Deep Learning, Reinforcement Learning, and Heuristic Search

Forest Agostinelli
University of South Carolina

Students

Ph.D. Students

Rojina Panta

Vedant Khandelwal

Misagh Soltani

Cale Workman

B.S. Students

Christian Geils

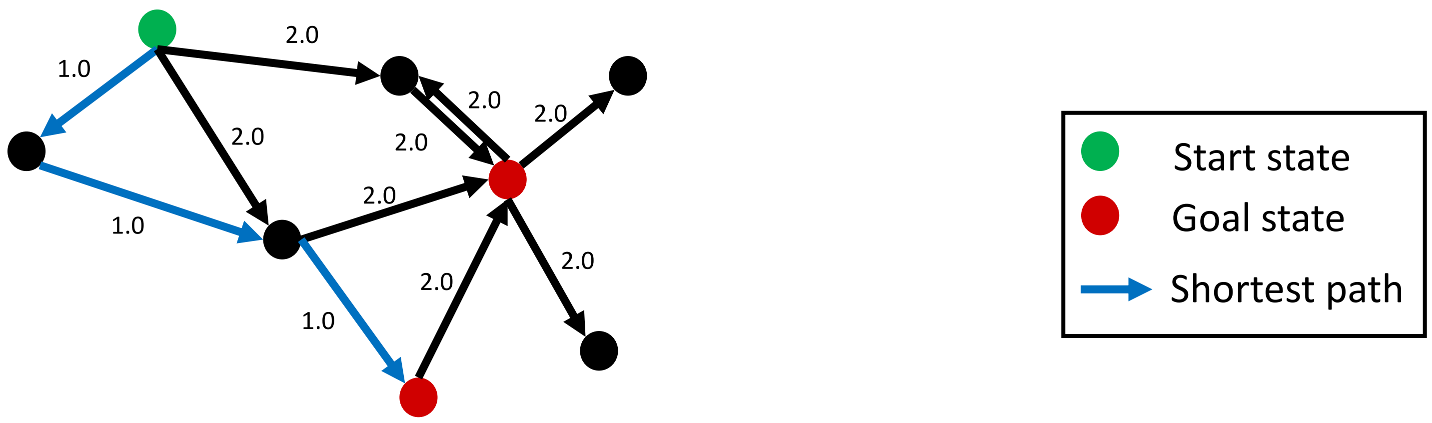
William Edwards

Outline

- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

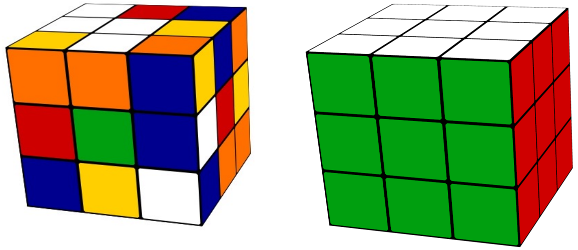
Pathfinding

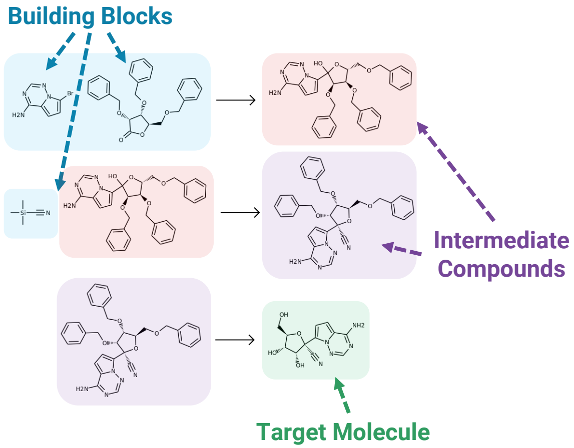
- The objective of pathfinding is to find a sequence of actions that forms a path between a given start state and a given goal
  - A goal is a set of states
  - Preference for minimum cost paths
- A pathfinding problem can be represented as a weighted directed graph where nodes represent states, edges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

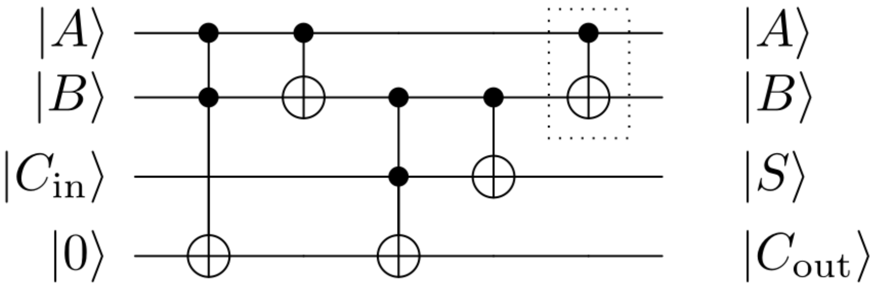


Pathfinding Domains

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







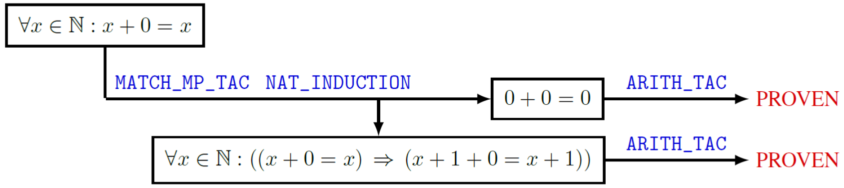
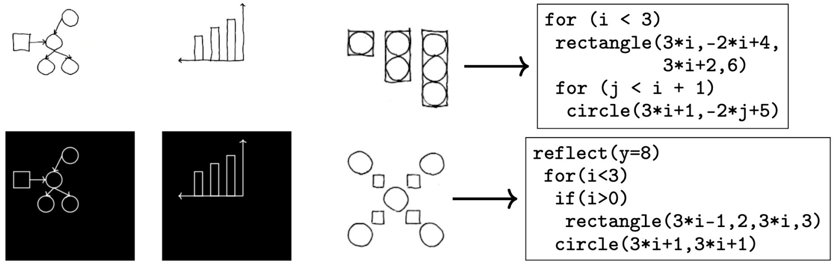


Figure 1: Formally proving $\forall x \in \mathbb{N} : x + 0 = x$.



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 test function
  - Goal specification language
- **Objective: Create a domain-independent algorithm that learns domain-specific heuristics**

Scope of Problems

- What pathfinding problems can be solved with deep reinforcement learning and heuristic search?
  - Sufficient data
  - Sufficient expressivity of deep neural network (DNN)
- Deep learning is data hungry
- Defining models and generating data is easy for many important and difficult problems
  - Theorem proving
  - Program synthesis
  - Quantum algorithm compilation
- Also possible for other environments
  - Chemical synthesis
  - Robotics (sim2real)
- What if we don't have enough data or time?
  - Foundation models
  - Generative model of domains?

Outline

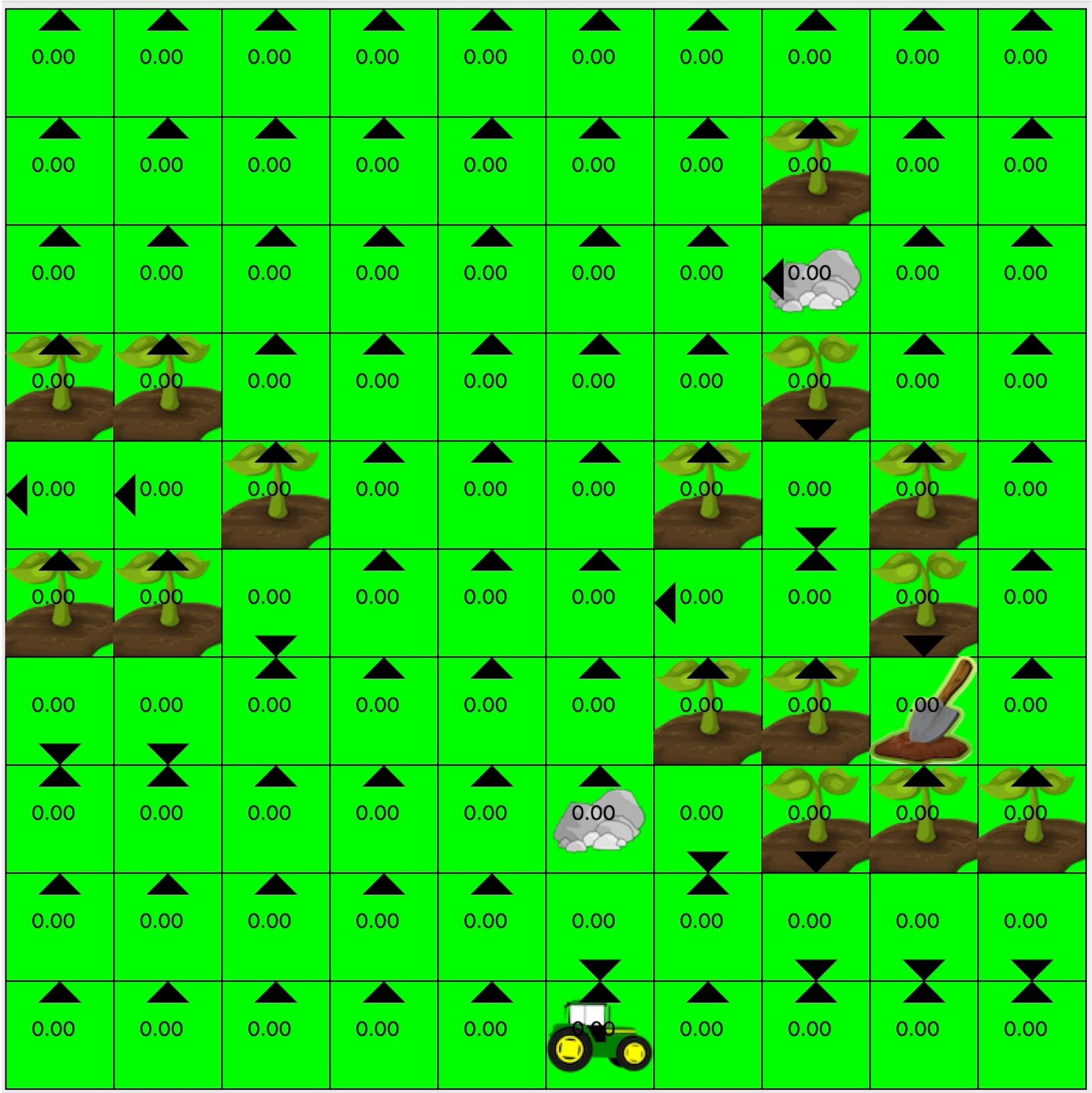
- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

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

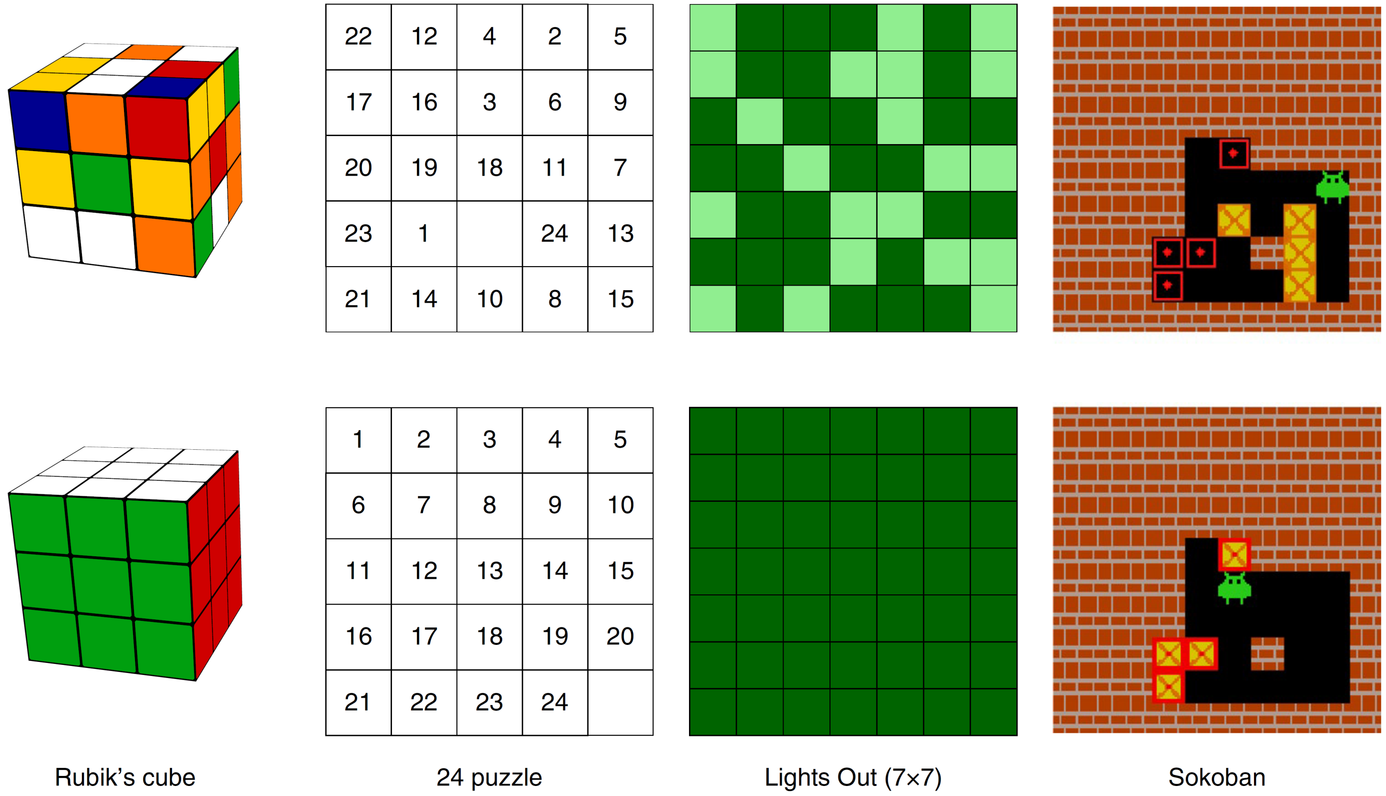
- Actions: up, down, left, right
- Transition costs
  - 1 if square is blank
  - 10 if square has a rock
  - 50 if square has a plant
- Goal: shovel
- Updates propagate outwards from the goal
  - $h(s) = \min_a (c^a(s) + h(T(s,a)))$



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
  - $L(\theta) = \left(\min_a (c^a(s) + h_{\theta^-}(T(s,a))) - h_\theta(s)\right)^2$
  - 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

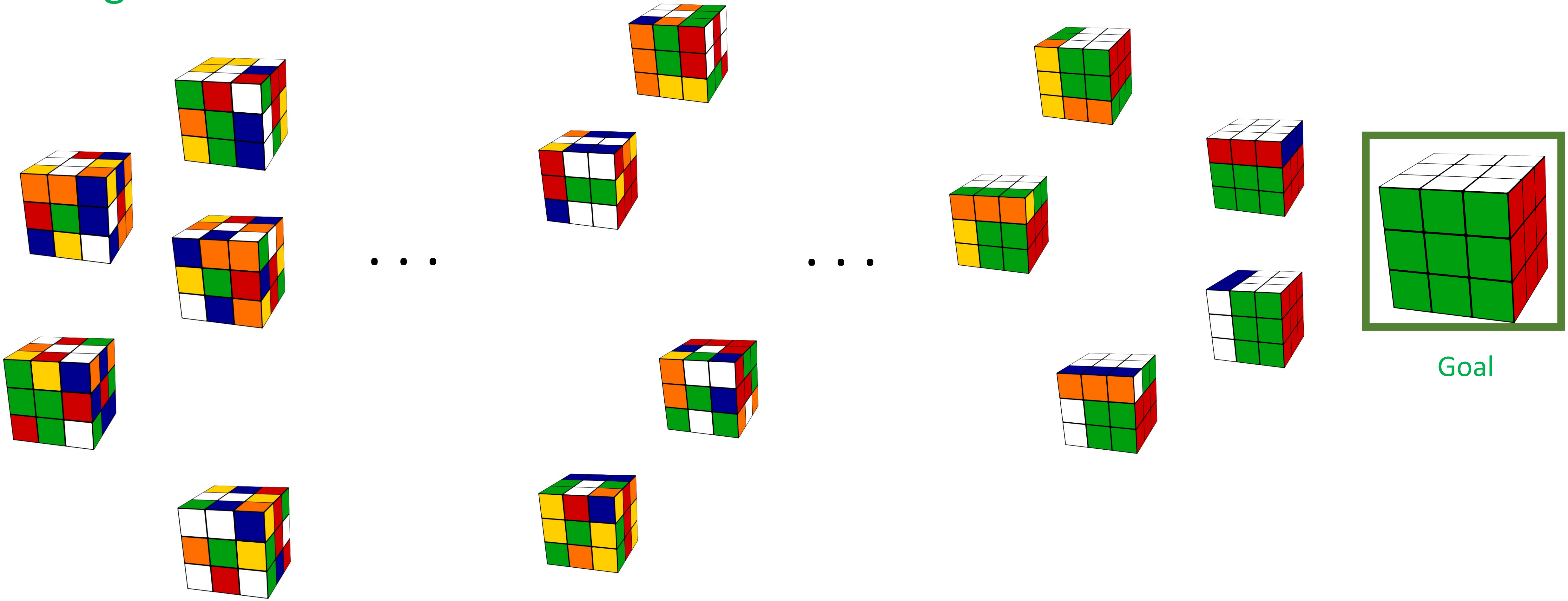


1. Rubik's Cube
2. 15-puzzle
3. 24-puzzle
4. 35-puzzle
5. 48-puzzle
6. Lights Out
7. Sokoban

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

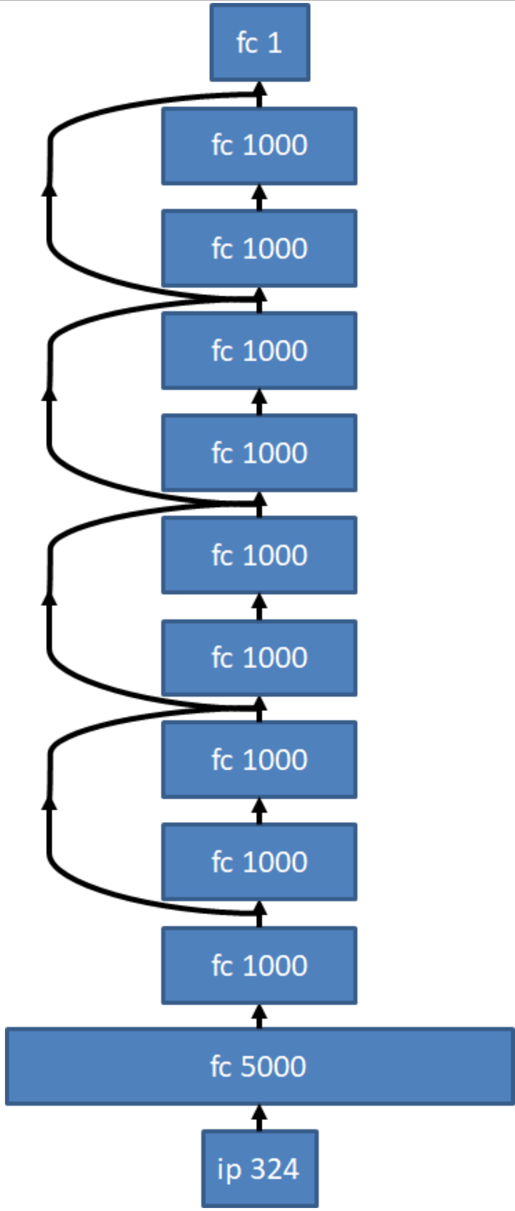
Generating States

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



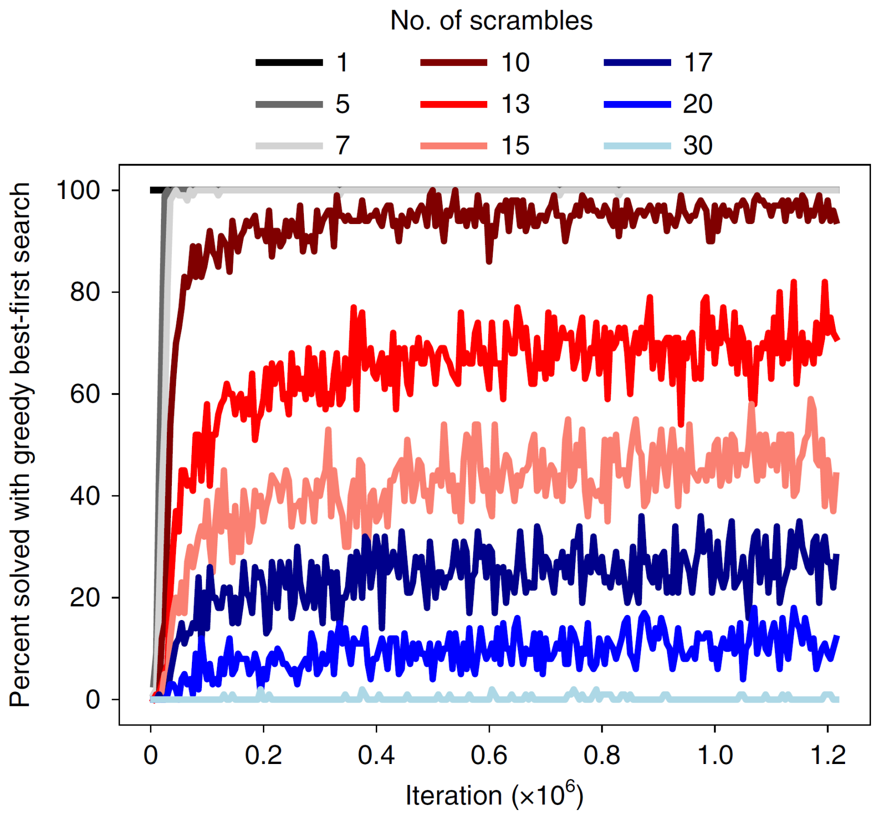
Training

- 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
- Supervised learning yields similar performance
- We need heuristic search!



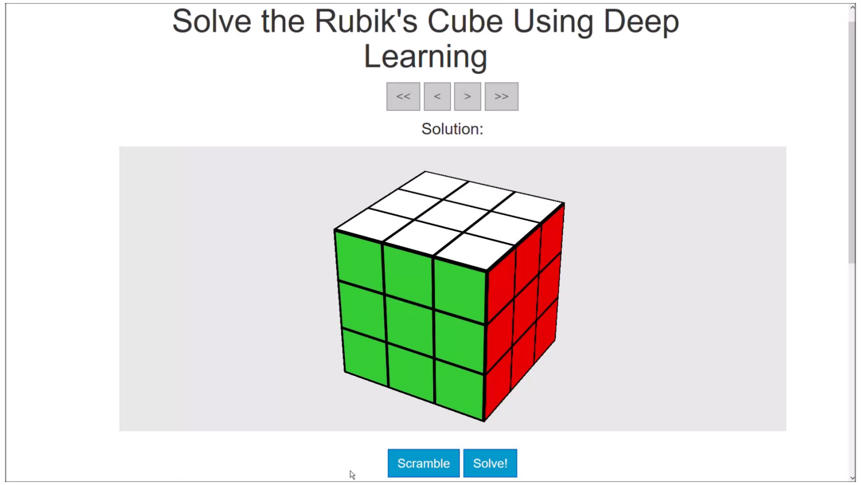
Batch Weighted A* Search

- 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 the lower bound >= the upper bound

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.

Outline

- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

Q-learning

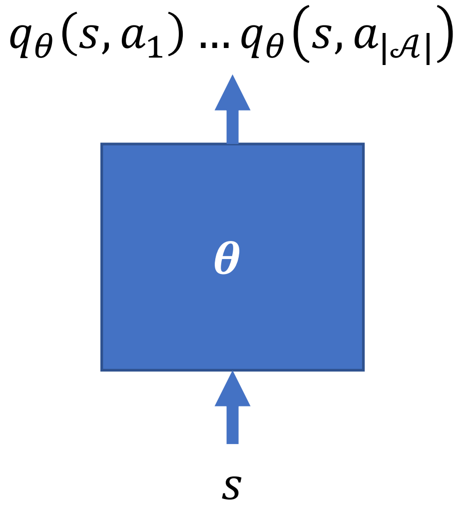
- In the context of pathfinding, Q-learning is used to compute the cost of a path when in a given state, taking a given action, and taking a shortest path from the next state
  - $Q(s,a) = Q(s,a) = c^a(s) + h(T(s,a))$
  - $h(s) = \min_a Q(s,a)$
- Tabular Q-learning applies the following update to each 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
  - $L(\theta) = \left(c^a(s) + \min_{a'} q_{\theta^-}(T(s,a),a') - q_\theta(s,a)\right)^2$
- 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
  - $\pi(a|s) = \frac{e^{(-\frac{h_\theta(s,a)}{T})}}{\sum_{a'=1}^{|\mathcal{A}|} e^{(-\frac{h_\theta(s,a')}{T})}}$

From A* Search to Q* Search

- A* search: the number of nodes generated and number of heuristic function applications during each iteration of search grows linearly with the size of the action space
- Deep Q-networks (DQNs) can compute the estimated cost of taking all actions with a single forward pass
- Q* search: the number of nodes generated and number of heuristic function applications is independent of the size of the action space



Batch Weighted Q* Search

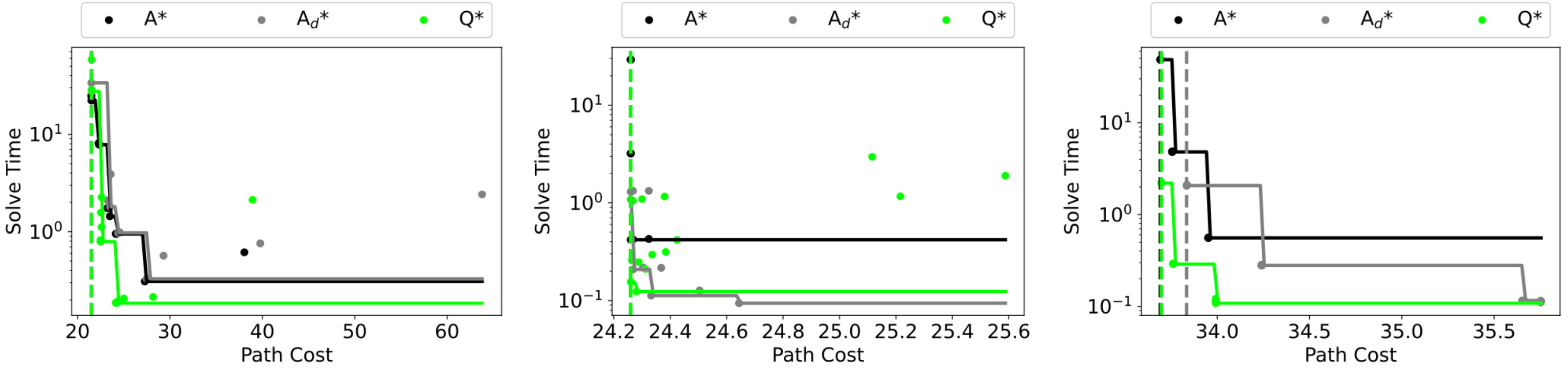
- Given a node, compute the transition cost and heuristic value for all child nodes with a single pass through a DQN
- Store tuples of nodes and actions in OPEN
  - Only part that grows linearly with action space
- Apply popped actions to popped nodes
- Batch weighted version can also be used
- Guaranteed to be bounded suboptimal if
  - The heuristic function never overestimates
    - $c^a(s) + \min_{a'} q^*(T(s,a), a')$
  - If we terminate when the lower bound >= the upper bound

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}

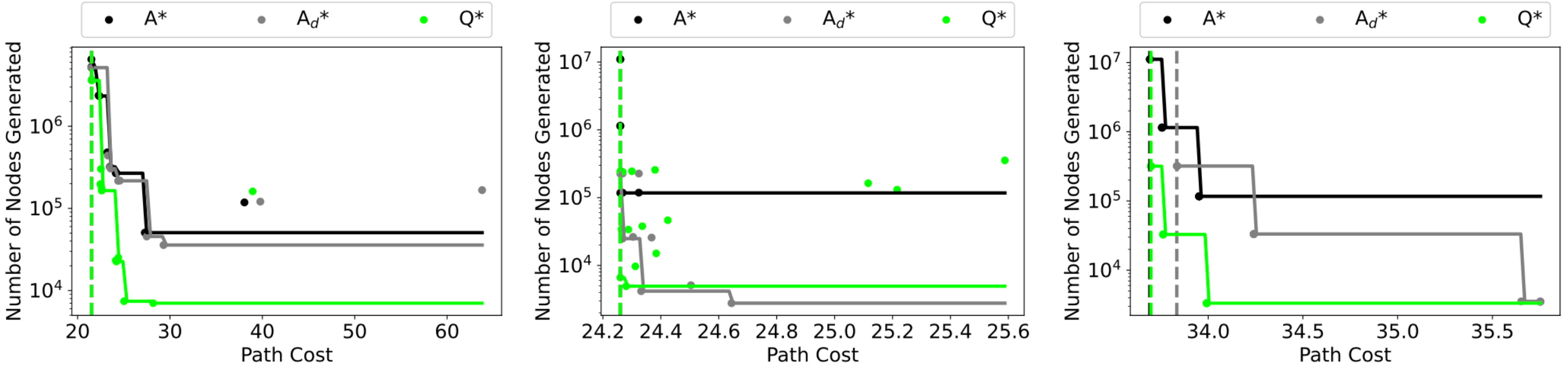
Results

- 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.



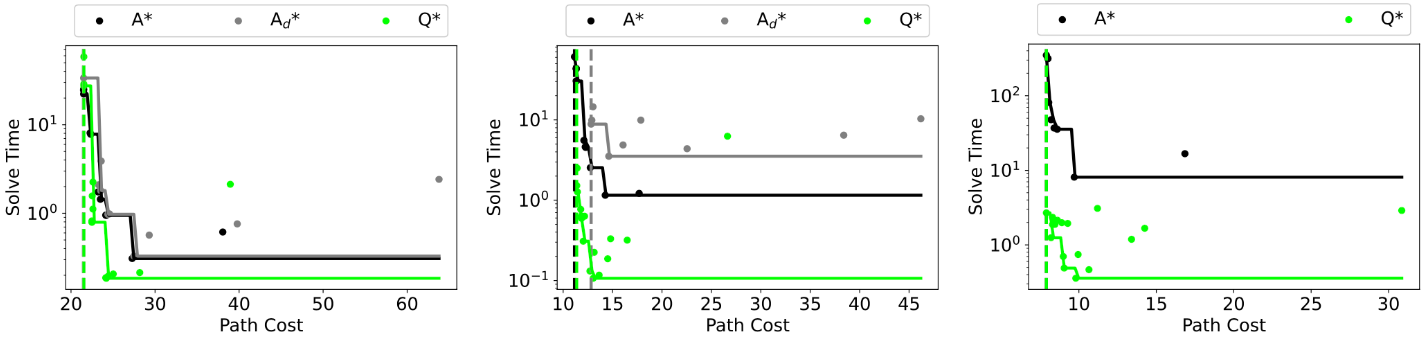
(a) RC(12) (b) Lights Out (c) 35-Pancake

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*

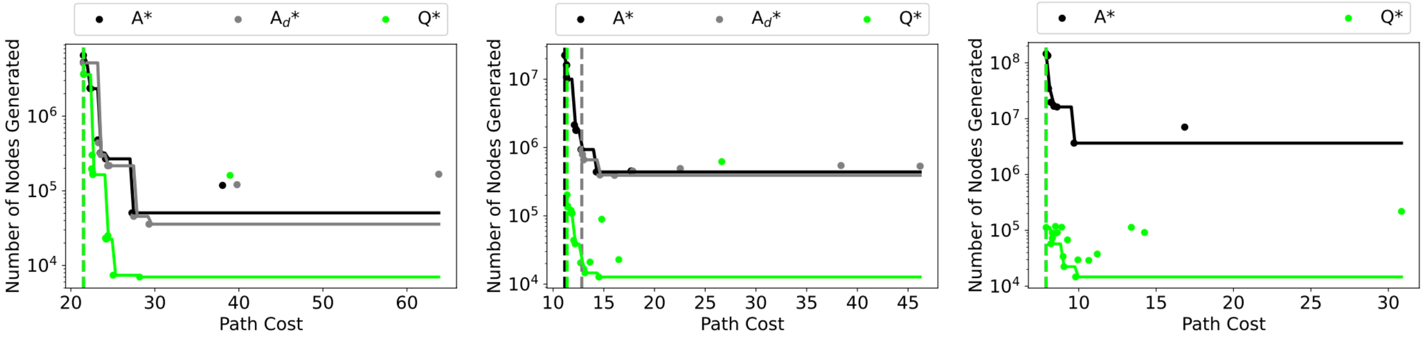
Results

- 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

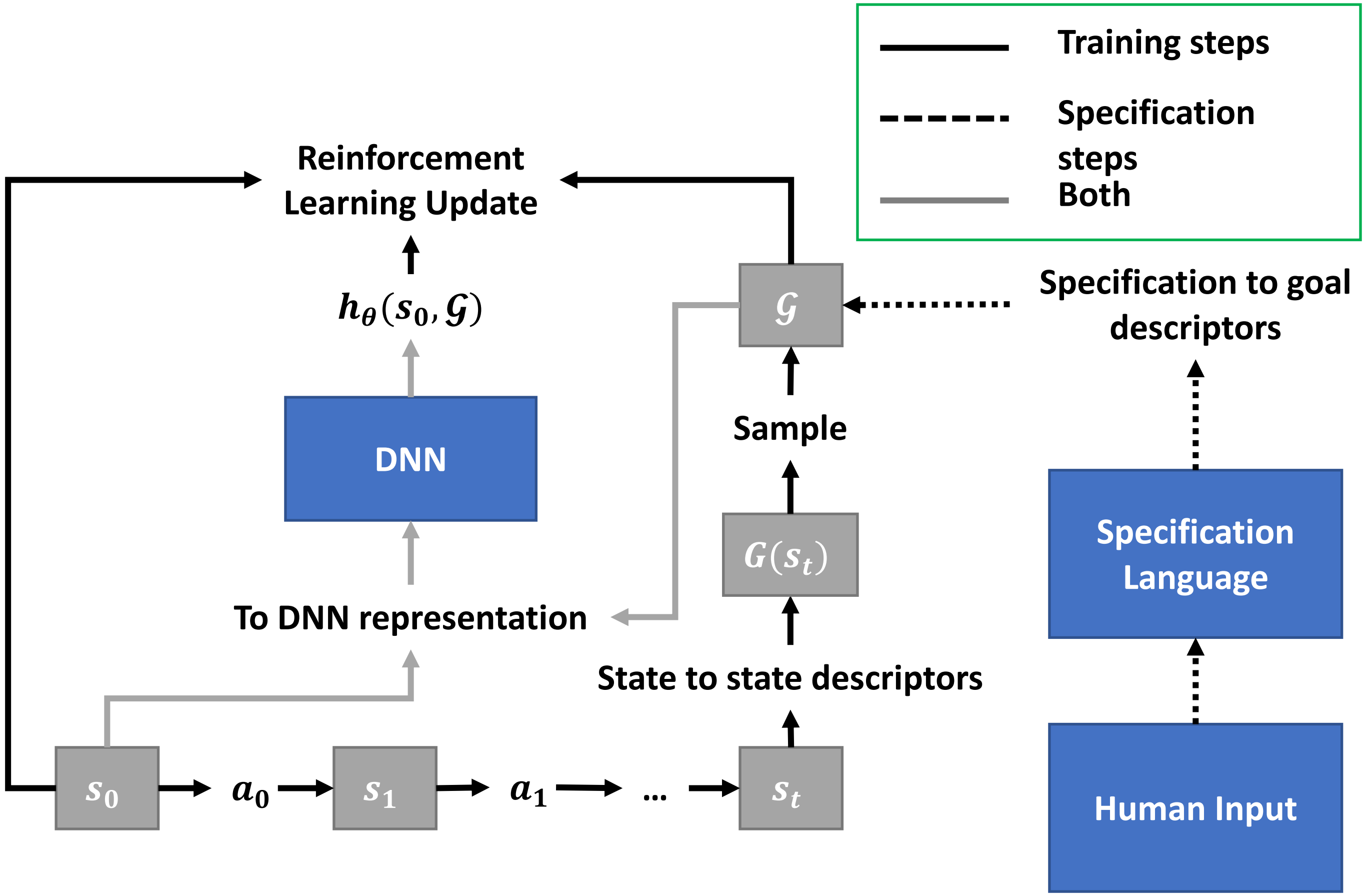
- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

Rojina Panta

Vedant Khandelwal

Generalizing Over Goals: Overview

- In the previous work, the goal is predetermined
- We build on hindsight experience replay to generalize over sets of goal states
- In our work
  - State descriptors: assignments of values to variables
  - Specification language: Answer set programming (ASP)
  - ASP will be used to describe goals at a high-level using formal logic and an answer set solver will be used to find assignments that represent a subset of the goal

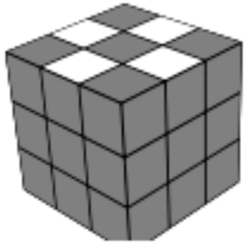


State Representation

- In a given pathfinding domain, there are V variables
  - A variable, x_i, can be assigned a single value from its (variable) domain, $D(x_i)$
- An assignment is an assignment is a set of assignments of values to variables $\{x_i = v_i\}$
  - All $v_i \in D(v_i)$
  - If x_i is not in the assignment then it is unassigned
- An assignment is a complete assignment iff all variables have been assigned values
- A state is a complete assignment
- For example, for the Rubik's cube, variables are stickers and values are their colors

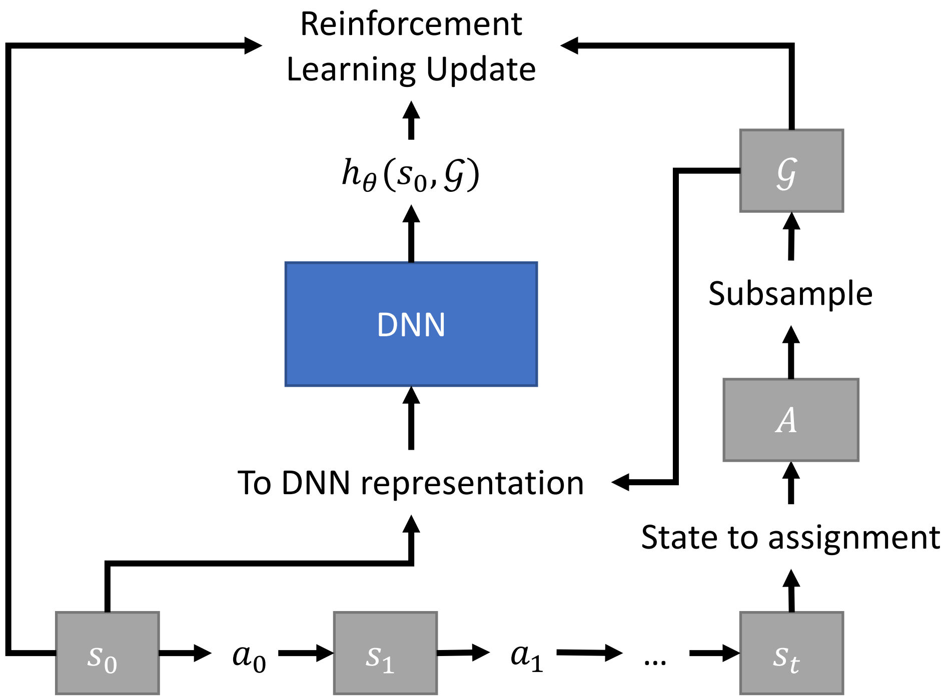
Goal Representation

- An assignment is a partial assignment iff at least one variable has not been assigned a value
- A goal is a complete or partial assignment
- An assignment, A, represents a set of states, $\mathcal{S}_A$
  - A complete assignment always represents a set of states of size 1
- A state, s, is in $\mathcal{S}_A$ iff $A \subseteq s$
  - In other words, all assignments in A are present in s
  - An empty assignment represents the set of all possible states
- For example, a visualization of an assignment for the "white cross" pattern for the Rubik's cube and a state that is in the set of states represented by this assignment



Training

- Generate a start state
- Take a random walk whose length is somewhere between 0 and T
  - Future work could use artificial curiosity
- Convert the end state to its representation as an assignment
- Subsample to obtain a goal
- Convert this representation into one suitable for the DNN
  - One-hot representation
  - Graph
  - Etc.
- RL Update
  - $L(\theta) = \left(\min_{a}(c^{a}(s) + h_{\theta^-}(T(s,a)), \mathcal{G}) - h_{\theta}(s, \mathcal{G})\right)^2$



Experiments

- ASP will be used to find assignments; therefore, we compare our method (DeepCubeA$_g$) to other methods capable of finding paths to goals that can be represented as complete or partial assignments
- 500-1,000 test start and goal pairs
- 200 second time limit to solve test states
- **DeepCubeA**
  - Predefined goal
- **Fast Downward Planner**
  - Can automatically construct heuristics given a formal definition of the domain (including the transition function) in the planning domain definition language (PDDL)
  - Goal count heuristic, fast forward heuristic, causal graph heuristic
  - A* search
- **PDBs**
  - Divides into subproblems and enumerates all possible combinations of the subproblem to create heuristic
  - Predefined goal
  - IDA* search

Performance

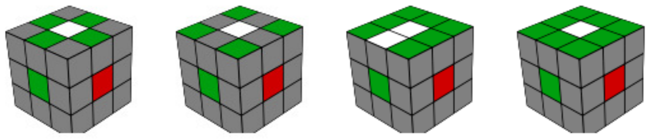
- Canon: Canonical goal states
- Rand: Random assignment selected as goal
  - Can be as small as the empty assignment
  - Methods that require a pre-definied goal cannot be applied to this scenario without considerable overhead
- PDBs+: Also includes group theory knowledge
- DeepCubeA$_g$ consistently outperforms fastdownard in terms of percentage of states solved

Puzzle	Solver	Path Cost	% Solved	% Opt	Nodes	Secs	Nodes/Sec
RC (Canon)	PDBs$^+$	**20.67**	**100.00%**	**100.0%**	**2.05E+06**	**2.20**	**1.79E+06**
	DeepCubeA	21.50	**100.00%**	60.3%	6.62E+06	24.22	2.90E+05
	DeepCubeA$_g$	22.03	**100.00%**	35.00%	2.44E+06	41.99	5.67E+04
	FastDown (GC)	-	0.00%	0.0%	-	-	-
	FastDown (FF)	-	0.00%	0.0%	-	-	-
	FastDown (CG)	-	0.00%	0.0%	-	-	-
RC (Rand)	DeepCubeA$_g$	15.22	**99.40%**	-	1.91E+06	32.24	5.19E+04
	FastDown (GC)	7.18	32.80%	-	2.67E+06	13.79	1.41E+05
	FastDown (FF)	6.49	31.20%	-	4.87E+05	13.83	2.93E+04
	FastDown (CG)	7.85	33.80%	-	1.12E+06	11.62	5.81E+04
15-P (Canon)	PDBs	**52.02**	**100.00%**	**100.0%**	**3.22E+04**	**0.002**	**1.45E+07**
	DeepCubeA	52.03	**100.00%**	99.4%	3.85E+06	10.28	3.93E+05
	DeepCubeA$_g$	**52.02**	**100.00%**	**100.0%**	1.81E+05	2.61	6.94E+04
	FastDown (GC)	36.75	0.80%	0.80%	9.05E+07	102.11	8.66E+05
	FastDown (FF)	52.75	80.80%	24.80%	2.92E+06	42.11	6.93E+04
	FastDown (CG)	41.95	4.40%	1.20%	2.00E+07	80.58	2.47E+05
15-P (Rand)	DeepCubeA$_g$	**33.98**	**100.00%**	-	**1.11E+05**	**1.60**	**6.16E+04**
	FastDown (GC)	14.92	38.00%	-	1.61E+07	18.77	5.46E+05
	FastDown (FF)	32.66	89.20%	-	1.24E+06	17.39	5.65E+04
	FastDown (CG)	20.45	51.20%	-	3.90E+06	21.41	1.20E+05
24-P (Canon)	PDBs	**89.41**	**100.00%**	**100.00%**	8.19E+10	4239.54	**1.91E+07**
	DeepCubeA	89.49	**100.00%**	96.98%	6.44E+06	19.33	3.34E+05
	DeepCubeA$_g$	90.47	**100.00%**	55.24%	**3.38E+05**	**5.22**	6.48E+04
	FastDown (GC)	-	0.00%	0.00%	-	-	-
	FastDown (FF)	81.00	1.01%	0.40%	2.68E+06	89.84	2.91E+04
	FastDown (CG)	-	0.00%	0.00%	-	-	-
24-P (Rand)	DeepCubeA$_g$	66.28	**99.60%**	-	3.10E+05	4.91	6.16E+04
	FastDown (GC)	9.86	10.00%	-	9.54E+06	11.88	4.27E+05
	FastDown (FF)	26.35	26.00%	-	5.99E+05	19.57	2.41E+04
	FastDown (CG)	13.75	12.60%	-	1.42E+06	14.42	6.85E+04
Sokoban	DeepCubeA	32.88	**100.00%**	-	**5.01E+03**	2.71	1.84E+03
	DeepCubeA$_g$	**32.02**	**100.00%**	-	1.80E+04	0.95	1.79E+04
	FastDown (GC)	31.94	99.80%	-	3.17E+06	5.93	5.85E+05
	FastDown (FF)	33.15	**100.00%**	-	2.92E+04	**0.32**	**7.49E+04**
	FastDown (CG)	33.12	**100.00%**	-	4.43E+04	0.51	7.25E+04

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

ASP Specifications

- We build on this using answer set programming to describe goals with first-order logic and use answer set solvers to solve for assignments that make these goals true
- For the Rubik's cube
  - Define basic background knowledge
    - Colors, faces, cubelets
    - Constraints: Cannot have two stickers of the same color on the same cubelet, cannot have two stickers from the same cubelet on opposite faces
  - Given basic background knowledge, specifications often only require a few lines of code
    - `face_same(F) :- face_col(F, FCol), #count{Cbl : onface(Cbl, FCol, F)}=9.`
    - `canon_solved :- #count{F : face_same(F)}=6.`
  - Our specifications contain combinations of common patterns
    - Note: the training procedure is unaware of what the specification will be at test time



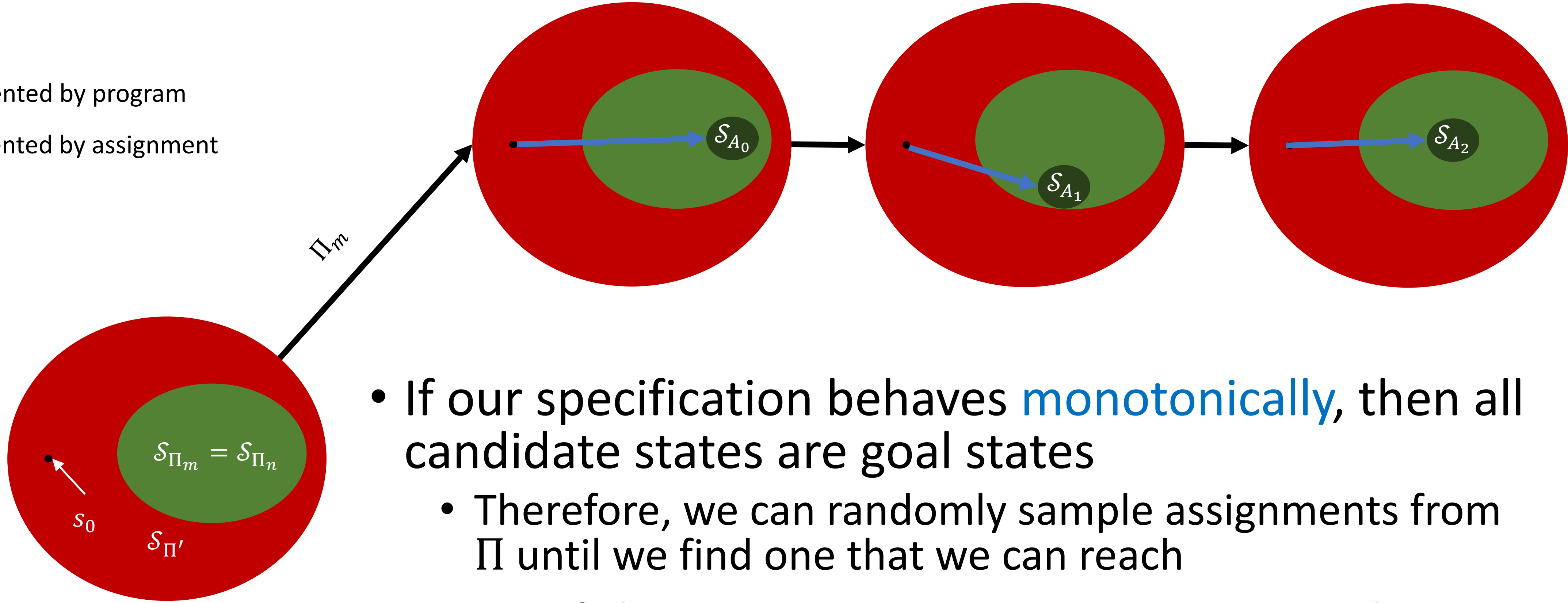
(a) Cross (b) X (c) Cup (d) Spot

Goal Reaching: Monotonic Specification

Π: Answer set program

$\mathcal{S}_\Pi$: set of states represented by program

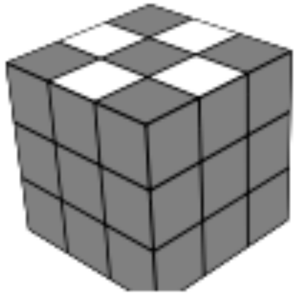
$\mathcal{S}_A$: set of states represented by assignment



- If our specification behaves monotonically, then all candidate states are goal states
  - Therefore, we can randomly sample assignments from Π until we find one that we can reach
- Some of these assignments may represent the empty set
- The answer set solver (we use clingo) used is agnostic to the cost of a shortest path

Handling Non-Monotonicity

- If negation as failure is used in a program, Π, then Π can exhibit non-monotonic behavior
  - A logic program is non-monotonic if some atoms that were previously derived can be retracted by adding new knowledge
  - Therefore, we can have a state that is a candidate state but not a goal state
- For example, a white cross with no yellow stickers on the white face
  - The assignment for this specification is just a white cross
  - However, there can be a state that is a specialization of this assignment, but has yellow on the white face

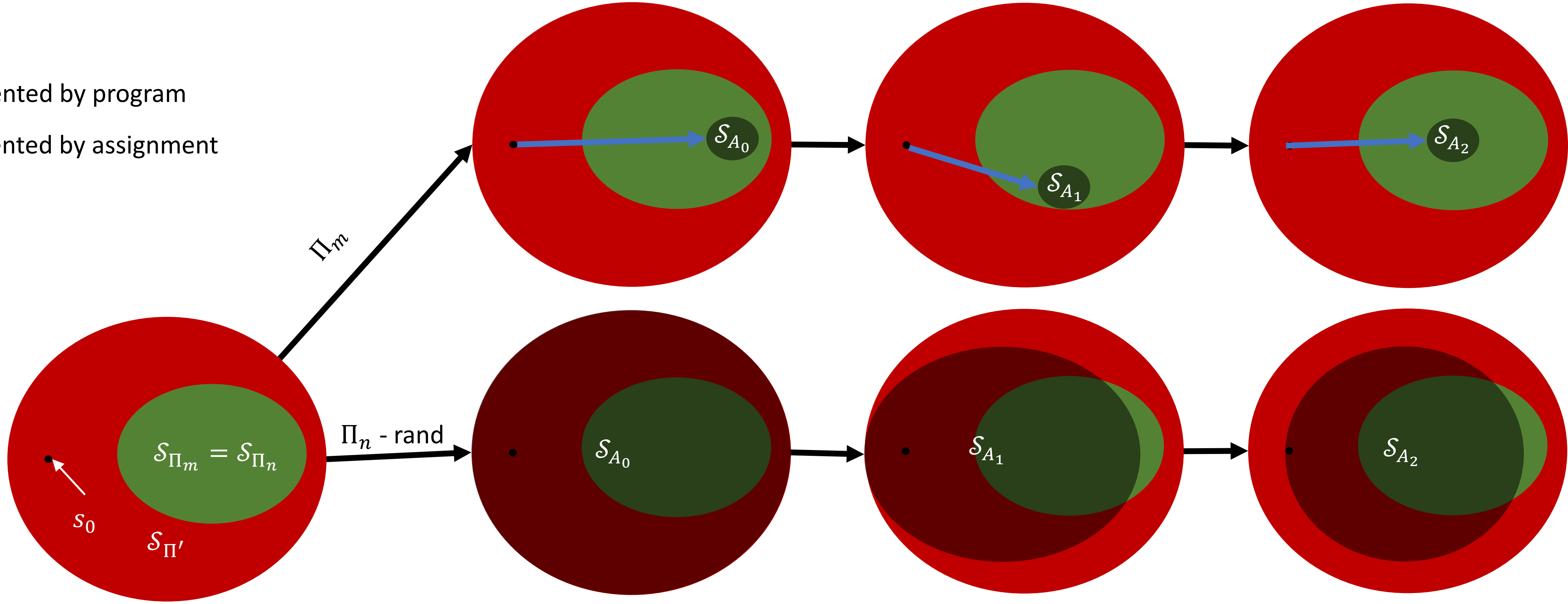


Goal Reaching: Non-monotonic

Π: Answer set program

$\mathcal{S}_\Pi$: set of states represented by program

$\mathcal{S}_A$: set of states represented by assignment

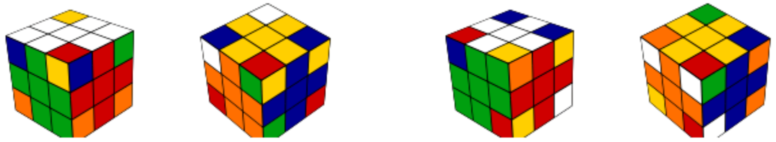


To reduce the size of candidate states while ensuring there is still at least one goal state, find another minimal assignment, A_2, such that

$$A \subset A_2$$
$$A_2 \in \alpha(\Pi)$$

Results

Goal	Path Cost	% Solved	# Models	Model Time	Search Time
Rubik's Cube (Canon)	24.41	100%	1	0.37	4.39
Rubik's Cube (Cross6)	13.11	100%	1	0.41	2.14
Rubik's Cube (Cup4)	24.33	100%	42.5	34.65	374.11
Rubik's Cube (CupSpot)	17.99	100%	27.68	38.66	241.08
Rubik's Cube (Checkers)	23.85	100%	1	0.49	4.2
Sokoban (Immov)	35.15	100%	6.37	6.83	16.16
Sokoban (BoxBox)	33.77	88%	1.89	0.58	6.08
Sokoban (AgentInBox)	34.42	77%	1.26	0.38	4.09



(a) Example 1 (b) Example 2

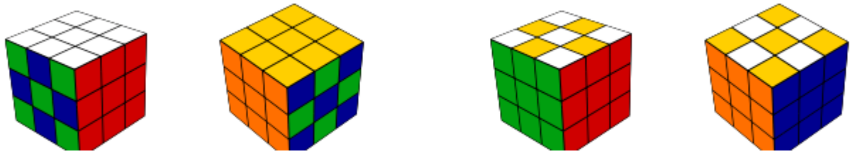
Cross6

(a) Example 1 (b) Example 2

Cup4

(a) Example 1 (b) Example 2

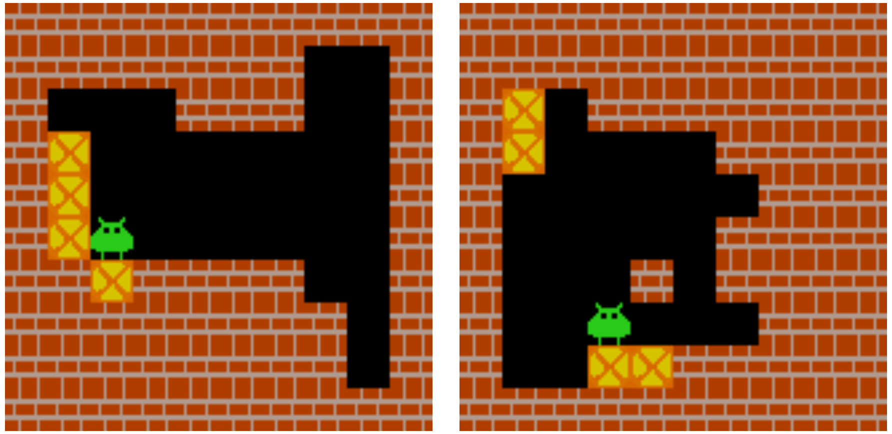
CupSpot



Checkers

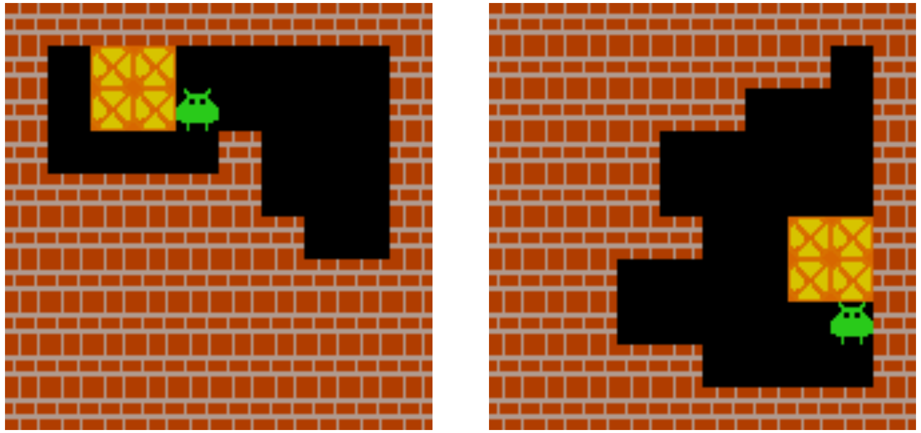
Agostinelli, Forest, Rojina Panta, and Vedant Khandelwal. "Specifying Goals to Deep Neural Networks with Answer Set Programming." *ICAPS 2024*

Results

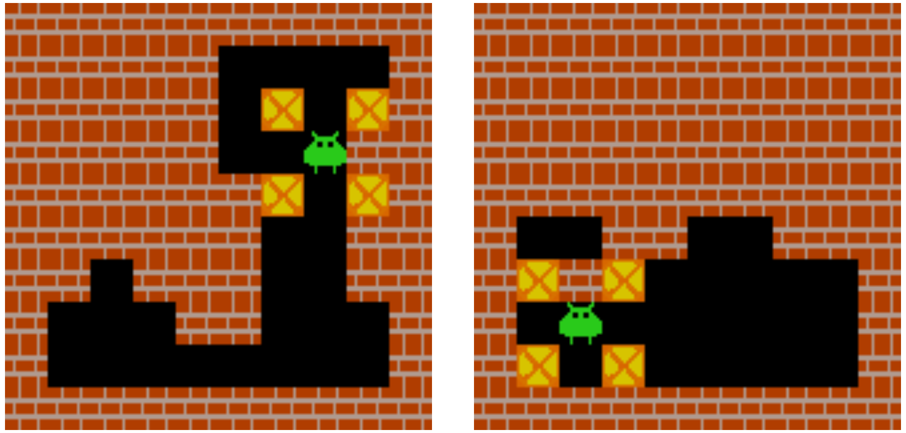


All boxes are immoveable

Goal	Path Cost	% Solved	# Models	Model Time	Search Time
Rubik's Cube (Canon)	24.41	100%	1	0.37	4.39
Rubik's Cube (Cross6)	13.11	100%	1	0.41	2.14
Rubik's Cube (Cup4)	24.33	100%	42.5	34.65	374.11
Rubik's Cube (CupSpot)	17.99	100%	27.68	38.66	241.08
Rubik's Cube (Checkers)	23.85	100%	1	0.49	4.2
Sokoban (Immov)	35.15	100%	6.37	6.83	16.16
Sokoban (BoxBox)	33.77	88%	1.89	0.58	6.08
Sokoban (AgentInBox)	34.42	77%	1.26	0.38	4.09



A box of boxes



Boxes at the four corners of the agent

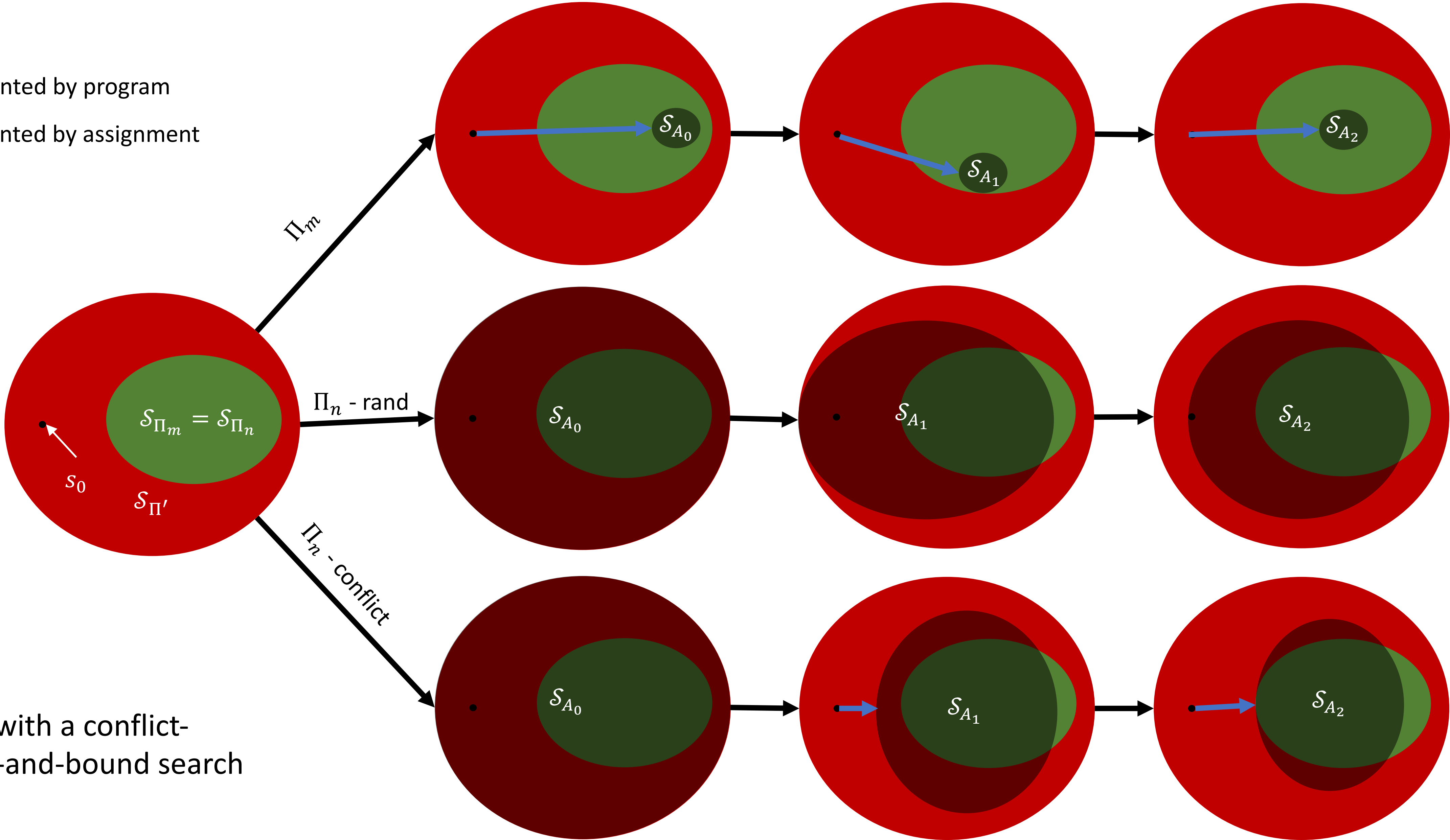
Agostinelli, Forest, Rojina Panta, and Vedant Khandelwal. "Specifying Goals to Deep Neural Networks with Answer Set Programming." *ICAPS 2024*

Goal Reaching: Non-monotonic

Π: Answer set program

$\mathcal{S}_{\Pi}$: set of states represented by program

$\mathcal{S}_{A}$: set of states represented by assignment

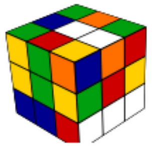


Combine this with a conflict-driven branch-and-bound search

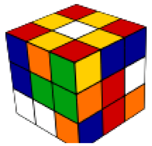
Results

Goal	SpecOp	Cost	%Solve	#Itr	#Assign	%reach	%not goal	Secs/Spec	Secs/Path	Secs
RC:∀diffCtrW	-	11.54	70	**3.34**	**33.43**	7.68	**0**	12.77	7.5	564.94
RC:¬∃sameCtrW	Rand	1.67	99	7.2	63.02	87.84	69.06	**0.06**	1.04	95.46
	Conflict	**1.26**	**100**	5.43	36.31	**99.34**	52.36	**0.06**	**0.07**	**5.98**
24p:r0SumEven	-	24.55	**100**	9.24	92.4	**100**	**0**	**0.2**	0.23	42.52
24p:¬r0SumOdd	Rand	3.16	**100**	4.27	33.6	**100**	38.71	**0.2**	**0.03**	6.64
	Conflict	**2.51**	**100**	**4.06**	**31.6**	**100**	22.13	0.21	0.04	**6.58**
24p:∀rSumEven	-	83.71	**100**	9.19	91.9	50.41	**0**	0.88	1.77	250.18
24p:¬∃rSumOdd	Rand	17.07	**100**	10.23	92.05	99.98	85.51	**0.1**	**0.08**	21.72
	Conflict	**12.87**	**100**	**8.66**	**77.1**	**100**	79.72	0.11	**0.08**	**17.08**

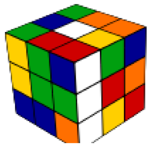
All stickers on the white face are different than the center sticker



Start



Mono: path cost 12



Non-mono: path cost 1

All rows sum to an even number

12	22	6	9	5
7	1	19	2	17
16	13	4	20	21
11	15	10	3	8
	14	18	24	23

Start

17	10	20	5	22
1	6	14	15	16
12	13	23		8
11	3	9	4	7
18	19	2	21	24

Mono: path cost 93

12	22	6	9	5
7	1	19	2	17
16	13	4	20	21
11	15	10		8
14	18	24	3	23

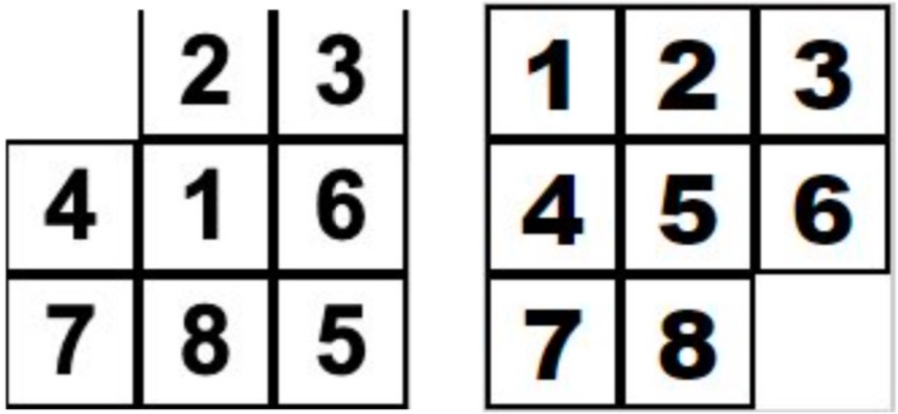
Non-mono: path cost 4

Outline

- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

Vedant Khandelwal

Example



Start state

Goal state

- If using only canonical actions, the cost-to-go is 16
- If including diagonal actions, the cost-to-go is 2
- To differentiate between these two scenarios, information about the domain must also be given to the heuristic function

Training

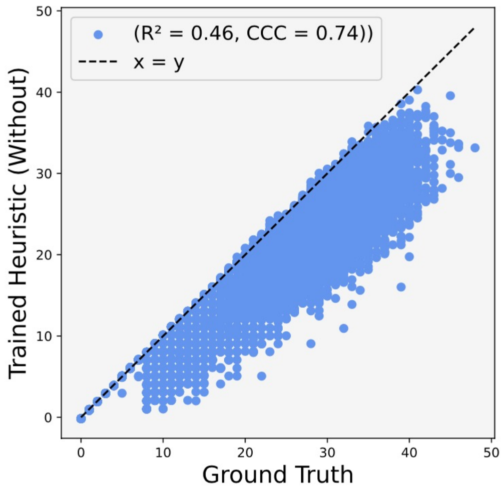
- For each example, randomly sample a domain
- For that domain, randomly sample a state
- RL Update
  - $L(\theta) = \left(\min_a (c^a(s) + h_{\theta^-}(T(s,a), D)) - h_\theta(s, D)\right)^2$
  - D: Domain

Preliminary Experiments

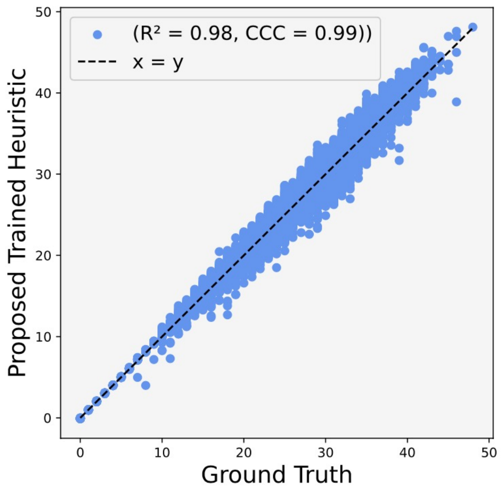
- For the 15-puzzle, generate different domains by sampling a subset of {U, D, L, R, UL, UR, DL, DR} actions for each tile position
  - 8 actions for each of the 16 positions, max $2^{8*16} \approx 3.4\times10^{38}$ domains
  - Ensure all sampled domains are reversible, for simplicity
- Represent the domain as a one-hot vector of which actions are allowed in each position
- Compare heuristic performance with true cost-to-go for random states from domains
  - True cost-to-go computed with merge-and-shrink heuristic
- Compare when training a heuristic function across domains without domain information
- Compare heuristic function with DeepCubeA trained for a fixed domain

Results

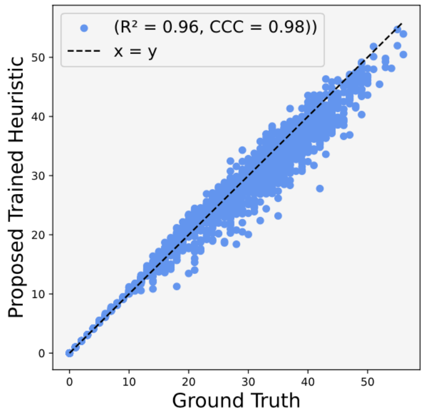
- Adding action information significantly improves performance
- Performs slightly worse when compared to DeepCubeA trained on that specific domain
    - However, unlike DeepCubeA, it does not need to be re-trained for that domain



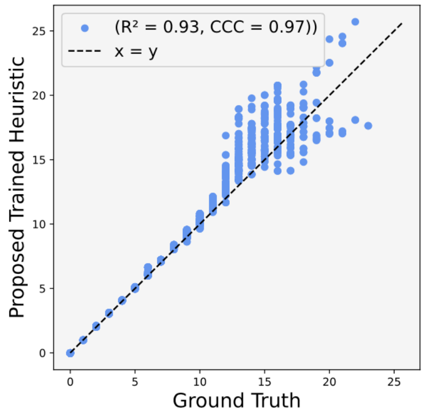
(a) Without Action Info



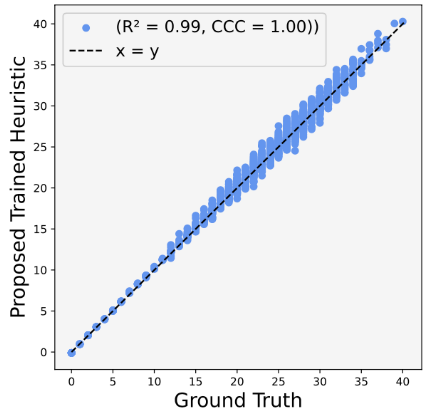
(b) With Action Info



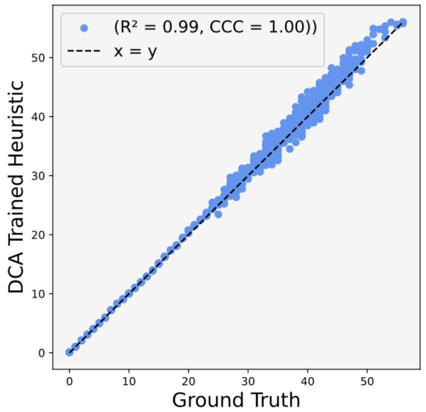
(a) C: P vs GT



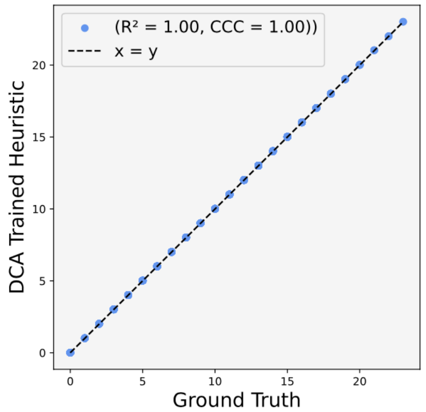
(b) D: P vs GT



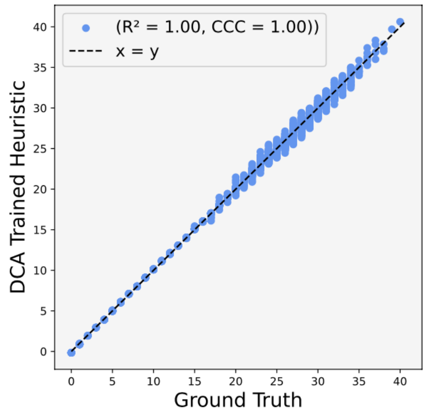
(c) C+D: P vs GT



(d) C: DCA vs GT



(e) D: DCA vs GT



(f) C+D: DCA vs GT

Khandelwal, Vedant, Amit Sheth, Forest Agostinelli. "Towards Learning Foundation Models for Heuristic Functions to Solve Pathfinding Problems." *arxiv, 2024*

Results

- Repeat training for 8-puzzle and 24-puzzle
- Proposed approach compares favorably to the fast downward planner with the fast forward heuristic
- Is slightly worse than DeepCubeA, which must be re-trained for each domain
- Future work could build on work by Felipe Trevizan and Sylvie Thiebaux on using graph neural networks to encode PDDL domains

Domain	Solver	Len	Opt	Nodes	Secs	Nodes/Sec	Solved
8 Puzzle (**C**)	DeepCubeA	**18.38**	**100%**	3.59E+04	0.69	**5.2E+04**	**100%**
8 Puzzle (**C**)	Proposed	**18.38**	**100%**	7.17E+04	1.76	4.07E+04	**100%**
8 Puzzle (**C**)	FD(FF)	18.8	81%	**5.56E+02**	**0.11**	4.7E+03	**100%**
8 Puzzle (**D**)	DeepCubeA	**1.44**	**100%**	1.95E+01	**0.01**	2.92E+03	**100%**
8 Puzzle (**D**)	Proposed	**1.44**	**100%**	4.05E+01	**0.01**	**4.92E+03**	**100%**
8 Puzzle (**D**)	FD(FF)	**1.44**	**100%**	**2.45E+00**	0.2	1.23E+01	**100%**
8 Puzzle (**C+D**)	DeepCubeA	**11.84**	**100%**	6.2E+04	**1.18**	**5.26E+04**	**100%**
8 Puzzle (**C+D**)	Proposed	**11.84**	**100%**	6.23E+04	1.56	3.97E+04	**100%**
8 Puzzle (**C+D**)	FD(FF)	12.9	54.2%	**8.68E+01**	**0.13**	6.59E+02	**100%**
15 Puzzle (**C**)	DeepCubeA	**52.03**	**99.4%**	**1.82E+05**	**4.31**	4.21E+04	**100%**
15 Puzzle (**C**)	Proposed	52.18	93.76%	3.62E+05	10.39	3.49E+04	**100%**
15 Puzzle (**C**)	FD(FF)	52.75	24.80	2.92E+06	42.11	**6.93E+04**	80.80%
15 Puzzle (**D**)	DeepCubeA	**10.8**	**100%**	8.2E+02	**0.03**	2.43E+04	**100%**
15 Puzzle (**D**)	Proposed	10.81	99.8%	1.64E+03	0.05	**3.01E+04**	**100%**
15 Puzzle (**D**)	FD(FF)	10.86	96.8%	**4.18E+01**	0.21	1.96E+02	**100%**
15 Puzzle (**C+D**)	DeepCubeA	**25.66**	**100%**	1.78E+05	3.74	**4.78E+04**	**100%**
15 Puzzle (**C+D**)	Proposed	25.67	99.8%	1.78E+05	4.72	3.78E+04	**100%**
15 Puzzle (**C+D**)	FD(FF)	29.32	13.4%	**8.4E+03**	**1.17**	3.56E+03	**100%**
24 Puzzle (**C**)	DeepCubeA	**89.48**	**96.98%**	**3.34E+05**	**8.05**	**4.15E+04**	**100%**
24 Puzzle (**C**)	Proposed	92.80	22.03%	7.6E+05	24.06	3.16E+04	**100%**
24 Puzzle (**C**)	FD(FF)	81.00	0.40	2.68E+06	89.84	2.91E+04	1.01%
24 Puzzle (**D**)	DeepCubeA	**14.9**	**100%**	2.55E+04	**0.47**	**5.46E+04**	**100%**
24 Puzzle (**D**)	Proposed	14.92	99.8%	5.1E+04	1.35	3.78E+04	**100%**
24 Puzzle (**D**)	FD(FF)	15.16	89.2%	**2.64E+02**	**0.12**	2.05E+03	**100%**
24 Puzzle (**C+D**)	DeepCubeA	**31.33**	**100%**	2.27E+05	**4.83**	**4.69E+04**	**100%**
24 Puzzle (**C+D**)	Proposed	31.34	99.6%	2.27E+05	6.78	3.34E+04	**100%**
24 Puzzle (**C+D**)	FD(FF)	36.81	13.8%	**1.7E+04**	5.35	1.77E+03	99.4%

Khandelwal, Vedant, Amit Sheth, Forest Agostinelli. “Towards Learning Foundation Models for Heuristic Functions to Solve Pathfinding Problems.” *arxiv, 2024*
Chen, Dillon Z., Sylvie Thiébaux, and Felipe Trevizan. "Learning Domain-Independent Heuristics for Grounded and Lifted Planning." *AAAI*. Vol. 38. No. 18. 2024.

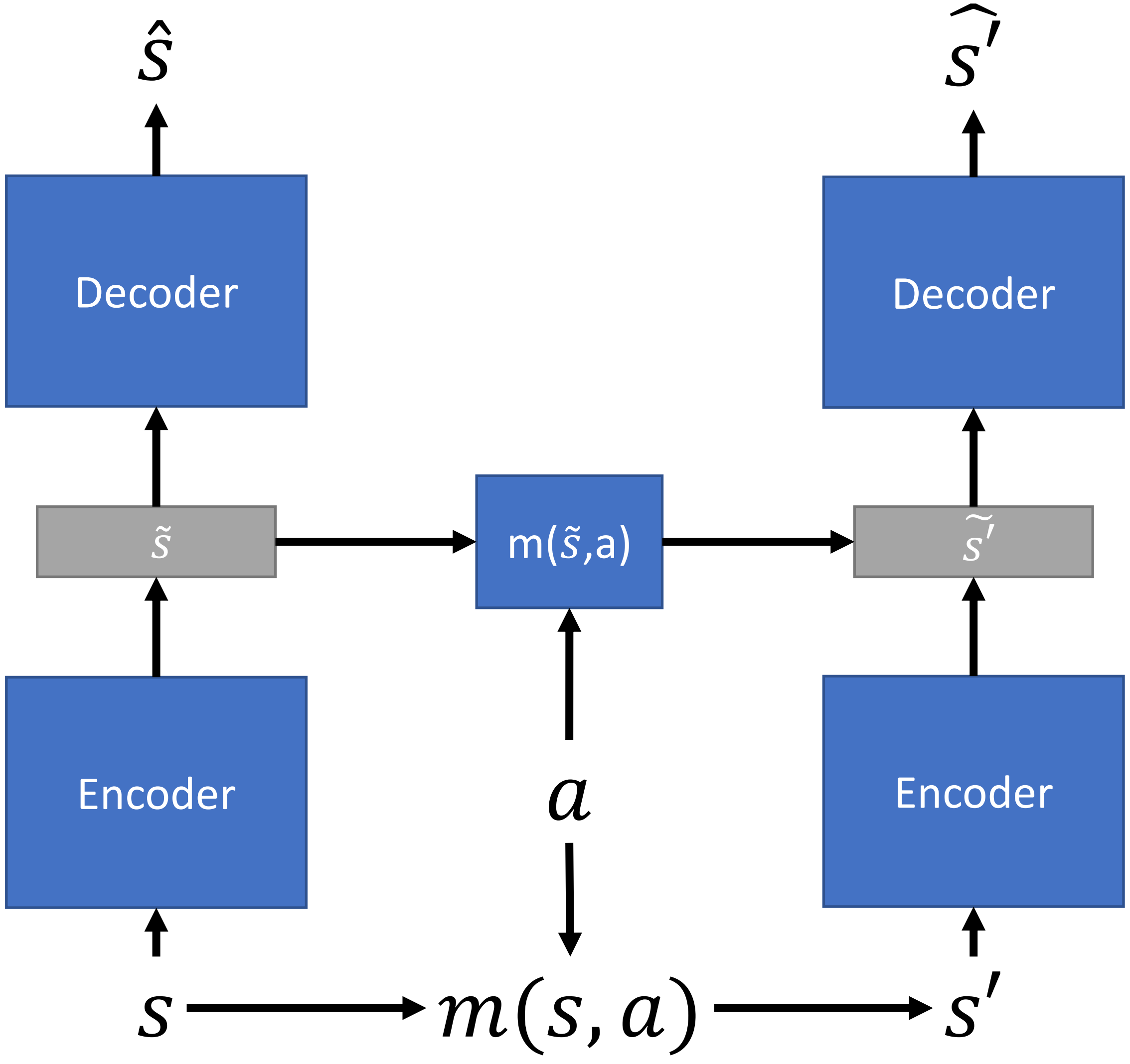
Outline

- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

Misagh Soltani

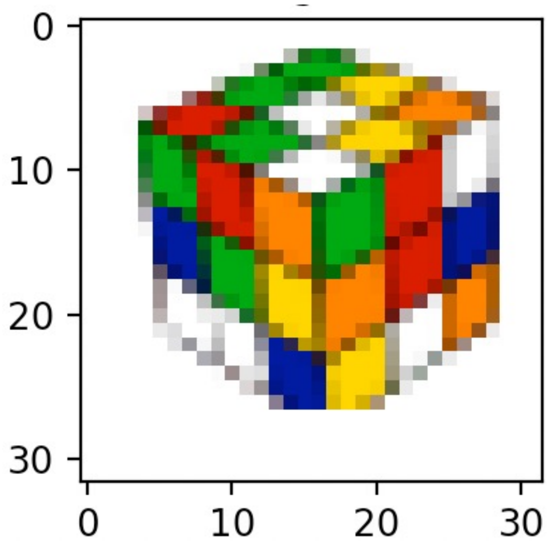
Learning Discrete World Models

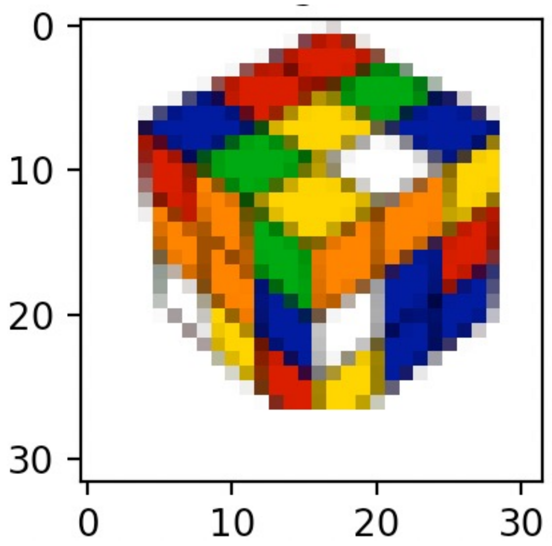
- Addressing previous shortcomings
  - Small errors in prediction can be corrected by simply rounding
  - Can reidentify states by comparing two vectors
- Encoder
  - Maps the state to a discrete representation
  - To allow training with gradient descent, use a straight through estimator
- Decoder
 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

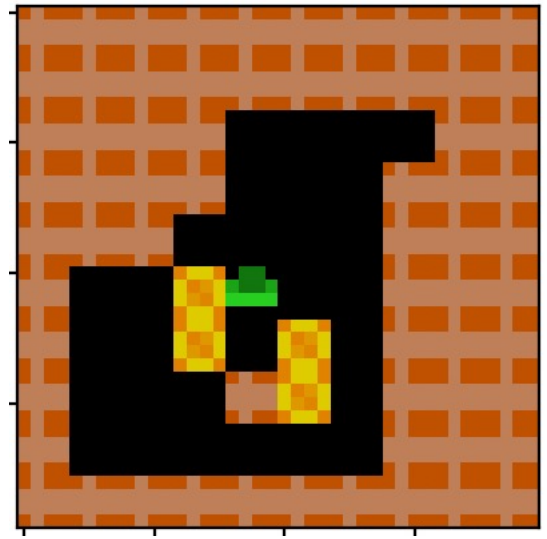


Experiments

- Rubik's cube
  - Two 32x32 RGB images showing both sides of the cube
- Sokoban
  - One 40x40 RGB image
- Generate offline dataset of 10,000 episodes of 30 random steps, each

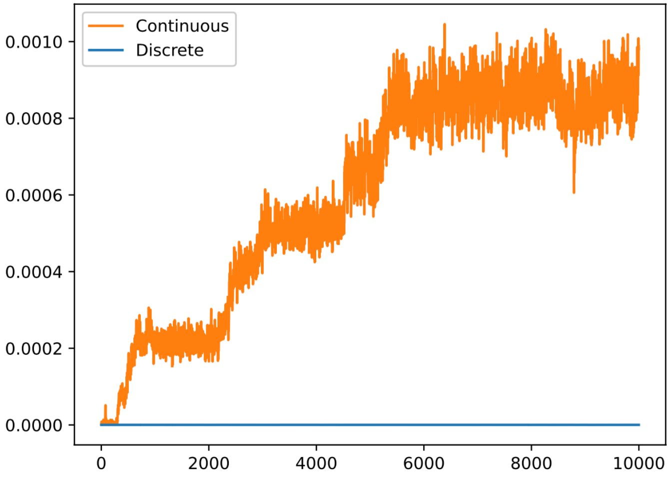




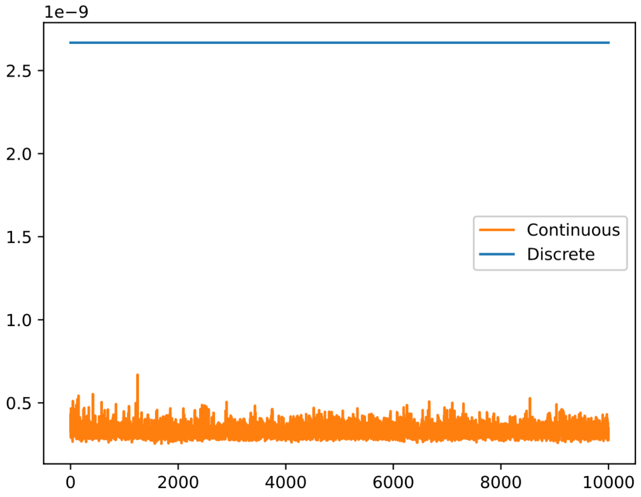


Discrete vs Continuous Model Performance

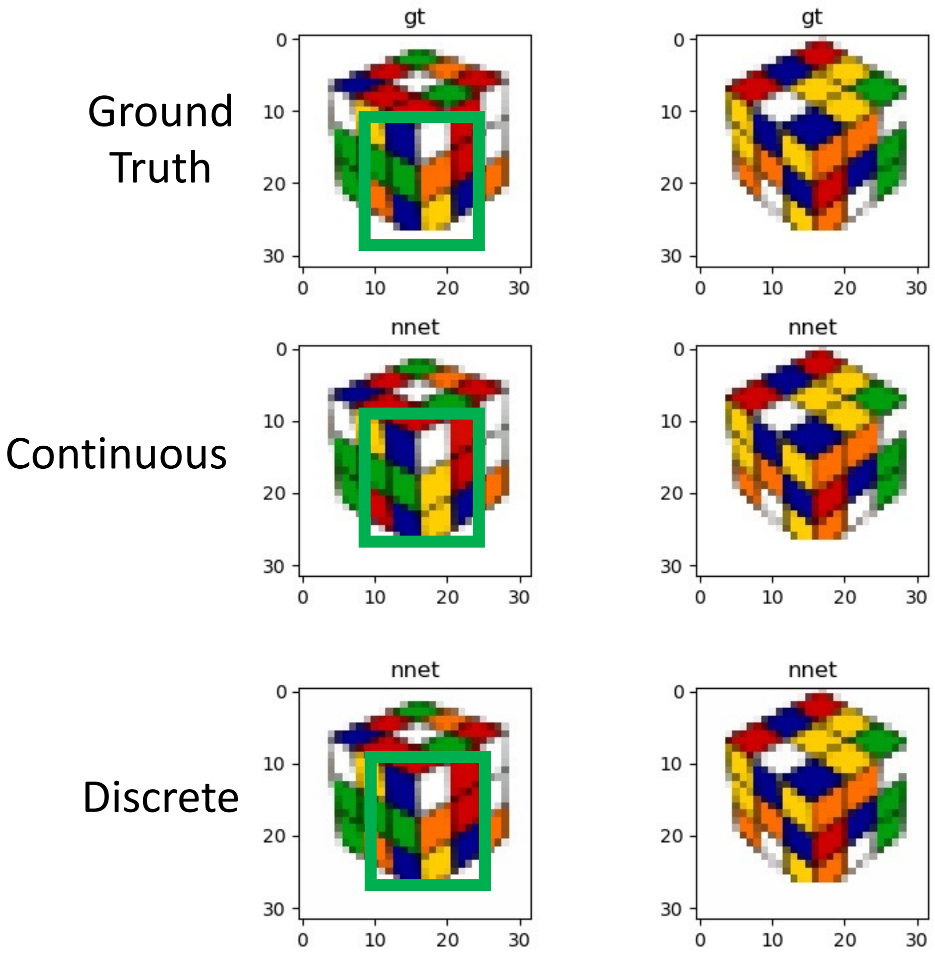
- The continuous model eventually accumulates error for the Rubik's cube



(a) Rubik's Cube



(b) Sokoban



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*

Outline

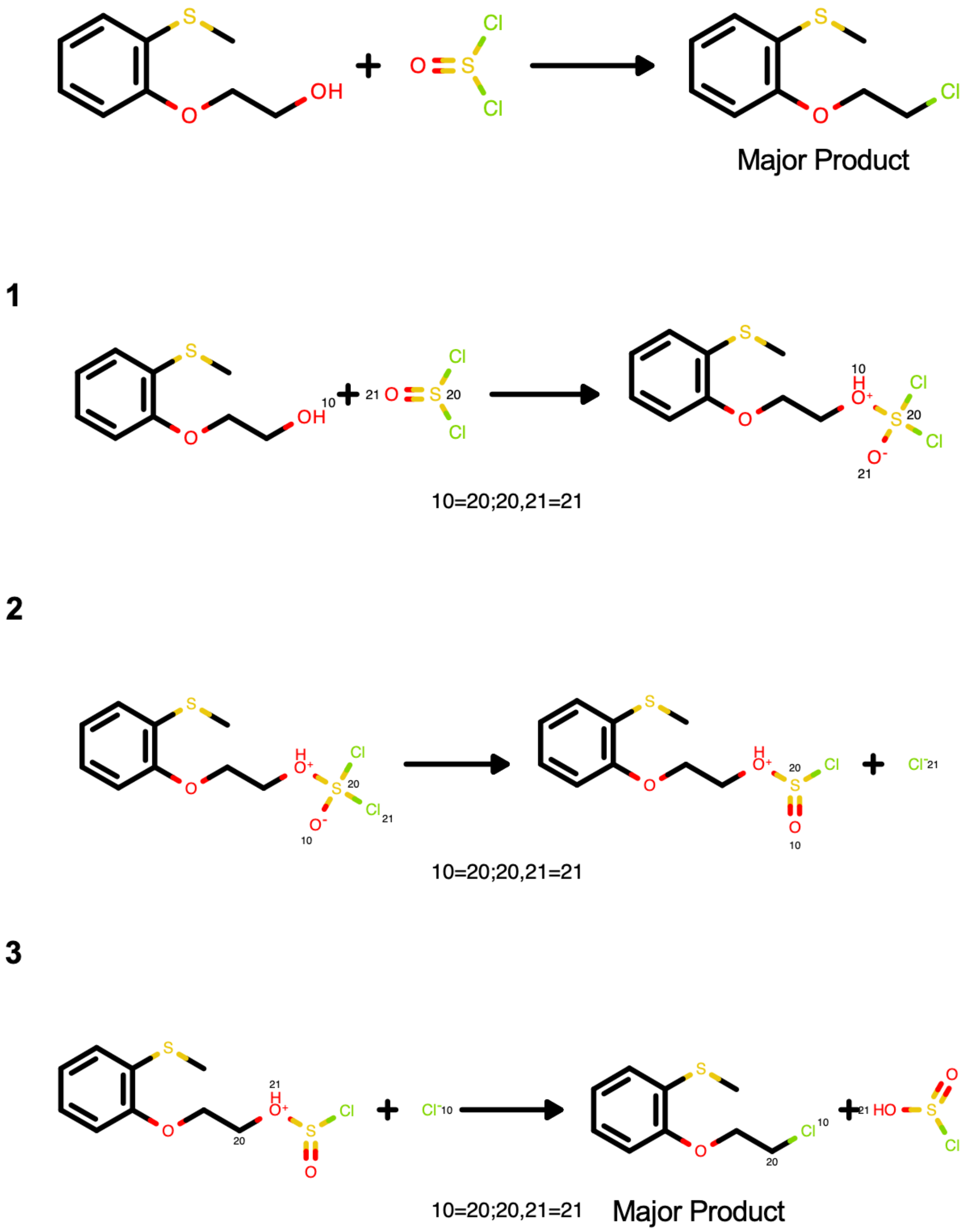
- Background
- Approximate value iteration and batch weighted A* search
- Approximate Q-learning and batch weighted Q* seaerch
- Generalization
  - Generalizing over goals
  - Generalizing over domains
  - Generalizing to domains with unknown transition functions
- Applications

Rojina Panta

Christian Geils

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

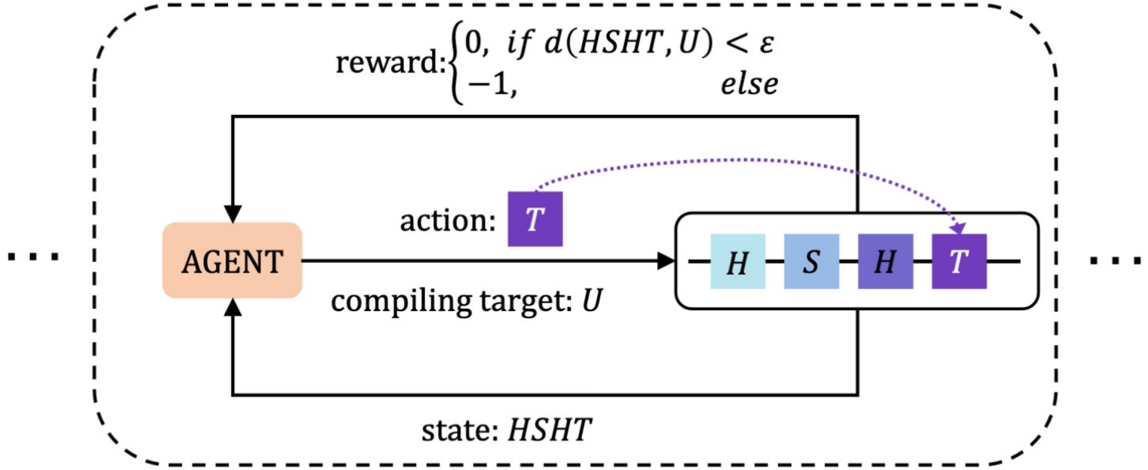
Step/s	Solver	Path Cost	% Solved	Nodes	Secs	Nodes/Sec
Steps=0	DeepCubeA	0.00	**100.00%**	3.09E+2	3.87	79.97
	Uniform Cost Search	0.00	**100.00%**	3.09E+2	4.61	67.13
	Tanimoto Similarity	0.00	**100.00%**	3.09E+2	3.71	83.42
Steps=1	DeepCubeA	1.00	**100.00%**	7.49E+2	9.70	77.26
	Uniform Cost Search	1.00	**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*

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

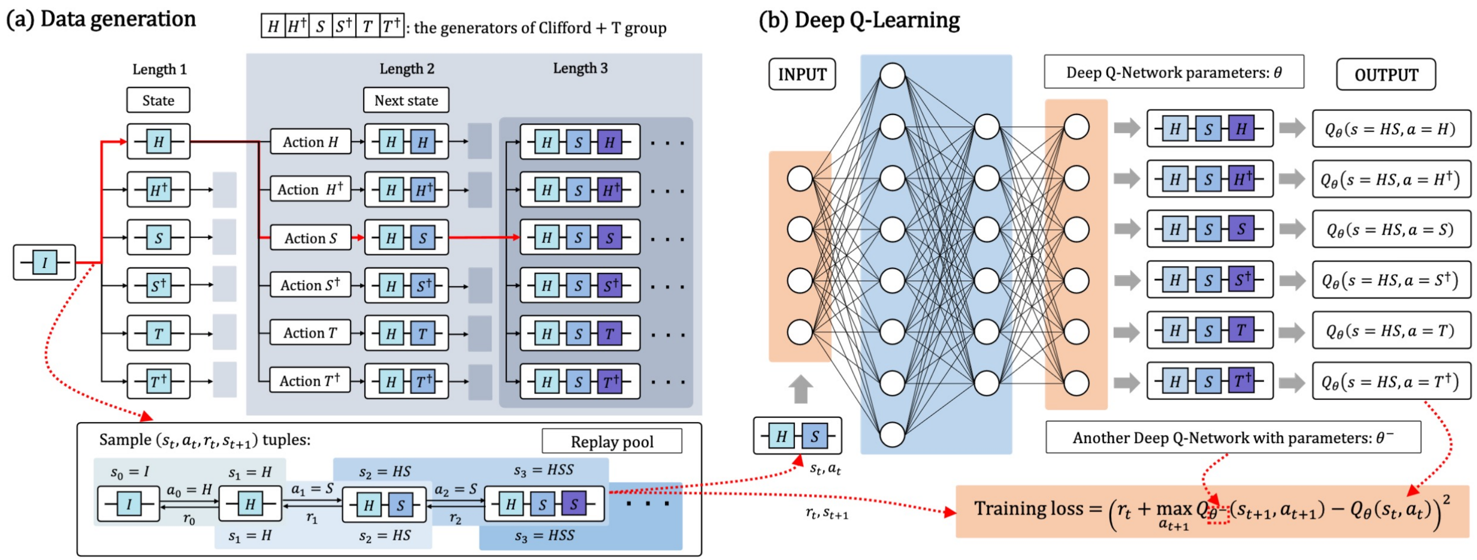
Quantum compiling in the framework of RL

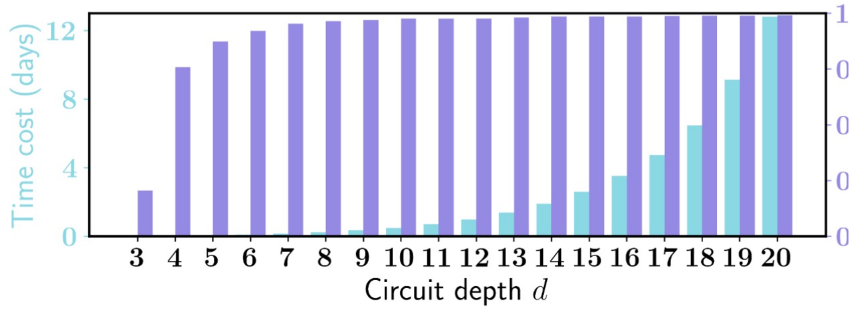


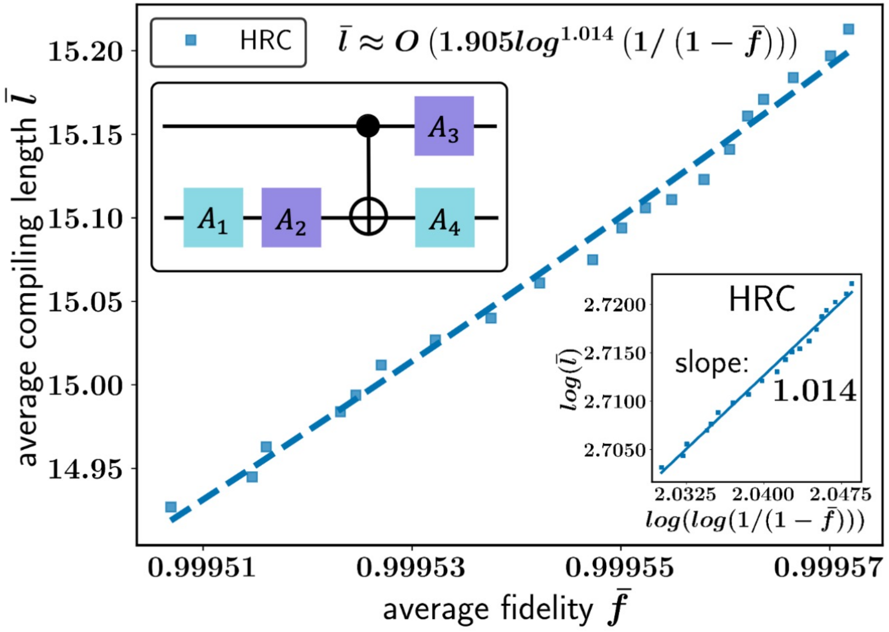
The Markov Decision Process	
action space $\mathcal{A}$	hardware-specific universal basis set
state space $\mathcal{S}$	the circuits composed of $\{A_j; A_j \in \mathcal{A}\}$
reward function r	reward $r = \begin{cases} 0, & if\ d(\prod_{j=0}^{L-1} A_j, U) < \varepsilon \\ -1, & else \end{cases}$
transition 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
- Accuracy increases given more time for synthesis





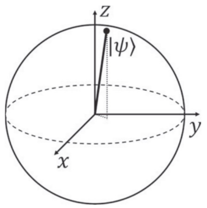
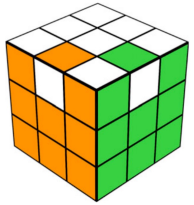


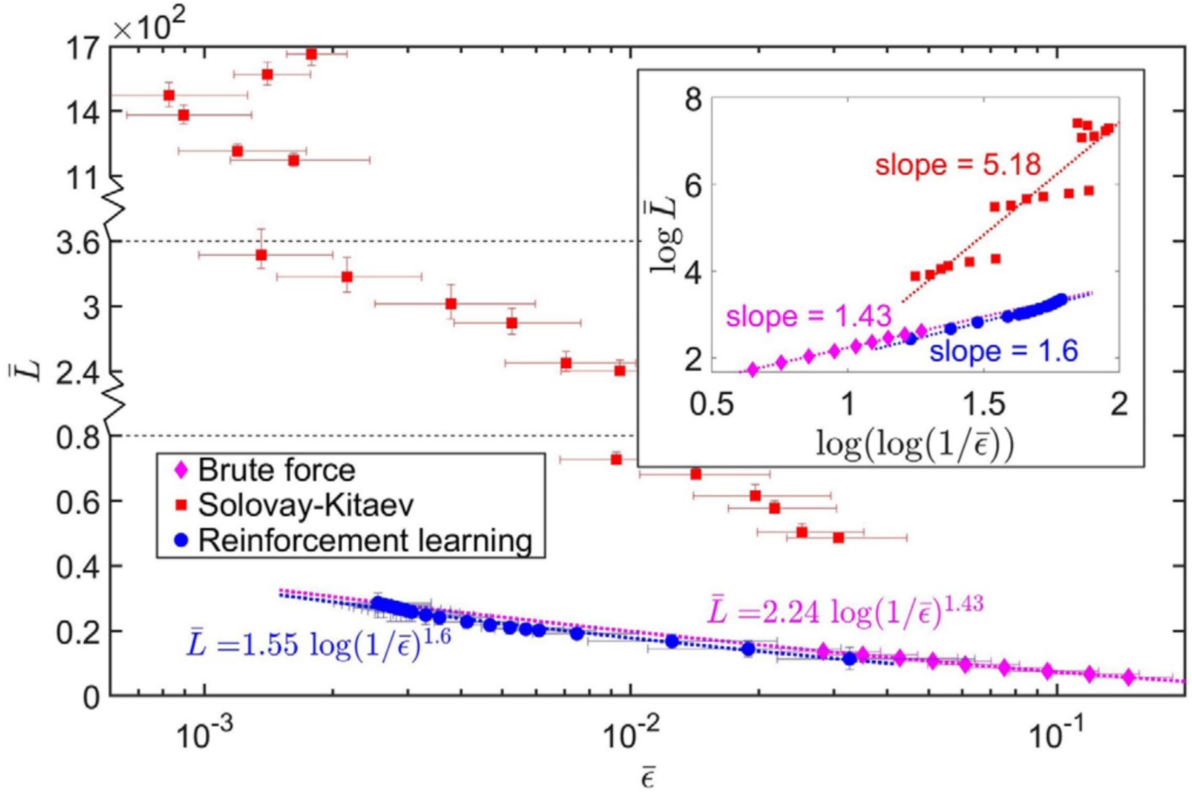
Quantum compilation on two-qubit universal basis set

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.

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, et al. "Q* Search: Heuristic Search with Deep Q-Networks." *ICAPS PRL Workshop 2024*
  - Agostinelli, Forest, Rojina Panta, and Vedant Khandelwal. "Specifying Goals to Deep Neural Networks with Answer Set Programming." *ICAPS 2024*
  - Agostinelli, Forest. "A Conflict-Driven Approach for Reaching Goals Specified with Negation as Failure." *ICAPS 2024 HAXP Workshop*
  - Khandelwal, Vedant, Amit Sheth, Forest Agostinelli. "Towards Learning Foundation Models for Heuristic Functions to Solve Pathfinding Problems." *arxiv, 2024*
  - Agostinelli, Forest and Soltani, Misagh "Learning Discrete World Models for Heuristic Search." *Reinforcement Learning Conference 2024*
  - Panta, Rojina, et al. "Finding Reaction Mechanism Pathways with Deep Reinforcement Learning and Heuristic Search." *ICAPS PRL Workshop 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/