Potential-based Reward Shaping in Sokoban

Zhao Yang[1], Mike Preuss[2], and Aske Plaat[3]

[1] LIACS, Leiden University, the Netherlands
z.yang@liacs.leidenuniv.nl
[2] LIACS, Leiden University, the Netherlands
m.preuss@liacs.leidenuniv.nl
[3] LIACS, Leiden University, the Netherlands
aske.plaat@gmail.com

Abstract. Learning to solve sparse-reward reinforcement learning problems is difficult, due to the lack of guidance towards the goal. But in some problems, prior knowledge can be used to augment the learning process. Reward shaping is a way to incorporate prior knowledge into the original reward function in order to speed up the learning. While previous work has investigated the use of expert knowledge to generate potential functions, in this work, we study whether we can use a search algorithm(A*) to automatically generate a potential function for reward shaping in Sokoban, a well-known planning task. The results showed that learning with shaped reward function is faster than learning from scratch. Our results indicate that distance functions could be a suitable function for Sokoban. This work demonstrates the possibility of solving multiple instances with the help of reward shaping. The result can be compressed into a single policy, which can be seen as the first phrase towards training a general policy that is able to solve unseen instances.[4]

Keywords: Reinforcement Learning · Potential-based Reward Shaping · Sokoban.

1 Introduction

Sokoban is a well-known puzzle game that is often used as a benchmark for evaluating reinforcement learning (RL) agents [8,10]. Sokoban is a sparse reward task, furthermore, it suffers from dead-ends: one bad action can render the whole instance unsolvable. Sokoban is a deceptively simple puzzle. RL agents struggle to learn a behavior policy, unless they used planning as part of their learning effort. A simple example from [17] is shown in Fig. 1. Although this example has only three boxes, it might already be challenging to solve for humans.

Human problem solving used heuristics, rules of thumb that are based on experience, that work most of the time, but not always. Heuristics usually increase our ability to solve problems greatly. Reward shaping [5,13] is proposed

[4] We open-sourced all the code we used. It can be found, after the review, at https://anonymous.org/blind-review

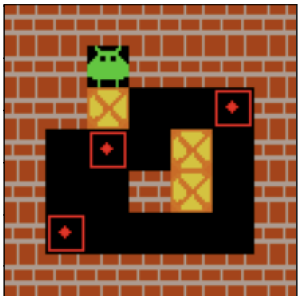

Fig. 1: An example of 3-boxes Sokoban instance.

for incorporating prior (heuristic) knowledge to accelerate learning in RL. It reshapes the original reward function by adding another reward function which is formed by prior knowledge in order to get an easy-learned reward function, that is often also more dense. Examples of prior knowledge are heuristics from context structures, demonstrations from experts, etc.

In this paper, we show that reward shaping can be applied to sparse reward tasks for faster learning. More accurately, we choose the distance function (automatically provided by the A* search algorithm) as the potential function, and subsequently performed potential-based reward shaping in Sokoban. Our results demonstrate that learning with shaped reward functions outperforms learning from scratch by a large margin. In contrast to neural networks, that are able to generalize to unseen tasks but require much training data, our reward shaping can be seen as the first step towards the final goal that aims to train an agent which is able to solve multiple unseen new Sokoban instances. With reward shaping, the ability of solving multiple instances is compressed into a single behavior policy without extensive training.

The paper is structured as follows: first we briefly describe related work in section 2; then details about our method are provided in section 3; followed by experimental design and results; lastly, we discuss limitations and potential future works of this paper, then draw conclusions in section 5.

2 Related Work

Reinforcement learning (RL) algorithms are used to solve decision making problems which could be formed into Markov Decision Process (MDPs), and they train policies by interacting with environments [14,19]. Recently, RL achieves super human performance in the board game Go [18], Atari games [1] and StarCraft [20]. In this section, we will briefly describe related work about both potential-based reward shaping and Sokoban.

2.1 Reward Shaping

Reward shaping offers a way to add useful information to the reward function of the original MDP. By reshaping, the original sparse reward function will be denser and is more easily-learned. The heuristics can come from different sources, such as demonstrations either from human or another RL agent [2,11], or expert's guidance, etc.

The optimal policy is determined by the reward function, small transformations of the reward function might cause intractable problems [15]. Ng. et al. proved that by following the potential-based reward shaping, the optimal policy will be invariant [13]. The original reward function R is augmented by another reward function F, shown in Eq. 1 and if and only if F is the subtraction between a function ϕ of the next state s' and the current state s then the optimal policy will keep unchanged. The function ϕ is called potential function. Examples of good potential function could be Manhattan distance in navigation tasks or pre-trained state value functions, etc.

$$\begin{aligned} R'(s,a,s') &= R(s,a,s') + F(s,a,s') \\ &= R(s,a,s') + \phi(s') - \phi(s) \end{aligned} \tag{1}$$

Brys et al. extracted potential function from demonstrations by checking if agent's state-action pairs are in demonstrations or not and apply it to Cart Pole and Mario [2]. Hussein et al. trained a neural network from demonstrations as the potential function and added it to the original reward function of DQN in grid navigation tasks[12]. Grzes provided more insights and analysis for potential-based reward shaping and extended it to multi-agent RL scenario [7]. While most previous methods have focused on extracting potential functions from expert demonstrations, we investigate whether potential functions can also be extracted from a search. In our case, we use the distance function which is provided by the A* search algorithm as the potential function.

2.2 Sokoban

Sokoban is a challenging puzzle game and has been proved to be PSPACE-complete [3] and NP-hard [4] problem. It also plays an important role in benchmarking RL agents. Many models are proposed to solve Sokoban. Both model-based methods [9,10,21], as well as model-free methods can reach competitive performance [8]. Curriculum learning has been used to solve a difficult Sokoban instance [6]. The works mentioned above try to solve Sokoban using special-designed models, while we are focusing on using general reward shaping techniques to speed up the learning.

Fine-tuning pre-trained models is helpful in accelerating learning in Sokoban [22]. Reward shaping was applied to a single simple Sokoban instance by interacting with human experts in [16] to speed up the learning. In our work, we demonstrate potential-based reward shaping over many Sokoban instances range from 1-box to 3-boxes, where no human expert involved.

3 Methods

In this section, we will explain the methods and techniques that we used. We report the problem model, the heuristic that was used in A*, and details about the reward shaping.

3.1 Reinforcement Learning

MDP is short for Markov Decision Process, and it models decision making problems into a 4-tuple, $\langle S, A, R(s,a,s'), P(s,a,s')\rangle$. In our paper, we follow the MDP notation proposed in [19]. S is a set of states s called the state space, it will be different states in Sokoban instance in our case. A is a set of actions a called the action space, in our case it will contain all actions that the agent can take (no operation, going up/down/left/right). $R(s,a,s')$ is a reward function that determines immediate rewards that the agent will get after performs an action a which leads the agent from the current state s to the next state s'. $P(s,a,s')$ is the probability that action a leads the agent from the current state s to the next state s'. Reinforcement learning methods solve MDP using data$(s,a,s',R(s,a,s'))$ collected by interacting with the environment to train a policy aims to maximize the accumulated reward shown in Eq. 2,

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2}$$

where $\gamma \in [0,1]$ is the discount factor and r_t is the immediate reward the agent gets in time step t.

We use the RL algorithm A2C to train the agent to learn to solve Sokoban. The policy is represented by a neural network and the architecture of the neural network we are using for experiments is the same as the architecture used in [22], which consists of 3 convolutional layers and 2 fully-connected layers. All hyperparameters of A2C are also kept the same as described in [22]. More details can be found in the Appendix B.

3.2 A* Heuristics

A* is a heuristic search algorithm, it extends the Dijkstra's algorithm by adding heuristics. The heuristic we used in our case is the overall Manhattan distance between untargeted boxes and goals[5], formula shown in Eq 3.

$$h(s) = \sum_{b \in B, t \in T} (|x_b - x_t| + |y_b - y_t|) \tag{3}$$

, where (x_b, y_b) is the location of boxes while (x_t, y_t) is the location of targets, h is the heuristic for the current state s. B is all boxes which are not on targets

[5] The implementation we are using is from `https://github.com/KnightofLuna/sokoban-solver`

yet and T is all targets where there are no boxes on. If a Sokoban instance is solvable, A* will return the solution otherwise it will return nothing. As such, it could also be used to check the solvability of a Sokoban instance.

3.3 Reward Shaping

Values of potential functions of states should be higher if states are 'better' and vice versa. For this reason the **minus** of the distance function is used as the potential function in our case. The distance function will take the current state as input, and output how many steps the agent needs to take towards the goal state.

The shaped reward function will be(shown in Eq. 4):

$$\begin{aligned} R'(s,a,s') &= R(s,a,s') + F(s,a,s') \\ &= R(s,a,s') + \phi(s') - \phi(s) \\ &= R(s,a,s') - d(s') + d(s) \end{aligned} \tag{4}$$

where s is the current state while s' is the next state, and a is the action that leads s to s'. ϕ is the potential function and $d(s)$ is the distance function from the current state s to the goal state. In our case, we use the A* search algorithm[6] to provide the distance information.

In Sokoban, some actions can lead to unsolvable situations. An example is shown in Fig. 2. A box is pushed into the corner and it is not possible to pull it back. The instance has become completely unsolvable. Then a natural question is what distance we should assign to states which are unsolvable. The algorithm, however, can still get some rewards by learning sub-optimal policies, such as pushing one of the boxes onto one of the targets. In order not to break the sub-optimal policy invariance, we don't shape the reward function and just keep the original reward function after the instance has become unsolvable. To conclude, our shaped reward function is shown in Eq. 5.

$$F(s,a,s') = \begin{cases} \phi(s) - 1 = -d(s) - 1 & \text{if } s \text{ is solvable and } s' \text{ is unsolvable} \\ 0 & \text{if both } s' \text{ and } s \text{ are unsolvable} \\ \phi(s') - \phi(s) = -d(s') + d(s) & \text{otherwise} \end{cases} \tag{5}$$

4 Experiments

The agent is evaluated every 1,000 environment steps on 20 randomly selected instances. We use 100 * 1-box instances, 100 * 2-boxes instances and 60 * 3-boxes instances(since more boxes are more expensive, we use 60 instead of 100). The results shown are averaged over 5 runs with different random seeds.

[6] `https://en.wikipedia.org/wiki/A*_search_algorithm`

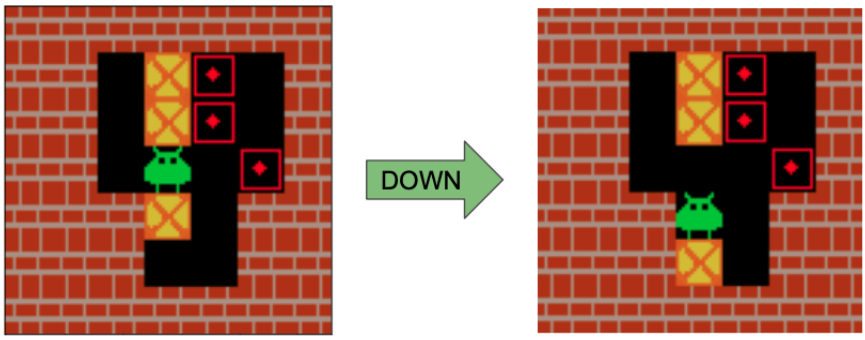

Fig. 2: How an unsolvable situation happens.

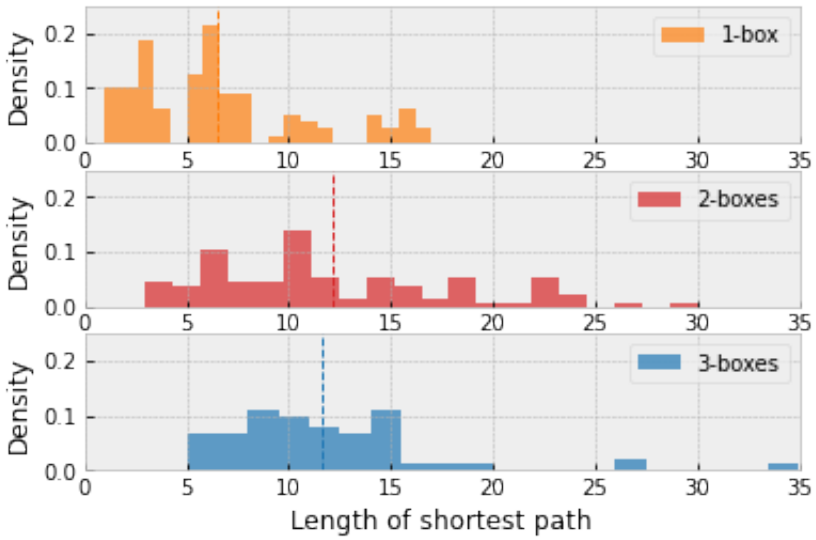

Fig. 3: The shortest path of different Sokoban instances, dash lines are means. Top: solutions of 1-box instances, and the mean of solutions is 6.52. Mid: solutions of 2-boxes instances, and the mean of solutions is 12.14. Bottom: solutions of 3-boxes instances, and the mean of solutions is 11.62.

The length of the shortest path of an instance could indicate the difficulty of the instance. Fig. 3 shows the distribution of shortest paths of instances we are using; as expected the more boxes, the longer the shortest path.

The learning results on 1-box instances is shown in Fig. 4. Even without reward shaping, the agent can quickly learn to master given instances within 60k environment steps. From the top subplot in Fig. 3 we see that, solutions of 1-box instances are mostly shorter than 10. This also indicates that RL is able

to solve simple sparse-reward problems. By adding reward shaping, the agent is about four times faster than learning from scratch. Both learning with reward shaping and learning from scratch are able to solve given instances within the given steps.

Learning on 2-box and 3-box instances is more difficult than learning on 1-box instances. Reinforcement learning from scratch almost learns nothing on 2-box and 3-box instances. Solutions of multiple box instances are generally longer than solutions of single-box instances. Learning with reward shaping performs better by a large margin, the agent is able to solve given instances within 50k environment steps, while learning from scratch only reaches a solved ratio of around 0.2. This again demonstrates that RL is not good at solving sparse-reward problems.

Fig. 4: Learning over 100 * 1-box instances.

5 Discussion and Conclusion

In this work, we showed that the distance function can be used as potential function in potential-based reward shaping to further speed up the learning in Sokoban. Meanwhile, we have seen that the distance function can be used as potential function in grid-world navigation tasks, since grid-world navigation tasks can be treated as special types of Sokoban where there are no needs for pushing boxes but only moving the agent to the target. Our experiments showed that abilities of solving multiple instances can be quickly learned and compressed into a single behavior policy, which can be seen as the first step towards training a general policy which is able to solve unseen Sokoban instances quickly. For

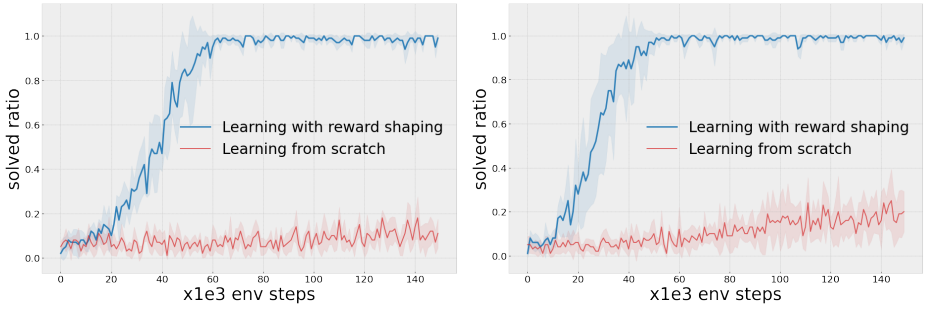

Fig. 5: Learning over 100 * 2-boxes instances(left) and 60 * 3-boxes instances(right).

instance, if the agent is exposed to a Sokoban generator for training and the goal is to train the agent to be able to solve new unseen instances. Then learning with reward shaping will be way faster than learning from scratch to reach this 'final' goal.

A limitation of our approach is that since we are using search algorithms to provide the distance function, scalability is limited. For more difficult instances, search algorithms can not find solutions within a reasonable time, and the reward shaping we did in this work will not be usable. In the future, it would be interesting to work on scalable heuristic functions to use as potential functions in Sokoban. On the other hand, as we mentioned, our methods are the first used to train a baseline agent quickly, then reuse or transfer the trained neural networks for further training or tasks.

Acknowledgement

The financial support to Zhao Yang is from the China Scholarship Council(CSC). Computation support is from ALICE/Leiden and the DSLab. The authors thank Michiel van der Meer, Hui Wang, Matthias Müller-Brockhausen and all members from the Leiden Reinforcement Learning Group for helpful discussions. Especially thanks to Thomas Moerland for detailed feedback.

References

1. Badia, A.P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z.D., Blundell, C.: Agent57: Outperforming the atari human benchmark. In: International Conference on Machine Learning. pp. 507–517. PMLR (2020)
2. Brys, T., Harutyunyan, A., Suay, H.B., Chernova, S., Taylor, M.E., Nowé, A.: Reinforcement learning from demonstration through shaping. In: Twenty-fourth international joint conference on artificial intelligence (2015)
3. Culberson, J.: Sokoban is pspace-complete (1997)
4. Dor, D., Zwick, U.: Sokoban and other motion planning problems. Computational Geometry **13**(4), 215–228 (1999)

5. Dorigo, M., Colombetti, M.: Robot shaping: Developing autonomous agents through learning. Artificial intelligence **71**(2), 321–370 (1994)
6. Feng, D., Gomes, C.P., Selman, B.: A novel automated curriculum strategy to solve hard sokoban planning instances. Advances in Neural Information Processing Systems **33**, 3141–3152 (2020)
7. Grzes, M.: Reward shaping in episodic reinforcement learning (2017)
8. Guez, A., Mirza, M., Gregor, K., Kabra, R., Racanière, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., et al.: An investigation of model-free planning. In: International Conference on Machine Learning. pp. 2464–2473. PMLR (2019)
9. Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., Silver, D.: Learning to search with mctsnets. In: International conference on machine learning. pp. 1822–1831. PMLR (2018)
10. Hamrick, J.B., Friesen, A.L., Behbahani, F., Guez, A., Viola, F., Witherspoon, S., Anthony, T., Buesing, L.H., Veličković, P., Weber, T.: On the role of planning in model-based deep reinforcement learning. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=IrM64DGB21
11. Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al.: Deep q-learning from demonstrations. In: Thirty-second AAAI conference on artificial intelligence (2018)
12. Hussein, A., Elyan, E., Gaber, M.M., Jayne, C.: Deep reward shaping from demonstrations. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 510–517. IEEE (2017)
13. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: Theory and application to reward shaping. In: In Proceedings of the Sixteenth International Conference on Machine Learning. pp. 278–287. Morgan Kaufmann (1999)
14. Plaat, A.: Learning to Play: Reinforcement Learning and Games. Springer Verlag, Heidelberg, See https://learningtoplay.net (2020)
15. Randløv, J., Alstrøm, P.: Learning to drive a bicycle using reinforcement learning and shaping. In: ICML. vol. 98, pp. 463–471. Citeseer (1998)
16. Raza, S.A., Johnston, B., Williams, M.A.: Reward from demonstration in interactive reinforcement learning. In: The Twenty-Ninth International Flairs Conference (2016)
17. Schrader, M.P.B.: gym-sokoban. https://github.com/mpSchrader/gym-sokoban (2018)
18. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)
19. Sutton, R.S., Barto, A.G.: Reinforcement learning, An Introduction, Second Edition. MIT Press (2018)
20. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
21. Weber, T., Racanière, S., Reichert, D.P., Buesing, L., Guez, A., Rezende, D.J., Badia, A.P., Vinyals, O., Heess, N., Li, Y., et al.: Imagination-augmented agents for deep reinforcement learning. arXiv preprint arXiv:1707.06203 (2017)
22. Yang, Z., Preuss, M., Plaat, A.: Transfer learning and curriculum learning in sokoban. arXiv preprint arXiv:2105.11702 (2021)

A Environment

The environment we are using is from the implementation [17] with slight modification. The rewards of the environment shown in the Tab. 1. Solving the instance by pushing all boxes onto targets returns 10.0; pushing one box onto a target gets 1.0 and pushing it off gets -1.0; in order to incentivize the agent solve the instance quickly, an -0.1 is given for each step the agent makes.

actions	reward
push all boxes on targets	10.0
push one box onto target	1.0
push one box onto target	-1.0
each step	-0.1

Table 1: Rewards in the environment.

B Neural Network Details

The model we are using contains three convolutional layers with kernel size 8x8, 4x4, 3x3, strides of 4, 2, 1, and number of output channels 32, 64, 64. Then followed by a fully connected layer with 512 units. In the end, the outputs are fed into two heads: outputting the policy logits and the state value. `ReLU` is used as the activation function after each layer and `RMSprop` is the optimizer we used. The input is pixel image gets from the environment directly, which is 3x80x80.

learning rate	$7 \cdot 10^{-4}$
gamma	0.99
entropy coef	0.1
value loss coef	0.5
eps	10^{-5}
alpha	0.99
rollout storage size	5
No. of environments for collecting trajectories	30

Table 2: Hyper-parameters of the neural network and training.

C Overall Training Loop

Algorithm 1: Overall RL training loop

Initialization: policy π, number of training steps N, environment `env`
$s \leftarrow$ `env.reset()`;
while $n < N$ **do**
  $a \leftarrow \pi(s)$;
  s', r, $\leftarrow$ `env.step`(a);
  `/* calculate the potential value under different situations. */`
  if *s and s' are solvable* **then**
    $f \leftarrow -d(s') + d(s)$; `/* f is the potential value. */`
  else if *s is solvable and s' is not solvable* **then**
    $f \leftarrow -d(s) - 1$;
  else if *s, s' are not solvable* **then**
    $f \leftarrow 0$;
  $r' \leftarrow r + f$; `/* Reshape the reward. */`
  execute A2C update on π using the shaped reward r';
  $n \leftarrow n + 1$;
end
return π ;