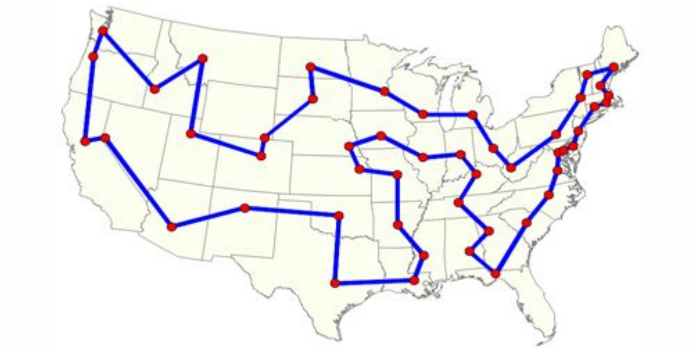
Combinatorial Optimization by Neural Reinforcement Learning

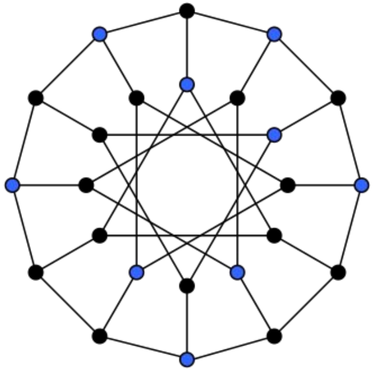
By Lei Zhang

Combinatorial optimization

- Finding an optimal solution from a finite set of possible solutions
  - In operations research, applied math, computer science
  - Feasible solutions are discrete, often evolves graphical structures
  - Many are NP-hard



Travelling Salesman Problem (TSP)



Maximum independent set (MIS)

Machine learning for combinatorial optimization literature

- Routing problems: TSP, Vehicle routing problem(VRP), etc.
  - Vinyals et al., *Pointer networks*, NIPS 2015
    - TSP, Pointer network with **supervised learning,** near optimal solution for 2D TSP up to 50 nodes
  - Bello et al., *Neural combinatorial optimization with reinforcement learning*, ICLR 2017
    - TSP: pointer network, and **RL** (policy gradient),near optimal up to 100 nodes
  - Nazari et al, *Reinforcement Learning for solving the vehicle routing problem*, NeurIPS 2018
    - VRP: RNN, policy gradient
  - Kool et al, *Attention, learn to solve routing problem!* ICLR 2019
    - TSP & VPR, Attention network, policy gradient with baseline
- Graph theoretical problems: Maximum independent set, minimum vertex cover, maximum cut, etc.
  - Dai et al., *Learning combinatorial optimization algorithms over **graphs***, NeurIPS 2017
    - Structure2vec, greedy algorithm, DQN
  - Li et al., Combinatorial Optimization with **graph convolutional networks** and guided tree search, NeurIPS 2018
  - Abe et al., *Solving NP-hard problems on graphs by **reinforcement learning** without domain knowledge*
    - Graph Isomorphism Networks and the Monte-Carlo Tree Search
- A survey by Benjio, et al.
  - Machine learning for combinatorial optimization: a methodological tour d'horizon, Nov 2018

Process Flexibility Design Problem

Stochastic Mixed Integer Linear Program

Goal: design a flexibility structure to maximize expected profit

$$\max_{f \in F} E_{d \sim D}[P(f, d)]$$

- F: set of feasible solutions -- bipartite graph
- D: demand distribution
- P(f, d): a function to calculate maximum profit -- a linear programing problem

Applications of process flexibility

Manufacturing

Health care

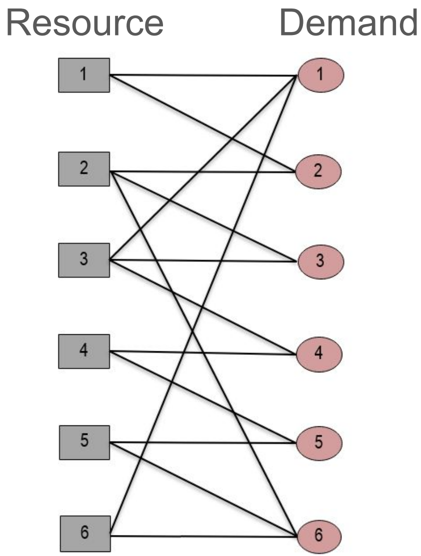
E-commerce

Delivery logistics

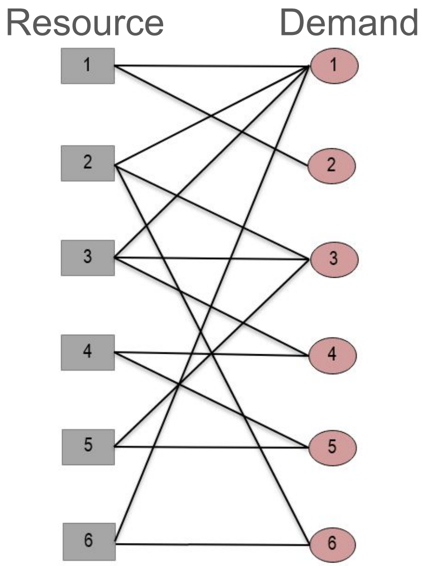
Service operations

Designing flexibility structure

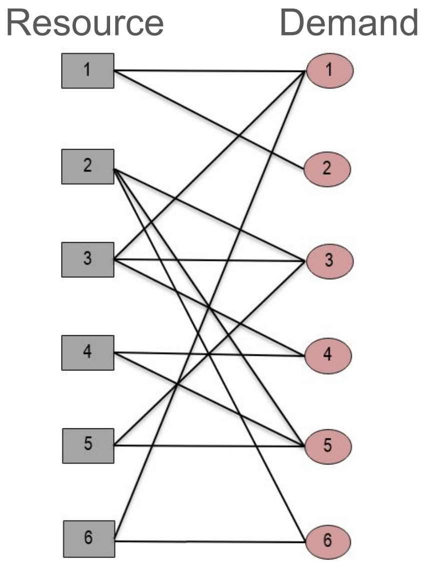
How to design a structure with 15 arcs?



Design A



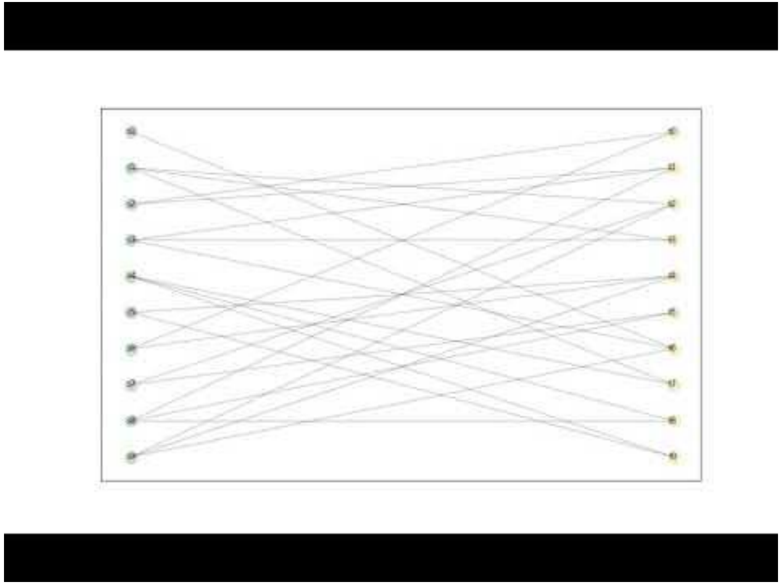
Design B



Design C

The neural reinforcement learning approach

- Generating structures like playing games
  - Starting from zero arcs, at each step play an action (add/remove one arc), until target arc # is reached
- State: adjacency matrix
- Action: the arc to be **added/removed**
- Reward:
  - If done: average profits of **50 simulation instances**
  - Else: 0
- Algorithm: Proximal Policy Gradient (PPO)



Proximal Policy Gradient

Take the biggest possible improvement step on a policy without stepping so far which leads to performance collapse, by penalizing KL-divergence of pi_old and pi_new or clipping

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Clipping function

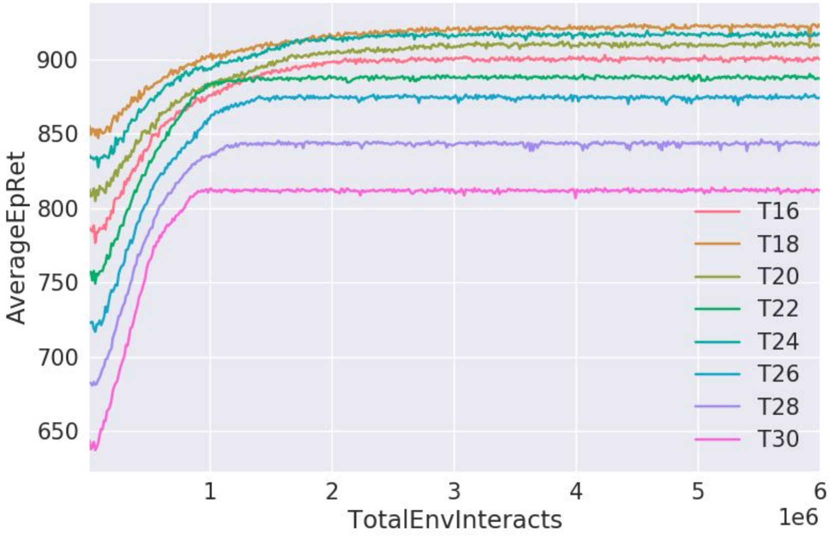
$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

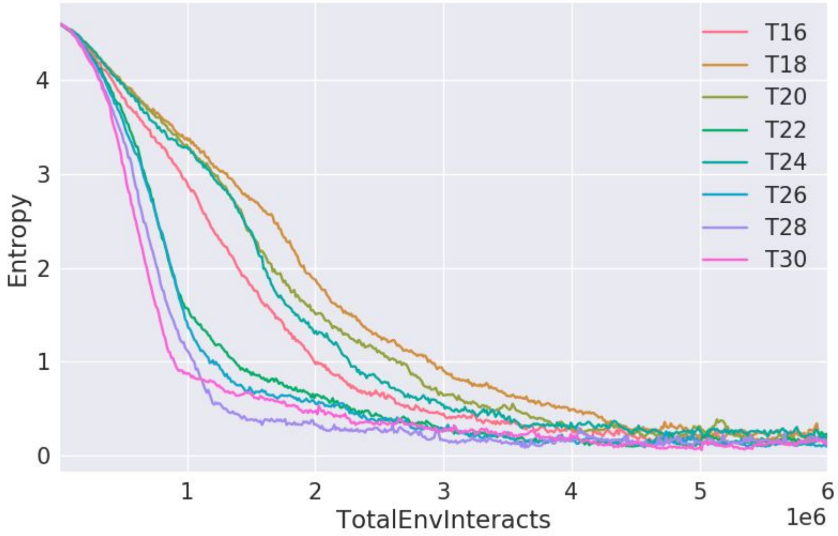
Advantage function estimation

$$\begin{aligned}
\hat{A}_t^{(1)} &:= \delta_t^V &&= -V(s_t) + r_t + \gamma V(s_{t+1}) \\
\hat{A}_t^{(2)} &:= \delta_t^V + \gamma \delta_{t+1}^V &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\
\hat{A}_t^{(3)} &:= \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})
\end{aligned}$$

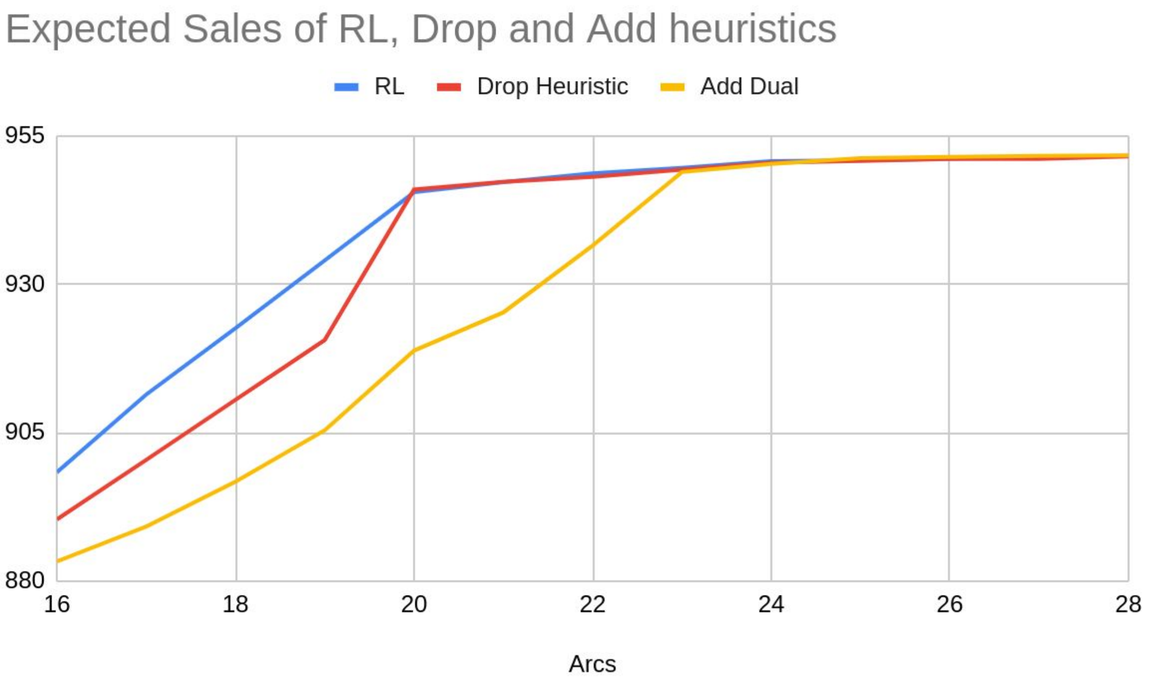
$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$$

Training results of 10x10 flexibility environment





Performance comparison of RL and heuristics



Main findings

- First attempt to apply neural RL to stochastic combinatorial optimization problems
  - Stochasticity plays to RL’s advantage
- Game play
  - More sample efficient, and larger learning capacity
  - Others: greedy, or one shot solution
- Extensible to move variations of the same problem
- Takes a long time to develop and tune parameters
  - Justifiable if the gain is high

Limitations and next steps

- Model is not extensible
  - Graph neural net
  - RNN
- Can also try Monte Carlo Tree Search