Solving Sokoban with Reinforcement Learning
Progress Report

Omar Tounsi, Joseph Lin, Jihyo Park

November 16, 2024

Objective

The objective of this project is to apply reinforcement learning (RL) algorithms to the Sokoban puzzle game and assess their performance. By comparing RL approaches to traditional search-based algorithms such as depth-first search (DFS) and breadth-first search (BFS), the study aims to explore how these algorithms differ in terms of efficiency and performance. The code for our our preliminary implementation is available on GitHub [4].

Motivation

Sokoban is a classic puzzle game where a player must push boxes into designated cells on a grid, with solid walls acting as boundaries and obstacles. Sokoban is a challenging game because the player must carefully plan each move to push boxes to their designated locations without getting stuck during the process. The player can only push boxes, not pull them. This means that if a box is pushed into a corner or against a wall without room to maneuver, it becomes stuck, making the puzzle unsolvable. The computational complexity of Sokoban is known to be PSPACE-complete, which means solving Sokoban is as hard as the most difficult problems that can be solved using a polynomial amount of memory (PSPACE). Sokoban presents a challenging environment for testing reinforcement learning due to its complex spatial reasoning, the long-term effect of actions, and the potential for deadlocks. This project provides an opportunity to assess the strengths and limitations of RL in game-playing, while also benchmarking its performance against traditional search-based algorithms.

Problem Definition

Sokoban can be modeled as a Markov Decision Process (MDP), where the state space represents the entire grid, including the positions of boxes, target spots, and the agent. The action space comprises the agent's possible movements: up, down, left, and right. The transition function is deterministic, and the reward function is designed to incentivize progress and penalize mistakes. The reward system we chose assigns +1 for placing a box on a target spot, -1 for attempting invalid actions (like moving into an obstacle), +100 for solving the puzzle (placing all boxes on target spots), and -100 for creating an irreversible deadlock by getting a box stuck. This approach ensures that the agent is encouraged to progress towards completing the puzzle while being penalized for inefficient behavior. For a given puzzle, the objective is to learn a policy that places all the boxes correctly in the minimum number of steps.

Plan

Our project plan involves developing RL algorithms, starting with Monte Carlo methods for policy evaluation, and testing their performance on a variety of Sokoban puzzles with different complexities, and comparing the results with non-RL Sokoban solvers (DFS, BFS, etc). Our RL implementation uses an episodic, finite-horizon setting where an episode terminates upon either solving the puzzle or reaching a deadlock. To measure the effectiveness of the algorithms, metrics such as the percentage of boxes placed correctly, the number of actions taken by the agent, and policy convergence speed are recorded.

The very large state space of Sokoban puzzles has made it impractical to use algorithms like value iteration and policy iteration, which rely on explicitly enumerating all possible states. To overcome this limitation, future work will explore reinforcement learning algorithms that leverage function approximators, such as Deep Q-Learning (DQN). DQN uses a neural network to approximate the Q-function, allowing the agent to generalize Q-value predictions across states without the need to store them explicitly. This is particularly advantageous for Sokoban, where the exponential growth of possible configurations makes traditional tabular methods infeasible.

Candidate Data Sets

Some of the puzzles we tested our algorithms on were sourced from https://www.sourcecode.se/sokoban/levels.php, from Andrej Cerjak's AC Easy collection. In the future, as we move to more complex algorithms, we intend to use the Boxoban dataset [3], developed by DeepMind. Boxoban provides a rich collection of over 900,000 training Sokoban puzzle levels with varying difficulty. Each level has a size of 10x10 with four boxes to be placed. The dataset is publicly accessible on GitHub. We implemented a script that efficiently parses combined Sokoban level files from the Boxoban dataset into individual level files, facilitating easier access of each level in the future.

Related Work

When applying reinforcement learning (RL) to complex puzzles, "Solving Sokoban with Forward-Backward Reinforcement Learning" by Yaron Shoham and Gal Elidan presents a notable approach [5]. The authors address the challenge of sparse rewards in environments like Sokoban, where feedback is typically provided only upon task completion. Their method involves training a backward-looking agent with a simplified goal, then enhancing the state representation of a forward-looking agent with features derived from the backward agent's behavior. This strategy enables the forward agent to utilize information from backward plans without directly imitating their policy. The results demonstrate that this approach not only surpasses existing learned solvers that generalize across levels but also competes with state-of-the-art handcrafted systems, all while learning from a limited set of practice levels and employing straightforward RL techniques.

"An Investigation of Model-Free Planning: Boxoban Levels" [2] by Arthur Guez et al. explores the application of model-free reinforcement learning to complex puzzle environments using the Boxoban dataset. The authors aim to assess the performance of model-free planning methods in environments characterized by sparse rewards and high complexity, such as Sokoban-like puzzles. By utilizing the Boxoban dataset, the study provides a benchmark for evaluating reinforcement learning agents. The paper highlights the limitations of traditional model-free approaches when applied to intricate planning problems and proposes modifications to improve their efficiency and adaptability. This work underscores the importance of structured datasets like Boxoban in advancing research on reinforcement learning for sequential decision-making tasks.

Preliminary Experimental Results

To evaluate the effectiveness of Monte Carlo policy evaluation in solving Sokoban puzzles, the every-visit algorithm was applied to five puzzles of varying difficulty levels: Trivial, Easy1, Easy2, Medium, and Hard. Such puzzles can be solved optimally (and near instantly) using BFS, DFS, A* search, or Uniform Cost Search. This was confirmed using an online application stored on a GitHub repository named sokoban-solver [1].

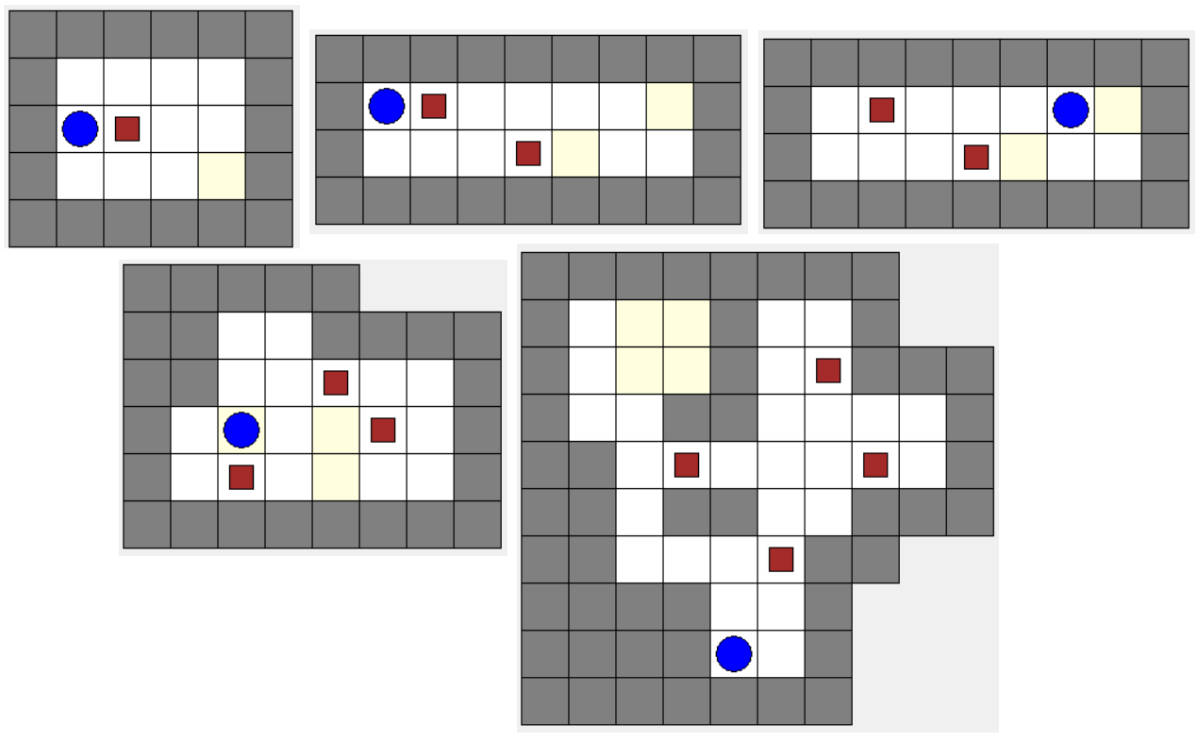

Figure 1: Puzzles used for experiments (Trivial, Easy1, Easy2, Medium, and Hard, clockwise)

The following hyperparameters were chosen with the goal of balancing exploration and exploitation:

- Exploration Parameter (ϵ): Initialized to 0.9 with a decay factor of 0.995 per episode, encouraging exploration in the early episodes and a gradual shift toward exploitation as the training progressed.
- Discount Factor (γ): Set to 0.85 to reduce the emphasis on long-term rewards, thereby discouraging loop behaviors.
- Number of Episodes: Limited to 100,000 to allow sufficient exploration of the state-action space while managing computational feasibility.
- Convergence Threshold: Set to 0.001 for the easier puzzles and 0.0001 for the Medium and Hard puzzles, to ensure the Q-function stabilizes to an acceptable degree of accuracy before concluding training.

The algorithm successfully solved the Trivial, Easy1, and Easy2 puzzles, achieving optimal policies with complete box placement and minimal number of actions. However, for the Medium and Hard puzzles, the algorithm failed to converge within the allocated episodes, producing suboptimal policies where the agent successfully placed a box or two on a target spot but then fell into an infinite loop, repeatedly executing the same sequence of actions without making further progress. The reason for lowering the convergence threshold to 0.0001 for the Medium and Hard puzzles was to prevent the policies from converging prematurely. Upon increasing the maximum number of episodes to 1,000,000, the algorithm eventually converged for the medium puzzle at episode 257,942, however the policy remained suboptimal, which may indicate that the Q-function converged locally but not optimally. For the hard puzzle, the algorithm did not converge in a reasonable time. This suggests that our Monte Carlo implementation struggles with complex puzzles.

Table 1: Metrics for Monte Carlo Policy Evaluation

Puzzle	**Converged**	**Episodes**	**Time Taken (s)**	**Game Won**	**% Boxes Placed**	**Actions**
Trivial	Yes	17,052	40.00	True	100.0	5
Easy1	Yes	30,875	330.00	True	100.0	9
Easy2	Yes	23,498	184.00	True	100.0	14
Medium	No	100,000	N/A	False	33.0	Infinite
Hard	No	100,000	N/A	False	25.0	Infinite

The agent's tendency to get stuck in infinite loops for more complex puzzles but not for simpler ones may stem from several reasons. The state-action space in complex puzzles is significantly larger due to additional boxes, obstacles, and potential configurations. This increases the difficulty of adequately exploring all relevant states, even with a high initial exploration rate ϵ. As a result, the agent may fail to discover better actions and instead converge prematurely to repetitive behaviors. Furthermore, the reward sparsity of Sokoban likely plays a major role in provoking loops. In complex puzzles, rewards are often farther apart in the sequence of actions, making it challenging for the agent to associate intermediate actions with long-term success. These challenges are less pronounced in simpler puzzles where the state space is smaller, rewards are easier to obtain, and the likelihood of encountering deadlock is reduced.

Our Monte Carlo implementation failed to perform better than non-RL search algorithms. Not only the harder puzzles could not be solved by the every-visit algorithm, but it also took a significantly longer time to converge for the easier puzzles than traditional search algorithms, who can solve complex puzzles nearly instantly.

References

[1] dangarfield. sokoban-solver. https://github.com/dangarfield/sokoban-solver, 2022.

[2] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An investigation of model-free planning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2464–2473. PMLR, 09–15 Jun 2019.

[3] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, Timothy Lillicrap, and Victor Valdes. An investigation of model-free planning: boxoban levels. https://github.com/deepmind/boxoban-levels/, 2018.

[4] Jihyo Park Omar Tounsi, Joseph Lin. sokoban-rl. https://github.com/omartounsi7/sokoban-rl, 2024.

[5] Yaron Shoham and Gal Elidan. Solving sokoban with forward-backward reinforcement learning, 2021.