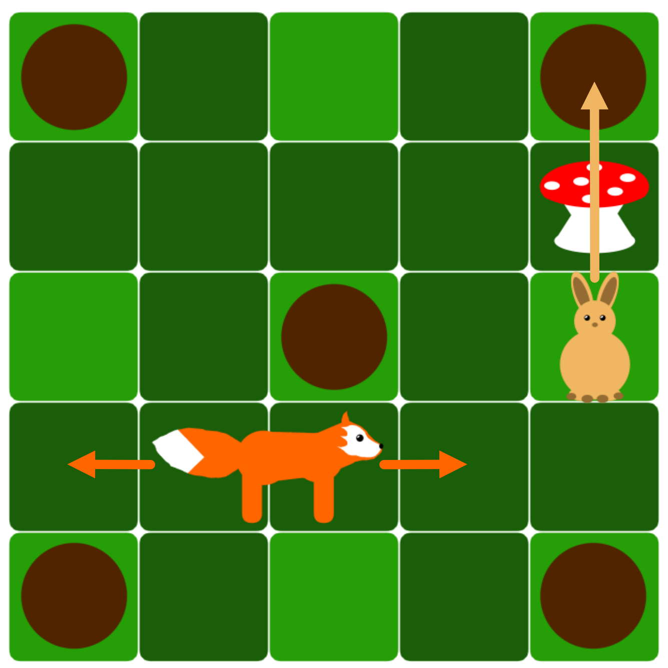
SOLVING JUMPIN' USING ZERO-DEPENDENCY REINFORCEMENT LEARNING

Rachel Ostic,[1] Oliver Benning,[1] Patrick Boily[2]
[1,2]University of Ottawa
[2]Data Action Lab, Ottawa
[2]Idlewyld Analytics and Consulting Services, Wakefield

JUMPIN' GAME

Single-player game by **SmartGames** with 60 sample puzzles and solutions in manual

- 5 × 5 grid
- Raised cells
- Burrows
- Mushrooms
  - Anywhere
  - Stationary during gameplay
- Foxes
  - Not on raised cells
  - Move forward and back
- Bunnies
  - Anywhere
  - Bounce over other pieces

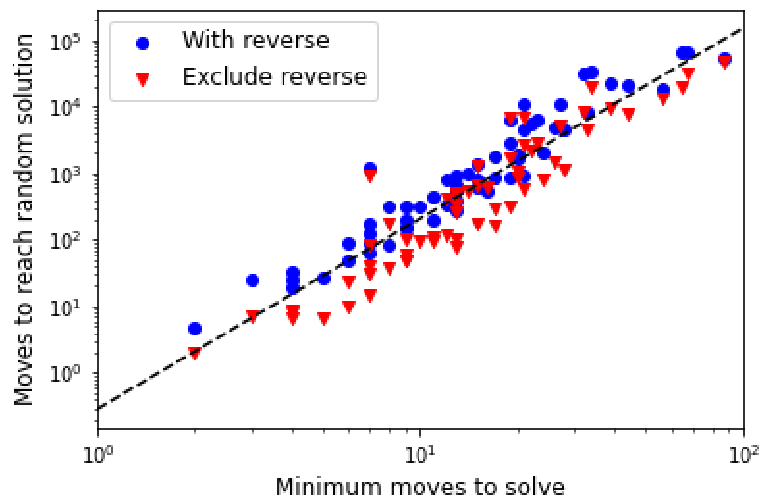


Win by placing all bunnies in burrows

Despite being simple to explain, it isn't easy to win!

INITIAL NOTES

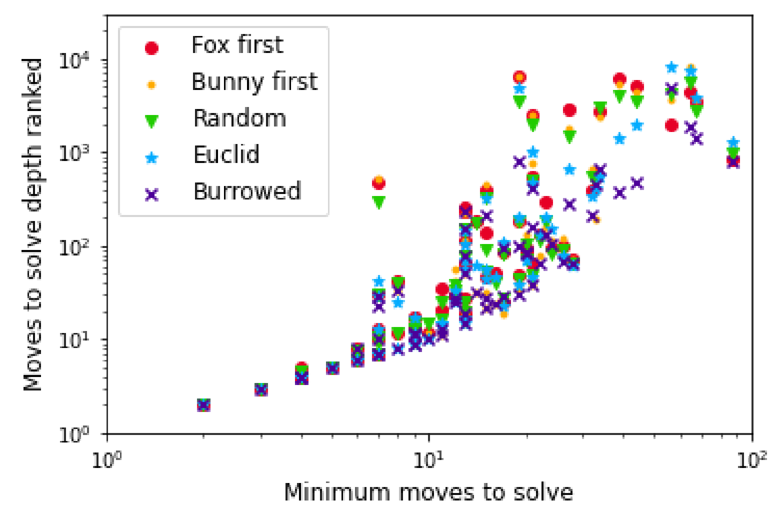
- **Difficulty** ranges from **2 to 80 moves** to solve
  - Not all manual solutions are optimal!
- How difficult is it to win by moving pieces at random?
  - Length of random solution roughly polynomial with actual puzzle difficulty
  - Fit: $y = 0.3\,x^{2.9}$



METHODS

Use **Python** to create three modules:

1. **Game mechanics**
   - Encode **rules**
   - Initialize game
   - Query and perform available moves
2. **Solution**
   - Build solution tree (**breadth** first or **depth** first)
   - Prune previously seen states
   - Move ranking strategies
3. **Train and test**
   - Create model templates
   - Implement **Q-learning** updates
   - Test performance



Comparison of move ranking strategies

- We're interested in finding easily-to-understand strategies
- Maximizing bunnies in burrows is consistently among the most efficient solutions

Q-learning implementation

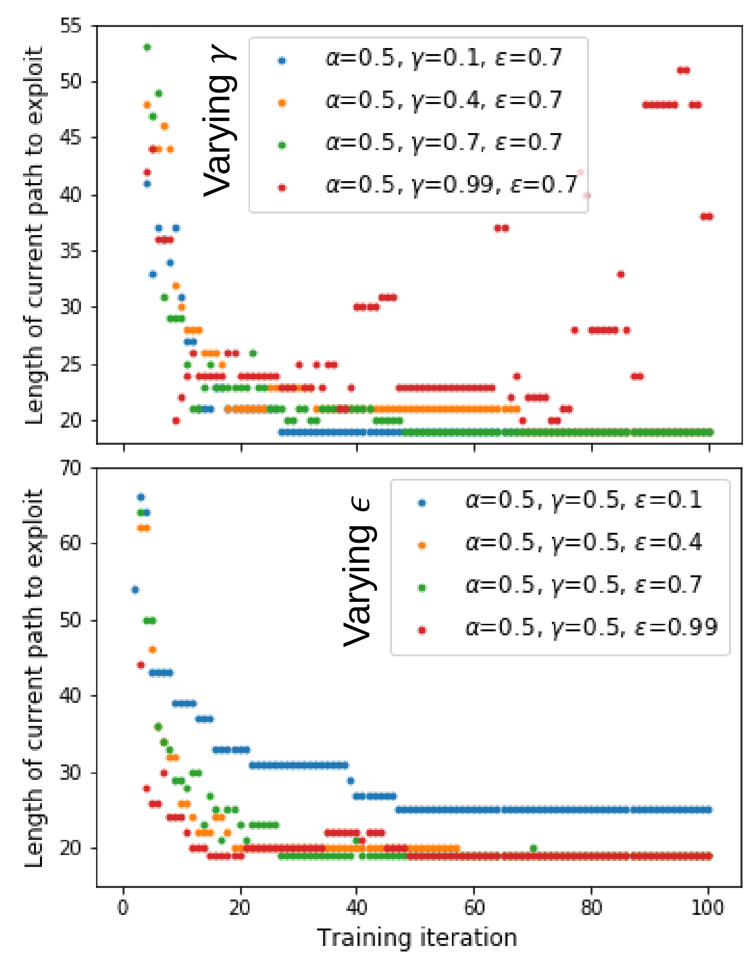
- After solving a puzzle, we assign a reward equal to the reciprocal of the solution length to reinforce shorter solutions more strongly
- Backtracking update to Q-table with Bellman equation:

$$q_{new}(S_t, a_t) = q(S_t, a_t) + \alpha\left(r_t + \gamma \max_{a \in \{a_{t+1}\}} q(S_{t+1}, a) - q(S_t, a_t)\right)$$

where S_t is the board state, a_t is the action taken, and r_t is the reward at step t

Exploring hyperparameters

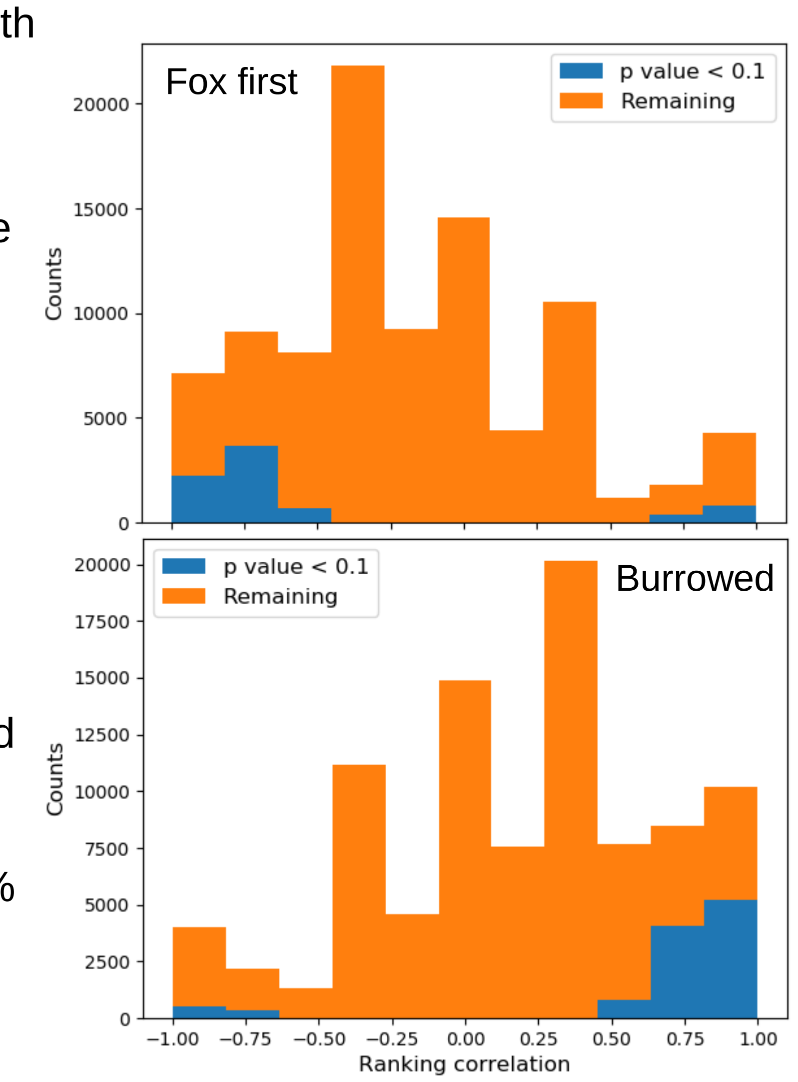
- The Bellman equation has two tunable parameters, learning rate α and discount factor γ
- Another, ϵ, for the exploration rate
  - α has little effect on convergence
  - Setting γ too close to 1 leads to divergent behavior
  - Varying ϵ balances amount of training to solve optimally vs. monotonic improvement



RESULTS

Parsing the Q-table

- After training 10,000 times on all 60 puzzles ($\alpha = 0.5$, $\gamma = 0.5$, $\epsilon = 0.1$), our Q-table contains 92,183 states and 430,545 actions
- We pass through these states, comparing the rankings by Q-value to those from other strategies
- Ranking correlations with Kendall τ show overall whether strategy matches Q-table
  - Generally negative if fox moves prioritized
  - Generally positive for moves that get bunnies closer to burrows

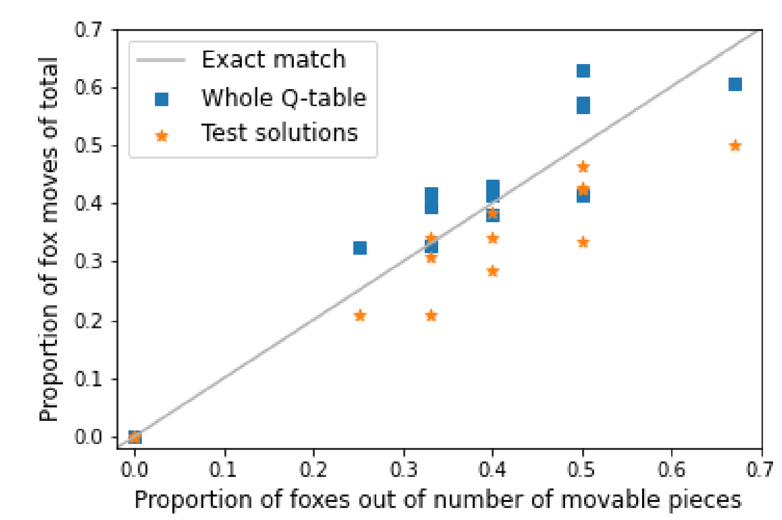


Moves from optimal solutions

- Top ranked move based on maximizing number of bunnies in burrows matches Q-table in 38% of cases
- Next best strategy: minimize average Euclidean distance to burrows over bunnies

Other trends

- The fraction of recommended fox moves appears to be correlated with piece distribution on the board



OUTLOOK

Distilled strategy?

- Consistently move bunnies so as to maximize number in burrows until a fox move opens up new options

Future directions

- Game mechanics module accommodates larger board size or extra pieces
- Swap out game rules for a more complex board game (e.g. chess)
- Incorporate different reinforcement learning techniques (e.g. deep Q-learning, policy gradient approach)