UofSC

AI INSTITUTE #AIISC
UNIVERSITY OF SOUTH CAROLINA

Deep Reinforcement Learning and Heuristic Search

Forest Agostinelli
University of South Carolina

Students

Rojina Panta

Vedant Khandelwal

Misagh Soltani

Cale Workman

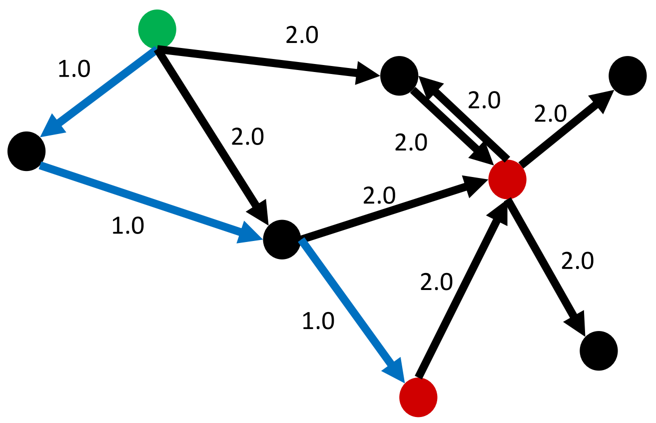
Christian Geils

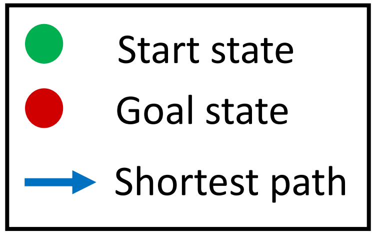
Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Pathfinding

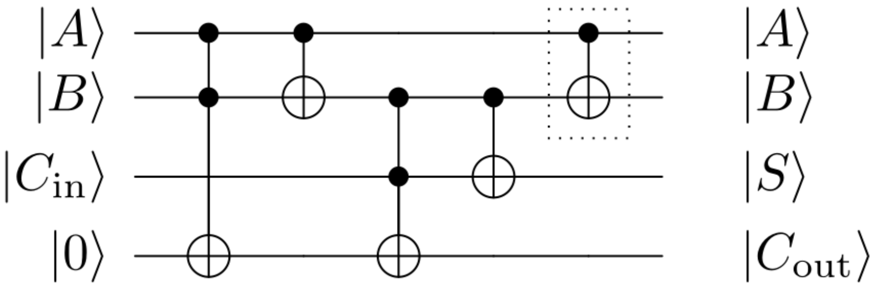
- The objective of pathfinding is to find a sequence of actions that forms a path between a given start state and a given goal
  - A goal is a set of states
  - Preference for minimum cost paths
- A pathfinding problem can be represented as a weighted directed graph where nodes represent states, edges represent actions that transition between states, and edge weights represent transition costs
  - The cost of a path is the sum of transition costs

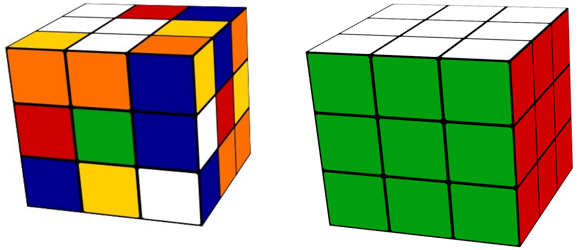




Pathfinding Domains

- Pathfinding problems can be found throughout mathematics, computing, and the natural sciences
  - Puzzle solving, chemical synthesis, quantum circuit synthesis, theorem proving, program synthesis, robotics





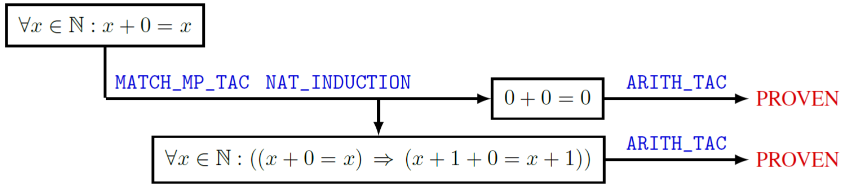
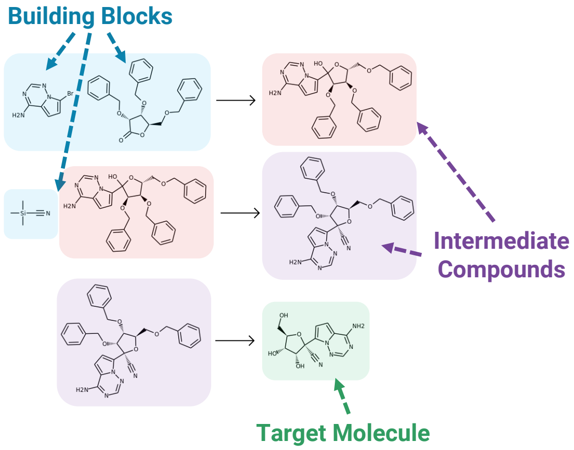
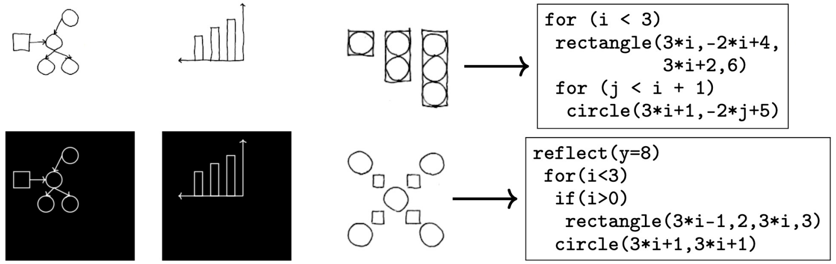


Figure 1: Formally proving $\forall x \in \mathbb{N} : x + 0 = x$.





Pathfinding Domain Definition

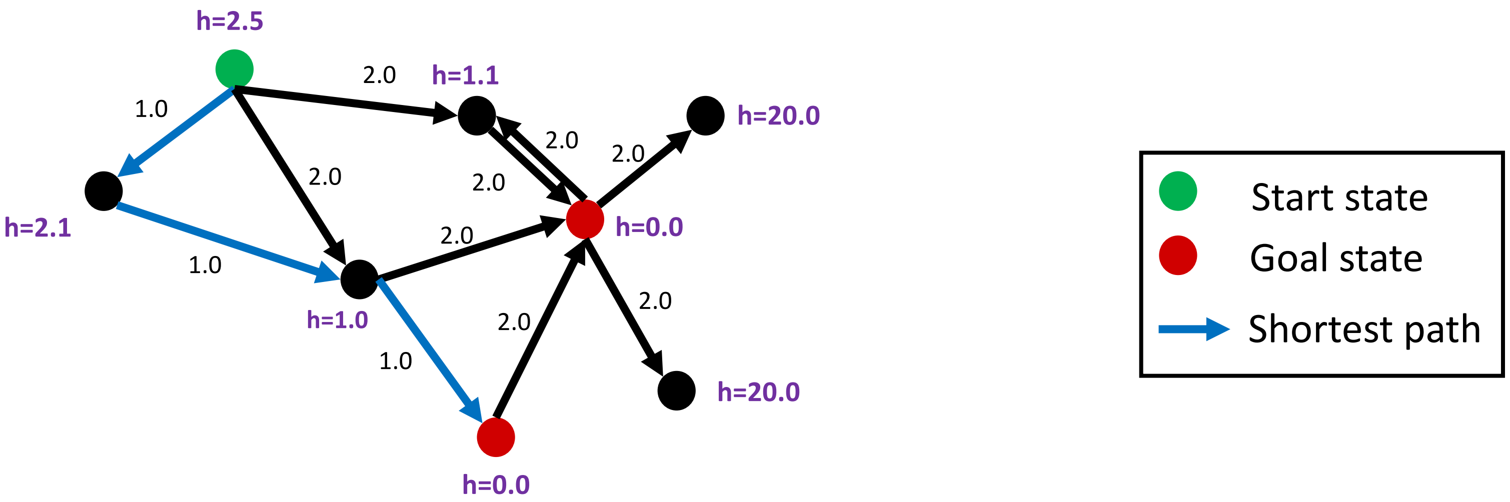
- The entire state space graph cannot be given to a pathfinding problem solver because the number of states in a pathfinding problem can be very large.
  - Rubik's cube: $\sim 10^{19}$
  - 48-puzzle: $\sim 10^{62}$
  - Organic chemistry: $\sim 10^{60}$ (exact number unknown)
- Assumptions on what is given
  - Action space
  - State transition function
  - Transition cost function
  - Goal specification language
  - Goal test function
- Objective: Create a domain independent algorithm
  - Input: Pathfinding domain definition, start state, goal specification
  - Output: Path to a goal state

Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Learned Heuristic Functions

- Heuristic function maps a state to an estimate of the cost of a shortest path from that state, also known as the cost-to-go

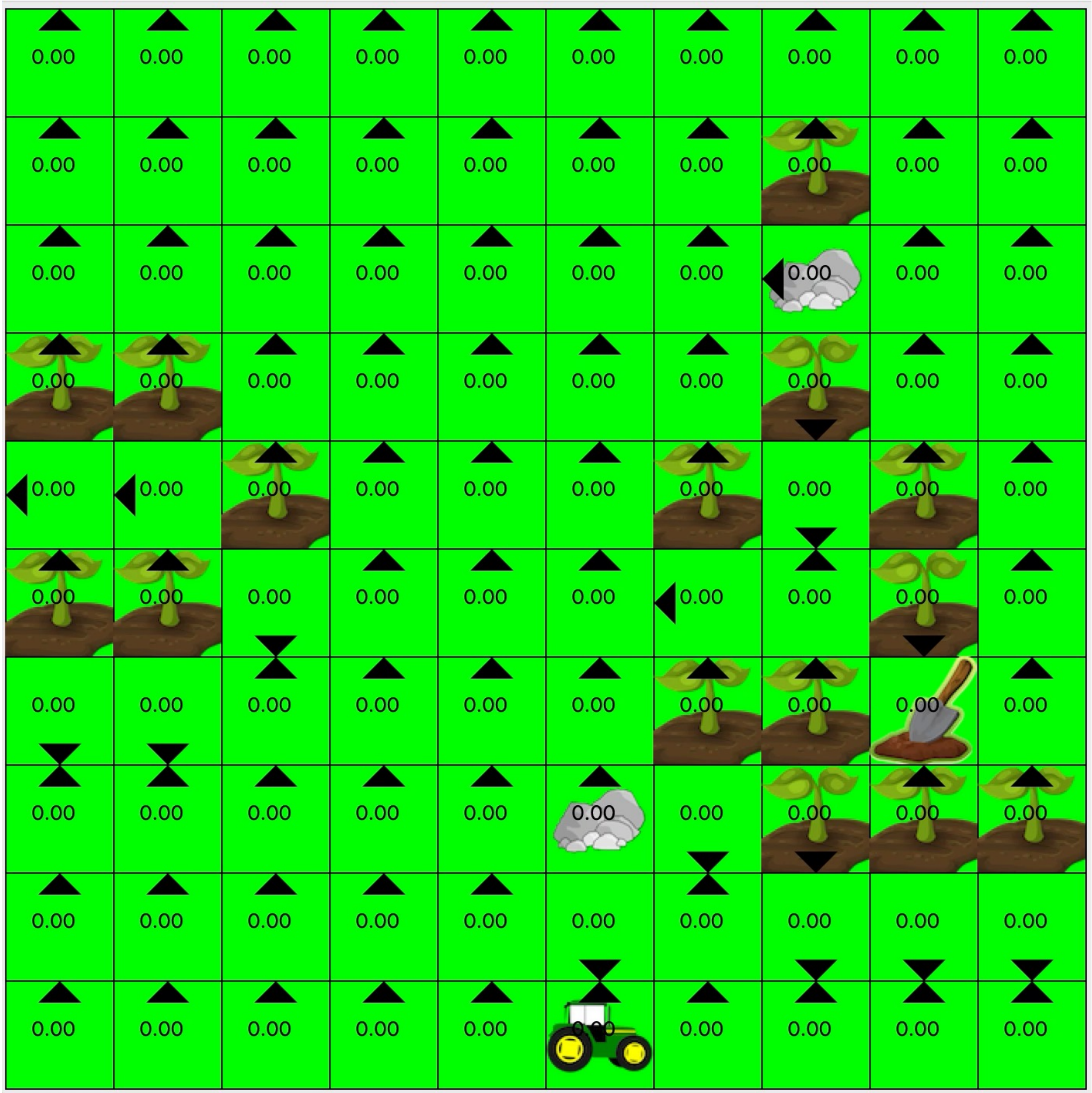


Value Iteration

- Value iteration is a dynamic programming algorithm and is a foundational algorithm in reinforcement learning
- In the context of pathfinding, value iteration is an algorithm for computing the cost-to-go of finding a shortest path for each state in the state space
- Tabular value iteration loops over all states and applies the following update until convergence (h stops changing)
  - $h(s) = \min_{a}(c^a(s) + h(T(s,a)))$
  - Guaranteed to converge to h^* in the tabular setting
- s: state
- a: action
- T: state transition function
- c^a: transition cost function

Value Iteration: Visualization

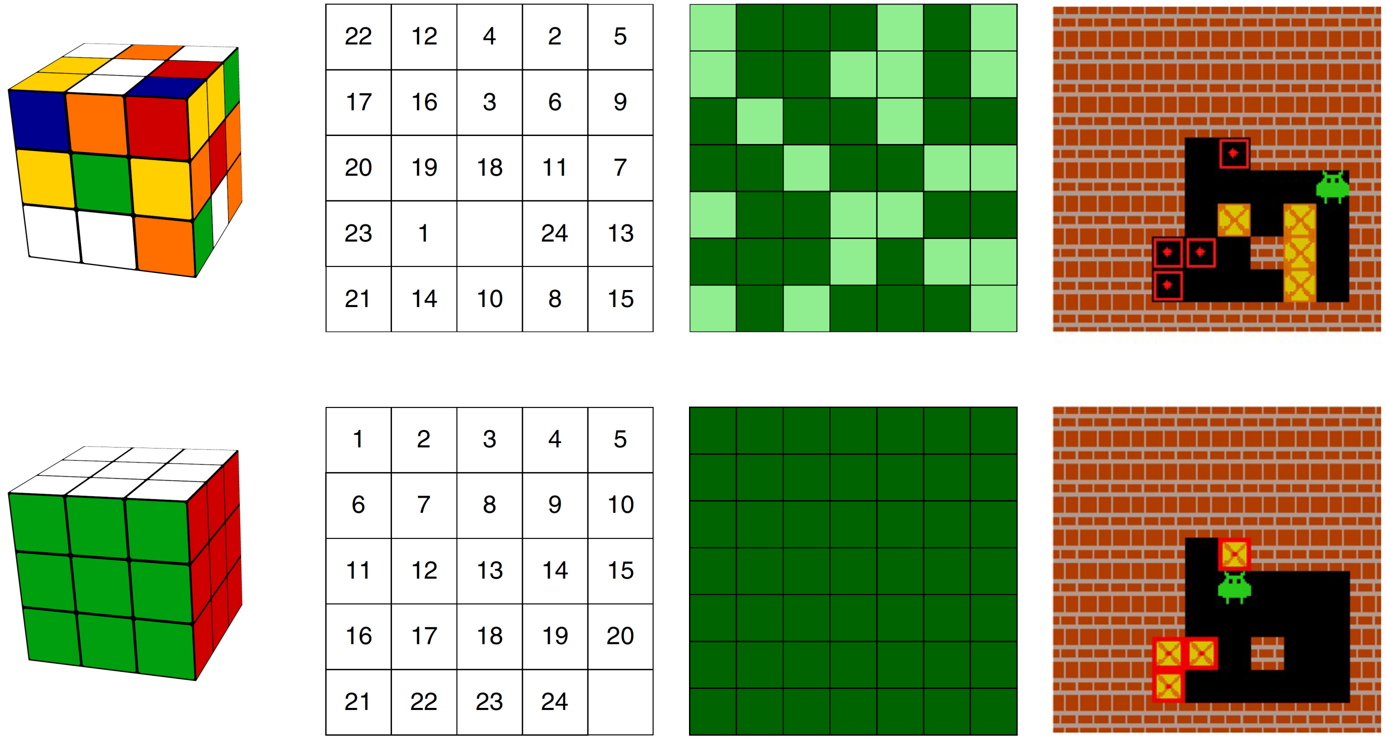
- Actions: up, down, left, right
- Transition costs
  - 1 if square is blank
  - 10 if square has a rock
  - 50 if square has a plant
- Goal: shovel
- Updates propagate outwards from the goal



Approximate Value Iteration

- As the state space grows, tabular value iteration becomes infeasible
- Approximate value iteration uses an approximation architecture to approximate the value iteration update
- When using a deep neural network as the approximation architecture, we refer to this as deep approximate value iteration (DAVI)
- The update is approximated using the following loss function
  - $L(\theta) = \left(\min_a\left(c^a(s) + h_{\theta^-}(T(s,a))\right) - h_\theta(s)\right)^2$
  - Target is set to zero if s is a terminal state
- s: state
- a: action
- T: state transition function
- c^a: transition cost function
- θ: parameters
- θ^-: parameters for target network
  - Is periodically updated to θ throughout training

Application to Puzzle Solving



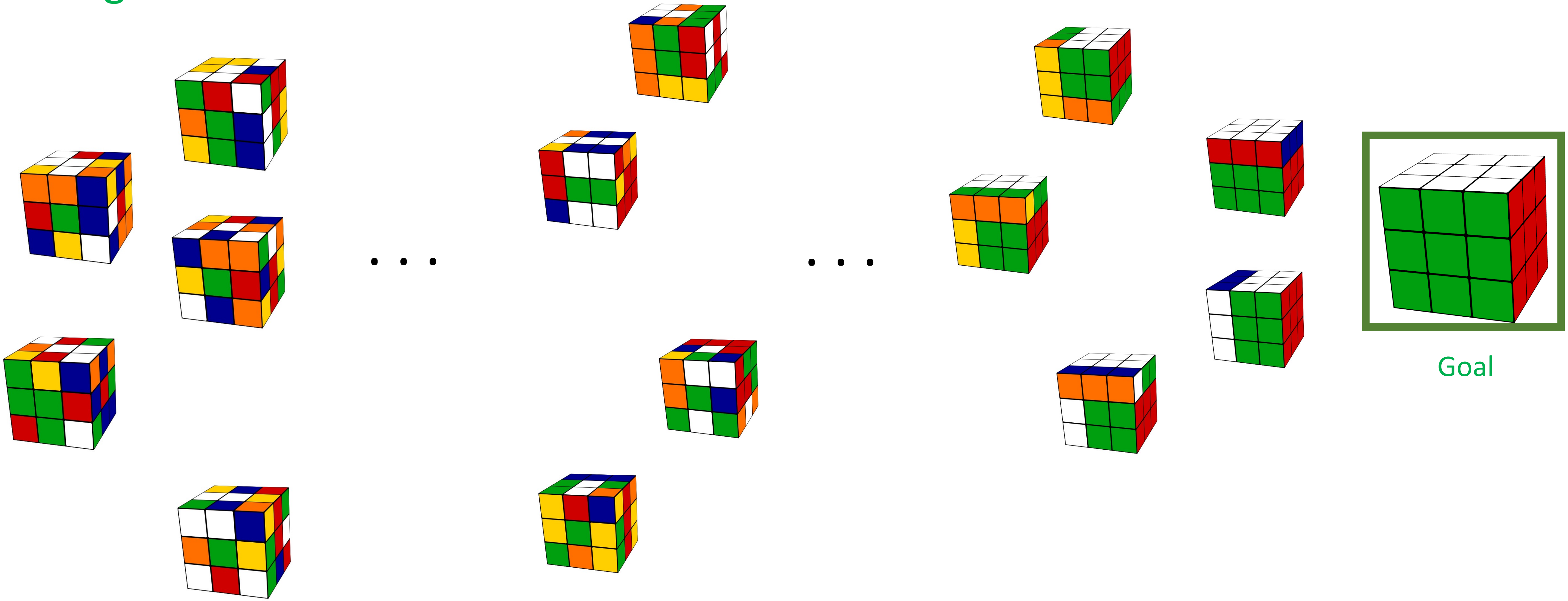
Rubik's cube   24 puzzle   Lights Out (7×7)   Sokoban

1. Rubik's Cube
2. 15-puzzle
3. 24-puzzle
4. 35-puzzle
5. 48-puzzle
6. Lights Out
7. Sokoban

Largest state space is 3.0×10^{62} (48-puzzle)

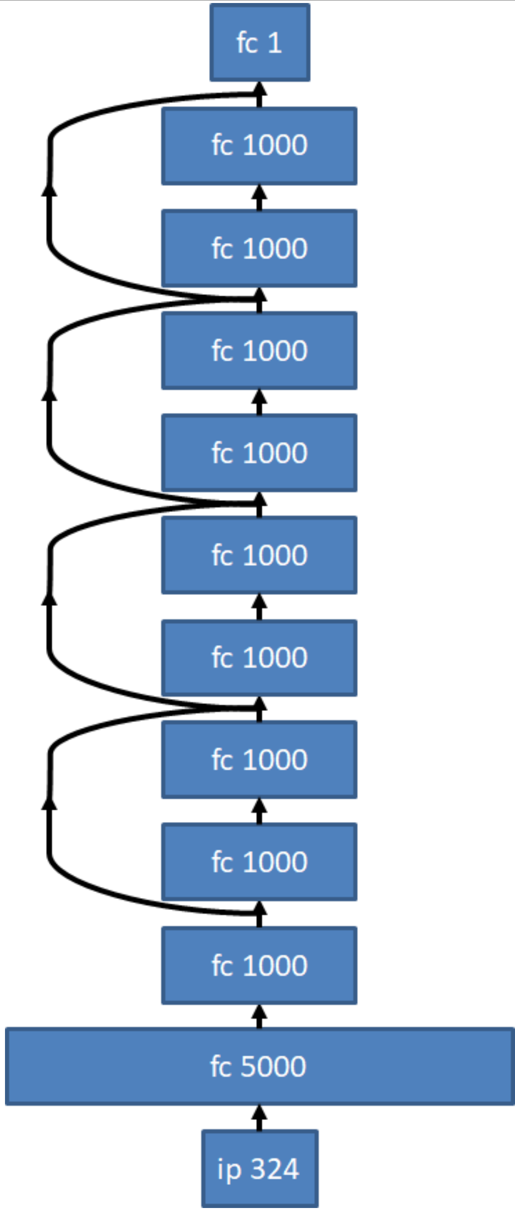
Generating States

- Prioritized sweeping: Generate training data by taking moves in reverse from the goal



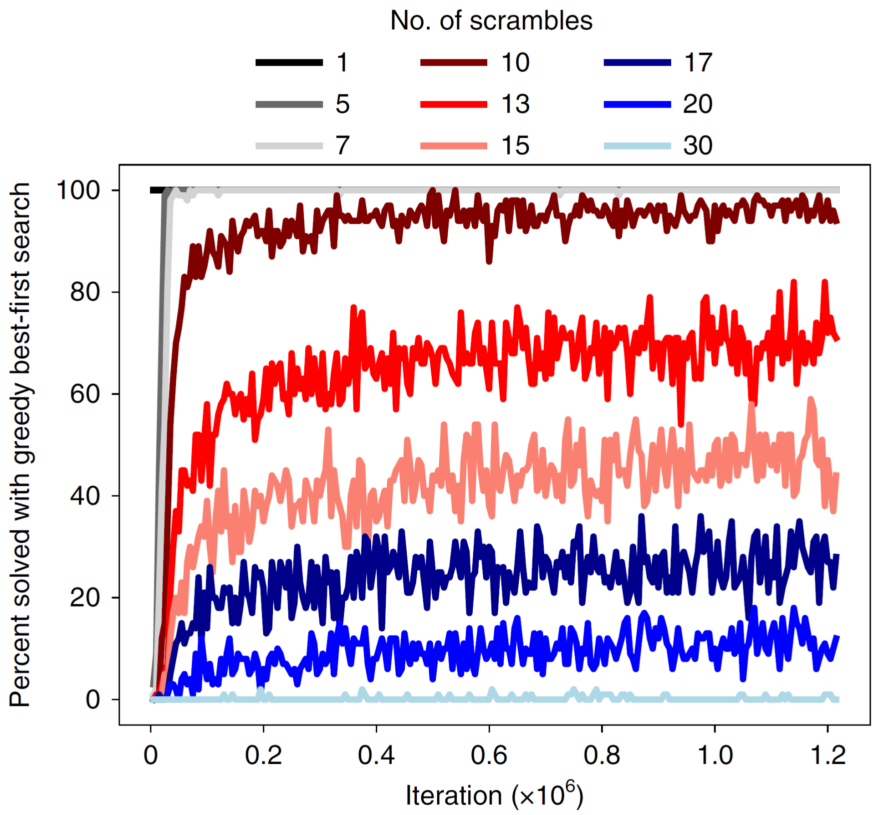
Training

- Deep neural network
  - Input layer -> Two fully connected layers -> Four residual blocks -> Linear output layer
  - Same type of architecture used for all puzzles
    - 24-puzzle has two more residual blocks
- Training
  - Batch size of 5,000
  - ~1,000,000 training iterations
  - Parameters for target network updated when loss goes below some target threshold
    - Future work updates based on greedy policy performance



Greedy Policy Performance

- Behave greedily with respect to the heuristic function
- $\pi(s) = \underset{a}{\operatorname{argmin}}(c^a(s) + h_\theta(T(s,a)))$
- Does not solve all states



Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Integration with A* Search

- Learned heuristic function can be used as a heuristic in A* search
- A* Search
  - Maintains a search tree where nodes are states and edges are actions
  - Initialized with a start node representing the start state
  - Expands nodes according to the priority
    - $f(n) = g(n) + h(n.s)$
    - $f(n)$: cost
    - $g(n)$: path cost (cost to get from start node to n)
    - $h(n.s)$: heuristic (estimated cost-to-go from $n.s$ to a closest goal state)
  - Terminates when a node associated with a goal state is selected for expansion
- Weighted A* Search
  - Decreasing the weight on the path cost may result in expanding fewer nodes while possibly increasing the length of paths found
  - $f(n) = \lambda * g(n) + h(n.s)$

Batch Weighted A* Search

- To take advantage of parallelism provided by GPUs, we can expand multiple nodes at once
- Guaranteed to be bounded suboptimal if
  - The heuristic function is admissible
  - If we terminate when
    - A node we expand from OPEN has a cost greater than or equal to the shortest path we have found so far
    - The number of children generated for that iteration is zero

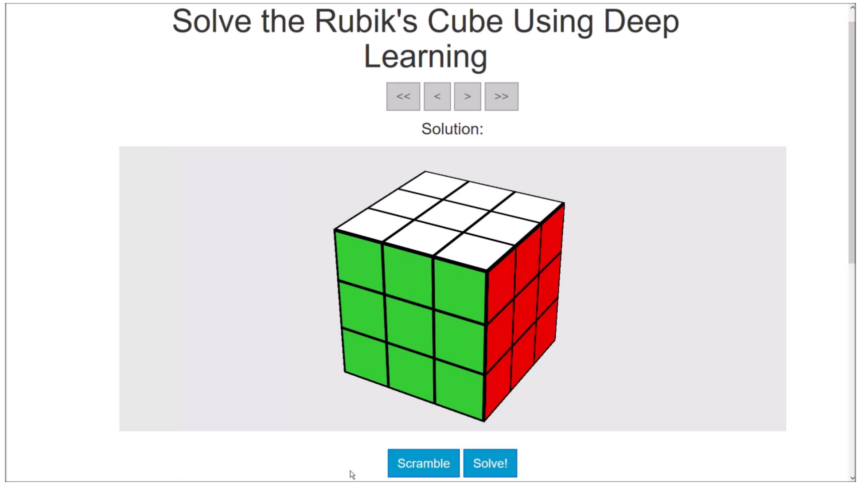
Algorithm 1 Batch Weighted A* Search (BWAS)

Input: $start$, DNN v_θ, batch size B, weight λ
OPEN $\leftarrow$ priority queue of nodes based on minimal f
CLOSED $\leftarrow$ maps states to their shortest discovered path costs
$UB, n_{UB} \leftarrow \infty$, NIL
$LB \leftarrow 0$
$n_{start} \leftarrow$ NODE($s = start, g = 0, p =$ NIL$, f = v_\theta(start)$)
PUSH n_{start} to OPEN
while not IS_EMPTY(OPEN) **do**
  generated $\leftarrow$ []
  while not IS_EMPTY(OPEN) and SIZE(generated) $< B$ **do**
    $n = (s, g, p, f) \leftarrow$ POP(OPEN)
    if IS_EMPTY (generated) **then**
      $LB \leftarrow \max(f, LB)$
    if IS_GOAL(s) **then**
      if $UB > g$ **then**
        $UB, n_{UB} \leftarrow g, n$
      continue loop
    for a in $|\mathcal{A}|$ **do**
      $s' \leftarrow A(s, a)$
      $g(s') \leftarrow g(s) + c^a(s)$
      if s' not in CLOSED or $g(s') <$ CLOSED[s'] **then**
        CLOSED[s'] $\leftarrow g(s')$
        APPEND(generated, $(s', g(s'), n)$)
  if $LB \geqslant \lambda \cdot UB$ **then**
    return PATH_TO_GOAL(n_{UB})
  generated_states $\leftarrow$ GET_STATES(generated)
  heuristics $\leftarrow v_\theta$(generated_states)
  for $0 \leqslant i \leqslant$ SIZE(generated) **do**
    $s, g, p \leftarrow$ generated[i]
    $h \leftarrow$ heuristics[i]
    $n_s \leftarrow$ NODE($s, g, p, f = \lambda \cdot g + h$)
    PUSH n_s to OPEN
return PATH_TO_GOAL(n_{UB})   *// failure if n_{UB} is NIL*

Agostinelli, Forest, et al. "Obtaining approximately admissible heuristic functions through deep reinforcement learning and A* search." *ICAPS PRL Workshop*. 2021.
Li, Tianhua, et al. "Optimal search with neural networks: Challenges and approaches." *Proceedings of the International Symposium on Combinatorial Search*. Vol. 15. No. 1. 2022.

DeepCubeA: Results

- When applied to seven different puzzles, it was able to solve all test instances and found a shortest path in the majority of verifiable cases
- http://deepcube.igb.uci.edu/

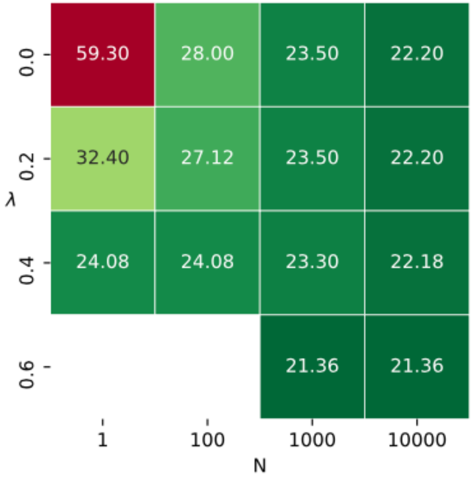


Puzzle	Solution Length	Percent Optimal	Time (seconds)
Rubik's Cube	21.50	60.3%	24.22
15-puzzle	52.03	99.4%	10.28
24-puzzle	89.49	96.98%	19.33
35-puzzle	124.64	N/A	28.45
48-puzzle	253.35	N/A	74.46
Lights Out	24.26	100.0%	3.27
Sokoban	32.88	N/A	2.35

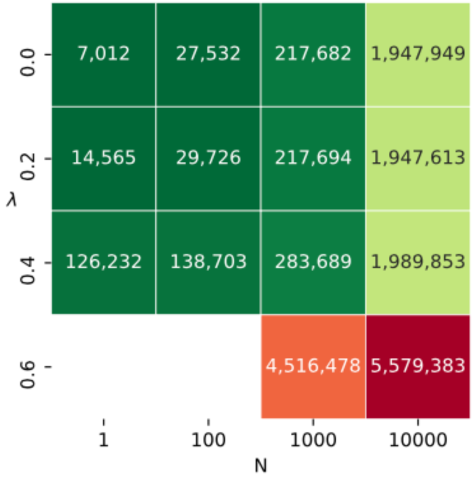
Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.

Effect of Batch and Weight

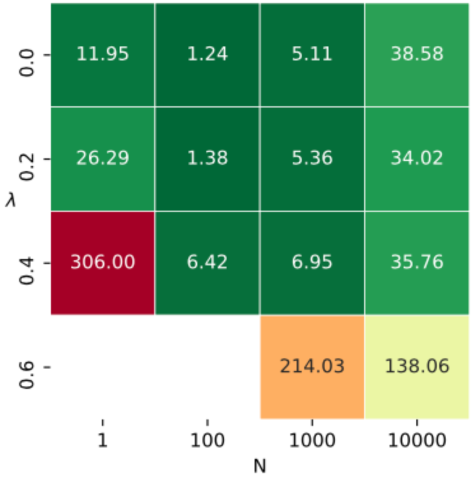
- Increasing the batch size decreases the path cost, increases the nodes/second
- Decreasing the weight generally leads to longer solutions but faster run times



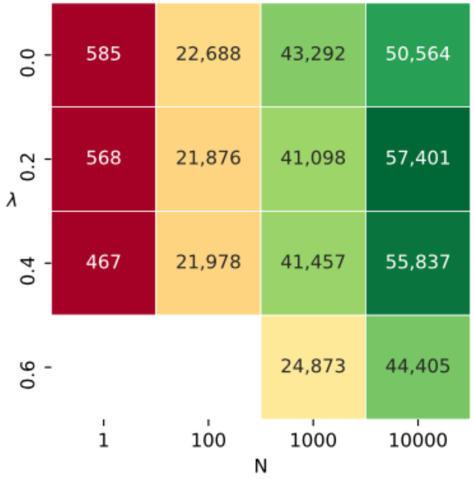
(a) Solution Length



(b) Nodes Generated



(c) Solve Time



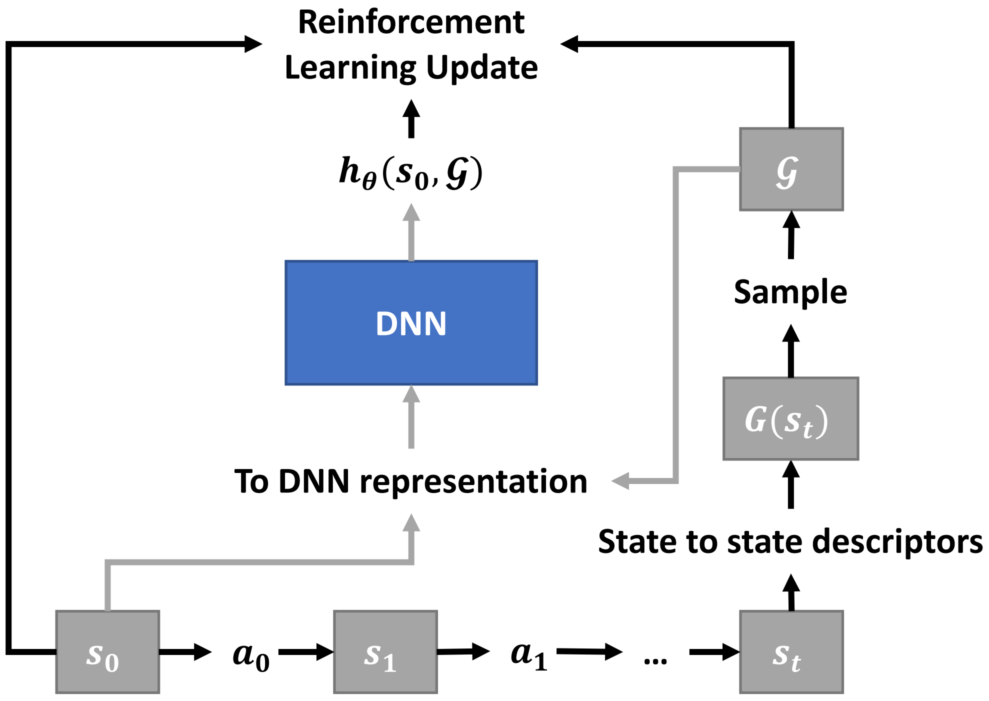
(d) Nodes/Second

Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Generalizing Over Goals

- In the previous work, the goal is predetermined
- Building on hindsight experience replay, we can generalize over goal states or sets of goal states
  - Generate a start state
  - Take a random walk whose length is somewhere between 0 and T
    - Future work could use artificial curiosity
  - Convert terminal state to a set of descriptors
  - Subsample to obtain a goal
  - Convert this representation into one suitable for the DNN
    - One-hot representation
    - Graph
    - Etc.
  - RL Update



Agostinelli, Forest, Rojina Panta, and Vedant Khandelwal. "Specifying Goals to Deep Neural Networks with Answer Set Programming." *ICAPS 2024*

Generalizing Over Goals: Training

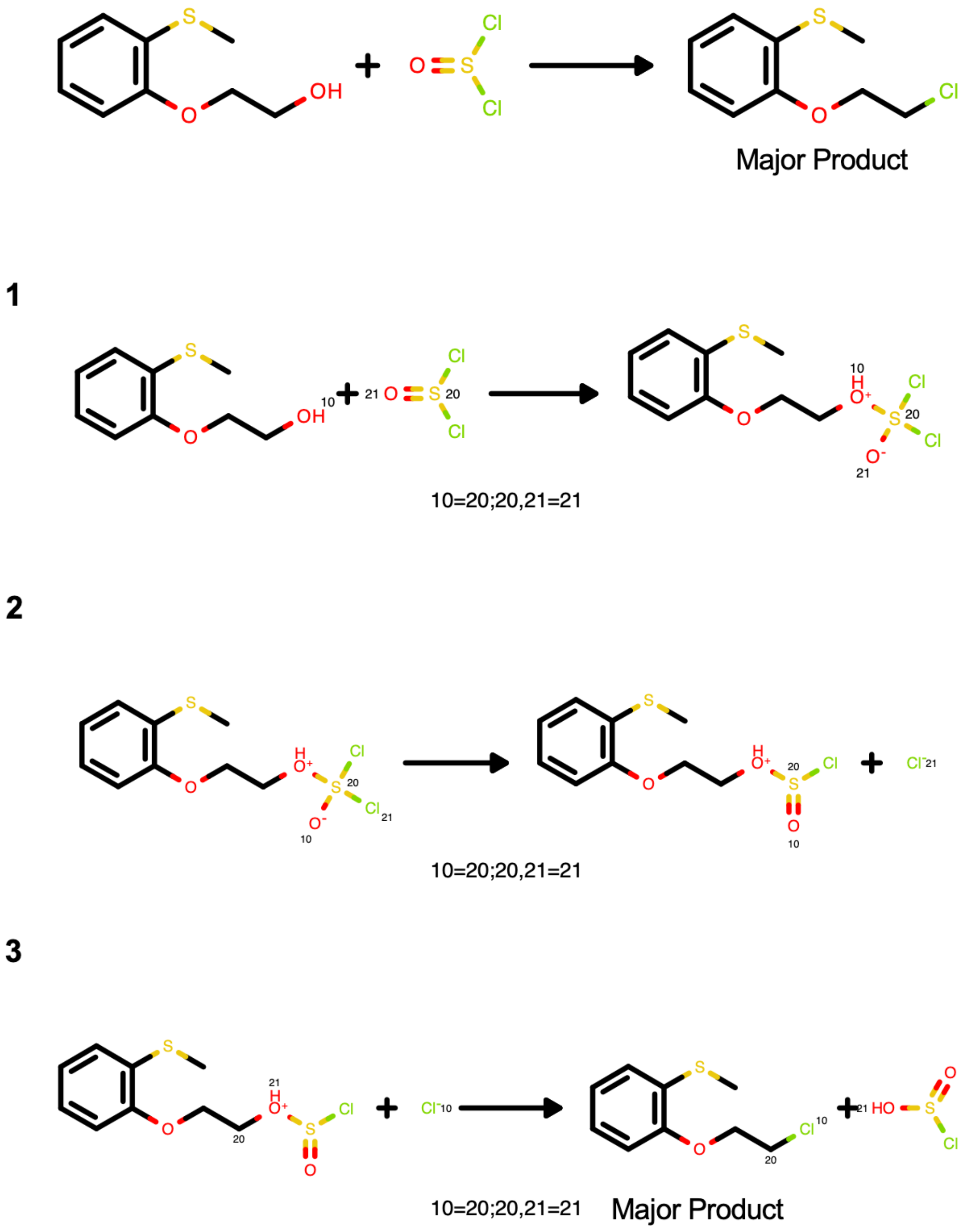
- $L(\theta) = \left(\min_a (c^a(s) + h_{\theta^-}(T(s,a),\mathcal{G})) - h_\theta(s,\mathcal{G})\right)^2$
- Given randomly generated start and goal pairs, additional data generated by following an epsilon-greedy policy
    - Can help identify depression regions
- Parameters for target network updated when the greedy policy improves
    - Tested every ~5,000 iterations

Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Reaction Mechanisms

- Chemical reactions are composed of smaller steps called reaction mechanisms
- Knowledge of the reaction mechanisms that compose a chemical reaction allows practitioners to
  - Validate reaction feasibility
  - Improve reaction efficiency
  - Predict reaction outcome under different conditions
- Most chemical reaction prediction methods skip reaction mechanisms and predict products directly from reactants



Reaction Mechanism Domain

- We create the state transition function using OrbChain, a model for reaction mechanism steps
  - Can take over a second to expand a state, limiting training data
- For simplicity, we assume all transition costs are 1
  - Future work will use negative log probabilities of reaction mechanism steps as transition costs
- We use extended-connectivity fingerprints to represent a molecule to the heuristic function
  - Future work will use a learned representation using graph neural networks
- We generate data using small molecules from the United States Patent and Trademark Office (USPTO) dataset of chemical reactions
  - Using random walks, we generate new molecules
- The heuristic function also takes a goal state as input
  - $L(\theta) = \left(\min_{a}\left(c^{a}(s) + h_{\theta^-}(T(s,a), s_g)\right) - h_{\theta}(s, s_g)\right)^2$

Results

- Generate test data by performing a random walk between 0 and 6 steps
- The learned heuristic function outperforms uniform cost search and A* search with the Tanimoto similarity metric

Step/s	Solver	Path Cost	% Solved	Nodes	Secs	Nodes/Sec
Steps=0	DeepCubeA	0.00	**100.00%**	3.09E+2	3.87	79.97
	Uniform Cost Search	0.00	**100.00%**	3.09E+2	4.61	67.13
	Tanimoto Similarity	0.00	**100.00%**	3.09E+2	3.71	83.42
Steps=1	DeepCubeA	1.00	**100.00%**	7.49E+2	9.70	77.26
	Uniform Cost Search	1.00	**100.00%**	4.26E+4	553.33	76.95
	Tanimoto Similarity	1.00	**100.00%**	3.13E+4	429.29	72.97
Steps=2	DeepCubeA	2.07	**100.00%**	1.63E+4	267.16	60.87
	Uniform Cost Search	1.67	20.00%	1.32E+5	1497.77	87.96
	Tanimoto Similarity	1.75	26.67%	1.10E+5	1229.10	89.13
Steps=3	DeepCubeA	2.77	86.67%	4.14E+4	578.88	71.54
	Uniform Cost Search	-	0.00%	-	-	-
	Tanimoto Similarity	-	0.00%	-	-	-
Steps=4	DeepCubeA	3.33	60.00%	6.36E+4	821.64	77.36
	Uniform Cost Search	3.00	6.67%	1.43E+5	1962.28	73.01
	Tanimoto Similarity	3.00	6.67%	2.47E+4	272.15	90.64
Steps=5	DeepCubeA	3.40	33.33%	8.40E+4	968.49	86.69
	Uniform Cost Search	-	0.00%	-	-	-
	Tanimoto Similarity	-	0.00%	-	-	-
Steps=6	DeepCubeA	3.20	33.33%	6.14E+4	933.86	65.73
	Uniform Cost Search	-	0.00%	-	-	-
	Tanimoto Similarity	-	0.00%	-	-	-

Panta, Rojina, et al. "Finding Reaction Mechanism Pathways with Deep Reinforcement Learning and Heuristic Search." *ICAPS PRL Workshop 2024*

Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Q-learning

- In the context of pathfinding, Q-learning is used to compute the cost of a path when in a given state, taking a given action, and taking a shortest path from the next state
  - $Q(s,a) = Q(s,a) = c^a(s) + h(T(s,a))$
  - $h(s) = \min_a Q(s,a)$
- Tabular Q-learning applies the following update to each state seen in an episode
  - $Q(s,a) = Q(s,a) + \alpha[c^a(s) + \min_{a'} Q(T(s,a),a') - Q(s,a)]$
  - α is the learning rate
  - Guaranteed to converge to q^* in the tabular setting if certain conditions are met

Approximate Q-learning

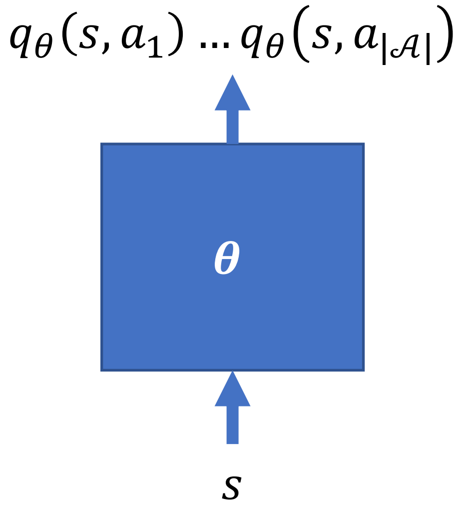
- Q-learning loss
  - $L(\theta) = \left(c^{a}(s) + \min_{a'} q_{\theta^{-}}(T(s,a), a') - q_{\theta}(s,a)\right)^2$
- s: state
- a: action
- T: state transition function
- c^{a}: transition cost function
- θ: parameters
- θ^{-}: parameters for target network
  - Is periodically updated to θ throughout training

Approximate Q-learning

- Q-learning loss
  - $L(\theta) = \left(c^a(s) + \min_{a'} q_{\theta^-}(T(s,a),a') - q_\theta(s,a)\right)^2$
- For each training iteration, an action to update is sampled randomly
- Since it is possible most actions are not part of a shortest path, this could bias the estimator to overestimate the cost-to-go
- Therefore, we sample actions according to a Boltzmann distribution
  - $\pi(a|s) = \dfrac{e^{\left(-\frac{h_\theta(s,a)}{T}\right)}}{\sum_{a'=1}^{|\mathcal{A}|} e^{\left(-\frac{h_\theta(s,a')}{T}\right)}}$

Deep Q-Networks

- Deep Q-networks (DQNs) can compute the estimated cost of taking all actions with a single forward pass
- We create a search algorithm that exploits this to find paths more efficiently and with less memory



Outline

- Background and overview
- Learned heuristic functions and heuristic search
    - Approximate value iteration
    - Batch weighted A* search
    - Generalizing over goals
    - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
    - Q-learning
    - Batch weighted Q* search
    - Applications to quantum computing
- Learned discrete world models and heuristic search

A* Search and Large Action Spaces

- Computation and memory grows linearly with the size of the action space
- Node expansion requires applying every action
- For all child nodes, the heuristic function must be applied
  - Particularly expensive for DNNs with many parameters
- Child nodes are then pushed to OPEN

Batch Weighted Q* Search

- Given a node, compute the transition cost and heuristic value for all child nodes with a single pass through a DQN
- Store tuples of nodes and actions in OPEN
  - Only part that grows linearly with action space
- Apply one action to one node each iteration
- Batch weighted version can also be used
- Guaranteed to be bounded suboptimal if
  - The heuristic function never overestimates
    - $c^a(s) + \min_{a'} q^*(T(s,a), a')$
  - If we terminate when
    - A node we expand from OPEN has a cost greater than or equal to the shortest path we have found so far
    - The number of children generated for that iteration is zero

Algorithm 2 Batch Weighted Q* Search (BWQS)

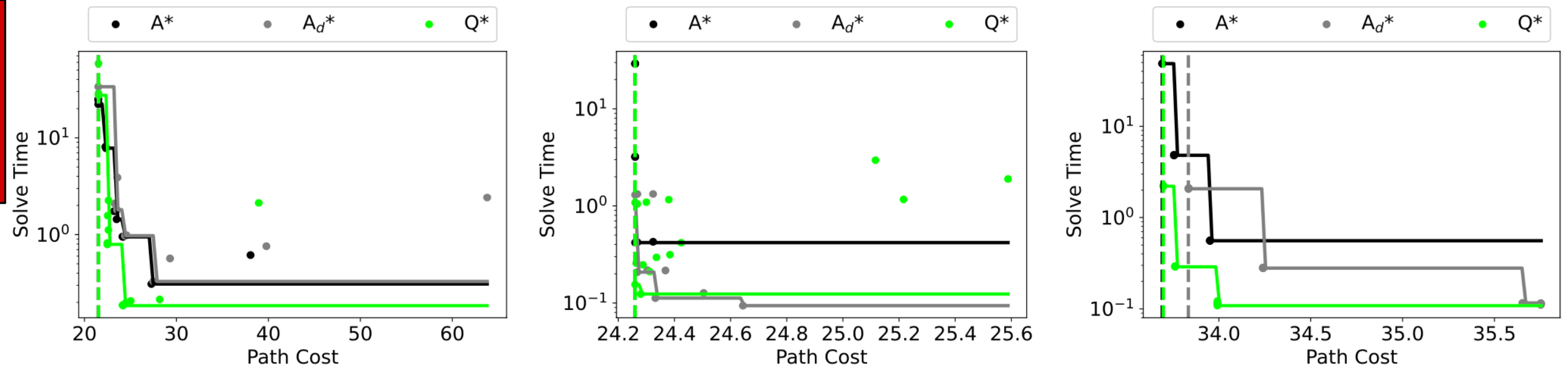
Input: $start$, DNN q_ϕ, batch size B, weight λ
OPEN $\leftarrow$ priority queue of nodes based on minimal f
CLOSED $\leftarrow$ maps states to their shortest discovered path costs
$U, n_U \leftarrow \infty$, NIL
$LB \leftarrow 0$
$n_{start} \leftarrow$ NODE($s = start, g = 0, p =$ NIL$, a =$ NO_OP$, f = 0$)
PUSH n_{start} to OPEN
while not IS_EMPTY(OPEN) **do**
  generated $\leftarrow$ []
  while not IS_EMPTY(OPEN) and SIZE(generated) $< B$ **do**
    $n = (s, a, g, p, f) \leftarrow$ POP(OPEN)
    if IS_EMPTY (generated) **then**
      $LB \leftarrow \max(f, LB)$
    $s' \leftarrow A(s, a)$
    $g(s') \leftarrow g(s) + c^a(s)$
    if IS_GOAL(s') **then**
      if $U > g + c^a(s)$ **then**
        $U, n_U \leftarrow g + c^a(s), n$
      continue loop
    if s' not in CLOSED or $g(s') <$ CLOSED$[s']$ **then**
      CLOSED$[s'] \leftarrow g(s')$
      for a' in $|\mathcal{A}|$ **do**
        APPEND(generated, $(s', g(s'), a', n)$)
  if $LB \geqslant \lambda \cdot U$ **then**
    return PATH_TO_GOAL(n_U)
  generated_states_actions $\leftarrow$ GET_STATES(generated)
  transition_costs, heuristics $\leftarrow q_\phi$(generated_states_actions)
  for $0 \leqslant i \leqslant$ SIZE(generated) **do**
    $s, a, g, p \leftarrow$ generated$[i]$
    $g' \leftarrow g +$ transition_costs$[i]$
    $h \leftarrow$ heuristics$[i]$
    $n_{(s,a)} \leftarrow$ NODE($s, a, g, p, f = \lambda \cdot g' + h$)
    PUSH $n_{(s,a)}$ to OPEN
return PATH_TO_GOAL(n_U) *// failure if n_U is NIL*

Experiments

- Domains: Rubik's cube, Lights Out, 35-pancake puzzle
- Case study: Adding combinations of actions to the Rubik's cube: 12 actions, 156 actions, 1884 actions
- Comparisons
  - A* search
  - Deferred heuristic evaluation: assign heuristic of parent to children
- Did batch weighted search for all search methods
  - Weight in {0.0, 0.2, 0.4, 0.6, 0.8, 1.0}
  - Batch size in {100, 1000, 10000}

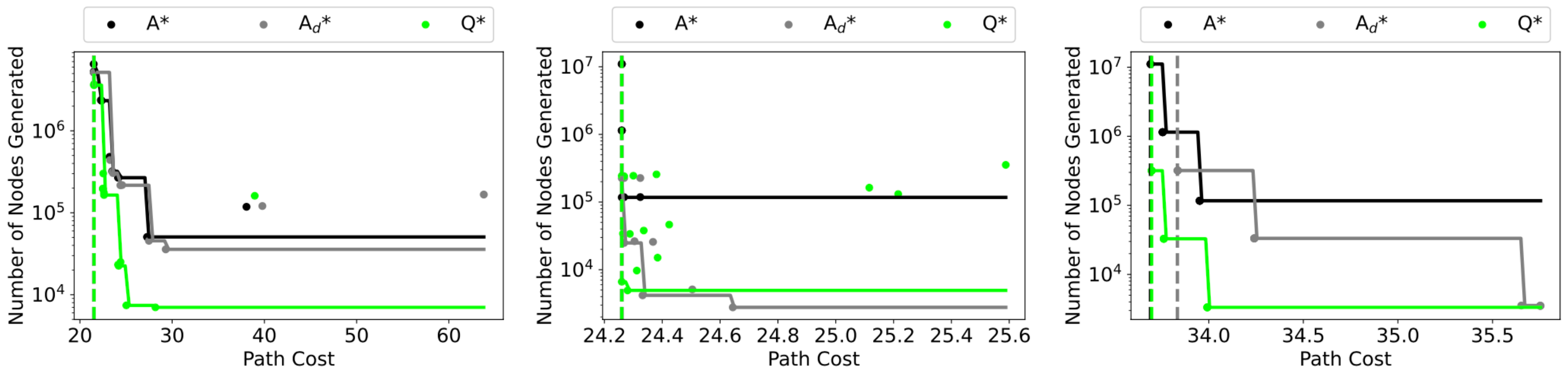
Results

- Each point is a different search parameter setting
- Dashed line: Best path cost
- Solid line: Best of all parameter settings at that path cost
- Q* search often outperforms A* and deferred A* by orders of magnitude
- Best average path cost is either the same or slightly longer



(a) RC (b) Lights Out (c) 35-pancake

Figure 1: Relationship between the average path cost and the average time to find a solution.



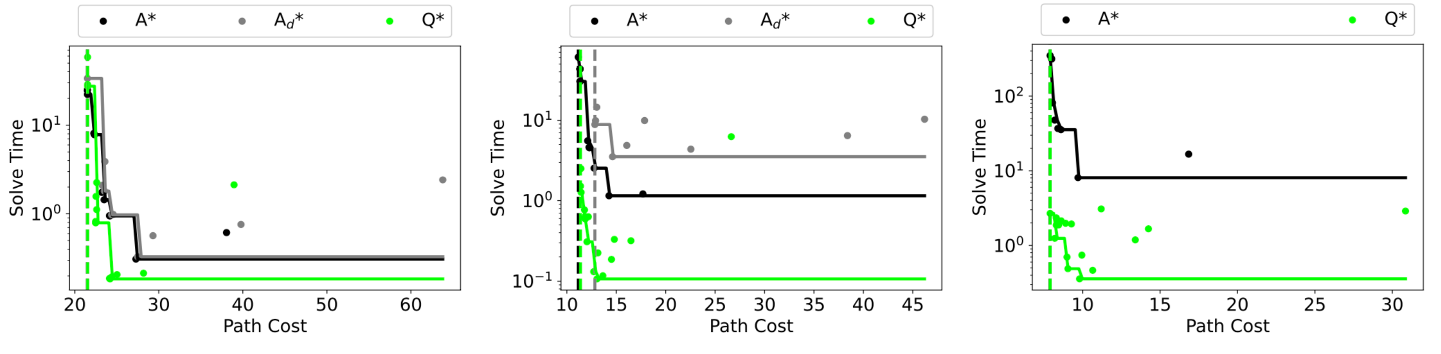
(a) RC(12) (b) Lights Out (c) 35-Pancake

Figure 2: Relationship between the average path cost and the average node generations.

Agostinelli, Forest, et al. "Q* Search: Heuristic Search with Deep Q-Networks." ICAPS PRL Workshop 2024

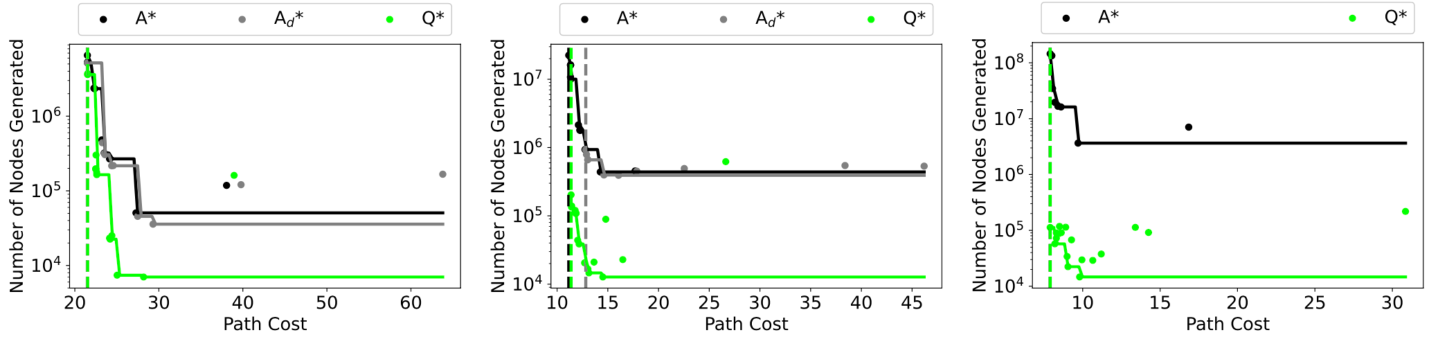
Results

- With 157 times more actions, Q* is only 3.7 times slower and uses 2.3 times more memory



(a) RC(12) (b) RC(156) (c) RC(1884)

Figure 3: Action space size ablation study on Rubik's cube: average path cost vs average time to find a solution.



(a) RC(12) (b) RC(156) (c) RC(1884)

Figure 4: Action space size ablation study on Rubik's cube: average path cost vs average node generations.

Puzzle	Actions	Method	Time	Nodes Gen
RC(156)	x13	A*	3.5(1.6)	8.7(2.2)
		Q*	**0.9(0.7)**	**1.4(1.3)**
RC(1884)	x157	A*	37.0(6.5)	62.7(5.2)
		Q*	**3.7(4.0)**	**2.3(3.6)**

Agostinelli, Forest, et al. "Q* Search: Heuristic Search with Deep Q-Networks." ICAPS PRL Workshop 2024

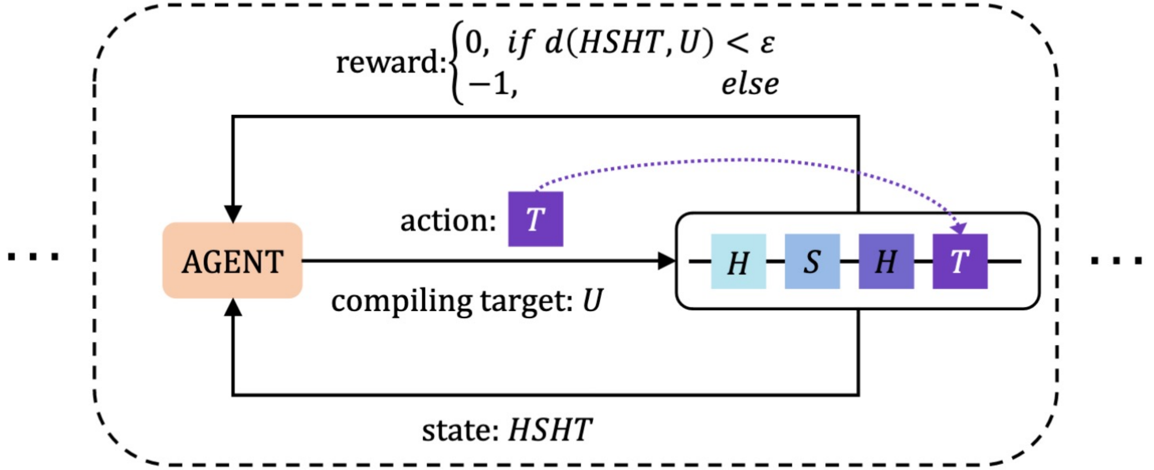
Outline

- Background and overview
- Learned heuristic functions and heuristic search
  - Approximate value iteration
  - Batch weighted A* search
  - Generalizing over goals
  - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
  - Q-learning
  - Batch weighted Q* search
  - Applications to quantum computing
- Learned discrete world models and heuristic search

Quantum Algorithm Compilation

- Given a quantum algorithm, a compiler must synthesize a quantum circuit for this algorithm from a given set of quantum gates
- If a given circuit is below an error threshold, then the problem is considered solved

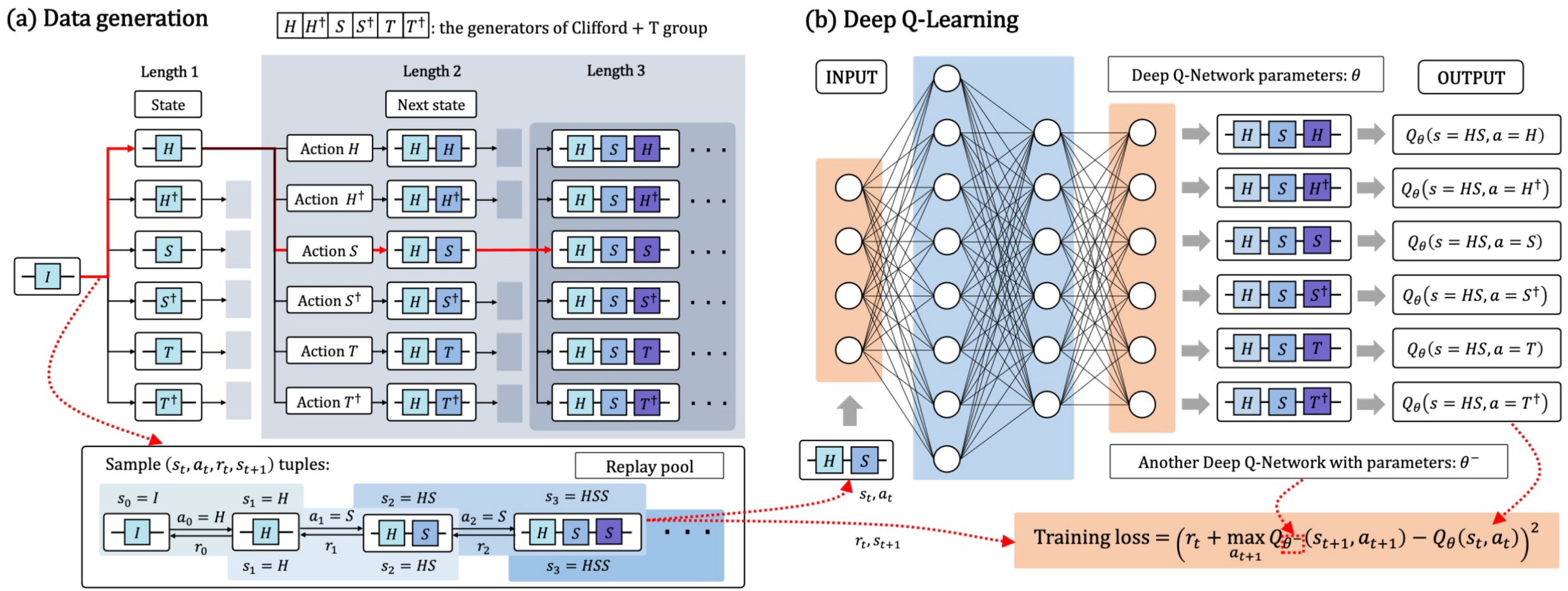
Quantum compiling in the framework of RL



The Markov Decision Process	
action space $\mathcal{A}$	hardware-specific universal basis set
state space $\mathcal{S}$	the circuits composed of $\{A_j; A_j \in \mathcal{A}\}$
reward function r	reward $r = \begin{cases} 0, & if\ d(\prod_{j=0}^{L-1} A_j, U) < \varepsilon \\ -1, & else \end{cases}$
transition probability $\mathcal{P}$	$P(s_{t+1} \mid s_t, a_t) = 1$ (deterministic)

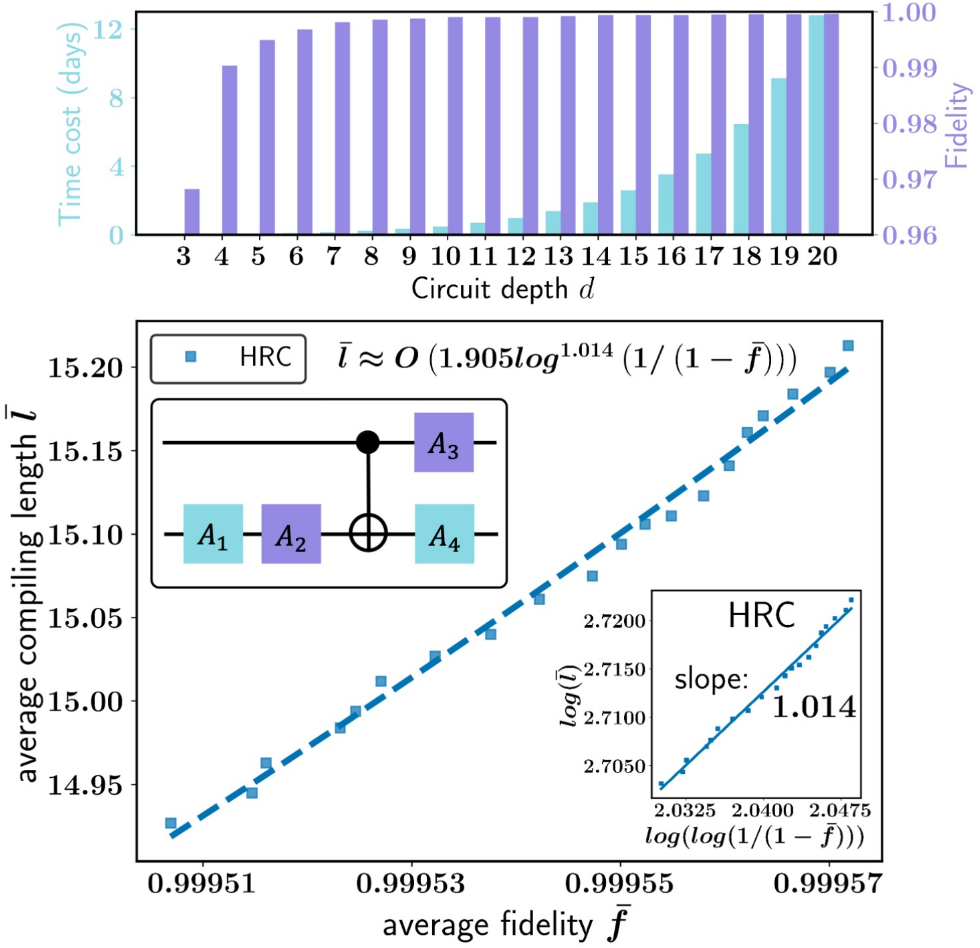
Quantum Algorithm Compilation

- Training data can be generated from a given gate set and a DQN trained to predict the distance of the current quantum circuit to the identity function
- Given a trained DQN, Q* search can be used to search for a circuit for a given algorithm

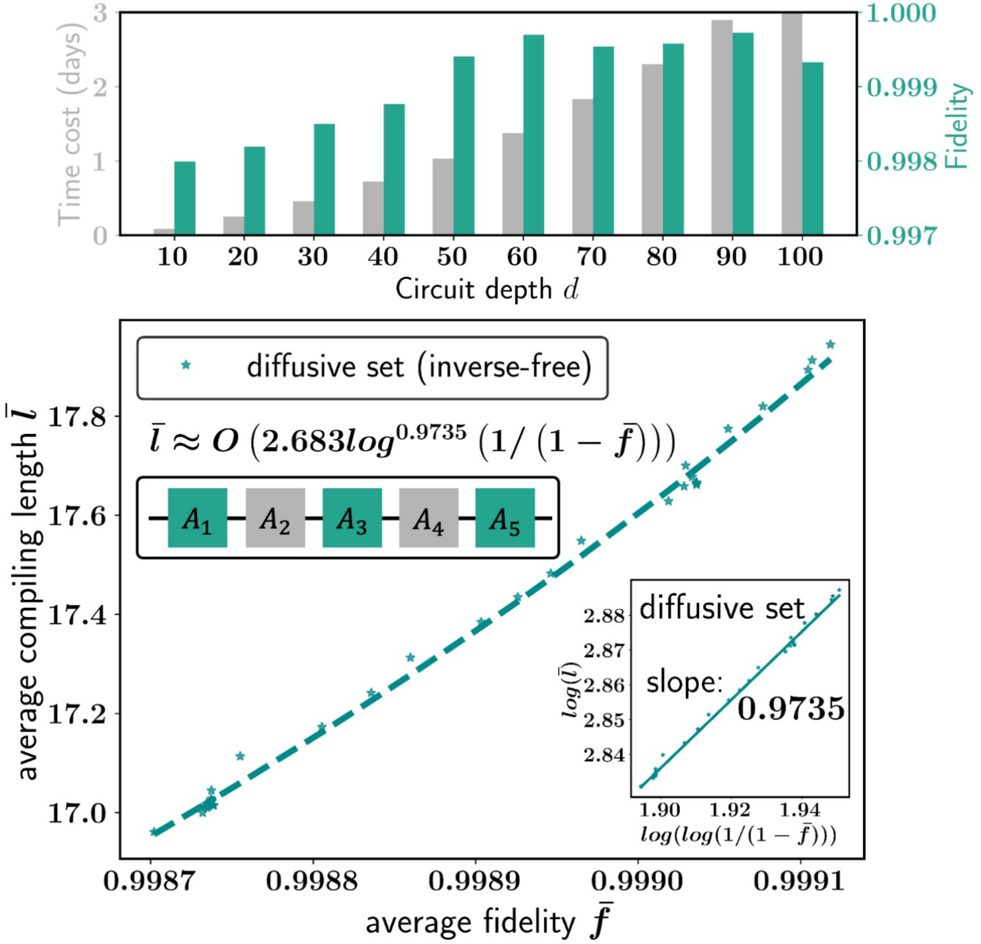


Q-learning and Q* Search

- Accuracy increases given more time for synthesis



Quantum compilation on two-qubit universal basis set

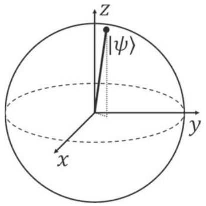
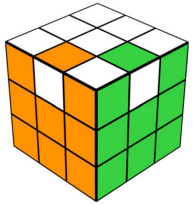


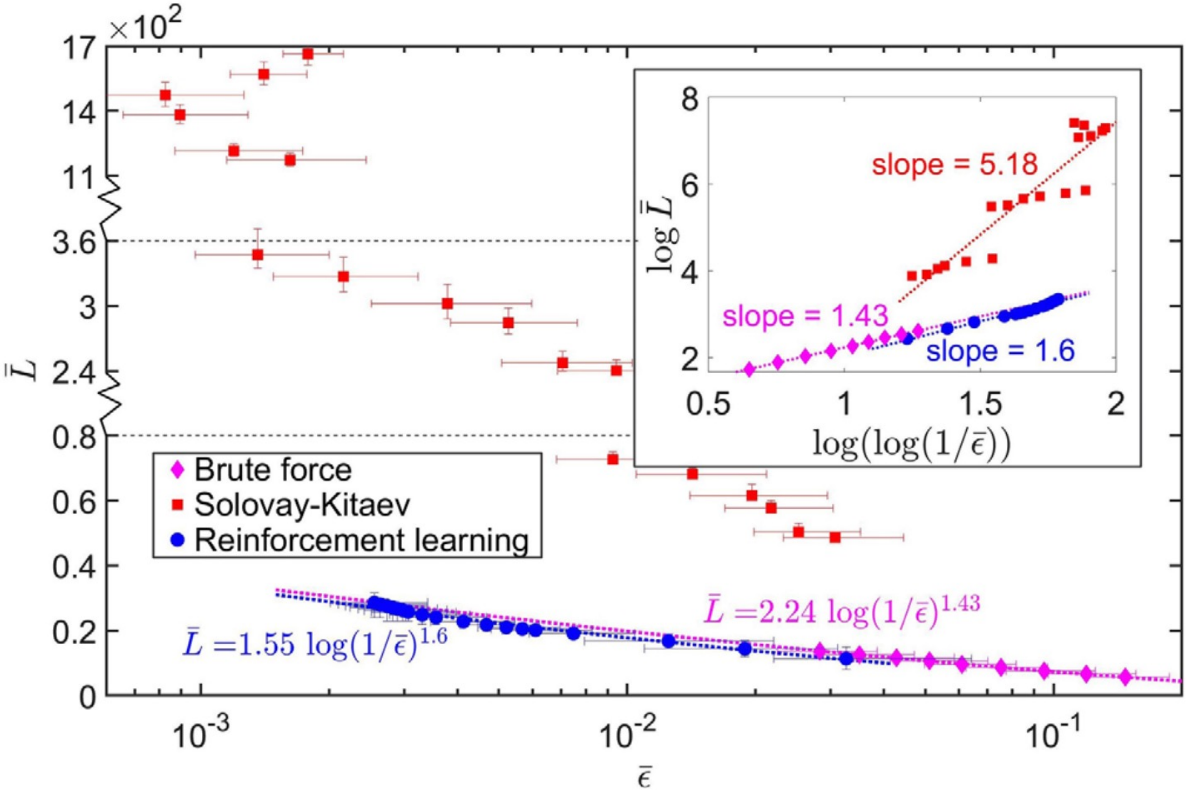
Quantum compilation on inverse-free universal basis set

Qiuhao, Chen, et al. "Efficient and practical quantum compiler towards multi-qubit systems with deep reinforcement learning." Quantum Science and Technology (2024).

Other Applications to Quantum Algorithm Compilation

- Topological quantum compiling
- Clifford synthesis
- Can produce near-optimal solutions

System		
Initial state	The unitary to be approximated	The scrambled cube
Target state	The identity matrix	The solved cube
Basic move	A gate from the universal set	Rotation of one face



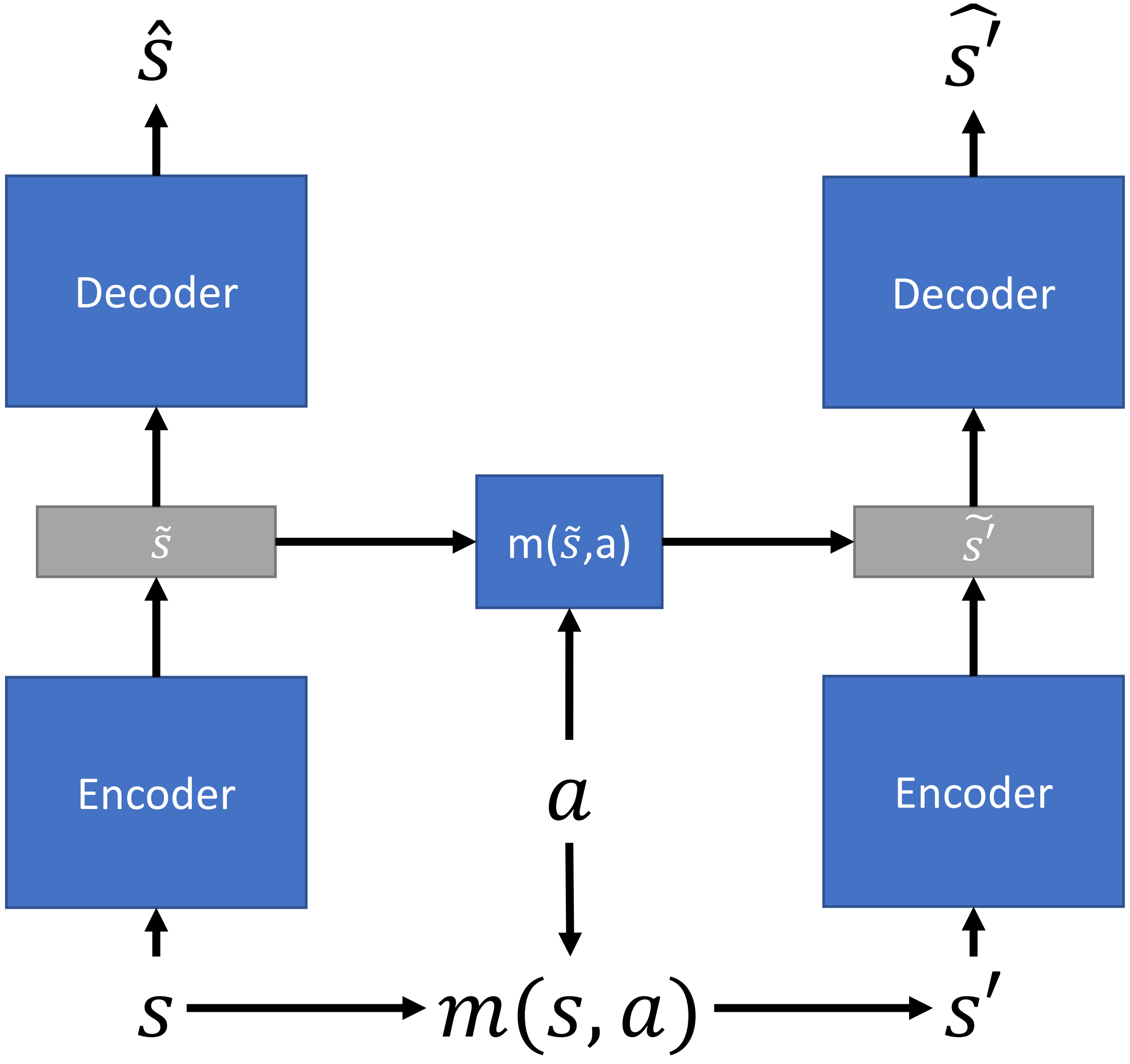
Zhang, Yuan-Hang, et al. "Topological Quantum Compiling with Reinforcement Learning." Physical Review Letters 125.17 (2020): 170501.
Bao, Ning, and Gavin S. Hartnett. "Twisty-puzzle-inspired approach to Clifford synthesis." *Physical Review A* 109.3 (2024): 032409.

Outline

- Background and overview
- Learned heuristic functions and heuristic search
    - Approximate value iteration
    - Batch weighted A* search
    - Generalizing over goals
    - Applications to reaction mechanism pathway prediction
- Learned action-heuristic functions and heuristic search
    - Q-learning
    - Batch weighted Q* search
    - Applications to quantum computing
- Learned discrete world models and heuristic search

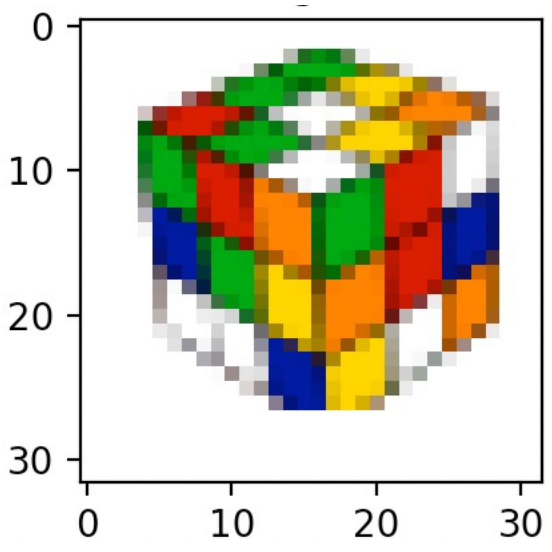
Learning Discrete World Models

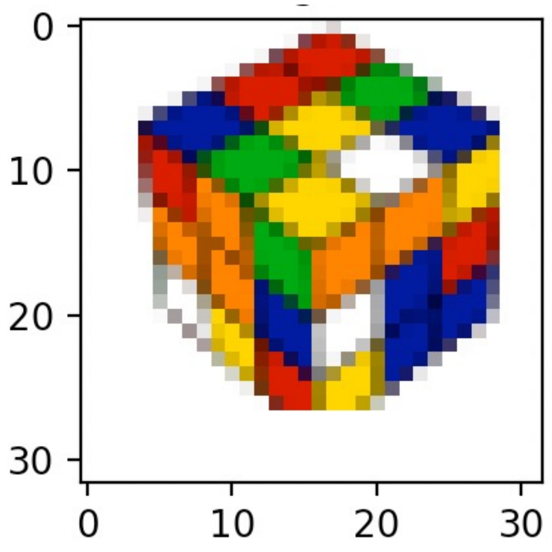
- Addressing previous shortcomings
  - Small errors in prediction can be corrected by simply rounding
  - Can reidentify states by comparing two vectors
- Encoder
  - Maps the state to a discrete representation
  - To allow training with gradient descent, use a straight through estimator
- Decoder
  - Maps the discrete representation to the state
  - Ensures the discrete representation is meaningful
- Environment model
  - Maps discrete states and actions to next discrete state

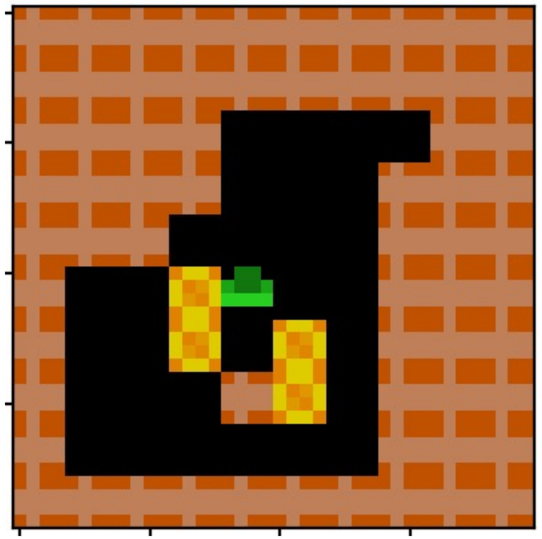


Experiments

- Rubik’s cube
  - Two 32x32 RGB images showing both sides of the cube
- Sokoban
  - One 40x40 RGB image
- Generate offline dataset of 300,000 episodes of 30 random steps, each

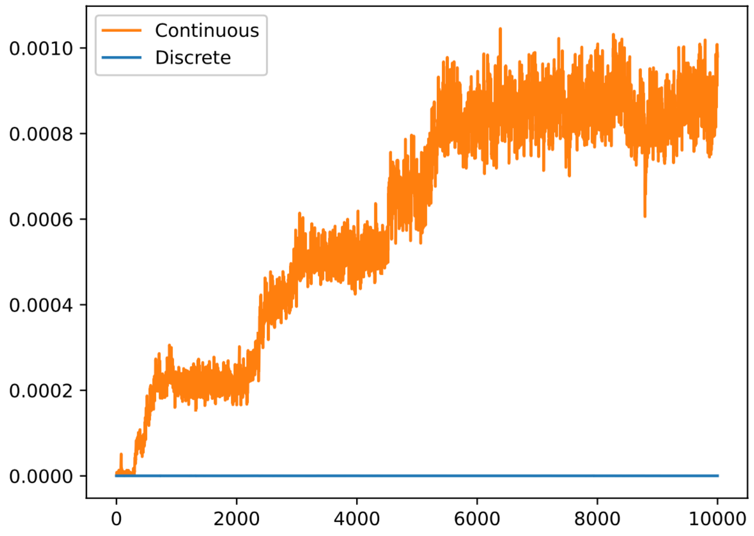




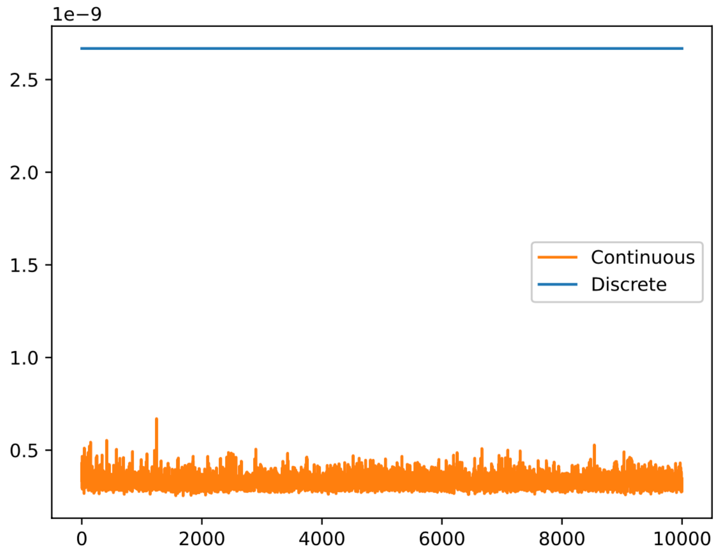


Discrete vs Continuous Model Performance

- The continuous model eventually accumulates error for the Rubik's cube



(a) Rubik's Cube



(b) Sokoban

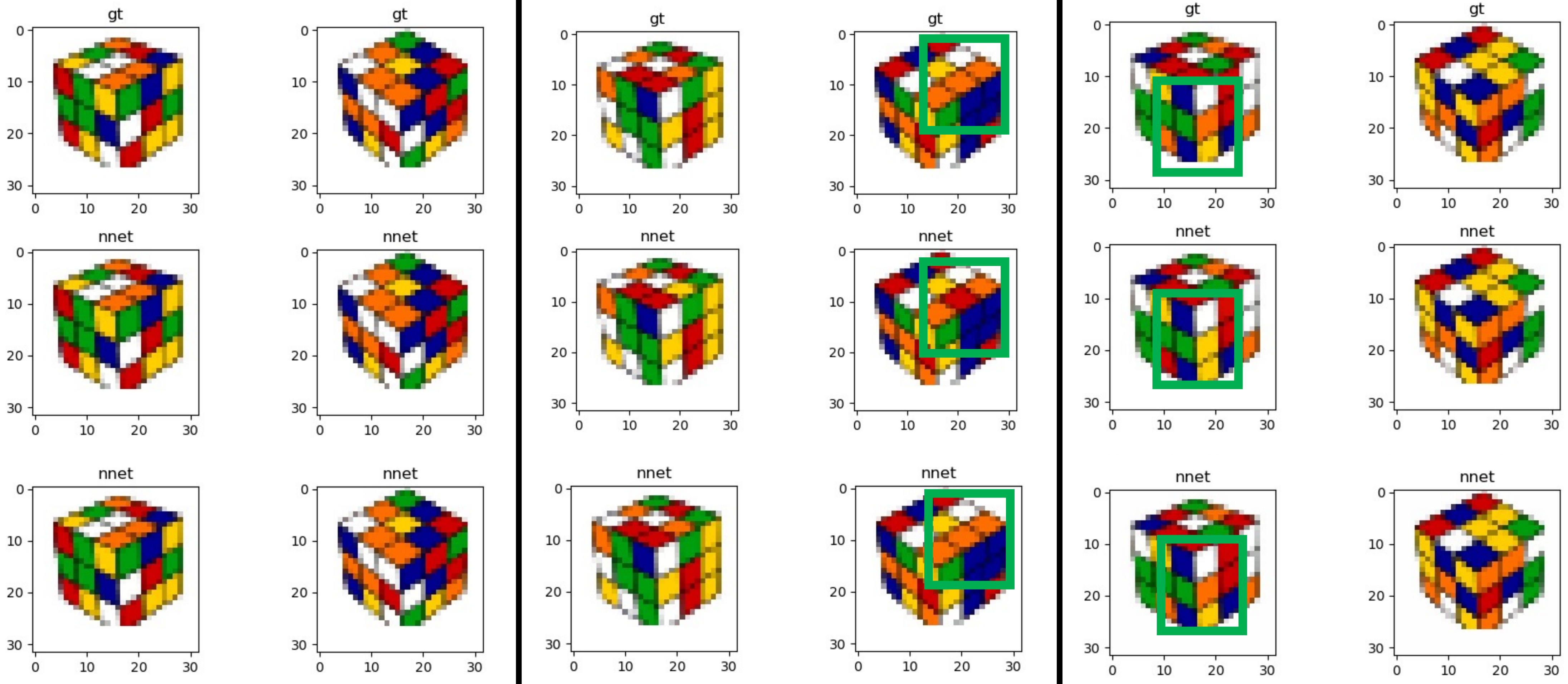
Discrete vs Continuous Model Performance

1000 steps | 4000 steps | 9000 steps

Ground Truth

Continuous

Discrete



Heuristic Learning and Search with Discrete Model

- DeepCubeAI – DeepCubeA + "Imagination"
  - Learn discrete world model with offline data
  - Use offline data and the learned world model to generate training data
  - Heuristic learning: Q-learning with hindsight experience replay
    - Generalize over goal states
  - Heuristic search: Q* search
    - Helps when model uses computationally expensive DNN

Domain	**Solver**	**Len**	**Opt**	**Nodes**	**Secs**	**Nodes/Sec**	**Solved**
RC	PDBs^{+}	20.67	100.0%	2.05E+06	2.20	1.79E+06	100%
	DeepCubeA	21.50	60.3%	6.62E+06	24.22	2.90E+05	100%
	Greedy (ours)	-	0%	-	-	-	0%
	DeepCubeAI (ours)	22.85	19.5%	2.00E+05	6.21	3.22E+04	100%
RC_{rev}	Greedy (ours)	-	0%	-	-	-	0%
	DeepCubeAI (ours)	22.81	21.92%	2.00E+05	6.30	3.18+04	99.9%
Sokoban	LevinTS	39.80	-	6.60E+03	-	-	100%
	LevinTS (*)	39.50	-	5.03E+03	-	-	100%
	LAMA	51.60	-	3.15E+03	-	-	100%
	DeepCubeA	32.88	-	1.05E+03	2.35	5.60E+01	100%
	Greedy (ours)	29.55	-	-	1.68	-	41.9%
	DeepCubeAI (ours)	33.12	-	3.30E+03	2.62	1.38E+03	100%

Agostinelli, Forest and Soltani, Misagh "Learning Discrete World Models for Heuristic Search." *Reinforcement Learning Conference 2024*

Questions?

- Papers

  - Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.
  - Agostinelli, Forest, Rojina Panta, and Vedant Khandelwal. "Specifying Goals to Deep Neural Networks with Answer Set Programming." *ICAPS 2024*
  - Panta, Rojina, et al. "Finding Reaction Mechanism Pathways with Deep Reinforcement Learning and Heuristic Search." *ICAPS PRL Workshop 2024*
  - Agostinelli, Forest, et al. "Q* Search: Heuristic Search with Deep Q-Networks." ICAPS PRL Workshop 2024
  - Agostinelli, Forest and Soltani, Misagh "Learning Discrete World Models for Heuristic Search." *Reinforcement Learning Conference 2024*

- Code
  - Many of these algorithms are publicly available on GitHub
  - https://github.com/forestagostinelli/deepxube

Email: foresta@cse.sc.edu

Website: https://cse.sc.edu/~foresta/

DeepCubeA Performance

Puzzle	Solver	Length	Percentage of optimal solutions	No. of nodes	Time taken (s)	Nodes per second
Rubik's cube	PDBs[7]	–	–	–	–	–
	PDBs+[24]	20.67	100.0	2.05×10^{6}	2.20	1.79×10^{6}
	DeepCubeA	21.50	60.3	6.62×10^{6}	24.22	2.90×10^{5}
Rubik's cube_h	PDBs[7]	–	–	–	–	–
	PDBs+[24]	26.00	100.0	2.41×10^{10}	13,561.27	1.78×10^{6}
	DeepCubeA	26.00	100.0	5.33×10^{6}	18.77	2.96×10^{5}
15 puzzle	PDBs[9]	52.02	100.0	3.22×10^{4}	0.002	1.45×10^{7}
	DeepCubeA	52.03	99.4	3.85×10^{6}	10.28	3.93×10^{5}
15 puzzle_h	PDBs[9]	80.00	100.0	1.53×10^{7}	0.997	1.56×10^{7}
	DeepCubeA	82.82	17.65	2.76×10^{7}	69.36	3.98×10^{5}
24 puzzle	PDBs[9]	89.41	100.0	8.19×10^{10}	4,239.54	1.91×10^{7}
	DeepCubeA	89.49	96.98	6.44×10^{6}	19.33	3.34×10^{5}
35 puzzle	PDBs[9]	–	–	–	–	–
	DeepCubeA	124.64	–	9.26×10^{6}	28.45	3.25×10^{5}
48 puzzle	PDBs	–	–	–	–	–
	DeepCubeA	253.35	–	1.96×10^{7}	74.46	2.63×10^{5}
Lights Out	DeepCubeA	24.26	100.0	1.14×10^{6}	3.27	3.51×10^{5}
Sokoban	LevinTS[32]	39.80	–	6.60×10^{3}	–	–
	LevinTS(*)[32]	39.50	–	5.03×10^{3}	–	–
	LAMA[32]	51.60	–	3.15×10^{3}	–	–
	DeepCubeA	32.88	–	1.05×10^{3}	2.35	5.60×10^{1}