Deep Reinforcement Learning and Heuristic Search

Forest Agostinelli
University of South Carolina
• 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

\[
\begin{array}{c|c}
|A\rangle & |A\rangle \\
|B\rangle & |B\rangle \\
|C_{in}\rangle & |S\rangle \\
|0\rangle & |C_{out}\rangle \\
\end{array}
\]

\[
\forall x \in \mathbb{N} : x + 0 = x \\
\text{MATCH_MP_TAC, DAT_INDUCTION} \\
0 + 0 = 0 \\
\text{ARITH_TAC} \\
\text{PROVEN} \\
\forall x \in \mathbb{N} : (x + 0 = x) \Rightarrow (x + 1 + 0 = x + 1) \\
\text{ARITH_TAC} \\
\text{PROVEN}
\]

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: \(~10^{19}\)
  • 48-puzzle: \(~10^{62}\)
  • Organic chemistry: \(~10^{60}\) (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

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

• 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

Largest state space is $3.0 \times 10^{62}$ (48-puzzle)

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

\[ \pi(s) = \arg\min_a \left( c^a(s) + h_\theta(T(s, a)) \right) \]

• 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
  • \( f(n) = \lambda \cdot g(n) + h(n.s) \)
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 $v_0$, batch size $B$, weight $\lambda$
OPEN ← priority queue of nodes based on minimal $f$
CLOSED ← maps states to their shortest discovered path costs
$UB, n_{UB} \leftarrow \infty, NIL$
$LB \leftarrow 0$

$n_{start} \leftarrow NODE(s=start, g=0, p=\text{NIL}, f=v_0(start))$
PUSH $n_{start}$ to OPEN

while not IS_EMPTY(OPEN) do
  generated ← []
  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 $g(s') < \text{CLOSED}[s']$ then
        $\text{CLOSED}[s'] \leftarrow g(s')$
        APPEND(generated, $(s', g(s'), n)$)
    end for
  end while
  if $LB \geq \lambda \cdot UB$ then
    return PATH_TO_GOAL($n_{UB}$)
  end if
  generated_states ← GET STATES(generated)
  heuristics ← $v_0$ (generated_states)
  for $0 \leq i \leq \text{SIZE}(\text{generated})$ do
    $s, g, p \leftarrow \text{generated}[i]$
    $h \leftarrow \text{heuristics}[i]$
    $n_o \leftarrow NODE(s, g, p, f = \lambda \cdot g + h)$
    PUSH $n_o$ to OPEN
  end for
  return PATH_TO_GOAL($n_{UB}$) // failure if $n_{UB}$ is NIL


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/](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>
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.

![Graphs showing the effect of batch and weight on solution length, nodes generated, solve time, and nodes/second.](image)
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

\[ L(\theta) = \left( \min_a (c^a(s) + h_\theta(T(s,a), G)) - h_\theta(s, G) \right)^2 \]

- 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.
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
  • $L(\theta) = \left( \min_a \left( c^a(s) + h_\theta(T(s, a), s_g) \right) - h_\theta(s, s_g) \right)^2$
• 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>

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
  • \( Q(s, a) = Q(s, a) + \alpha [c^a(s) + \min_{a'} Q(T(s, a), a') - Q(s, a)] \)
  • \( \alpha \) is the learning rate
  • Guaranteed to converge to \( q^* \) in the tabular setting if certain conditions are met
Approximate Q-learning

- Q-learning loss
  \[ L(\theta) = \left( c^a(s) + \min_{a'} q_{\theta^-}(T(s, a'), a') - q_{\theta}(s, a) \right)^2 \]
  
  - \( s \): 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
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
  \[ \pi(a | s) = \frac{e^{-\frac{h_\theta(s, a)}{T}}}{\sum_{a' = 1}^{\left| A \right|} e^{-\frac{h_\theta(s, a')}{T}}} \]
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 \( q_\theta \), batch size \( B \), weight \( \lambda \)

OPEN \( \leftarrow \) priority queue of nodes based on minimal \( f \)

CLOSED \( \leftarrow \) maps states to their shortest discovered path costs

\( U, n_U \leftarrow \infty, \text{NIL} \)

\( LB \leftarrow 0 \)

\( n_{\text{start}} \leftarrow \text{NODE}(s = \text{start}, g = 0, p = \text{NIL}, a = \text{NO.OP}, f = 0) \)

PUSH \( n_{\text{start}} \) to OPEN

while not IS.EMPTY(OPEN) do
  generated \( \leftarrow [] \)
  while not IS.EMPTY(OPEN) and SIZE(generated) < \( B \) do
    \( n = (s, a, g, p, f) \leftarrow \text{POP(OPEN)} \)
    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 \( g(s') < \text{CLOSED}[s'] \) then
      CLOSED[s'] \( \leftarrow g(s') \)
      for \( a' \) in \( |A| \) do
        APPEND(generated, (s', g(s'), a', n))
    if LB \( \geq \lambda U \) then
      return PATH_TO_GOAL(n_U)
  generated_states, actions \( \leftarrow \text{GET_STATES(generated)} \)
  transition_costs, heuristics \( \leftarrow q_\theta \) (generated_states, actions)
  for \( 0 \leq i \leq \text{SIZE(generated)} \) do
    s, a, g, p, f \( \leftarrow \) generated[i]
    g' \( \leftarrow g + \text{transition.costo}[i] \)
    h \( \leftarrow \text{heuristics}[i] \)
    \( n_{(s,a)} \leftarrow \text{NODE}(s, a, g, p, f = \lambda \cdot g' + h) \)
    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

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.

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

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

<table>
<thead>
<tr>
<th>Puzzle</th>
<th>Actions</th>
<th>Method</th>
<th>Time</th>
<th>Nodes Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC(156)</td>
<td>x13</td>
<td>A*</td>
<td>3.5(1.6)</td>
<td>8.7(2.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q*</td>
<td><strong>0.9(0.7)</strong></td>
<td><strong>1.4(1.3)</strong></td>
</tr>
<tr>
<td>RC(1884)</td>
<td>x157</td>
<td>A*</td>
<td>37.0(6.5)</td>
<td>62.7(5.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q*</td>
<td><strong>3.7(4.0)</strong></td>
<td><strong>2.3(3.6)</strong></td>
</tr>
</tbody>
</table>

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

<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>reward $r = \begin{cases} 0, &amp; \text{if } d(\text{HSHT}, U) &lt; \varepsilon \ -1, &amp; \text{else} \end{cases}$</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.
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

Other Applications to Quantum Algorithm Compilation

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


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

Ground Truth

Continuous

Discrete

1000 steps

4000 steps

9000 steps
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>3.18E+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>-</td>
<td>1.68</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
Questions?

• Papers
  • Agostinelli, Forest, Rojina Panta, and Vedant Khandelwal. “Specifying Goals to Deep Neural Networks with Answer Set Programming.” ICAPS 2024
  • Agostinelli, Forest and Soltani, Misagh “Learning Discrete World Models for Heuristic Search.” Reinforcement Learning Conference 2024
  • Agostinelli, Forest. “A Conflict-Driven Approach for Reaching Goals Specified with Negation as Failure.” ICAPS 2024 HAXP Workshop

• 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/