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* search
• Generalization
  • Generalizing over goals
  • Generalizing over domains
  • Generalizing to domains with unknown transition functions
• Applications
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

![Diagram of Pathfinding Domains]
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?
• Background
• Approximate value iteration and batch weighted A* search
• Approximate Q-learning and batch weighted Q* search
• 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
- \( \theta \): parameters
- \( \theta^- \): parameters for target network
  - Is periodically updated to \( \theta \) 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 $3.0 \times 10^{62}$ (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
  \[ \pi(s) = \arg\min_a (c^a(s) + h_\theta(T(s, a))) \]
- 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

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/

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

Outline

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

• \( Q(s, a) = Q(s, a) + \alpha [c^a(s) + \min_{a'} Q(T(s, a), a') - Q(s, a)] \)
• \( \alpha \) 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

  \[ \pi(a \mid s) = \frac{e^{-h_{\theta}(s,a)/T}}{\sum_{a'=1}^{\mid A \mid} e^{-h_{\theta}(s,a')/T}} \]
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 $\geq$ 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

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

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

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

• Background
• Approximate value iteration and batch weighted A* search
• Approximate Q-learning and batch weighted Q* search
• 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\textsubscript{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
Our experiments use 100 start states from the test

<table>
<thead>
<tr>
<th>Puzzle</th>
<th>Solver</th>
<th>Path Cost</th>
<th>% Solved</th>
<th>% Opt</th>
<th>Nodes</th>
<th>Secs</th>
<th>Nodes/Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-P (Canon)</td>
<td>PDBs+</td>
<td>20.67</td>
<td>100.00%</td>
<td>100.00%</td>
<td>2.05E+06</td>
<td>2.20</td>
<td>1.79E+06</td>
</tr>
<tr>
<td></td>
<td>DeepCubeA</td>
<td>21.50</td>
<td>100.00%</td>
<td>60.3%</td>
<td>6.62E+06</td>
<td>24.22</td>
<td>2.90E+05</td>
</tr>
<tr>
<td></td>
<td>DeepCubeA_g</td>
<td>22.03</td>
<td>100.00%</td>
<td>35.00%</td>
<td>2.44E+06</td>
<td>41.99</td>
<td>5.67E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (GC)</td>
<td>-</td>
<td>0.00%</td>
<td>0.0%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FastDown (FF)</td>
<td>-</td>
<td>0.00%</td>
<td>0.0%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FastDown (CG)</td>
<td>-</td>
<td>0.00%</td>
<td>0.0%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>15-P (Rand)</td>
<td>DeepCubeA_g</td>
<td>15.22</td>
<td>99.40%</td>
<td>-</td>
<td>1.91E+06</td>
<td>32.24</td>
<td>5.19E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (GC)</td>
<td>7.18</td>
<td>32.80%</td>
<td>-</td>
<td>2.67E+06</td>
<td>13.79</td>
<td>1.41E+05</td>
</tr>
<tr>
<td></td>
<td>FastDown (FF)</td>
<td>6.49</td>
<td>31.20%</td>
<td>-</td>
<td>4.87E+05</td>
<td>13.83</td>
<td>2.93E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (CG)</td>
<td>7.85</td>
<td>33.80%</td>
<td>-</td>
<td>1.12E+06</td>
<td>11.62</td>
<td>5.81E+04</td>
</tr>
<tr>
<td></td>
<td>PDBs</td>
<td>52.02</td>
<td>100.00%</td>
<td>100.00%</td>
<td>3.22E+04</td>
<td>0.002</td>
<td>1.45E+07</td>
</tr>
<tr>
<td>24-P (Canon)</td>
<td>DeepCubeA</td>
<td>52.02</td>
<td>100.00%</td>
<td>100.00%</td>
<td>1.81E+05</td>
<td>2.61</td>
<td>6.94E+04</td>
</tr>
<tr>
<td></td>
<td>DeepCubeA_g</td>
<td>36.75</td>
<td>80.80%</td>
<td>24.80%</td>
<td>2.92E+06</td>
<td>42.11</td>
<td>6.93E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (GC)</td>
<td>52.75</td>
<td>80.80%</td>
<td>1.20%</td>
<td>2.00E+07</td>
<td>80.58</td>
<td>2.47E+05</td>
</tr>
<tr>
<td></td>
<td>FastDown (FF)</td>
<td>41.95</td>
<td>4.40%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FastDown (CG)</td>
<td>33.98</td>
<td>100.00%</td>
<td>-</td>
<td>1.11E+05</td>
<td>1.60</td>
<td>6.16E+04</td>
</tr>
<tr>
<td>24-P (Rand)</td>
<td>DeepCubeA_g</td>
<td>36.75</td>
<td>80.80%</td>
<td>24.80%</td>
<td>2.92E+06</td>
<td>42.11</td>
<td>6.93E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (GC)</td>
<td>52.75</td>
<td>80.80%</td>
<td>1.20%</td>
<td>2.00E+07</td>
<td>80.58</td>
<td>2.47E+05</td>
</tr>
<tr>
<td></td>
<td>FastDown (FF)</td>
<td>41.95</td>
<td>4.40%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>24-P (Rand)</td>
<td>DeepCubeA_g</td>
<td>36.75</td>
<td>80.80%</td>
<td>24.80%</td>
<td>2.92E+06</td>
<td>42.11</td>
<td>6.93E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (GC)</td>
<td>52.75</td>
<td>80.80%</td>
<td>1.20%</td>
<td>2.00E+07</td>
<td>80.58</td>
<td>2.47E+05</td>
</tr>
<tr>
<td>Sokoban</td>
<td>DeepCubeA</td>
<td>32.88</td>
<td>100.00%</td>
<td>-</td>
<td>5.01E+03</td>
<td>2.71</td>
<td>1.84E+03</td>
</tr>
<tr>
<td></td>
<td>DeepCubeA_g</td>
<td>32.02</td>
<td>100.00%</td>
<td>-</td>
<td>1.80E+04</td>
<td>0.95</td>
<td>1.79E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (GC)</td>
<td>31.94</td>
<td>99.80%</td>
<td>-</td>
<td>3.17E+06</td>
<td>5.93</td>
<td>5.85E+05</td>
</tr>
<tr>
<td></td>
<td>FastDown (FF)</td>
<td>33.15</td>
<td>100.00%</td>
<td>-</td>
<td>2.92E+04</td>
<td>0.32</td>
<td>7.49E+04</td>
</tr>
<tr>
<td></td>
<td>FastDown (CG)</td>
<td>33.12</td>
<td>100.00%</td>
<td>-</td>
<td>4.43E+04</td>
<td>0.51</td>
<td>7.25E+04</td>
</tr>
</tbody>
</table>

- **Canon**: Canonical goal states
- **Rand**: Random assignment selected as goal
  - Can be as small as the empty assignment
  - Methods that require a pre-defined goal cannot be applied to this scenario without considerable overhead

- **PDBs+**: Also includes group theory knowledge
- **DeepCubeA_g** consistently outperforms fastdownward in terms of percentage of states solved
For the Rubik’s cube 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

Figure 2: Examples of patterns that are combined to create goals.

Figure 3: Achieved goal of having a cross on all 6 faces.

Figure 4: Achieved goal of having cups on red, green, blue, where the center cubelet and cross are the same color.

Figure 5: Achieved goal of having a cup adjacent to a spot.

Figure 6: Achieved goal of having two checkerboards on opposite faces with all of the other faces the same.

Table 1: The time it takes to find stable models for each goal, shown in Table 1.
If our specification behaves **monotonically**, then all candidate states are goal states

- Therefore, we can randomly sample assignments from $\Pi$ 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
• 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
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

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

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

Table 2: Performance of DeepCubeA when reaching goals specified with ASP.

Discussion
In Table 2 we see that the path cost for finding the Cross6 goal is almost half that of finding the canonical goal, even though the canonical goal is a subset of the Cross6 goal. This indicates that the trained heuristic function is capable of estimating the cost-to-go to a closest state in a set of goal states without needing to be explicitly told of a closest state. This ability to discover paths to goal states which, themselves, are not known until a path is found, could be extended to domains such as chemical synthesis. For example, this would allow practitioners to specify properties a molecule should or should not have, discover synthesis routes to such molecules and, as a result, discover molecules that meet these specifications.

In Algorithm 1, we sample a new stable model if we fail to find a goal state. From Table 2, we see that the number of models we need to sample for the canonical Rubik’s cube goal state and Cross6 is only one. However, for Cup4 and CupSpot, we must sample, on average, 42.5 and 27.68 models, respectively, to find a goal state. In cases where a goal state was not found, A* search failed to find a path to the sampled stable model. This may be because the sampled stable models represented only unreachable states. We discuss ways to overcome this in the Future Work Section.

For Sokoban, we see that the BoxBox and AgentInBox goals did not achieve a 100% success rate. Since we did not set a maximum iteration for Algorithm 1, all failure cases involved the algorithm terminating because all models were banned. Therefore, A* search failed to find a path to all stable models, which may indicate that the goal was not reachable for these start states. Figure 10 shows start states that failed to reach both the BoxBox and AgentInBox goals. The figure shows that there was not enough room to reach these goals.

Related Work
Action Schema Networks (ASNets) (Toyer et al. 2020) are neural networks that exploit the structure of the Planning Domain Definition Language (PDDL) to learn a policy that generalizes across problem instances. However, ASNets are...
Goal Reaching: Non-monotonic

\[ \Pi: \text{Answer set program} \]
\[ \mathcal{S}_\Pi: \text{set of states represented by program} \]
\[ \mathcal{S}_A: \text{set of states represented by assignment} \]

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

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

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

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

All rows sum to an even number

Agostinelli, Forest. “A Conflict-Driven Approach for Reaching Goals Specified with Negation as Failure.” ICAPS 2024 HAXP Workshop
Outline

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

- 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
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 $2^{8 \times 16} \approx 3.4 \times 10^{38}$ 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

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

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

Outline

• Background
• Approximate value iteration and batch weighted A* search
• Approximate Q-learning and batch weighted Q* search
• Generalization
  • Generalizing over goals
  • Generalizing over domains
  • Generalizing to domains with unknown transition functions
• Applications
• 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
The continuous model eventually accumulates error for the Rubik’s cube.
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

<table>
<thead>
<tr>
<th>Domain</th>
<th>Solver</th>
<th>Len</th>
<th>Opt</th>
<th>Nodes</th>
<th>Secs</th>
<th>Nodes/Sec</th>
<th>Solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>PDBs†</td>
<td>20.67</td>
<td>100.0%</td>
<td>2.05E+06</td>
<td>2.20</td>
<td>1.79E+06</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>DeepCubeA</td>
<td>21.50</td>
<td>60.3%</td>
<td>6.62E+06</td>
<td>24.22</td>
<td>2.90E+05</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Greedy (ours)</td>
<td>-</td>
<td>0%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>DeepCubeAI (ours)</td>
<td>22.85</td>
<td>19.5%</td>
<td>2.00E+05</td>
<td>6.21</td>
<td>3.22E+04</td>
<td>100%</td>
</tr>
<tr>
<td>RC&lt;sub&gt;rev&lt;/sub&gt;</td>
<td>Greedy (ours)</td>
<td>-</td>
<td>0%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>DeepCubeAI (ours)</td>
<td>22.81</td>
<td>21.92%</td>
<td>2.00E+05</td>
<td>6.30</td>
<td>3.18+04</td>
<td>99.9%</td>
</tr>
<tr>
<td>Sokoban</td>
<td>LevinTS</td>
<td>39.80</td>
<td>-</td>
<td>6.60E+03</td>
<td>-</td>
<td>-</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>LevinTS (*)</td>
<td>39.50</td>
<td>-</td>
<td>5.03E+03</td>
<td>-</td>
<td>-</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>LAMA</td>
<td>51.60</td>
<td>-</td>
<td>3.15E+03</td>
<td>-</td>
<td>-</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>DeepCubeA</td>
<td>32.88</td>
<td>-</td>
<td>1.05E+03</td>
<td>2.35</td>
<td>5.60E+01</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Greedy (ours)</td>
<td>29.55</td>
<td>-</td>
<td>1.68</td>
<td>-</td>
<td>-</td>
<td>41.9%</td>
</tr>
<tr>
<td></td>
<td>DeepCubeAI (ours)</td>
<td>33.12</td>
<td>-</td>
<td>3.30E+03</td>
<td>2.62</td>
<td>1.38E+03</td>
<td>100%</td>
</tr>
</tbody>
</table>

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* search
• 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

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

The Markov Decision Process

<table>
<thead>
<tr>
<th>action space $\mathcal{A}$</th>
<th>hardware-specific universal basis set</th>
</tr>
</thead>
<tbody>
<tr>
<td>state space $\mathcal{S}$</td>
<td>the circuits composed of ${A_j; A_j \in \mathcal{A}}$</td>
</tr>
<tr>
<td>reward function $r$</td>
<td>$r \begin{cases} 0, &amp; \text{if } d(HSHT, U) &lt; \epsilon \ -1, &amp; \text{else} \end{cases}$</td>
</tr>
<tr>
<td>transition probability $P$</td>
<td>$P(s_{t+1}</td>
</tr>
</tbody>
</table>
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.

Other Applications to Quantum Algorithm Compilation

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

Questions?

• Papers
  • 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
  • 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/