Sequential Stochastic Combinatorial Optimization Using Hierarchical Reinforcement Learning

ICLR | UCLA | 香 港 大 學 THE UNIVERSITY OF HONG KONG

Xinsong Feng, Zihan Yu, Yanhai Xiong, Haipeng Chen

WILLIAM & MARY

arxiv GitHub repo

SSCO

What is SSCO?

- Sequential stochastic combinatorial optimization
- Two-stage decision-making:
  - Allocate budgets across multiple time steps
  - Sequentially select optimal subsets of nodes to maximize cumulative rewards

Problem Formulation

$$\max_{\substack{K_1,K_2,\ldots,K_T,\\ S_1,S_2,\ldots,S_T}} \sum_{t=1}^{T} r_t(S_t)$$

$$\text{subject to} \quad \sum_{t=1}^{T} K_t \le K,$$

$$|S_t| \le K_t, \quad \forall t = 1,2,\ldots,T,$$

$$|K_t| \in \mathbb{N}, \quad \forall t = 1,2,\ldots,T,$$

$$S_t \subseteq V, \quad \forall t = 1,2,\ldots,T.$$

Challenges & Contributions

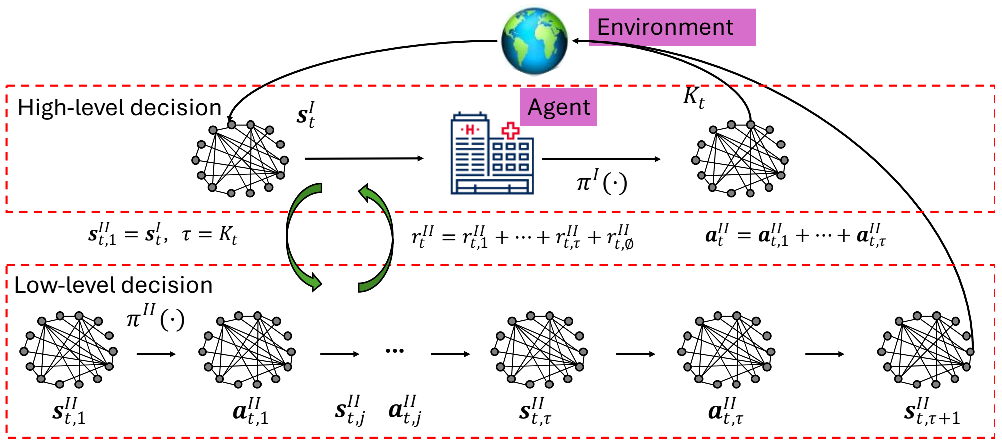
Challenges

- **Exponentially** many ways to split budget and pick node-sets over T steps
- **Stochastic** transitions and delayed feedback complicate reliable planning
- **Interdependent** high-level budgeting and low-level node selection on large graphs

Contributions

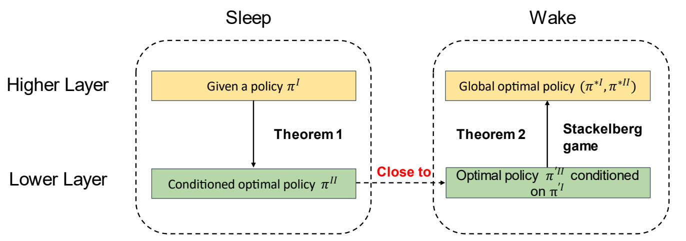
- We are the first to formally summarize and define the generic class of SSCO problems
- We design a novel HRL algorithm, Wake-Sleep Option, to solve the formulated SSCO

Hierarchical Markov Decision Process



Layer	State	Decision	Reward
High-level	(X, g)	Allocate budget $K_t \in [0, K_r]$	Cumulative gain from selected nodes
Low-level	(X, g)	Pick one node or null action	$r_{t,v}^{II} = \frac{m_{t,v}}{\sum_{u \in a_t^{II}} m_{t,u}} (r_t^I - r_{t,\emptyset}^{II})$

Wake-Sleep Option Framework



Layer-wise Method Selection

High-level → Monte Carlo

- Unbiased returns
- Stabilizes Q-network

Low-level → TD learning

- Fast, sample-efficient node-selection updates

Sleep stage: Freeze high-level Q-network and train low-level Q-network

Wake stage: Train both layers jointly

Experimental Results

Table 1: Experimental results for AIM, $n = 200$. All cases have p-values ≤ 0.05.

Method	$T, K = 10, 10$	$T, K = 10, 20$	$T, K = 10, 30$	$T, K = 20, 10$
WS-option	**76.00**	**118.56**	**129.06**	**80.95**
average-degree	67.92	104.54	122.50	72.18
average-score	74.36	116.10	128.29	80.31
normal-degree	69.28	101.50	109.39	63.47
normal-score	75.05	111.89	118.78	70.68
static-degree	70.02	105.25	122.37	70.57
static-score	74.81	118.13	128.01	71.68

Table 2: Experimental results for RP, $n = 100$. All cases have p-values ≤ 0.05.

Method	$T, K = 10, 10$	$T, K = 10, 20$	$T, K = 10, 30$	$T, K = 20, 10$
WS-option	**7.46**	**12.86**	**18.57**	**7.52**
greedy	6.29	12.02	15.68	6.73
GA	6.79	11.65	15.70	6.93

Table 3: Cumulative rewards when varying one layer's policy while the other layer remains fixed. All cases have p-values ≤ 0.05.

Setting	Lower layer fixed (using the learned policy)			
	WS-option	average	normal	static
$T, K = 10, 10$	**76.79**	71.45	75.27	74.85
$T, K = 10, 20$	**127.51**	126.35	120.26	125.46

Setting	Higher layer fixed (using the average policy)		
	WS-option	degree	score
$T, K = 10, 10$	**71.45**	62.55	69.15
$T, K = 10, 20$	**126.35**	118.75	125.02

Conclusion

- Hierarchical RL is powerful for breaking down hard combinatorial problems — we hope this idea inspires others to try similar decompositions in their work
- One challenge we didn't fully solve is scaling to very large graphs or datasets. There's still room to improve model efficiency and training stability