Macro Actions in Reinforcement Learning

Varun Sreedhara Bhatt
140260004

Kalpesh Krishna
140070017

Vihari Piratla
173050039

Abstract

Repeating the same action across multiple contiguous time-steps ("macro-actions") in a reinforcement learning setting speeds up the computation and performs better (on certain tasks) than the case when action is chosen at every time step. In this work, we compare multiple algorithms that exercise the macro-actions heuristic on a task to learn a defense agent in Half Field Offense problem (HFO). A version of agent which repeats an action for a static number of time steps (DI-SARSA) is found to perform much worse than an algorithm that predicts how many time steps to repeat a certain action (FiGAR) on HFO. Further, we propose and compare a few other simple techniques that align with the heuristic and improve over DI-SARSA.[1]

Introduction

In reinforcement learning, an agent interacts with the environment, obtaining observations and rewards. Through this, it attempts to learn the optimal actions it needs to take at each step. The environment is assumed to be a Markov Decision Process (MDP) with state space $\mathbf{S}$, action space $\mathbf{A}$, reward function $\mathbf{R} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}$, transition function $\mathbf{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}$ and discount factor γ. The agent is in state s^t at time t, takes an action a^t, gets reward r^t and goes to state s^{t+1}. Its objective is to maximise the expected long term reward

$$\mathbb{E}\left[r^0 + \gamma r^1 + \gamma^2 r^2 + ...\right]$$

Reinforcement learning algorithms like SARSA and Q-learning maintain a table containing the values of $Q(s, a)$ defined as the maximum expected long term reward obtained by starting in state s, taking action a and following the optimal policy after that. A greedy in limit with infinite exploration (GLIE) policy based on the Q values is used by the agent. The table of Q values is updated based on the rewards obtained from the environment.

When the state space is very large or even continuous, it is not possible to maintain a table of Q values. In these cases, function approximation is used to approximate $Q(s, a) = \sum_i w_i \phi_i(s)$ where $\phi_i(s)$ are the features corresponding to state s and w_i are the weights given to feature $\phi_i(s)$.

[1]We have open sourced our implementation of these techniques at https://github.com/martiansideofthemoon/macro-action-rl.

In both tabular and function approximation case, choosing the action to take and updating the parameters happen at every time step. This may be unnecessary in cases where neighboring states are very similar and hence, their Q values are close to each other. Instead, taking a single action for an interval of time would lead to less number of updates while not reducing the performance. In fact, in certain domains it even improves the performance since the resulting action space is smooth. Another advantage of repeating actions is that it can introduce temporal abstraction in the policy, easing the transition to temporally distant advantageous states.

Our contribution is a study of five algorithms learning to play as defensive agents on the HFO problem (Hausknecht *et al.* 2016), a two-dimensional soccer game. Our five algorithms are,

- **Fixed Decision Interval** - Sarsa(λ) updates are separated by a fixed decision interval as done in (Siddharth 2017).
- **FiGAR (Fine Grained Action Repitition)** - An independent agent learns to predict a decision interval alongside the defense agent, similar to idea given in (Sharma *et al.* 2017)
- **Reward Regularization** - Penalizing the defense agent when it changes the action.
- **Augmented Action Spaces** - By expanding the action space to include both action to take and interval to repeat it as mentioned in (Lakshminarayanan *et al.* 2016).
- **Conditional FiGAR** - Similar to FiGAR, an independent agent learns to predict decision intervals. However, unlike FiGAR, the prediction takes into account the action taken by the defense agent.

Related Work

(Siddharth 2017) explores the effect of repeating actions in a fixed interval by using an algorithm called DI (decision interval)-SARSA. This algorithm is a modification of SARSA algorithm where action is repeated for d steps and the reward obtained during this period ($r^{t-d+1} + \gamma r^{t-d+2} + ...\gamma^{d-1} r^t$) is used in the SARSA update at time t and a new action is chosen. DI-SARSA is applied on Acrobot and HFO defense domains and in both cases $d > 1$ gave better results compared to $d = 1$.

In (Lakshminarayanan *et al.* 2016), the action space is expanded from $\{a_0, a_2, ..., a_{n-1}\}$ to $\{a_0, a_2, ..., a_{n-1}, a_n, ..., a_{2n-1}\}$. The action taken is $a_{i\%n}$ if i^{th} action is chosen and it is repeated for d_1 times if $i < n$ and d_2 times otherwise. This action space expansion is used on ATARI games and the agent is trained with DQN algorithm. Improvements are seen over DQN when action space expansion is applied.

(Sharma *et al.* 2017) propose a method called FiGAR to improve existing reinforcement learning algorithms to learn temporal abstractions in the policy space. There are 2 networks, one which gives the policy using the existing algorithm and other which will give the number of repetitions. The loss function for the networks is a combination of losses of both the policies. This framework is applied to A3C, TRPO and DDPG algorithms and the agent is shown to learn better policies.

Problem & MDP Description

We use the Half-Field Offense framework (Hausknecht *et al.* 2016) to implement all our algorithms and run experiments. The HFO problem is a two-dimensional soccer game restricted to one half field of a soccer pitch. The task is episodic, and continues until the offensive side scores a goal, the defensive team captures the ball or the ball is kicked out of play. We focus on 2v2 game, where offensive agents and a goalkeeper are controlled by a fixed strategy, and our agent is a single defensive player.

The state space of this MDP is continuous. At any state, the agent is permitted to take one of the following high level actions - *MARK_PLAYER_i* (i is one of the opponents), *REDUCE_ANGLE_TO_GOAL*, *GO_TO_BALL* and *DEFEND_GOAL*, *MOVE* and *NO_OP*. An episode ends with a reward -1 whenever the offensive team scores a goal. In all other cases, the reward is +1. While an episode is proceeding, a reward of 0 is given at every step. A discount factor $\gamma = 1$ is used for this MDP(S, A, R, T, γ).

Methods

Since we are dealing with a continuous state-space, a total of 15 features are extracted as the observation and function approximation is used via tile coding. An ϵ-greedy policy is followed while selecting the next action.

SARSA(λ) update is given by:

$$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$$
$$e(s, a) \leftarrow e(s, a) + 1$$

where s is the current state, a is the action taken, s' is the next state and a' is the next action that will be taken, e is eligibility trace that is set to zero at the start of episode. Additionally, $\forall s, a$

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$$
$$e(s, a) \leftarrow \gamma\lambda e(s, a)$$

where α is the learning rate.

Since $Q(s, a)$ in our experiments is a function of weights of features, the update for Q is equivalent to

$$\theta^{t+1} \leftarrow \theta^t + \alpha\delta e(s, a)\nabla_\theta Q(s, a)$$

Fixed Decision Interval

We outline the fixed decision interval algorithm used in (Siddharth 2017) in **Algorithm 1**. Updates and action changes are only performed after a fixed decision interval d. In our experiments we tuned over intervals $1, 2, 4, 8, 16, 32, 64$. Note that $d = 1$ corresponds to a vanilla SARSA(λ) update.

FiGAR

Our implementation of FiGAR uses the framework mentioned in **Algorithm 1** of (Sharma *et al.* 2017). We now construct an independent agent (denoted by $Q_i(s, a')$) which predicts a decision interval for every state s. This agent has its own set of function approximation weights. Here, $a' \in \{1, 2, 4, 8, 16, 32, 64\}$. We outline our implementation in **Algorithm 2**. Notice that both $Q_i(s, a')$ and $Q_a(s, a)$ are updated together with the same reward. We decided to use the same λ for SARSA updates for both agents in our implementation, but we could use different λ_1 and λ_2 in the most general case.

Reward Regularization

This is a simple tweak to the vanilla algorithm. After every step, a constant (penalty p) is subtracted from the reward received whenever there is a change in action. This changes the underlying MDP $M = (S, A, R, T, \gamma)$ to $M_{reg} = (S, A, R_{reg}, T, \gamma)$. We model our reward $R_{reg}(s, a, s', a') = R(s, a, s')$ if $a = a'$ else $R_{reg}(s, a, s', a') = R(s, a, s') - p$. Here, a' is the previously taken action. The policy learnt for M_{reg} will have lesser action changes compared to that of M due to regularization. We outline our implementation in **Algorithm 3**.

Augmented Action Space

Our augmented action space implementation is based on (Lakshminarayanan *et al.* 2016). Let's say we wish to model I decision intervals (in our implementation, $I = 6$, $i \in \{1, 2, 4, 8, 16, 32, 64\}$). For every action a_1, we add I actions to the new action space, denoted by $a'_{1,i}$. This new action space has a total of $I * |A|$ actions. We outline our implementation in **Algorithm 4**. Note that in our implementation, an action space of $\{a_0, a_2, ..., a_{n-1}\}$ is expanded as $\{a_{0,0}, a_{0,1}, ..., a_{0,I}, a_{1,0}, ..., a_{1,I}, ..., a_{n-1,0}, ..., a_{n-1,I}\}$ as opposed to $\{a_{0,0}, ..., a_{n-1,0}, a_{0,1}, ..., a_{n-1,1}, ..., a_{0,I}, ..., a_{n-1,I}\}$ which is suggested by the paper.

Conditional FiGAR

This is a direct extension of the FiGAR algorithm described earlier. We augment the state space of $Q_i(s, a')$ to include the action chosen for the current time-step. Note that in FiGAR, the decision interval was predicted based on the current state, independent of the agent's predicted action. We augment the feature vector $\phi(S)$ to include the predicted action. We outline our implementation in **Algorithm 5**.

Experiment Details

Tile Coding (CMAC) with per dimension resolution of 0.1 is used to approximate action-value function. Our vanilla

input: λ, decision interval d
$Q(s, a) \leftarrow 0$;
for *episodes* $1...E$ **do**
    step $\leftarrow$ 0;
    s $\leftarrow$ initial state;
    action $\leftarrow$ ϵ-greedy(Q);
    while *episode not over* **do**
        $r, s' \leftarrow$ simulateHFO(action);
        step $\leftarrow$ step + 1;
        if *step % d == 0* **then**
            action $\leftarrow$ ϵ-greedy(Q);
            SARSA(λ) update of $Q(s, a)$;
            step $\leftarrow$ 0;
        end
        s $\leftarrow$ s';
    end
    SARSA(λ) update of $Q(s, a)$;
end

Algorithm 1: Fixed Decision Interval

input: λ
$Q_a(s, a) \leftarrow 0$;
$Q_i(s, a') \leftarrow 0$;
for *episodes* $1...E$ **do**
    s $\leftarrow$ initial state;
    action $\leftarrow$ ϵ-greedy(Q_a);
    d $\leftarrow$ ϵ-greedy(Q_i);
    step $\leftarrow$ 0;
    while *episode not over* **do**
        $r, s' \leftarrow$ simulateHFO(action);
        step $\leftarrow$ step + 1;
        if *step % d == 0* **then**
            action $\leftarrow$ ϵ-greedy(Q_a);
            d $\leftarrow$ ϵ-greedy(Q_i);
            SARSA(λ) update of $Q_a(s, a)$;
            SARSA(λ) update of $Q_i(s, a')$;
            step $\leftarrow$ 0;
        end
        s $\leftarrow$ s';
    end
    SARSA(λ) update of $Q_a(s, a)$;
    SARSA(λ) update of $Q_i(s, a')$;
end

Algorithm 2: FiGAR - SARSA(λ)

input: λ, penalty p
$Q(s, a) \leftarrow 0$;
for *episodes* $1...E$ **do**
    s $\leftarrow$ initial state;
    previous = NULL;
    action $\leftarrow$ ϵ-greedy(Q);
    while *episode not over* **do**
        $r, s' \leftarrow$ simulateHFO(action);
        if *action != previous* **then**
            $r \leftarrow r - p$;
        end
        previous $\leftarrow$ action;
        action $\leftarrow$ ϵ-greedy(Q);
        SARSA(λ) update of $Q(s, a)$;
        s $\leftarrow$ s';
    end
    SARSA(λ) update of $Q(s, a)$;
end

Algorithm 3: Regularized Reward

input: λ, $intervals$
$A' \leftarrow$ augment action space $|intervals|$ times;
$Q(s, a') \leftarrow 0$;
for *episodes* $1...E$ **do**
    s $\leftarrow$ initial state;
    step $\leftarrow$ 0;
    augmented $\leftarrow$ ϵ-greedy(Q);
    action $\leftarrow$ augmented / $|intervals|$;
    d $\leftarrow$ $intervals$[augmented % $intervals$];
    while *episode not over* **do**
        $r, s' \leftarrow$ simulateHFO(action);
        step $\leftarrow$ step + 1;
        if *step % d == 0* **then**
            augmented $\leftarrow$ ϵ-greedy(Q);
            action $\leftarrow$ augmented / $|intervals|$;
            d $\leftarrow$ $intervals$[augmented % $intervals$];
            SARSA(λ) update of $Q(s, a')$;
            step $\leftarrow$ 0;
        end
        s $\leftarrow$ s';
    end
    SARSA(λ) update of $Q(s, a')$;
end

Algorithm 4: Augmented Action Space

input: λ
$Q_a(s,a) \leftarrow 0$;
$Q_i([s,a],a') \leftarrow 0$;
for *episodes* $1...E$ **do**
  s $\leftarrow$ initial state;
  action $\leftarrow \epsilon$-greedy(Q_a);
  d $\leftarrow \epsilon$-greedy(Q_i);
  step $\leftarrow 0$;
  while *episode not over* **do**
    $r, s' \leftarrow$ simulateHFO(action);
    step $\leftarrow$ step + 1;
    if *step % d == 0* **then**
      action $\leftarrow \epsilon$-greedy(Q_a);
      Augment 'action' with s';
      d $\leftarrow \epsilon$-greedy(Q_i);
      SARSA(λ) update of $Q_a(s,a)$;
      SARSA(λ) update of $Q_i([s,a],a')$;
      step $\leftarrow 0$;
    end
    s $\leftarrow s'$;
  end
  SARSA(λ) update of $Q_a(s,a)$;
  SARSA(λ) update of $Q_i([s,a],a')$;
end

Algorithm 5: Conditional FiGAR - SARSA(λ)

agent uses a SARSA(λ) update (λ tuned in our experiments over set $0, 0.5, 0.8, 0.9, 0.95$) with a learning rate of 0.1. An ϵ-greedy policy is followed while selecting the next action, with $\epsilon = 0.1$. To make a fair comparison among all macro-action variants, we constrain the interval set to $1, 2, 4, 8, 16, 32, 64$ for FiGAR, Augmented Action Space and Conditional FiGAR. DI-SARSA is tuned on each of these intervals.

All the runs are repeated thrice for 50,000 episodes with different random seeds; The average cumulative reward, cumulative regret and percent of goals and successful blocks for different algorithms are shown in **Table 2** for Reward Regularization, **Table 3** for FiGAR, **Table 4** for Augmented Action Space, **Table 5** for DI-SARSA and **Table 6** for Conditional FiGAR. Also shown in the table are the standard deviation across runs and number of runs averaged on[2].
Additionally, we report averaged percentage goals vs episodes in **Figure 1, 2** for DI SARSA; **Figure 3** for FiGAR; **Figure 4, 5** for Reward Regularization; **Figure 6** for Augmented Action Space and **Figure 7** for Conditional FiGAR.

Table 1 contrasts the performance of fine-tuned algorithms with one another. The corresponding plot is shown in **Figure 8**.

Results and Discussions

We confirm the hypothesis presented in (Siddharth 2017) in our DI-SARSA experiments. Quite clearly from **Figure 1**, a fixed decision interval $d > 1$ outperforms a vanilla SARSA(λ) ($d = 1$). We also note the best step size to be 32. ((Siddharth 2017) reported 31).

Agent trained with FiGAR was most successful in defending. Regularized SARSA and Augmented SARSA beat DI-SARSA with at least ten point improvement in number of successful defends. DI-SARSA beats baseline SARSA because of temporal coarse coding, this also means the agent is updated only once in several time steps. Hence, the rate at which the agent learns is reduced. This could be the reason why DI-SARSA was easy to beat.

The FiGAR variants: *Augmented Action Space* and *Conditional FiGAR* are expected to be at least as good as FiGAR. However in both the cases, the complexity of action-value function increases, which in turn will require more updates for a good function approximation. This might have caused the low performance values for these two variants.

Regularized SARSA is a simple modification on SARSA that we have proposed. The algorithm imposes penalty on choosing different actions for states that are close. By doing so, we introduce a margin between $Q(s,a^*)$ and $max_{a \neq a^*} Q(s,a)$, where a^* is optimal action at state s. This method did better than SARSA(λ) on this task. However, after the empirical analysis, we noticed a few shortcomings in our choice of penalty per action switch, p. Let b denote the number of action switches on a path. We expect that the effective reward to be greater than the penalty accrued when the goal is not achieved. That is

$$R_{max} - b * p > R_{min} \Rightarrow b < \frac{R_{max}+R_{min}}{p}$$

In the case of HFO, R_{max} and R_{min} are 1 and -1 respectively. For any value of $p > 1$, the optimal path cannot have any action switches. This is reflected by almost same performance for p values of 4, 6, 8 in **Figure 4**. While DI-SARSA specifies the number of time steps for which the action should be repeated, the regularized version (soft-)specifies the maximum number of action switches allowed.

In **Figure 4**, for the case with $p = 2$, the fraction of goals increased after decreasing initially. If the agent has learned for more than 20,000 episodes, it already has a good approximation of $Q(s,a)$, further constraining the decision with penalty will force the agent to take the sub-optimal actions. Hence, annealing the penalty with episodes will potentially improve performance. We leave finding a suitable annealing strategy for future work.

Conclusion

We present a study of five different algorithms which force or encourage an agent to take "macro-actions", or in other words repeat actions for a number of steps. We note that all our algorithms beat the vanilla SARSA(λ), with FiGAR SARSA performing the best. Future work would include a theoretical analysis of the benefits of action repetition, more exhaustive tuning on HFO and empirical analysis of these five algorithms on more complex frameworks.

References

[Hausknecht *et al.* 2016] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrish-

[2] A few experiments were run on lesser number of random seeds due to less computation time.

Algorithm	Cumm. Regret $\pm\sigma$	Cumm. Reward $\pm\sigma$	%captured	%goal
SARSA	32534.000 + 287.800 (3)	11060.000 + 208.251 (3)	0.254	0.746
DI-SARSA	28405.333 + 146.780 (3)	15188.667 + 128.175 (3)	0.348	0.652
Reg. SARSA	23986.000 + 4900.388 (3)	19608.000 + 4634.171 (3)	0.450	0.550
FiGAR SARSA	19536.667 + 6150.605 (3)	24057.333 + 6654.715 (3)	**0.552**	**0.448**
Augmented SARSA	22797.000 + 9850.000 (2)	20797.000 + 7150.275 (2)	0.477	0.523
Conditional SARSA	28785.000 + 3898.890 (3)	14809.000 + 2765.450 (3)	0.340	0.660

Table 1: Performance measures for each fine-tuned algorithm

Figure 1: Tuning of DI-SARSA with step size, $\lambda = 0.8$

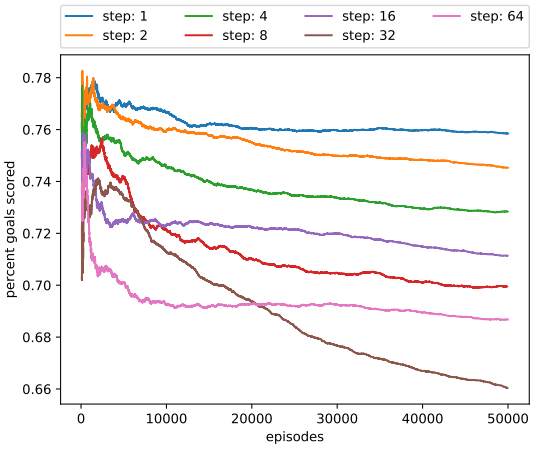
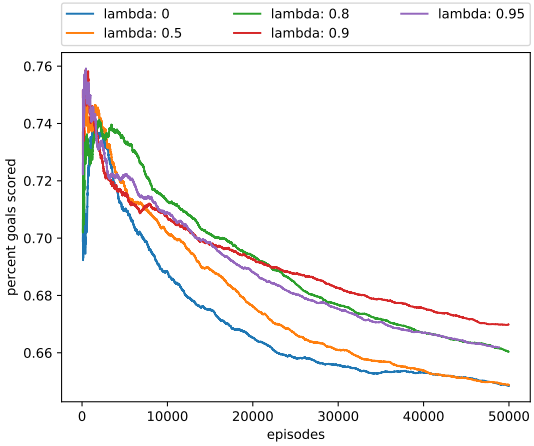


Figure 2: Tuning of DI-SARSA with λ, step = 32



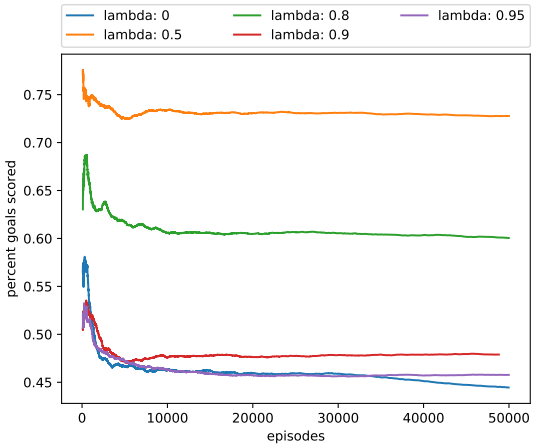
nan, and Peter Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, 2016.

[Lakshminarayanan *et al.* 2016] Aravind S Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic frame skip deep q network. *arXiv preprint arXiv:1605.05365*, 2016.

[Sharma *et al.* 2017] Sahil Sharma, Aravind S Lakshminarayanan, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

[Siddharth 2017] A Siddharth. On the effect of the frequency of decision making in temporal difference learning. 2017.

Figure 3: Tuning of FiGAR with λ



lambda	regReward	Num Episodes	Cumm. Regret $\pm\sigma$	Cumm. Reward $\pm\sigma$	%captured	%goal
0.50	0.0001	43594	33528.000 + 18.000 (2)	10066.000 + 68.370 (2)	0.231	0.769
0.50	0.001	43594	33038.333 + 276.116 (3)	10555.667 + 2000.392 (3)	0.242	0.758
0.50	0.01	43594	32923.333 + 281.750 (3)	10670.667 + 210.299 (3)	0.245	0.755
0.50	0.1	43594	33411.000 + 515.831 (3)	10183.000 + 403.952 (3)	0.234	0.766
0.50	0.5	43594	30359.000 + 2589.336 (3)	13235.000 + 2016.549 (3)	0.304	0.696
0.00	1.0	43594	28972.333 + 4580.272 (3)	14621.667 + 3328.254 (3)	0.335	0.665
0.50	1.0	43594	34404.000 + 315.000 (2)	9190.000 + 223.063 (2)	0.211	0.789
0.80	1.0	43594	31793.000 + 3994.260 (3)	11801.000 + 3011.231 (3)	0.271	0.729
0.90	1.0	43594	32598.667 + 1046.993 (3)	10995.333 + 2336.683 (3)	0.252	0.748
0.95	1.0	43594	27343.000 + 7769.632 (3)	16251.000 + 6953.874 (3)	0.373	0.627
0.00	2.0	43594	33454.500 + 440.500 (2)	10139.500 + 398.052 (2)	0.233	0.767
0.50	2.0	43594	31779.667 + 1683.164 (3)	11814.333 + 1236.690 (3)	0.271	0.729
0.80	2.0	43594	33414.667 + 1745.014 (3)	10179.333 + 1788.758 (3)	0.234	0.766
0.90	2.0	43594	24077.500 + 2223.500 (2)	19516.500 + 1792.520 (2)	0.448	0.552
0.95	2.0	43594	23986.000 + 4900.388 (3)	19608.000 + 4634.171 (3)	**0.450**	**0.550**
0.00	4.0	43594	32027.333 + 1624.043 (3)	11566.667 + 1201.101 (3)	0.265	0.735
0.50	4.0	43594	33710.333 + 1360.434 (3)	9883.667 + 963.355 (3)	0.227	0.773
0.80	4.0	43594	33193.000 + 1623.466 (3)	10401.000 + 1975.593 (3)	0.239	0.761
0.90	4.0	43594	33292.000 + 26.470 (3)	10302.000 + 73.115 (3)	0.236	0.764
0.95	4.0	43594	33281.000 + 1987.814 (3)	10313.000 + 1436.971 (3)	0.237	0.763
0.00	6.0	43594	34635.333 + 335.919 (3)	8958.667 + 244.755 (3)	0.206	0.794
0.50	6.0	43594	32864.667 + 2049.912 (3)	10729.333 + 1571.581 (3)	0.246	0.754
0.80	6.0	43594	29472.333 + 5675.704 (3)	14121.667 + 4767.923 (3)	0.324	0.676
0.90	6.0	43594	32927.333 + 2061.711 (3)	10666.667 + 1536.957 (3)	0.245	0.755
0.95	6.0	43594	33168.667 + 1948.053 (3)	10425.333 + 1442.663 (3)	0.239	0.761
0.00	8.0	43594	34397.000 + 340.213 (3)	9197.000 + 277.635 (3)	0.211	0.789
0.50	8.0	43594	32895.000 + 2117.346 (3)	10699.000 + 1556.574 (3)	0.245	0.755
0.80	8.0	43594	33543.333 + 947.879 (3)	10050.667 + 2028.201 (3)	0.231	0.769
0.90	8.0	43594	32608.667 + 1914.662 (3)	10985.333 + 1478.948 (3)	0.252	0.748
0.95	8.0	43594	34073.667 + 580.959 (3)	9520.333 + 537.674 (3)	0.218	0.782

Table 2: Regularized version of SARSA tuned on different values of regPenalty and λ

Figure 4: Tuning of Reward Penalty with penalty, $\lambda = 0.95$

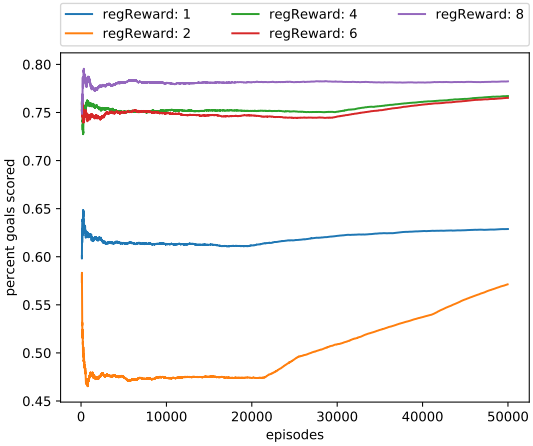
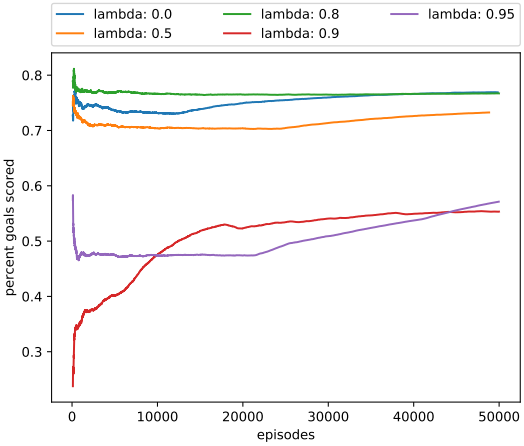


Figure 5: Tuning of Reward Penalty with λ, penalty = 2



lambda	Num Episodes	Cumm. Regret $\pm\sigma$	Cumm. Reward $\pm\sigma$	%captured	%goal
0.00	43594	19536.667 + 6150.605 (3)	24057.333 + 6654.715 (3)	**0.552**	**0.448**
0.50	43594	31761.000 + 77.240 (3)	11833.000 + 61.319 (3)	0.271	0.729
0.80	43594	26282.333 + 3396.982 (3)	17311.667 + 2423.101 (3)	0.397	0.603
0.90	43594	20903.000 + 10734.462 (3)	22691.000 + 7659.015 (3)	0.521	0.479
0.95	43594	19937.333 + 10876.495 (3)	23656.667 + 7714.036 (3)	0.543	0.457

Table 3: FiGAR on SARSA

lambda	Num Episodes	Cumm. Regret $\pm\sigma$	Cumm. Reward $\pm\sigma$	%captured	%goal
0.00	43594	32006.500 + 295.500 (2)	11587.500 + 211.306 (2)	0.266	0.734
0.50	43594	33919.667 + 388.304 (3)	9674.333 + 305.458 (3)	0.222	0.778
0.80	43594	32924.000 + 332.455 (3)	10670.000 + 270.202 (3)	0.245	0.755
0.90	43594	32117.000 + 1207.273 (3)	11477.000 + 887.541 (3)	0.263	0.737
0.95	43594	22797.000 + 9850.000 (2)	20797.000 + 7150.275 (2)	**0.477**	**0.523**

Table 4: Augmented Action Space

Figure 6: Tuning of Augmented Action SARSA with λ

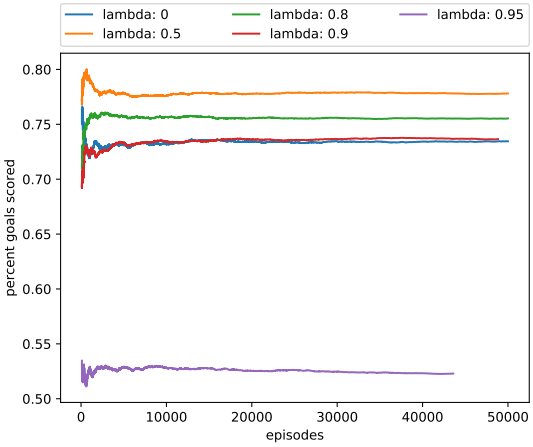


Figure 7: Tuning of Conditional FiGAR with λ

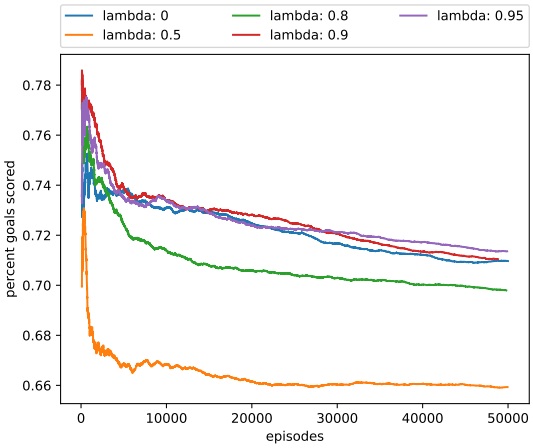
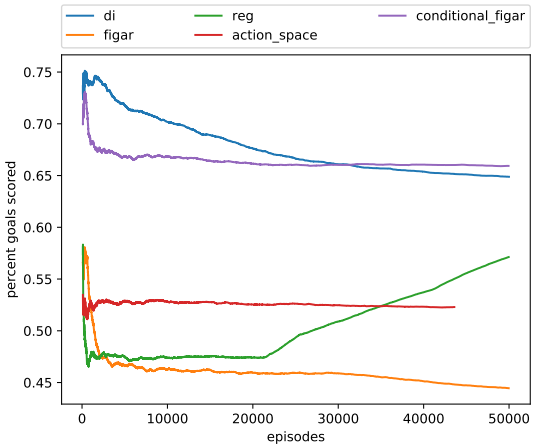


Figure 8: Fine-tuned comparison of all algorithms. Here `di` is DI-SARSA, `reg` is Regularized Reward, `action_space` is Augmented Action SARSA.



lambda	step	Num Episodes	Cumm. Regret $\pm\sigma$	Cumm. Reward $\pm\sigma$	%captured	%goal
0.00	1	43594	33128.333 + 386.559 (3)	10465.667 + 279.795 (3)	0.240	0.760
0.50	1	43594	32534.000 + 287.800 (3)	11060.000 + 208.251 (3)	**0.254**	**0.746**
0.80	1	43594	33102.000 + 371.280 (3)	10492.000 + 298.777 (3)	0.241	0.759
0.90	1	43594	33454.667 + 74.968 (3)	10139.333 + 76.956 (3)	0.233	0.767
0.95	1	43594	32916.333 + 227.639 (3)	10677.667 + 189.044 (3)	0.245	0.755
0.00	2	43594	32126.000 + 80.254 (3)	11468.000 + 85.259 (3)	0.263	0.737
0.50	2	43594	31326.667 + 111.828 (3)	12267.333 + 157.210 (3)	0.281	0.719
0.80	2	43594	32578.333 + 535.687 (3)	11015.667 + 465.552 (3)	0.253	0.747
0.90	2	43594	32294.000 + 439.560 (3)	11300.000 + 327.317 (3)	0.259	0.741
0.95	2	43594	32391.333 + 865.311 (3)	11202.667 + 628.730 (3)	0.257	0.743
0.00	4	43594	31619.333 + 757.157 (3)	11974.667 + 539.838 (3)	0.275	0.725
0.50	4	43594	31797.667 + 287.542 (3)	11796.333 + 226.129 (3)	0.271	0.729
0.80	4	43594	31789.667 + 434.316 (3)	11804.333 + 312.550 (3)	0.271	0.729
0.90	4	43594	31755.000 + 715.402 (3)	11839.000 + 517.151 (3)	0.272	0.728
0.95	4	43594	32067.000 + 711.743 (3)	11527.000 + 527.040 (3)	0.264	0.736
0.00	8	43594	31131.333 + 228.504 (3)	12462.667 + 179.353 (3)	0.286	0.714
0.50	8	43594	30819.333 + 472.752 (3)	12774.667 + 501.339 (3)	0.293	0.707
0.80	8	43594	30506.500 + 1347.500 (2)	13087.500 + 1320.286 (2)	0.300	0.700
0.90	8	43594	32093.000 + 89.993 (3)	11501.000 + 168.005 (3)	0.264	0.736
0.95	8	43594	31409.000 + 947.040 (3)	12185.000 + 819.300 (3)	0.280	0.720
0.00	16	43594	30635.000 + 466.275 (3)	12959.000 + 545.543 (3)	0.297	0.703
0.50	16	43594	31087.333 + 117.219 (3)	12506.667 + 174.485 (3)	0.287	0.713
0.80	16	43594	31098.000 + 228.370 (3)	12496.000 + 408.302 (3)	0.287	0.713
0.90	16	43594	30807.333 + 442.968 (3)	12786.667 + 407.942 (3)	0.293	0.707
0.95	16	43594	30754.333 + 291.320 (3)	12839.667 + 364.696 (3)	0.295	0.705
0.00	32	43594	28408.667 + 930.377 (3)	15185.333 + 1011.843 (3)	0.348	0.652
0.50	32	43594	28405.333 + 146.780 (3)	15188.667 + 128.175 (3)	**0.348**	**0.652**
0.80	32	43594	28975.000 + 529.777 (3)	14619.000 + 390.948 (3)	0.335	0.665
0.90	32	43594	29338.333 + 659.077 (3)	14255.667 + 492.916 (3)	0.327	0.673
0.95	32	43594	28983.667 + 451.658 (3)	14610.333 + 324.401 (3)	0.335	0.665
0.00	64	43594	29685.667 + 47.338 (3)	13908.333 + 97.589 (3)	0.319	0.681
0.50	64	43594	29726.667 + 142.783 (3)	13867.333 + 130.322 (3)	0.318	0.682
0.80	64	43594	30000.333 + 197.444 (3)	13593.667 + 203.316 (3)	0.312	0.688
0.90	64	43594	29945.667 + 314.194 (3)	13648.333 + 237.254 (3)	0.313	0.687
0.95	64	43594	29699.000 + 199.059 (3)	13895.000 + 212.555 (3)	0.319	0.681

Table 5: Decision Interval SARSA for different decision intervals. `step` = 1 corresponds to a vanilla SARSA(λ).

lambda	Num Episodes	Cumm. Regret $\pm\sigma$	Cumm. Reward $\pm\sigma$	%captured	%goal
0.00	43594	30933.667 + 213.920 (3)	12660.333 + 207.020 (3)	0.290	0.710
0.50	43594	28785.000 + 3898.890 (3)	14809.000 + 2765.450 (3)	**0.340**	**0.660**
0.80	43594	30517.333 + 458.005 (3)	13076.667 + 325.632 (3)	0.300	0.700
0.90	43594	31048.333 + 759.978 (3)	12545.667 + 602.929 (3)	0.288	0.712
0.95	43594	31200.333 + 124.596 (3)	12393.667 + 315.075 (3)	0.284	0.716

Table 6: Repetition prediction conditional on the action