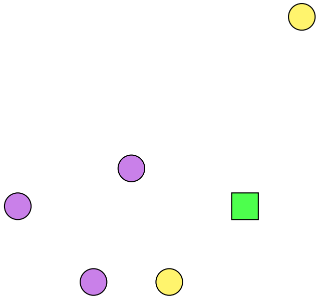
Example: epoch 1 routes



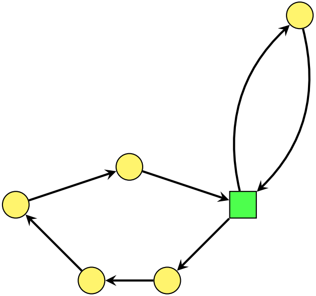
Example: end of epoch 1



Example: start of epoch 2, new requests arrive



Example: epoch 2 routes



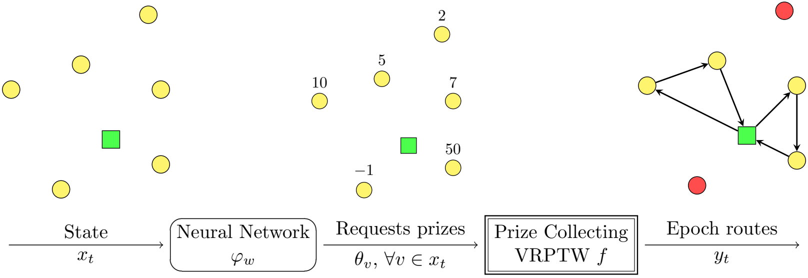
CO layer: Prize Collecting VRPTW

- Serving requests is optional
- Serving request v gives prize θ_v
- **Objective**: maximize total profit minus costs

$$\max_{y \in \mathcal{Y}(x_t)} \sum_{(u,v) \in x_t^2} (\theta_v - d_{u,v}) y_{u,v}.$$

- **Algorithm**: Prize Collecting Hybrid Genetic Search

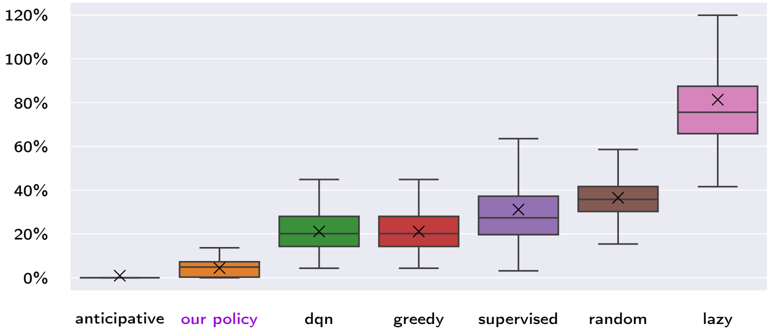
Policy based on a Deep Learning pipeline



$\Rightarrow$ we learn to imitate an anticipative policy

Results: 4.4% average gap

Benchmark on 2252 instances-seed combinations:



Qualifications Winner team of Euro-NeurIPS competition!

Rank	Date	Team	Static cost	Dynamic cost	Avg. cost	Static Rank	Dynamic Rank	Avg. rank
1	10/30/22	Kléopatra	180639.6	333490.8	2.570652e+05	5.0	1.0	3.0
2	10/30/22	OptiML	180639.1	339331.4	2.599852e+05	4.0	2.0	3.0
3	10/30/22	HowToRoute	180565.4	349115.4	2.648404e+05	2.0	6.0	4.0
4	10/31/22	Team_SB	180686.6	341169.1	2.609278e+05	9.0	3.0	6.0
5	10/29/22	ORberto Hood and the Barrymen	180677.0	346094.9	2.633860e+05	8.0	4.0	6.0
6	10/30/22	UPB	180670.8	349342.2	2.650065e+05	7.0	7.0	7.0
7	10/31/22	Miles To Go Before We Sleep	180562.9	352776.8	2.666698e+05	1.0	13.0	7.0
8	10/31/22	Kirchhoffslaw	180575.1	353443.5	2.670093e+05	3.0	15.0	9.0
9	10/20/22	dynamo	180728.3	350960.3	2.658443e+05	12.0	8.0	10.0
10	10/26/22	HustSmart	180799.3	346982.7	2.638910e+05	16.0	5.0	10.5

References

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. (2020).
Learning with Differentiable Perturbed Optimizers.
arXiv:2002.08676 [cs, math, stat].

Dalle, G., Baty, L., Bouvier, L., and Parmentier, A. (2022).
Learning with Combinatorial Optimization Layers: A Probabilistic Approach.

Vidal, T. (2021).
Hybrid Genetic Search for the CVRP: Open-Source Implementation and SWAP* Neighborhood.