Learning Options from Demonstrations: A Pac-Man Case Study

Marco Tamassia, Fabio Zambetta, William L. Raffe, Florian 'Floyd' Mueller, and Xiaodong Li

***Abstract*—Reinforcement Learning (RL) is a machine learning paradigm behind many successes in games, robotics and control applications. RL agents improve through trial-and-error, therefore undergoing a learning phase during which they perform sub-optimally. Research effort has been put into optimising behaviour during this period, to reduce its duration and to maximise after-learning performance. We introduce a novel algorithm that extracts useful information from expert demonstrations (traces of interactions with the target environment) and uses it to improve performance. The algorithm detects unexpected decisions made by the expert and infers what goal the expert was pursuing. Goals are then used to bias decisions while learning. Our experiments in the video game Pac-Man provide statistically significant evidence that our method can improve final performance compared to a state-of-the-art approach.**

***Index Terms*—Reinforcement learning, Temporal Difference Learning, Options Framework, Learning from Demonstration.**

I. Introduction

Reinforcement Learning (RL) is a paradigm for machine learning where behaviour is learned by agents through trial-and-error, as opposed to supervised learning, where algorithms are provided with labeled data. Researchers have achieved significant advancements using RL in various fields, including games [1], [2], robotics [3], and control [4].

Agents using RL techniques (RL agents, hereafter) are of practical interest because some of them do not require a model of the environment they act in, which matches most real world scenarios. To define what a "good" behaviour is, all that needs to be provided to an RL agent is a reward signal. To learn an optimal behaviour without prior information, the RL agent needs time to gather knowledge by interacting with the environment, via trial-and-error. It is to be expected that performance during this learning phase is sub-optimal. In particular, decisions might start out as random and improve over time until they converge to optimal behaviour.

An important part of RL research is aimed at improving the performance during this learning phase [5]. One way in which this can be achieved is by introducing bias in the decision process of the agent while it is still taking some random decisions. Sutton *et al.* achieve this by introducing one or more predefined policies, called "options" that the agent can commit to for some time [6]. It is important that options drive the agent in a profitable way, making good decisions whose consequences the agent will take note of. However, if not well-engineered, this technique may worsen performance [7].

The authors are with RMIT University, Melbourne VIC, Australia.
E-mail: <first.last>@rmit.edu.au

If properly directed during training, an agent can learn better options and learn them earlier. Better options make more effective game players, robots and controllers. Earlier learning means a shorter training time and fewer interactions with the environment, which are often costly. These are two highly desirable characteristics for learning systems.

Hand-crafting options is expensive and error-prone in scenarios where human intuition is difficult to encode. This has prompted work in the automatic learning of options. Notable approaches are based on frequency of successful trajectories [8], detection of bottleneck states [9], [10], commonalities of multiple tasks [11], states clustering [12], graph partitioning [13], relative novelty [14], difficulty for states to be reached and maintained [15], causal analysis of state features [16] or successful trajectories [17] and intrinsic motivation [18]. These approaches all rely on the agent exploring the environment, as opposed to learning from pre-existing data.

In this work, we introduce a novel algorithm that learns options from demonstrations provided by experts; that is, agents familiar with the environment. We investigate whether options learned with our method can improve performance of Q-learning, a popular RL algorithm. Here, a demonstration is a trace of an expert interacting with the environment. The intuition is that options learned from experts are likely to produce a good bias for an RL agent learning phase. Other works on learning options from demonstration have recently been published: based on the frequency of sub-trajectories [19], change of abstractions [20], human interaction [21].

The method proposed in [21] is closely related to our work, both because humans were involved and the same test-bed, the video game Pac-Man, was used. In [21], participants were asked to provide a list of sub-tasks useful to play the game; these sub-tasks were then converted into options. Our approach differs in that we only require players to play the game in their usual manner.

Our approach to learning options from demonstrations is to detect "surprising" decisions made by the experts, infer their intentions and use them as goals. This idea reflects studies in cognitive research where infants were observed to get excited by unexpected events and motivated to explore further [22].

This work builds on our previous work [23]. In this study we change the method of selection for detected goals: whereas previously we used a notion of distance between states, in this work we select the most frequently detected goals, which reduces computational time. Furthermore, in this study we use a more complex test-bed, the video game Pac-Man, which is more challenging than the simple grid-world previously used. Finally, we empirically demonstrate that this approach works

well even when the model of the environment is learned from the demonstrations themselves, rather than being given as input.

The paper is organised as follows: in Section II we give an introduction to Reinforcement Learning and to the framework introduced by Sutton *et al.*; in Section III we present our method for learning options from demonstration; in Section IV we present our experiments and report the results and in Section VI we summarise and discuss future research directions.

II. Background

In this section, we give the background necessary to understand this work. First, we give an introduction to Markov Decision Process (MDP) (II-A), the most common model used in RL. Then we introduce the Options framework (II-B), which allows an RL agent to choose among actions and options, predefined policies that the agent can commit to for some time.

A. MDP and Reinforcement Learning

A Markov Decision Process (MDP) [24] is a model of a stochastic, stationary environment; that is, an environment whose response to an agent's actions is non-deterministic and whose distribution does not change over time. Formally, an MDP M is a tuple $M = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where $\mathcal{S}$ is a set of *states*; $\mathcal{A}$ is a set of *actions*; $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a function expressing the environment *state transition probabilities*; $R : \mathcal{S} \to \mathbb{R}$ is a function associating a *reward* to each state; $\gamma < 1$ is a *discount factor* used to decrease future rewards, which encodes the notion of inflation.

At each point in time t an agent senses the state of the environment $s_t \in \mathcal{S}$ and performs action $a_t \in \mathcal{A}$ in response. The state of the environment changes stochastically according to the distribution $P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1})$. Finally, the agent receives a reward $r_t = R(s_{t+1})$. The purpose of an agent is to maximise the discounted, cumulative reward collected over time: $\sum_t \gamma^{t-1} r_t$. The agent wants to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximises such reward.

Q-learning [25] finds such a policy by learning the expected future cumulative reward of taking action a in state s and behaving optimally – according to its knowledge – afterwards. This information is encoded in the *state-action value function*, or *Q-function*.

Q-learning updates its estimate of the Q-function at every state transition, using (s_t, a_t, r_t, s_{t+1}):

$$Q(s_t, a_t) \overset{\alpha}{\leftarrow} r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t), \qquad (1)$$

where $x \overset{\alpha}{\leftarrow} y$ is short for $x \leftarrow x + \alpha y$ and $0 < \alpha \leq 1$ is the learning rate. This algorithm converges with probability 1 to the optimal Q-function, provided that each state has a non-zero probability of being visited.

An agent focusing only on maximising rewards (exploitation) could be preventing itself from gathering additional knowledge (exploration) that would allow for better decisions. To provide the proper degree of exploration over exploitation, an *exploration strategy* is normally used. In this work, we used the Annealing ϵ-*greedy* strategy, which chooses a random action with probability ϵ and the best (greedy) action with probability $1 - \epsilon$. The value of ϵ is decreased over time (annealed).

MDP actions are limited in that an agent cannot commit to a certain behaviour over time, because decisions are independent of each other; they just depend on the current state. The *options framework* addresses this limitation.

B. Options Framework

The concept of an *option* [6] generalises that of an action (actions are referred to as *primitive actions*) to include temporally extended courses of action. An option, like any other action, can be chosen by the agent policy; however, unlike with primitive actions, when an option is chosen, its internal policy is followed for some period of time.

An option o is defined as a tuple $o = (\mathcal{I}, \pi, \beta)$ where:

- $\mathcal{I}$ is the *initiation set*; that is, the set of states from which the agent can select option o;
- π is the policy to be followed when option o is selected;
- $\beta : \mathcal{S} \to \mathbb{R}$ expresses the probability of termination of option o and depends on the state.

When the agent selects a primitive action, the action is executed normally. On the other hand, when it selects an option, the option is activated and its policy determines this and the next actions to be performed; this continues until the option stochastically terminates. Upon option termination, the following rule is used to update the state-option value [6]:

$$Q(s_t, o_t) \overset{\alpha}{\leftarrow} r + \gamma^k \max_{o \in \mathcal{O}} Q(s_{t+k}, o) - Q(s_t, o_t), \qquad (2)$$

where $r = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{k-1} r_{t+k}$ is the discounted cumulative reward obtained during the option execution and $\mathcal{O}$ is the set of all actions and options. This rule leads Q to converge to the optimal state-option value function.

$\mathcal{I}$, π and β can be defined manually or learned algorithmically. We focus on learning options from expert demonstrations.

III. Generating Options from Demonstration

This section explains how our algorithm learns options from demonstrations, not requiring interaction with the environment.

Our approach focuses on options that have a goal-based policy. This is in line with previous work in the area [8], [9], [13], [14]. The approach we take is slightly different, however, in that it allows for policies with multiple goals. Each set of goals is then used to compute the components of an option ($\mathcal{I}$,π,β). This allows for more flexibility in choices. Details on how this is done are given in section IV-B.

A. Identifying Useful Subgoals

Our procedure requires one or more demonstrations given as input. We define a demonstration as a sequence of state and action pairs: $\langle (s_1, a_1), (s_2, a_2), \ldots, (s_n, a_n) \rangle$. The first step is to identify useful sub-goals in the demonstrations.

Our idea is that experts reason through a few, high-level goals. We aim to identify demonstration sequences of $(s_i, a_i), \ldots, (s_j, a_j)$ that can be explained as steps the expert took to reach a goal, s_{j+1}. That is, the above sequence could be rewritten as $(s_i, \pi_{s_{j+1}}(s_i)), \ldots, (s_j, \pi_{s_{j+1}}(s_j))$ where $\pi_{s_{j+1}}$ is a policy that minimises the expected number of steps to reach state s_{j+1}.

Algorithm 1 Goals extraction; uses a multiset structure, which is a set where information about the number of occurrences of each element is preserved.

1: **procedure** EXTRACT_GOALS(T, demo, ρ)
2: last_s, _ $\leftarrow$ last(demo)
3: goals $\leftarrow$ multiset $\varnothing$
4: prev_s $\leftarrow$ last_s
5: cur_goal $\leftarrow$ last_s
6: **for all** (s, a) in reverse(demo) **do**
7: $\hat{R}(s) \leftarrow \begin{cases} \rho & \text{if } s\text{=cur\_goal} \\ 0 & \text{otherwise} \end{cases}$
8: $\pi \leftarrow$ Value_Iteration$(T, \hat{R})$
9: **if** $\pi(\text{s}) \neq a$ **then**
10: cur_goal $\leftarrow$ prev_s
11: add cur_goal to goals
12: prev_s $\leftarrow$ s
13: **return** goals

Algorithm 1 achieves this by analysing the demonstration backwards. Because the procedure infers goals from expert actions, we assume that the experts are competent in the environment and that this is reflected in the demonstrations produced. The procedure works as follows:

- the current goal is set to the last visited state;
- the demonstration is analysed backwards, step by step;
- at every step (s_t, a_t), the best action a_t^* to reach the current goal from state s_t is computed via Value Iteration [24] (lines 7-8, the model of the environment can be learned from the samples themselves if not already available); Value Iteration is a dynamic programming algorithm that computes a state value function;
- if $a_t \neq a_t^*$, state s_{t+1} is set as the current goal;
- the current goal is added to the goals multiset; this is to keep track of how much each goal is used.

The underlying idea is that decisions of the expert that go against the expectations of the algorithm are intrinsically interesting. The algorithm tries to infer what the expert goal was when it took the unexpected action.

Algorithm 1 finds an exact solution only in deterministic MDPs. In stochastic MDPs, a state and action pair does not uniquely determine the next state. As a consequence, the next state in the demonstration is not necessarily the goal that the expert wanted to achieve. Not knowing the intent of the expert, we assume that the next state is what the expert wished to achieve. This is a heuristic that seems to work well in practice.

IV. EXPERIMENTS

This section details the experimental setting that has been used to test the proposed method. Results of the experiments are also presented here. The experiments are designed to show the benefits of using options learned by our algorithm. Notice that the purpose of this work is not to implement the best Pac-Man agent but, to show that using options learned from experts gives learning agents an advantage. These are two different problems: engineering Pac-Man agents to achieve the best performance versus *autonomously* learning how to effectively play. We choose to focus on the second problem, and use Pac-Man as a test-bed.

A. Experimental settings

We implemented our framework in Python using the Numpy library[1] [26] and parallelized using GNU Parallel[2] [27].

Q-learning: The discount factor for Q-learning was $\gamma = 0.97$ (a commonly chosen value [24]), while the reward peak for goal-driven policies was arbitrarily set to $\rho = 50$. The exploration strategy used in the experiments is Annealing ϵ-greedy, with $\epsilon = 0.1$ (also, a common choice [24]). The annealing schedule is based on the episode number k and is defined as $\epsilon(k) = 0.1/(1 + e^{\frac{3k}{1000}-3})$. These parameters have been chosen to create a sigmoid function where most of the descent happens between episodes 500 and 1500.

Pac-Man: Our algorithm has been tested in the video game *Pac-Man*[3]. We used the implementation of Pac-Man developed by University of California, Berkeley, for their Artificial Intelligence course [28][4].

In this game, the player controls Pac-Man, an agent moving in a two-dimensional environment whose purpose is to collect all the white pills. Pac-Man is chased by a number of ghosts, each of which will kill Pac-Man if he collides with them. There are also a number of special capsules that make all the ghosts scared for a limited period of time. If Pac-Man collides with a scared ghost, the ghost dies and reappears at the centre of the game area in a non-scared state. Players receive a score based on their performance:

- -1 at every time-step;
- +10 for every eaten pill;
- 0 for every eaten capsule;
- +200 for every killed ghost;
- +500 upon victory (eating all pills);
- -500 upon defeat (being eaten by a ghost).

Notice that this implementation is slightly different from the most popular ones in AI literature, such as those used in the *Ms. Pac-Man Competition*[5] and in the *Ms. Pac-Man vs Ghost-Team Competition*[6]. In particular, in this implementation there is a time-step penalty and the behaviour of ghosts is more simplistic, each ghost deciding a random direction at every intersection. The simulator we chose is, however, not uncommon, and has in particular been used in [21].

State space: Each state of the game is characterised by distance and direction from the closest of each of the following: capsule, food, scared ghost and non-scared ghost.

The distance information can assume values *close*, *midrange* and *far*, depending on the length of the shortest path as computed by the Dijkstra algorithm [29]. Directional information represents the direction that Pac-Man should follow to reach said objects via the shortest path.

[1] http://www.numpy.org/
[2] http://www.gnu.org/software/parallel/
[3] http://www.gamasutra.com/view/feature/3938/the_pacman_dossier.php
[4] http://ai.berkeley.edu
[5] http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html
[6] http://csee.essex.ac.uk/staff/sml/pacman/kit/AgentVersusGhostsOld.html

Automatic agents: The experiments compare the goals detected by the algorithm from two automatic agents. That is, these automatic agents are two of the experts in our setting.

Both agents include a perfect model of the game. They both compute a heuristic value for the state that would follow each available action, and select the best one. The score is computed taking into account information about non-scared ghosts and pills as follows:

$$score_c = \begin{cases} -\max(0, 7-g)^2 - \frac{p}{100} & \text{if there are pills} \\ -\max(0, 7-g)^2 + 1000 & \text{otherwise} \end{cases}, \quad (3)$$

where g is the length of the shortest-path to the nearest non-scared ghost and p is the length of the shortest-path to the nearest pill.

Conservative agent: The first agent, which we call *conservative*, uses Equation 3 as is to evaluate the states' value. This formula encourages reaching the closest pill while staying 7 steps away from the closest non-weak ghosts. The agent, thus, avoids taking risks and stays at a safe distance from threats. While this does not take into account the possibility of ambushes, it is effective in practice.

Aggressive agent: The second agent, which we call *aggressive*, takes into account the same information as the conservative as well as information on scared ghosts and capsules. The formula it uses is as follows:

$$score_a = score_c - 1000 \cdot c - 100 \cdot n - 10 \cdot w, \quad (4)$$

where c is the number of capsules present on the level, n is the number of scared ghosts and w is the length of the shortest-path to the nearest scared ghost. The formula in Equation 4 encourages eating capsules and scared ghosts. This behaviour is risky because it does not take into account for how long the ghosts are going to remain scared.

Human players: The other two experts in our experiments are humans. Specifically, two of the authors played Pac-Man on a small level and all of their games are recorded and treated as experts data.

B. Experimental procedure

The structure of the experiments for each agent $A \in \{\text{cons}, \text{aggr}\}$ (Section IV-A) is as follows:

1) Sets D_{Cons} and D_{Aggr} of 1000 games are recorded for each automatic agent.
2) Sets D_{P1} and D_{P2} of 30 games are recorded from two of the authors.
3) A model of the environment T is learned analysing all the above mentioned games.
4) Algorithm 1 is used to extract goals from games:
5) For each $A \in \{\text{Cons}, \text{Aggr}, \text{P1}, \text{P2}\}$:
   a) $G_A = \biguplus_{d \in D_A} \text{EXTRACT\_GOALS}(T, d)$, where $\biguplus$ indicates a multiset sum. A multiset is a set where information about the number of occurrences of each element is preserved. Notice that Algorithm 1 returns a multiset.
   b) States with equal values for selected features (see subsection at the end of this section for details) are aggregated; let F be the set of such features: $G_{A,\boldsymbol{v}} = \{s_i \in \mathcal{S} \mid \Pi_F(s_i) = \boldsymbol{v}\}$, for all combinations $\boldsymbol{v}$ of features in F, where Π_F is the relational algebra projection operator, which strips the input of all features except those in F. Notice that each $G_{A,\boldsymbol{v}}$ is a multiset, as opposed to a set.
   c) Rank all multisets $G_{A,\boldsymbol{v}}$ based on their cardinality in descending order.
   d) For each of the first top 4 multisets $\overline{G}$ of goals:
      i) Reward function $\overline{R}$ is created; $\overline{R}$ is 0 everywhere except for states in $\overline{G}$, where it is ρ;
      ii) Value Iteration is run with reward $\overline{R}$ and model T to find policy $\overline{\pi}$;
      iii) Set $\overline{G}^*$ is computed: all states that can indirectly reach states in $\overline{G}$ are included in $\overline{G}^*$;
      iv) An option $o = (\mathcal{I}, \pi, \beta)$ is created:
      - $\mathcal{I} = \overline{G}^* \setminus \overline{G}$;
      - $\pi = \overline{\pi}$;
      - $\beta(s) = \begin{cases} 0.2 & \text{if } s \in \overline{G}^* \setminus \overline{G} \\ 1 & \text{otherwise} \end{cases}$.

      v) o is added to $\mathcal{O}_A$

An extra set of options is then created: $\mathcal{O}_{\text{subr}}$, containing the options proposed in [21] for Pac-Man; namely: go to the closest food, go to the closest capsule, go to the closest ghost, avoid the closest ghost; policies are handcrafted, termination probability is 0.2, initiation set is $\mathcal{S}$.

Q-learning is then run 200 times for 3000 episodes with agents using $\mathcal{A}$, $\mathcal{A} \cup \mathcal{O}_{\text{aggr}}$, $\mathcal{A} \cup \mathcal{O}_{\text{cons}}$, $\mathcal{A} \cup \mathcal{O}_{\text{P1}}$, $\mathcal{A} \cup \mathcal{O}_{\text{P2}}$ or $\mathcal{A} \cup \mathcal{O}_{\text{subr}}$ as the options sets.

Features: The features in F, which we used to aggregate goals, are those that do not carry directional information. This choice was made to address redundancy in the goals and also by the limited utility of directional information in the definition of goals. Since directional information is descriptive of specific details rather than high-level ideas, it is less useful to characterise a goal.

V. Results

In this section we report the results of our experiments. First, we quantitatively analyse the data, and then we discuss the top detected goals and the performance of Q-learning with and without options learned with our algorithm.

Q-learning performance

The experiments showed that using options learned by our algorithm leads to final better performance than using the options proposed in [21]. Figure 1 shows the distribution of per-episode total reward for each agent over all episodes, while Figure 2 shows it for 300 extra episodes run after learning, with exploration and learning deactivated. It can be observed that our approach produces superior final performance to that of [21], while being inferior during learning. This is confirmed by a statistical analysis: we computed the effect size between all pairs of agents. Table I reports the two-samples t-tests p-values and the Cohen coefficients found for the largest effect sizes. All of the p-values are < 0.001, normally considered small,

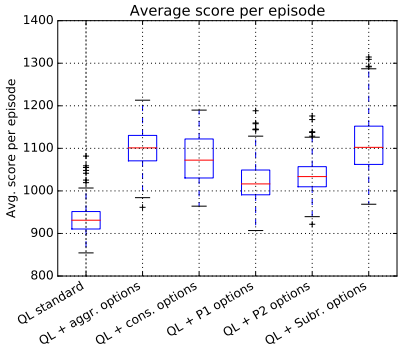

Figure 1. Per-episode return during learning, averaged over 3000 games and 200 repetitions.

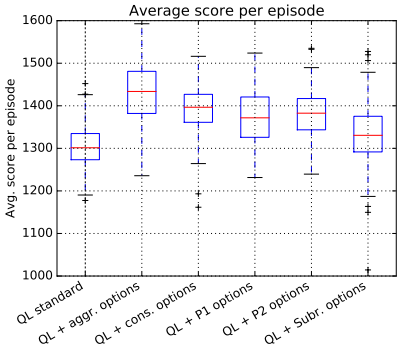

Figure 2. Per-episode return after learning, averaged over 300 games and 200 repetitions.

Stronger	Weaker	Eff. size	p-value
Aggr.	No opt.	2.123	< 0.001
Cons.	No opt.	1.738	< 0.001
P2	No opt.	1.527	< 0.001
Aggr.	Subr.	1.466	< 0.001
P1	No opt.	1.230	< 0.001
Cons.	Subr.	1.017	< 0.001
Aggr.	P1	0.865	< 0.001
P2	Subr.	0.848	< 0.001
Aggr.	P2	0.791	< 0.001
P1	Subr.	0.662	< 0.001

Table I
Largest effect sizes of average reward per episode after learning (as in Figure 2).

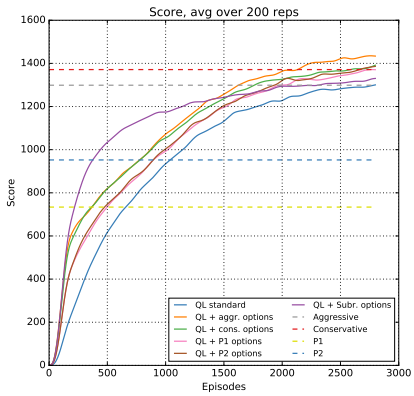

Figure 3. Episode score (average across 200 runs, moving average of size 200.)

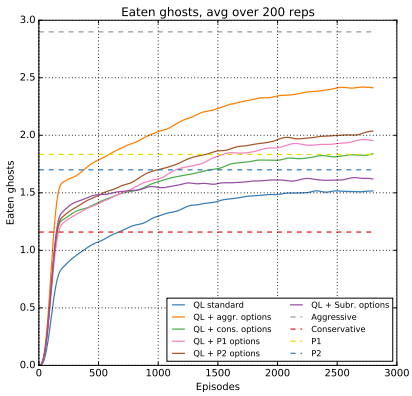

Figure 4. Eaten ghosts per episode (average across 200 runs, moving average of size 200.)

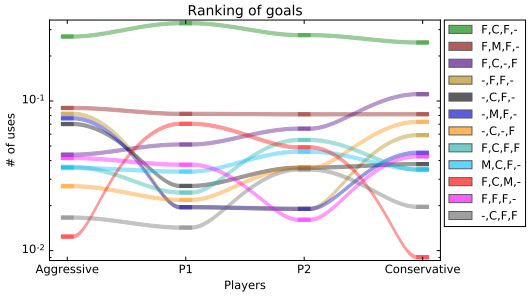

Figure 5. Frequency of use per player of the 12 most used goals (Y axis has a logarithmic scale). The values in the legend characterise each goal; they indicate, in order: distance from the closest capsule, food, non-scared ghost and scared ghost. Abbreviations: - is none, C is close, M is mid-range, F is far.

and most of the effect sizes are > 0.8, normally considered large. This confirms that the differences found are unlikely to be due to chance and large enough to be of practical relevance.

Performance of all the agents over time is shown in Figure 3. It is interesting to observe that the skill showed in the demonstrations does not seem to correlate with the performance of the learning agents. The difference in performance between the agents can be explained in most cases by the number of eaten ghosts, as shown in Figure 4. In fact, it seems agents using options produced by our approach show superior performance for this reason: the agent using options from [21] (labeled "QL + Subr. options") is the only outlier, being surpassed quite early during learning in terms of eaten ghosts, but maintaining an advantage in terms of reward (i.e., score) for some time.

Extracted goals

Analysing the extracted goals can give useful insights into how the proposed algorithm works. Figure 5 shows the 12 most used goals (across all players); for each goal and each player, it shows the relative frequency of use. From the figure, it is possible to infer what are the most used goals for each player and, therefore, the options generated (the top 4) for each agent. The features used to describe goals are, in order, distance from the closest capsule, the closest food, the closest non-scared ghost and the closest scared ghost. For brevity, we will indicate them as $(\mathrm{C}, \mathrm{F}, \mathrm{NSG}, \mathrm{SG})$ in the following discussion.

Figure 5 gives insights on the strategy preferred by each player. Most noticeable is that all players have a strong preference for goals $(\mathrm{far}, \mathrm{close}, \mathrm{far}, \mathrm{none})$ and $(\mathrm{far}, \mathrm{mid}, \mathrm{far}, \mathrm{none})$, indicating a tendency to stay away from non-scared ghosts (NSG = far) and to go after food (F = close and F = mid).

The conservative agent has the strongest tendency to ignore scared ghosts, as indicated by its preference for goal $(\mathrm{far}, \mathrm{close}, \mathrm{none}, \mathrm{far})$ over $(\mathrm{far}, \mathrm{mid}, \mathrm{far}, \mathrm{none})$ (in particular, SG = far). The other goal in the top 4 of the conservative agent is $(\mathrm{none}, \mathrm{close}, \mathrm{none}, \mathrm{far})$, again indicating a tendency to leave scared ghosts alone (SG = far) but, at the same time, to eat capsules (C = none).

On the other hand, the aggressive agent shows a strong preference for eating scared ghosts, as indicated by its preference for goals $(\mathrm{none}, \mathrm{far}, \mathrm{far}, \mathrm{none})$ and $(\mathrm{none}, \mathrm{mid}, \mathrm{far}, \mathrm{none})$, both having C = none and SG = none; i.e., all capsule eaten and no scared ghosts alive. Reaching these conclusions from the table is not intuitive because our approach is based on detection of goal states rather than actions. Figure 5 shows, as goals, the states following the action of eating. On the other hand, a human would intuitively consider the action of eating as a goal in itself. While this is not possible to detect with our algorithm, it is a direction for future research to narrow the gap between human reasoning and the algorithm output.

Interestingly, both human players are more flexible in terms of how near they dare to be to non-scared ghosts, as indicated

by goal $(\text{far}, \text{close}, \text{mid}, \text{none})$; this is an obvious consequence of the difference between hard-coded rules and fuzzy brains, but it is remarkable that the algorithm detected it. This is also confirmed by another behaviour of the automatic players, which use goals $(\text{none}, \text{far}, \text{far}, \text{none})$ and $(\text{none}, \text{mid}, \text{far}, \text{none})$ more often than human players: the automatic agents will always move away from food in order to stay away from ghosts since it is in their rules; the human players, on the other hand, are prone to taking risks.

Comparison with Subramanian et al. *:* As stated before, the work most similar to ours in the literature is [21]. We realise that, during learning, agents using our options produce inferior performance compared to that of an agent using options by Subramanian *et al.* However, our approach produces superior performance after learning. In addition, our approach does not require users to perform any specific tasks other than playing the game; conversely, in [21], participants were asked to explicitly contribute suggestions on how to improve the AI. We believe there is value in the weaker assumptions of our approach.

The difference in performance between our experiments and those reported in [21] can be explained by the different state representations and Q-learning algorithms used: in [21], the state representation is richer than ours, and they used approximated Q-learning to handle it; we, on the other hand, are using tabular Q-learning, which cannot handle a state representation as rich. We chose to use tabular Q-learning because our extraction algorithm can be simply defined on a discrete state-space; however, extending this work to approximated Q-learning is a promising avenue of future research.

VI. Conclusion

In this work, we investigated whether options learned from expert demonstrations extracted by our novel algorithm can improve the performance of Q-learning. We have extended the algorithm proposed in [23] that extracts sub-goals from expert demonstrations and builds options based on them. We tested the algorithm in a more complex scenario, the video game Pac-Man. We compare our approach to the work by Subramanian *et al.* [21]. In our experiments, agents using options generated by our algorithm, on average, produce better final performance than the agent using options proposed in [21]. The results we obtained suggest that our option-learning framework provides a viable way for harnessing expert knowledge. A comparison with other methods in the same family would nevertheless be interesting.

The proposed algorithm can be improved: currently, our approach relies on domain knowledge regarding what features are to be ignored for aggregating states. Possible future work may include a mechanism to learn these features from the demonstrations themselves. Another important limitation of this work is that the approach relies on tabular Q-learning, as opposed to approximated Q-learning. Improving this aspect will allow the algorithm to deal with richer state representations. Another avenue for future work is to replace multi-peaked rewards with potential-based shaping rewards, which could bias the agents even better during learning [30]. Finally, the goals learned by our technique could prove useful tools to characterise player styles; we therefore envision a larger study to investigate such application.

Acknowledgements

The authors acknowledge support from ARC LP130100743. We would like to thank the RMIT VxLab for their support.

References

[1] P. Stone and R. S. Sutton, "Scaling reinforcement learning toward robocup soccer", in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 537–544.

[2] J. Baxter, A. Tridgell, and L. Weaver, "Knightcap: A chess programm that learns by combining TD(lambda) with game-tree search", in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 28–36.

[3] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey", English, in *Reinforcement Learning*, ser. Adaptation, Learning, and Optimization, M. Wiering and M. van Otterlo, Eds., vol. 12, Springer Berlin Heidelberg, 2012, pp. 579–610.

[4] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning", English, in *Experimental Robotics IX*, ser. Springer Tracts in Advanced Robotics, J. Ang Marcelo H. and O. Khatib, Eds., vol. 21, Springer Berlin Heidelberg, 2006, pp. 363–372.

[5] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*, ser. Adaptation, Learning, and Optimization. Springer-Verlag Berlin Heidelberg, 2012.

[6] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning", *Artificial Intelligence*, vol. 112, no. 1–2, pp. 181–211, 1999.

[7] N. K. Jong, T. Hester, and P. Stone, "The utility of temporal abstraction in reinforcement learning", in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS '08, Estoril, Portugal: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 299–306.

[8] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density", in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 361–368.

[9] M. Stolle and D. Precup, "Learning options in reinforcement learning", English, in *Abstraction, Reformulation, and Approximation*, ser. Lecture Notes in Computer Science, S. Koenig and R. Holte, Eds., vol. 2371, Springer Berlin Heidelberg, 2002, pp. 212–223.

[10] I. Menache, S. Mannor, and N. Shimkin, "Q-cut—dynamic discovery of sub-goals in reinforcement learning", English, in *Machine Learning: ECML 2002*, ser. Lecture Notes in Computer Science, T. Elomaa, H. Mannila, and H. Toivonen, Eds., vol. 2430, Springer Berlin Heidelberg, 2002, pp. 295–306.

[11] M. Pickett and A. G. Barto, "Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning", in *Proceedings of the Nineteenth International Conference on Machine Learning*, Morgan Kaufmann, 2002, pp. 506–513.

[12] 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*, ser. ICML '04, Banff, Alberta, Canada: ACM, 2004, pp. 71–78.

[13] 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*, ser. ICML '04, Banff, Alberta, Canada: ACM, 2004, pp. 95–102.

[14] Ö. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning", in *Proceedings of the 22nd International Conference on Machine learning*, ser. ICML '05, Bonn, Germany: ACM, 2005, pp. 816–823.

[15] A. Bonarini, A. Lazaric, and M. Restelli, "Incremental skill acquisition for self-motivated learning animats", English, in *From Animals to Animats 9*, ser. Lecture Notes in Computer Science, S. Nolfi, G. Baldassarre, R. Calabretta, J. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi, Eds., vol. 4095, Springer Berlin Heidelberg, 2006, pp. 357–368.

[16] A. Jonsson and A. Barto, "Causal graph based decomposition of factored MDPs", *J. Mach. Learn. Res.*, vol. 7, pp. 2259–2301, Dec. 2006.

[17] N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich, "Automatic discovery and transfer of maxq hierarchies", in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, Helsinki, Finland: ACM, 2008, pp. 648–655.

[18] C. M. Vigorito and A. G. Barto, "Intrinsically motivated hierarchical skill learning in structured environments", *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 132–143, Jun. 2010.

[19] P. Zang, P. Zhou, D. Minnen, and C. Isbell, "Discovering options from example trajectories", in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09, Montreal, Quebec, Canada: ACM, 2009, pp. 1217–1224.

[20] G. Konidaris, S. Kuindersma, R. Grupen, and A. S. Barreto, "Constructing skill trees for reinforcement learning agents from demonstration trajectories", in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., Curran Associates, Inc., 2010, pp. 1162–1170.
[21] K. Subramanian, C. L. Isbell, and A. L. Thomaz, "Learning options through human interaction", in *In 2011 IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT)*, Citeseer, 2011.
[22] A. E. Stahl and L. Feigenson, "Observing the unexpected enhances infants' learning and exploration", *Science*, vol. 348, no. 6230, pp. 91–94, 2015. eprint: http://www.sciencemag.org/content/348/6230/91.full.pdf.
[23] M. Tamassia, F. Zambetta, W. Raffe, and X. Li, "Learning options for an MDP from demonstrations", English, in *Artificial Life and Computational Intelligence*, ser. Lecture Notes in Computer Science, S. Chalup, A. Blair, and M. Randall, Eds., vol. 8955, Springer International Publishing, 2015, pp. 226–242.
[24] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st. Cambridge, MA, USA: MIT Press, 1998.
[25] C. J. C. H. Watkins, "Learning from delayed rewards.", PhD thesis, University of Cambridge, 1989.
[26] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation", *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.
[27] O. Tange, "Gnu parallel - the command-line power tool", *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, Feb. 2011.
[28] J. DeNero and D. Klein, "Teaching introductory artificial intelligence with pac-man", in *Proceedings of the Symposium on Educational Advances in Artificial Intelligence*, 2010.
[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.
[30] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping", in *ICML*, vol. 99, 1999, pp. 278–287.

Marco Tamassia is a Ph.D. student in RMIT University (Melbourne, Australia), supervised by Dr. Fabio Zambetta. In his Ph.D., he has been working on Reinforcement Learning, Augmented Reality and Video Games. He obtained his B.CompSci and M.CompSci in University of Verona and has been a Research Fellow in the PAVIS lab. at the Italian Institute of Technology (IIT), where he worked on Multi-Agent Coordination Systems. He is a member of the RMIT Evolutionary Computing and Machine Learning Group (ECML).

William L. Raffe is a Research Fellow at the School of Science (Computer Science and Software Engineering) at RMIT University (Melbourne, Australia), where he also previously received a B.CompSci(Honours) in 2009 and a PhD(CompSci) in 2014. His research focus revolves around a variety of game design and computational intelligence topics, including player modelling, virtual agent learning and control, search-based procedural content generation, game analytics, and human computer interaction in augmented and virtual reality. He is a senior member of the RMIT Evolutionary Computing and Machine Learning Group (ECML) and of the Artificial Intelligence and Game Design node of the Centre for Game Design Research (CGDR).

Fabio Zambetta is a Senior Lecturer at RMIT University, Australia where he coordinates the Games and Graphics Programming degree. His main research interests include artificial intelligence in games, reinforcement learning and virtual, augmented, mixed reality. He is an IEEE member and a member of the IEEE Games Technical Committee.

Florian "Floyd" Mueller is a professor at RMIT University, directing the Exertion Games Lab. The Exertion Games Lab works on supporting people experiencing their bodies as digital play, situated within a broader interaction design agenda that supports people's values such as an active life. Floyd has been a Fulbright Fellow at Stanford University, having worked on the topic of exertion games across four continents, including at organizations such as the MIT Media Lab, Media Lab Europe, Fuji-Xerox Palo Alto Laboratories, Xerox Parc, the University of Melbourne, CSIRO and Microsoft Research Asia.

Xiaodong Li (M'03-SM'07) received his B.Sc. degree from Xidian University, Xi'an, China, and Ph.D. degree in information science from University of Otago, Dunedin, New Zealand, respectively. He is a Professor with the School of Science (Computer Science and Software Engineering), RMIT University, Melbourne, Australia. His research interests include evolutionary computation, neural networks, data analytics, multiobjective optimization, multimodal optimization, and swarm intelligence. He serves as an Associate Editor of the IEEE Transactions on Evolutionary Computation, Swarm Intelligence (Springer), and International Journal of Swarm Intelligence Research. He is a founding member of IEEE CIS Task Force on Swarm Intelligence, a vice-chair of IEEE Task Force on Multimodal Optimization, and a former chair of IEEE CIS Task Force on Large Scale Global Optimization. He is the recipient of 2013 ACM SIGEVO Impact Award and 2017 IEEE CIS "IEEE Transactions on Evolutionary Computation Outstanding Paper Award".