Learning to Solve Job Shop Scheduling under Uncertainty

G. Infantes[1], S. Roussel[2], P. Pereira[1], A. Jacquet[1], E. Benazera[1]

[1]Jolibrain
Toulouse, France

[2]Onera
Université de Toulouse
Toulouse, France

CPAIOR 2024

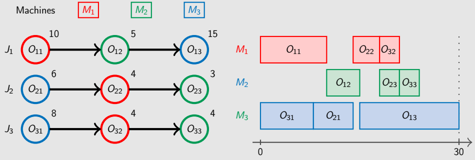
Outline

1. Introduction
2. Components
3. Wheatley : bringing all together
4. Experimental Results
5. Conclusion and Perspectives

Introduction

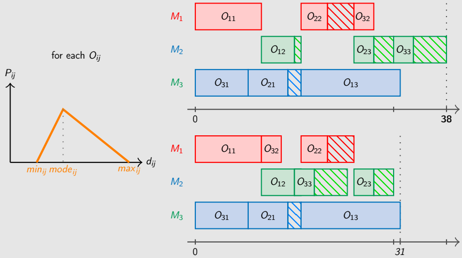
Classical JSSP

- N jobs
- of M tasks (w/ order)
- using M different machines
- deterministic durations



Stochastic JSSP

- probabilistic durations
- off-line (no reschedule)
- minimize expected makespan



Approach

Main ideas

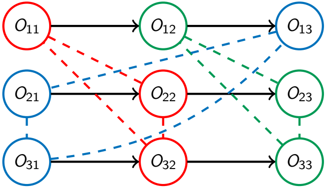
- Consider JSSP as a graph
- Sequential decision problem: add precedencies in the graph
- Use Deep Reinforcement Learning to learn "dispatch rules"
- Use Graph Neural Network inside DRL to get generalization

Expected benefits

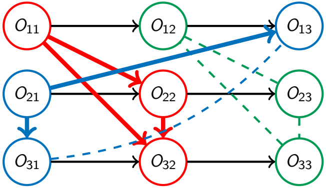
- no time discretization, nor epoch computation
- RL intrinsically accounts for stochasticity

Markov Decision Process

State: Disjunctive Graph



Transitions: select task to dispatch



Reward

- reward is $-$makespan
- durations drawn from distributions
- makespan evaluated *only on complete schedule*

Reinforcement Learning

Learn a policy $\pi(state) = action$ by collecting evidence (trials)

Deep Neural Networks

Stack parameterized functions

$$y_\theta^{(L)} = f_{\theta_L}^{(L)} \circ f_{\theta_{L-1}}^{(L-1)} \circ \dots f_{\theta_0}^{(0)}(x)$$

Prediction Error on {(x,t)}

$$E = loss(y^{(L)}, t)$$

Forward : compute all $y^{(l)}$ compute $E(y^{(L)}, t)$

Compute gradient of the error wrt parameters

- after last layer $\delta^{(L+1)} = \frac{\partial E}{\partial y^{(L)}}$
- backpropagate $\delta^{(l)} = \frac{\partial f_{\theta_{l+1}}^{(l+1)}}{\partial y^{(l)}} \delta^{(l+1)}$
- compute all gradient $\frac{\partial E}{\partial \theta_l} = \frac{\partial E}{\partial y^{(l)}} \frac{\partial y^{(l)}}{\partial \theta_l} = \delta^{(l)} y^{(l-1)}$

Update parameters towards error reduction using gradient, on minibatches

Graph Neural Networks

Inductive bias

- MLP: $y_j^{(l)}$ use all $y_j^{(l-1)}$: $y_j^{(l)} = a^{(l)}(\sum_k w_{jk}^{(l)}(y_k^{(l-1)}) + b_j^{(l)})$
- CNN: $y_j^{(l)}$ use euclidean neighbors of $y_j^{(l-1)}$ (+ pooling/zoom out)
- GNN: $y_j^{(l)}$ use neighbors of $y_j^{(l-1)}$ in the given graph

Message-Passing GNN : use graph as computational lattice

$$a_v^{(k)} = COMB^{(k)}(a_v^{(k)}, AGG^{(k)}(\{MSG^{(k)}(h_u^{k-1}) : u \in N(v)\}))$$

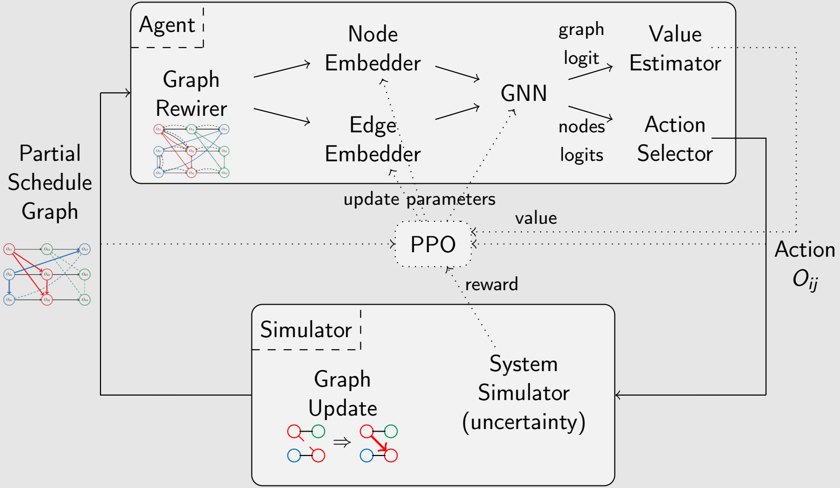
Node classification/regression

Use final values a_v^N

Graph classification/regression

- $POOL(\{a_v^N\})$ with $POOL \in \{MIN, MAX, SUM, AVG...\}$
- Virtual node g s.t. $N(g) = \{u \in G\}$

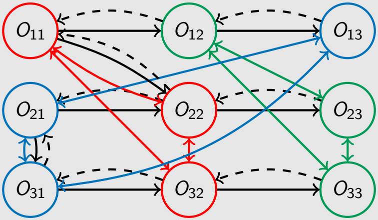
General Architecture



Algorithm

Init actor `// actor is ~ current policy` π_θ
Init critic `// critic is ~ value function estimator`
for $i = 1, 2, \ldots N$ **do**
  `// Collect dataset`
  Generate train instances
  Collect trials data $\mathcal{D}_i = ((s_t, a_t, r_t, s_{t+1}), \ldots)$ using current actor
  For each trial : sample makespan using system simulator ($=$ final cost)
  Compute advantages on trials using current critic
  Rewire graphs in trials data
  `// PPO update algorithm`
  repeat
    Sample a minibatch of n data points over shuffled collected data
    Update actor over the minibatch data towards advantage maximization
    Update critic by MSE regression
  until *max number of iterations or KL(current policy, update policy)* $>$ *threshold*
  Evaluate current policy (actor) on validation instances

Rewiring and Embedding



$\longrightarrow$ Jobs precedence ($\mathcal{C}$)
+ scheduling precedence choices

$\dashrightarrow$ Backward jobs precedence
+ backward scheduling prec. choices

$\leftrightarrow$ $\leftrightarrow$ $\leftrightarrow$ } Machines conflicts

Precedences

jobs definition + reverse (!)

Conflicts

Both directions

Nodes: MLP embedding

scheduled, selectable, duration, completion time (min max mode)

Edges: discrete embedding

type: precedence, reverse precedence, conflict

Some details

Sparse reward

Makespan evaluated on complete schedules

Action selection

- One to one matching between nodes and actions
  $\rightarrow$ action selection boils down to node regression
- final action logits = [node logit || graph logit]

Message Passing : Gatv2 + edge attributes (allows soft rewiring(!))

$$e_{ij}^{(l)} = a^{T^{(l)}} LeakyReLU(W_{src}^{(l)} h_i^{(l)} + W_{dst}^{(l)} h_j^{(l)} + W_{edge}^{(l)} f_{ij})$$

$$\alpha_{ij}^{(l)} = softmax_i(e_{ij}^{(l)}) = \frac{exp(e_{ij}^{(l)})}{\sum_{j' \in N_i} exp(e_{ij'}^{(l)})}$$

$$h_i^{(l+1)} = \sum_{j \in N(i)} \alpha_{ij}^{(l)} W_{update}^{(l)} h_j^{(l)}$$

Generalization

Setup

- Generate taillard instances of different sizes
- Learn on given fixed size
- Test on other problems, different sizes

	Deterministic			Stochastic		
Evaluation	W-6x6	W-10x10	W-15x15	W-6x6	W-10x10	W-15x15
6×6	**508**	521	521	**700**	714	715
10×10	927	**890**	915	1269	**1217**	1232
15×15	1557	**1388**	1392	2297	**1889**	**1889**
20×15	1798	**1583**	1622	2585	**2181**	2188
20×20	2314	1959	**1888**	3632	2643	**2608**

Table: Comparison of Wheatley wrt training instance sizes.

Learning on 10×10 $\rightarrow$ good compromise training time / performance

Deterministic

Compare W-10×10 to baselines, on different problems sizes

Evaluation	W-10x10	L2D	Best PDR	OR-Tools
6×6	521 (7.4)	571 (17.7)	545 (12.4)	**485 (0)**
10×10	890 (9.6)	993 (22.3)	948 (16.8)	**812 (0)**
15×15	1389 (17.2)	1501 (26.7)	1419 (19.8)	**1185 (0)**
20×15	1583 (16.9)	-	1642 (21.3)	**1354 (0)**
20×20	1959 (24.9)	2026 (29.2)	1870 (19.3)	**1568 (0)**
30×10	1829 (5.5)	-	1878 (8.9)	**1725 (0)**
30×15	2043 (14.5)	-	2092 (17.3)	**1784 (0)**
30×20	2377 (22.0)	-	2331 (19.7)	**1948 (0)**
50×15	3060 (8.3)	-	3079 (9.0)	**2825 (0)**
50×20	3322 (14.9)	-	3295 (14.0)	**2891 (0)**
60×10	3357 (1.7)	-	3376 (2.3)	**3301 (0)**
100×20	5886 (6.9)	-	5786 (5.1)	**5507 (0)**

Good generalization abilities

Stochastic

Compare W-10×10 learned on stochastic instances to baselines

Evaluation	W-10x10	Wd-10x10	MOPNR	CP-stoc	OR-Tools mode	OR-Tools real
6×6	714 (16.3)	817 (33.1)	699 (13.8)	**669 (9.0)**	728 (18.6)	*614 (0)*
10×10	1217 (21.5)	1464 (46.1)	1252 (25.0)	**1177 (17.5)**	1262 (25.9)	*1002 (0)*
15×15	1889 (29.3)	2406 (64.7)	1988 (36.1)	**1872 (28.1)**	1925 (31.8)	*1461 (0)*
20×15	**2181 (30.5)**	2729 (63.3)	2314 (38.5)	2222 (33.0)	2244 (34.3)	*1571 (0)*
20×20	2643 (36.4)	3511 (81.2)	2708 (40.0)	2631 (35.8)	**2619 (35.1)**	*1938 (0)*
30×10	**2425 (14.1)**	3511 (65.2)	2532 (19.1)	2476 (16.5)	2598 (22.2)	*2126 (0)*
30×15	**2792 (26.7)**	3251 (47.5)	2964 (34.5)	2892 (31.2)	2943 (33.5)	*2204 (0)*
30×20	3305 (36.9)	4186 (73.3)	3390 (40.4)	3355 (39.0)	**3299 (36.6)**	*2415 (0)*
50×15	**4043 (16.5)**	4413 (27.1)	4262 (22.8)	4239 (22.1)	4435 (27.7)	*3472 (0)*
50×20	**4520 (26.8)**	5351 (50.1)	4679 (31.2)	4682 (31.3)	4758 (33.4)	*3566 (0)*
60×10	**4315 (6.3)**	4475 (10.2)	4451 (9.6)	4442 (9.4)	4579 (12.8)	*4061 (0)*
100×20	**7591 (11.8)**	8377 (23.3)	7956 (17.1)	8203 (20.8)	8188 (20.5)	*6793 (0)*

Conclusion

GNNs are able to learn patterns on precedence graphs and generalize

With RL, stochasticity is naturally handled

Code is available at https://github.com/jolibrain/wheatley

Going forward: WIP

Extending to real-world factory scheduling

- Resources, options, more complex graphs : multi-mode RCPSP
- Opening calendars
- Due dates / tardiness
- Scaling up to thousands of tasks

Earth observation

- over subscribed, time windows : graph varies much more
- lexicographical utility $\rightarrow$ non chronological choice : STN

Beyond RL

- Replace RL with diffusion denoising on graphs
- can be seen as learning heuristic for local / large neighborhood search