Neural Network Solvers for Combinatorial Optimization

Graph 11. Other CO Solvers (LLMs)

1

Outline of the CO Lectures

- Introduction to ML for Combinatorial Optimization (CO)
- Autoregressive (AR) CO Solvers
  - Reinforcement learning with RNN-based networks [Bello*, Pham*, et al., **ICLR 2017**]
  - Reinforcement learning with Transformer-based networks [W Kool et al., **ICLR 2019**]
- Non-autoregressive (NAR) CO Solvers
  - Reinforcement learning with DIMES [R Qiu*, Z Sun* & Y Yang, **NearIPS 2022**]
  - Supervised learning with DIFUSCO [Z Sun & Y Yang, **NearIPS 2023**]
- Pre-trained Large Language Models [C Yang et al., ICLR 2024]

4/4/2024 @Yiming Yang, 11-741 S24 Graph11 Other CO Solvers (LLMs) 2

2

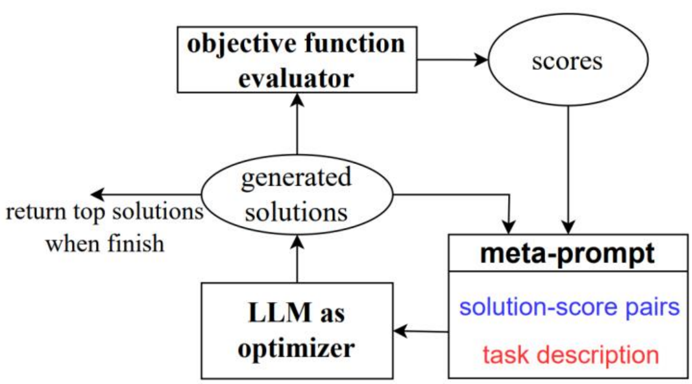
Large Language Models as Optimizers,

(Google DeepMind: C Yang*, C Chen*, et al., ICLR 2024)

- Key Idea
  - Proposing **OPRO** (Optimization by Prompting) as a generic optimizer for solving any problems (e.g., regression or TSP) described in natural language;
  - Each prompt contains a task description and a few solution/score pairs for previously solved problem instances;
  - Using a LLM (with the prompt) to generate solutions for each new problem instance;
  - Evaluate each new solution and adding the new solution/score pair to the prompt;
  - Repeating the above steps until a termination condition is met.

3

Learn to Solve Problems with OPRO



Yang, Chengrun, et al. "Large language models as optimizers." arXiv preprint arXiv:2309.03409 (2023).

4

Showcase of OPRO in Solving TSP

You are given a list of points with coordinates below: (0): (-4, 5), (1): (17, 76), (2): (-9, 0), (3): (-31, -86), (4): (53, -35), (5): (26, 91), (6): (65, -33), (7): (26, 86), (8): (-13, -70), (9): (13, 79), (10): (-73, -86), (11): (-45, 93), (12): (74, 24), (13): (67, -42), (14): (87, 51), (15): (83, 94), (16): (-7, 52), (17): (-89, 47), (18): (0, -38), (19): (61, 58).
Below are some previous traces and their lengths. The traces are arranged in descending order based on their lengths, where lower values are better.

<trace> 0,13,3,16,19,2,17,5,4,7,18,8,1,9,6,14,11,15,10,12 </trace>
length:
2254

<trace> 0,18,4,11,9,7,14,17,12,15,10,5,19,3,13,16,1,6,8,2 </trace>
length:
2017

<trace> 0,11,4,13,6,10,8,17,12,15,3,5,19,2,1,18,14,7,16,9 </trace>
length:
1953

<trace> 0,10,4,18,6,8,7,16,14,11,2,15,9,1,5,19,13,12,17,3 </trace>
length:
1840

Give me a new trace that is different from all traces above, and has a length lower than any of the above. The trace should traverse all points exactly once. The trace should start with <trace> and end with </trace>.

Input (Part I):
Node coordinates

Input (Part II):
Previous solutions

Instruction

Figure 18: An example of the meta-prompt for Traveling Salesman Problems with problem size $n = 20$. The blue text contains solution-score pairs; the orange text are meta-instructions.

5

Yang, Chengrun, et al. "Large language models as optimizers." arXiv preprint arXiv:2309.03409 (2023).

5

Evaluation Results

n	optimality gap (%)					# steps (# successes)		
	NN	FI	text-bison	gpt-3.5-turbo	gpt-4	text-bison	gpt-3.5-turbo	gpt-4
10	13.0 ± 1.3	3.2 ± 1.4	**0.0** ± 0.0	**0.0** ± 0.0	**0.0** ± 0.0	40.4 ± 5.6 **(5)**	46.8 ± 9.3 **(5)**	**9.6** ± 3.0 **(5)**
15	9.4 ± 3.7	1.2 ± 0.6	4.4 ± 1.3	1.2 ± 1.1	**0.2** ± 0.2	N/A (0)	202.0 ± 41.1 **(4)**	**58.5** ± 29.0 **(4)**
20	16.0± 3.9	**0.2**± 0.1	30.4 ± 10.6	4.4 ± 2.5	1.4 ± 0.6	N/A (0)	438.0 ± 0.0 (1)	**195.5** ± 127.6 **(2)**
50	19.7 ± 3.1	**9.8** ± 1.5	219.8 ± 13.7	133.0 ± 6.8	11.0 ± 2.6	N/A (0)	N/A (0)	N/A (0)

- Baseline NN (Nearest Neighbor Heuristic)
  - At each step, select the closest node from the current partial solution
- Baseline FI (Farthest Insertion)
  - At each step, add a new node that maximize the minimal insertion cost which is defined as

$$c(k) = \min_{i,j} d(i,k) + d(k,j) - d(i,j)$$

6

Concluding Remarks

- **Concept proving**
  - ORPO shows that LLMs with prompts in a loop can learn to optimize (mimicking gradient descent?)
- **Main limitations**
  - It cannot scale to large graphs due to the limited input length; similarly, it cannot handle a large training set of <solution, value> pairs.
- **Strong baselines are missing**
  - Comparison with DIMES and DIFUSCO on graphs with n=10000 nodes?
  - Comparison with classic exact solvers?
  - Comparison with LLMs for code generation?