Reusing Skills for First-Time Solution of Navigation Tasks in Platform Videogames

(Extended Abstract)

Michael Dann
School of Science
RMIT University
Melbourne, Australia
michael.dann@rmit.edu.au

Fabio Zambetta
School of Science
RMIT University
Melbourne, Australia
fabio.zambetta@rmit.edu.au

John Thangarajah
School of Science
RMIT University
Melbourne, Australia
john.thangarajah@rmit.edu.au

ABSTRACT

We consider the problem of performing real-time navigation in domains where a "god's eye view" is provided. One setting where this challenge arises is in platform videogames, occurring whenever the player wishes to reach an item or powerup on the current screen. Previous agents for these games rely on generating many low-level simulations or training runs for each fixed task. Human players, on the other hand, can solve navigation tasks at a high level by visualising sequences of abstract "skills". Based on this intuition, we introduce a novel planning approach and apply it to *Infinite Mario*. Despite facing randomly generated, maze-like tasks, our agent is capable of deriving complex plans in real-time, without exploiting precise knowledge of the game's code.

Keywords

Reinforcement Learning, Videogames, Options Framework

1. INTRODUCTION

Figure 1 shows a typical navigation task faced in platform videogames, taken from *Infinite Mario* [13]. In order to reach the fire flower at the top-right of screen, the player must run and jump around a "maze-like" series of platforms. Strong human players have little trouble visualising such trajectories. At typical frame rates of 24fps or higher, this implies that they posses a lookahead depth of several hundred time steps. However, existing artificial agents generally struggle on this type of task. Agents that learn by randomly exploring the environment may take a very long time to receive positive feedback, while simulation-based agents may require significant streamlining to achieve even a few seconds of planning depth [2].

Human players appear to contend with granular time scales by planning in abstract, high-level steps. For example, in Figure 1, a suitable plan might be described in human terms as follows: first the player must jump to ascend the first, second and third platforms on the left, then jump from the top-left platform to the top-right platform, then run to the flower. Observe that this plan consists of "skills" (e.g. *jumping* and *running*) whose durations and exact endpoints are uncertain, rather than low-level actions such as pressing a particular button at a particular frame. It also relies on exploiting the "god's eye view" provided to visualise the overall trajectory.

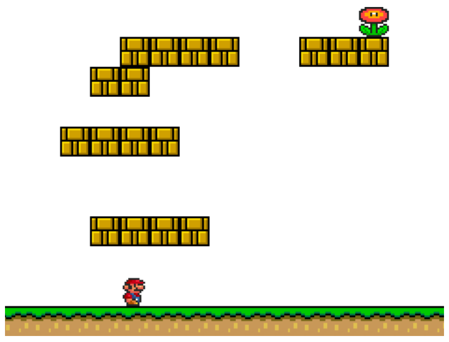

Figure 1: Reaching a powerup is a typical planning problem faced in platform videogames.

While skill acquisition has been a focus of machine learning research for some time [1, 4, 6, 7, 8, 9, 10, 12, 15], existing methods lack the flexibility of the thought process just described. The common approach of identifying "bottlenecks", that is, states that must be visited in order to navigate from one distinct region of the state space to another, leads to the acquisition of *task-specific* skills, e.g. a skill for jumping to the specific platform above Mario in Figure 1. Human players, on the other hand, seem to exploit *reusable* skill knowledge, e.g. knowledge of how to jump to any platform. Furthermore, human players can often find viable high-level plans upfront for previously unseen problems, whereas skill-based learning methods generally require the overall plan to be learned through trial and error, even when the underlying skills are transferred. The aim of this work is to devise an artificial approach that overcomes these issues, i.e. one that can find high-level plans spanning hundreds of frames in real-time.

2. APPROACH

Our proposed approach can be viewed as a hybrid between skill-based learning and traditional discrete planning. It consists of three main steps:

1. Rather than attempting to identify problem-specific

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

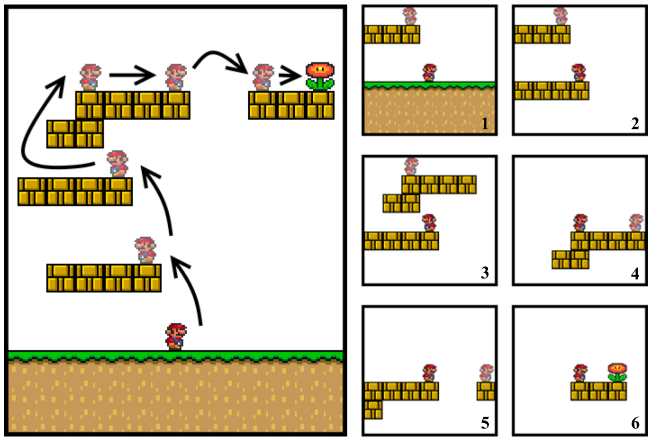

Figure 2: A schematic illustration of our approach, using an example from *Infinite Mario*.

bottlenecks upfront, or trying to train distinct skills such as *running* and *jumping*, we teach the agent a general skill for performing local navigation. We observe that in many classical videogames the world conforms to a grid, and teach the agent a single *agent space option* [3] for navigating to nearby grid squares.

2. After this skill has been acquired, we teach the agent to estimate the likelihood that a given local movement can be executed successfully. Theoretically, the agent ought to learn high probabilities for straightforward movements such as running a few squares to the left unobstructed, and low probabilities for impossible movements such as running through walls.

3. We model the game world as a graph, with edge weights between nearby grid squares equal to the log-likelihood of the agent being able to perform the corresponding movement. Given a target destination on the current screen, we calculate the path with the greatest likelihood of success through Dijkstra's algorithm.

A schematic illustration of a plan being segmented into local movements is shown in Figure 2. Note that under this method, the bottlenecks fall out at the end of the process; a reversal of the traditional order.

Since the approach described is probabilistic, plans may sometimes fail. During execution, replanning is triggered whenever the next waypoint falls out of the movement skill's range, or if the likelihood of reaching the next waypoint falls below a fixed percentage of the originally estimated likelihood. This fixed percentage is referred to as the *replanning threshold*, k.

3. RESULTS

We tested our approach in randomly generated, maze-like levels of *Infinite Mario*. At the start of each episode, a random grid square on the current screen was assigned as the goal. The task was reset and the level structure re-randomised whenever the goal was reached or the episode length exceeded 15 seconds.

The local movement skill was trained to perform movements within a Manhattan distance of 5 grid squares. Both

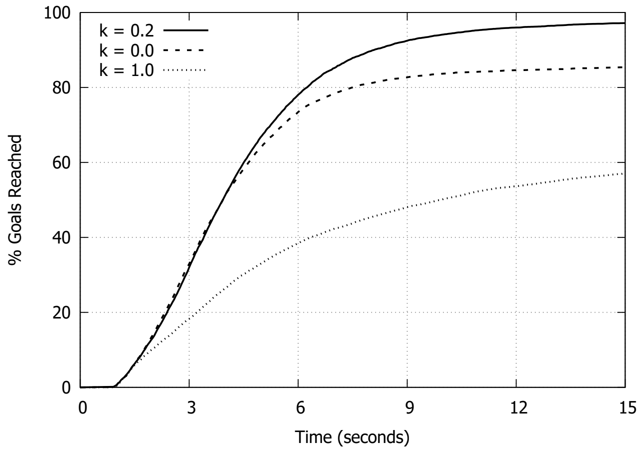

Figure 3: Success rates versus time taken for different settings of the replanning threshold, k.

the skill and its likelihood-of-success estimator were trained from a high-level state representation similar to that in [14], using *Q-Learning* [11, Section 6.5] with *experience replay* [5]. Action-values and likelihoods were approximated via a feed forward architecture with 2 hidden layers each containing 300 neurons. For Dijkstra's algorithm, the search space was configured to be the set of grid squares visible on current screen. Through trial and error, we found that the best setting of the replanning threshold was around $k = 0.2$.

As Figure 3 shows, the strongest configuration of our agent ($k = 0.2$) was broadly successful, reaching 97.1% ($\pm$ 0.2%) of goals within the time limit. However, changing the replanning threshold to $k = 1$ (which equates to continual replanning) was severely detrimental. It appears that there were often many viable plans with similar success probabilities. Small changes in the protagonist's position and velocity were often enough to alter the plan rankings, causing the agent to switch plans frequently without making any progress. Setting the threshold to $k = 0$ so that the agent never replanned (unless the next waypoint became out range) was less detrimental, because the agent rarely made critical mistakes. This configuration was still able to reach 85.4% ($\pm$ 0.4%) of goals within the time limit. However, it was unable to recover when it became stuck on difficult steps. The $k = 0.2$ agent was generally able to recover, albeit with slow completion times. Therefore, its advantage over the $k = 0$ agent only became prominent towards the right of the graph.

4. CONCLUSION

In this paper we introduced an approach for navigating environments where a "god's eye view" is provided but the low-level dynamics are unknown. We tested this approach in *Infinite Mario* and showed that it was capable of solving complex navigation tasks in real-time. We also showed that a variable replanning threshold may yield considerable improvement versus the extremes of continual replanning and never replanning. In the future, we plan to extend our approach to domains that involve non-navigational elements, e.g. enemy avoidance, and domains where the layout of obstacles does not conform to a grid. We also plan to benchmark our approach against pure low-level simulation.

REFERENCES

[1] B. L. Digney. Learning Hierarchical Control Structures for Multiple Tasks and Changing Environments. In *Proceedings of the 5th International Conference on Simulation of Adaptive Behavior*, volume 5, pages 321–330, 1998.

[2] E. J. Jacobsen, R. Greve, and J. Togelius. Monte Mario: Platforming with MCTS. In *Genetic and Evolutionary Computation Conference*, pages 293–300. ACM, 2014.

[3] G. Konidaris and A. Barto. Building Portable Options: Skill Transfer in Reinforcement Learning. In *International Joint Conference on Artificial Intelligence*, volume 7, pages 895–900, 2007.

[4] G. Konidaris and A. Barto. Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining. In *Advances in Neural Information Processing Systems*, pages 1015–1023, 2009.

[5] L.-J. Lin. Reinforcement Learning for Robots Using Neural Networks. Technical report, Defense Technical Information Center (DTIC) Document, 1993.

[6] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic Abstraction in Reinforcement Learning via Clustering. In *Proceedings of the 21st International Conference on Machine Learning*, page 71. ACM, 2004.

[7] I. Menache, S. Mannor, and N. Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. In *Machine Learning: ECML 2002*, pages 295–306. Springer, 2002.

[8] M. Pickett and A. G. Barto. Policyblocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. In *International Conference on Machine Learning*, volume 2, pages 506–513, 2002.

[9] Ö. Şimşek and A. G. Barto. Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. In *Proceedings of the 21st International Conference on Machine Learning*, page 95. ACM, 2004.

[10] Ö. Şimşek, A. P. Wolfe, and A. G. Barto. Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. In *International Conference on Machine Learning*, pages 816–823. ACM, 2005.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, volume 1. Cambridge Univ Press, 1998.

[12] S. Thrun, A. Schwartz, et al. Finding Structure in Reinforcement Learning. *Advances in Neural Information Processing Systems*, pages 385–392, 1995.

[13] J. Togelius, S. Karakovskiy, and R. Baumgarten. The 2009 Mario AI Competition. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[14] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber. Super Mario Evolution. In *IEEE Conference on Computational Intelligence in Games*, pages 156–161. IEEE, 2009.

[15] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, et al. Strategic Attentive Writer for Learning Macro-Actions. In *Advances in Neural Information Processing Systems*, pages 3486–3494, 2016.