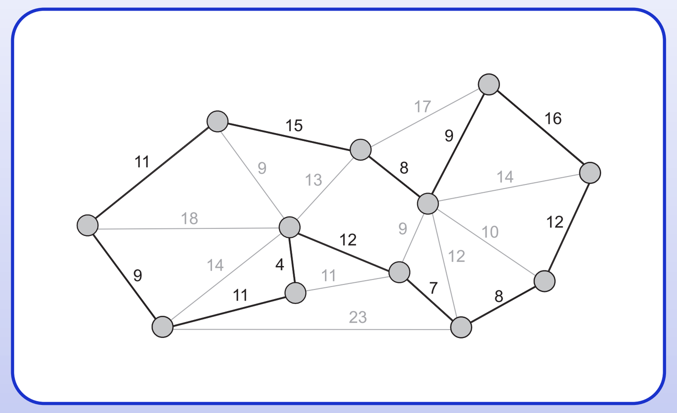
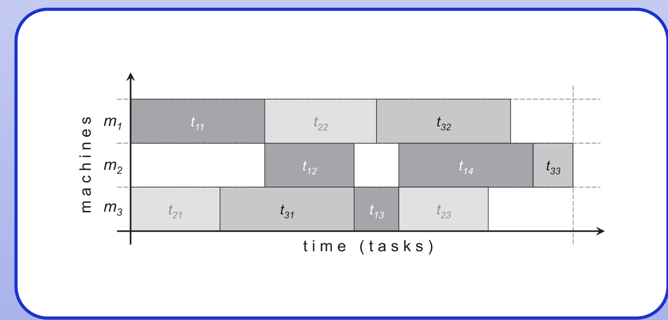
A Machine Learning Approach to Stochastic Resource Control

Balázs Csanád Csáji (balazs.csaji@uclouvain.be), Department of Mathematical Engineering, Université catholique de Louvain, Belgium

Motivations

- Resource allocation problems arise in many *real-world* domains, e.g.: production control, personnel / fleet / inventory management, scheduling of computer programs, controlling a cellular mobile network.
- They are usually *strongly NP-hard*, e.g., TSP and JSP, and they do not have any good polynomial time approximation algorithm, either.
- Moreover, real-world problems are often *large-scale*, there are significant *uncertainties* and the environment may even *change* over time.
- The extension of most classical algorithms, such as "branch & cut" or "constraint propagation", to the stochastic case is non-trivial.
- We suggest *machine learning* approaches to handle these problems.

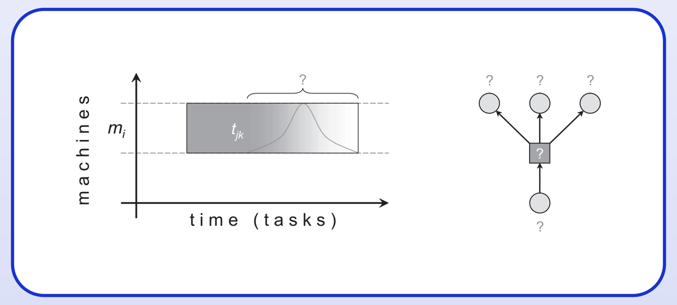


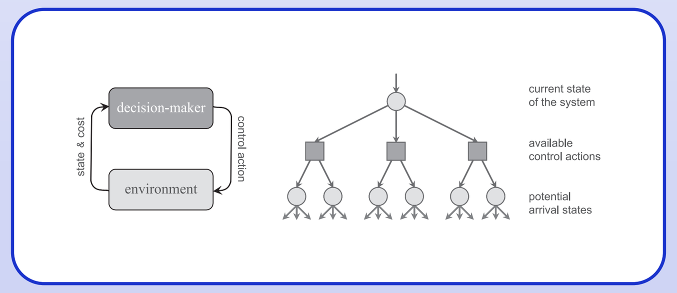


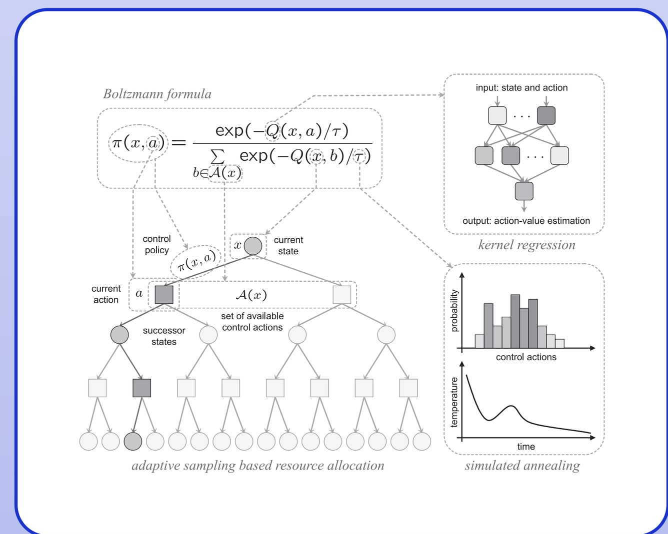
Markovian Resource Control

We showed that stochastic resource allocation problems can be reformulated as *Markov decision processes*, more precisely, as *stochastic shortest path* problems, with favorable properties, e.g., all policies are proper.

An *adaptive sampling* based method was applied, by which the optimal value function is iteratively approximated. The updates are performed at the end of each episode (a simulation of the resource allocation process).







Sampling and Regression Based Reinforcement Learning

1. **Initialize** Q_0, $\mathcal{L}_0$, τ and let $i = 1$.
2. **Repeat** (for each episode)
3. **Set** π_i to a soft and semi-greedy policy w.r.t. Q_{i-1}, e.g., $\pi_i(x,a) = \exp(-Q_{i-1}(x,a)/\tau) / \left[\sum_{b \in \mathcal{A}(x)} \exp(-Q_{i-1}(x,b)/\tau)\right]$.
4. **Simulate** a state-action trajectory from x_0 using policy π_i.
5. **For** $j = t_i$ **to** 1 (for each state-action pair in the episode) **do**
6. **Determine** the features of the state-action pair, $y_j^i = h(x_j^i, a_j^i)$.
7. **Compute** the new action-value estimation for x_j^i and a_j^i, e.g., $z_j^i = (1-\gamma_i) Q_{i-1}(x_j^i, a_j^i) + \gamma_i \left[g(x_j^i, a_j^i) + \alpha \min_{b \in \mathcal{A}(x_{j+1}^i)} Q_{i-1}(x_{j+1}^i, b)\right]$.
8. **End loop** (end of state-action processing)
9. **Update** sample set $\mathcal{L}_{i-1}$ with $\left\{\langle y_j^i, z_j^i \rangle : j = 1, \dots, t_i\right\}$, the result is $\mathcal{L}_i$.
10. **Calculate** Q_i by fitting a smooth regression function to the sample of $\mathcal{L}_i$.
11. **Increase** the episode number, i, and **decrease** the temperature, τ.
12. **Until** some terminating conditions are met, e.g., i reaches a limit or the estimated approximation error to Q^* gets sufficiently small.

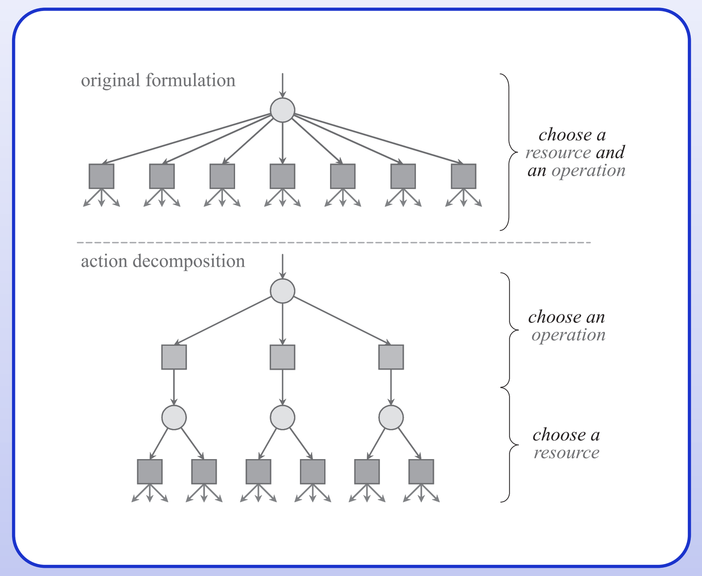
Output: the action-value function Q_i (or $\pi(Q_i)$, e.g., the greedy policy w.r.t. Q_i).

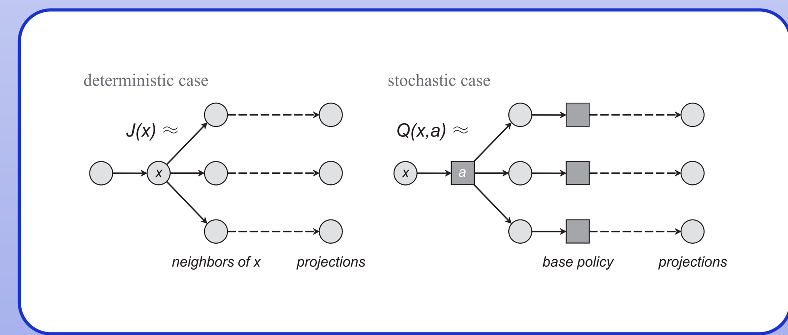
The notations of the pseudo-code are as follows: variable i is the episode number, t_i is the length of episode i and j is a parameter for time-steps inside an episode. The Boltzmann temperature is denoted by τ, π_i is the control policy applied in episode i and x_0 is the initial state. State x_j^i and action a_j^i correspond to step j in episode i. Function h computes features for state-action pairs while γ_i denotes learning rates. Finally, $\mathcal{L}_i$ denotes the regression sample and Q_i is the fitted (state-action) value function in episode i.

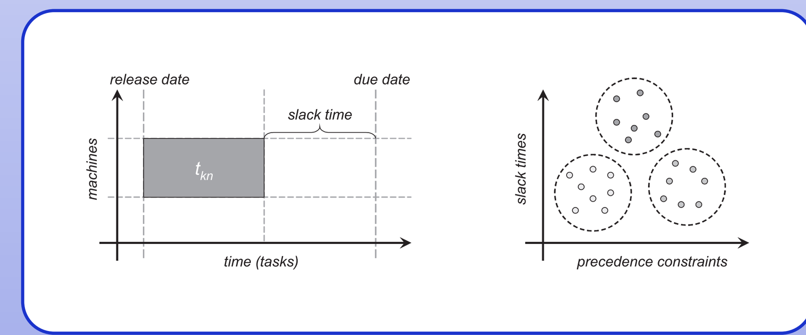
Additional Improvements

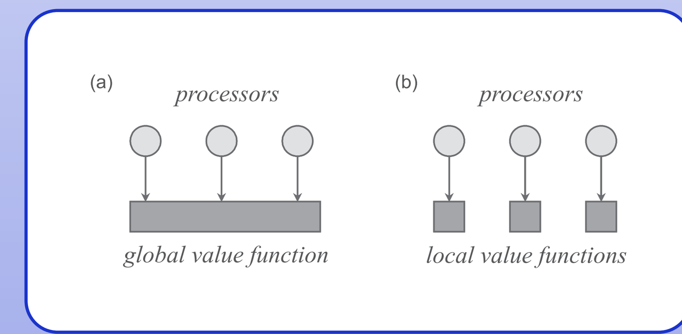
We applied several additional improvements to the core solution method:

- *Action space decomposition*: in order to decrease the available actions in the states, the action space was decomposed (figure on the right side).
- *Rollout methods*: to guide the initial exploration and to gather samples for the regression, limited-lookahead rollout algorithms were applied.
- *Clustering*: in order to divide the problem into several smaller subproblems, we clustered the tasks according to their expected slack times.
- *Distributed sampling*: we can exploit having more than one processors.









Experiments & Conclusions

The method has proven to be very efficient on benchmark and industrial problems.

- It outperformed most previous algorithms on hard job-shop problems (top).
- It showed excellent performance on a simulated industrial problem (middle).
- Clustering not only resulted in speedup but also in performance gains (down).

Advantages over previous approaches:

- It is *general*, since it is applicable to a large class of resource control problems.
- It takes *uncertainties* into account.
- The method can also quickly *adapt* to disturbances and environmental changes.
- There are theoretical *guarantees* of approximating the global optimal solution.
- It *scales-well* with the size of the problem without dramatic performance losses.
- It can use *domain specific* knowledge, as well (e.g., in the rollouts or explorations).
- It is an iterative, *any-time* solution.

Performance on benchmark datasets of hard flexible job-shop problems:

benchmark		1000 iterations		5000 iterations		10 000 iterations	
dataset	flexib	avg err	std dev	avg err	std dev	avg err	std dev
sdata	1.0	8.54 %	5.02 %	5.69 %	4.61 %	3.57 %	4.43 %
edata	1.2	12.37 %	8.26 %	8.03 %	6.12 %	5.26 %	4.92 %
rdata	2.0	16.14 %	7.98 %	11.41 %	7.37 %	7.14 %	5.38 %
vdata	5.0	10.18 %	5.91 %	7.73 %	4.73 %	3.49 %	3.56 %
average	2.3	11.81 %	6.79 %	8.21 %	5.70 %	4.86 %	4.57 %

Performance on industry-related problems (simulation of a real factory):

configuration		1000 iterations		5000 iterations		10 000 iterations	
machs	tasks	avg err	std dev	avg err	std dev	avg err	std dev
6	30	4.01 %	2.24 %	3.03 %	1.92 %	2.12 %	1.85 %
16	140	4.26 %	2.32 %	3.28 %	2.12 %	2.45 %	1.98 %
25	280	7.05 %	2.55 %	4.14 %	2.16 %	3.61 %	2.06 %
30	560	7.56 %	3.56 %	5.96 %	2.47 %	4.57 %	2.12 %
50	2000	8.69 %	7.11 %	7.24 %	5.08 %	6.04 %	4.53 %
100	10000	15.07 %	11.89 %	10.31%	7.97 %	9.11 %	7.58 %

Speedup and performance relative to the number of tasks in a cluster:

configuration		performance after 10 000 iterations per cluster				
clusters	tasks	late jobs	avg error	std dev	speed	speedup
1	1000	28.1	6.88 %	2.38 %	423 s	1.00 ×
5	200	22.7	5.95 %	2.05 %	275 s	1.54 ×
10	100	20.3	4.13 %	1.61 %	189 s	2.24 ×
20	50	13.9	3.02 %	1.54 %	104 s	3.28 ×
30	33	14.4	3.15 %	1.51 %	67 s	6.31 ×
40	25	16.2	3.61 %	1.45 %	49 s	8.63 ×
50	20	18.7	4.03 %	1.43 %	36 s	11.65 ×