How to Train Neural Networks for Classical Planning

Markus Fritzsche, PhD Student, Linköping University
Machine Reasoning Lab, AIICS, IDA
Supervisors: Prof. Hector Geffner, Simon Ståhlberg

LINKÖPINGS UNIVERSITET

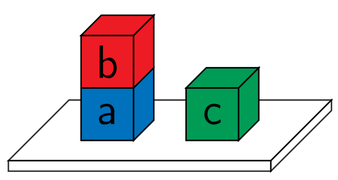
Background and Motivation

- Planning Problem: $P = \langle D, I \rangle$ with
  - Domain $D = \langle \mathcal{P}, \mathcal{A} \rangle$ consisting of
    - Predicate symbols $\mathcal{P}$
    - Action schema $\mathcal{A}$
  - Instance $I = \langle O, \mathcal{I}, \mathcal{G} \rangle$ consisting of
    - Object symbols O
    - Initial state $\mathcal{I}$
    - Goal state $\mathcal{G}$

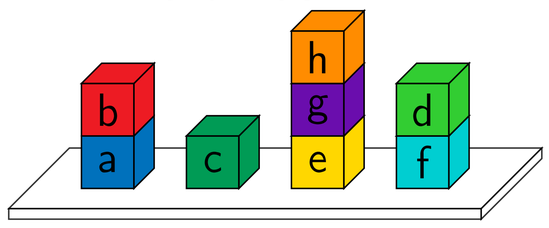
Example

- Stack all blocks on top of each other in alphabetical order.
- **Typically, solved using heuristic search.**

Easy Task



Hard Task



Motivation

- Idea: **Learn the optimal heuristic on small/simple problems, generalize to large/complex ones.**
- Challenge: Generalization is hard for neural networks when the test domain differs from the training domain.

Graph Representation of Planning States

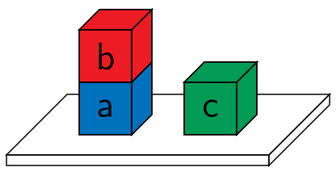
States as Set of Atoms

- State: $S = \{p(o_1, \ldots, o_k) \mid p \in \mathcal{P}, o_1, \ldots, o_k \in O\}$
- Goal: $\mathcal{G} = \{p(o_1, \ldots, o_k) \mid p \in \mathcal{P}, o_1, \ldots, o_k \in O\}$

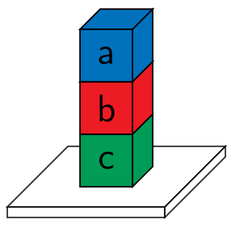
State Example

objects: a, b, c
atoms: $on(b, a)$
$clear(b)$
$clear(c)$
$on\_table(a)$
$on\_table(c)$
goal: $on(a, b)$
$on(b, c)$
actions: $pick\_up(b)$
$pick\_up(c)$

Initial State

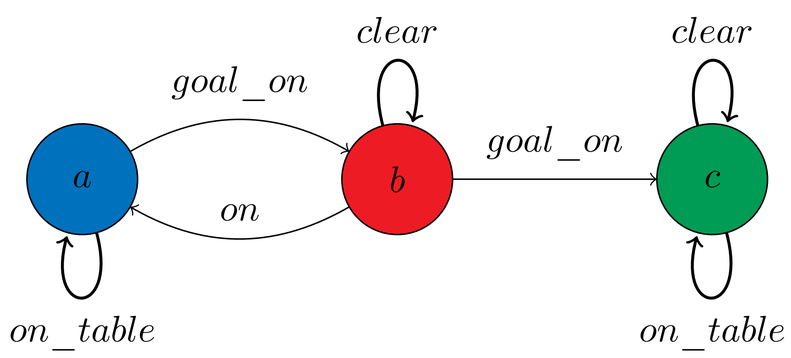


Goal State



State as Graph Representation

- Labeled graph: $G = (V, E, l)$ with
  - Nodes: $V = \{o \mid o \in O\}$
  - Edges: $E = \{(o_1, \ldots, o_k) \mid p(o_1, \ldots, o_k) \in S \cup \mathcal{G}\}$
  - Labeling function: $l(o_1, \ldots, o_k) = \{p \mid p(o_1, \ldots, o_k) \in S \cup \mathcal{G}\}$
- Example:



GNNs to Calculate the Optimal Heuristic

The Optimal Heuristic

- We want to calculate the length of the shortest path from the current state to the goal state.
- $h^* = \sum_{1 \leq i \leq l} c(a_i)$ where
  - a_i is an action
  - $(a_1, \ldots, a_l)$ is a plan that solves the task **with minimum cost**
  - c is a cost function
- i.e., we want to learn $f_\Theta(s) = h^*(s)$ where f is a neural network with parameters Θ.

Graph Neural Networks (GNNs)

- Node initialization: $h_v^0 = 0^d \quad \forall v \in V$ where d is the dimension of the node representation.
- Message: $m_{v_1,\ldots,v_k}^i = MSG_{l(v_1,\ldots,v_k)}(h_{v_1}^i, \ldots, h_{v_k}^i)$
- Aggregation: $agg\_m_v^i = AGG(\{m_{v_1,\ldots,v_k}^i \mid v \in (v_1, \ldots, v_k)\})$
- Combination: $h_v^{i+1} = COMB(h_v^i, agg\_m_v^i)$
- Readout: $h_G = READOUT(\{h_v^N \mid v \in V\})$ with N being the number of layers.

Challenges (My Research Focus)

Generalization

- Must learn features invariant to graph size, e.g., max aggregation vs sum aggregation.

Expressiveness

- GNNs are based on Weisfeiler-Lehman (WL) test, but this test **cannot distinguish all graph structures.**
- Hence, GNNs cannot learn the optimal heuristic for all tasks.

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM