Reusing Learned Policies Between Similar Problems

Mike Bowling Manuela Veloso
mhb@cs.cmu.edu veloso@cs.cmu.edu

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

We are interested in being able to leverage policy learning in complex problems upon policies learned for similar problems. This capability is particularly important in robot learning, where gathering data is expensive and time-consuming, and prohibits directly applying reinforcement learning. In this case, we would like to be able to transfer knowledge from a simulator, which may have an inaccurate or crude model of the robot and environment. We observed that when applying a policy learned in a simulator, some parts of the policy effectively apply to the real robots while other parts do not. We then explored learning a complex problem by reusing only parts of the solutions of similar problems. Empirical experiments of learning when part of the policy is fixed show that the complete task is learned faster, but the resulting policy is suboptimal. One of the main contributions of this paper is a theorem and its proof, which states the degree of suboptimality of a policy that is fixed over a subproblem, can be determined without the need for optimally solving the complete problem. We formally define a subproblem and build upon the value equivalence of the boundary states of the subproblem to prove the bound on suboptimality.

Keywords: reinforcement learning, subproblem reuse, bounding policy optimality

Contact: Mike Bowling, mhb@cs.cmu.edu, (412) 268-3069.

1 Introduction

Learning to act in complex environments, such as multi-robot ones, is a well-known difficult task. It seems however intuitive to think that the complexity of an environment could be handled by training and learning in simpler similar problems. Our motivation for this work is therefore twofold. We are interested in investigating the reuse of solutions to similar (possibly simpler) problems. And, we would like to apply our techniques to learning in a multi-robot setting.

Gathering real robot data is extremely time consuming and therefore prohibits using basic reinforcement learning techniques. Other learning approaches successfully solve this problem by refining a model of the robot. In our case, building a model of each robot is not feasible since each may have different hardware characteristics. In addition, model-based techniques still require complete exploration of the world and the robot's actions, and could still benefit from knowledge gained in similar problems.

We therefore followed a different approach. Instead of learning a model, we built a crude simulator of the robot's actions in the environment. The purpose was to learn optimal policies in the simulator, and use that to leverage learning in the real robots. Directly applying the policy learned from the simulator in the real robots, does not succeed. However we note (qualitatively) that some *parts* of the learned policies successfully apply to the real robots while others do not.

In this paper, we discuss this underlying real case study that motivates our formal investigation of reusing policies learned in similar problems. To effectively study the underlying issues, we explored this in simulated environments.

We reproduced in the simulator the use of subparts of a policy in similar problems. We realize that the learning of the new policy is faster and by comparing with learning from scratch, we can quantitatively determine how suboptimal is the policy learned when using a fixed policy for a subproblem.

It would be very interesting if we could determine quantitatively how suboptimal a policy built from a similar policy is without having to determine the optimal policy. We contribute in this paper an answer to this question. We introduce a theorem and its proof, which states that the degree of sub-optimality of a policy that is fixed over a subproblem, can be determined without the need for optimally solving the complete problem. We formally define a subproblem and build upon the value equivalence of the boundary states of the subproblem to prove the bound on policy suboptimality.

The paper is organized as follows. Section 2 presents the case study with the results of reusing policies from a crude simulator in a real robot. Section 3

introduces similarity between problems and shows the results of using similar policies in controlled experiments in a simulator. Section 4 introduces the theorem on the bound of the suboptimality of a similar policy. We present the complete detailed proof in the appendix. We analyze the value of the bounds in particular situations, and verify the bound for our results from Section 3. Section 5 discusses some related work, and Section 6 concludes.

2 A Motivating Example

Our work is in the context of robotic soccer [5]. The robots use several hardwired behaviors to select actions to achieve individual and team goals. However there are situations for which finding the appropriate analytical behavior is not trivial, and we would like to apply learning to these cases. Figure 1 illustrates an example of one of these problems. The robot is in a rectangular playing area, and the ball is placed at a fixed distance from the wall. The goal of the robot is to strike the ball in the desired direction. The robot is roughly square shaped with two independent motors, allowing it to move forwards and backwards as well as turning in place. Positive reinforcement is received for hitting the ball in the correct direction. Negative reinforcement is received for hitting the ball in an incorrect direction, leaving the playing area, or getting stuck along the wall. The goal is to learn a policy that would successfully hit the ball in the desired direction.

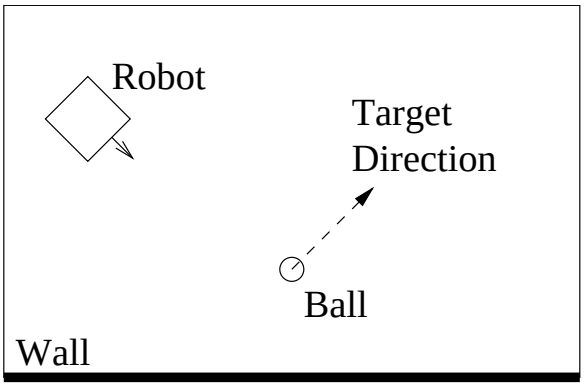

Figure 1: A simple problem in the robot soccer domain.

For learning, the state is encoded as a tuple (x, y, θ), which is the Cartesian position relative to the ball and the direction the robot is facing. This continuous

state is uniformly discretized into 2080 states[1]. The actions are discretized to four: turn left (in place), turn right, move forward, and move backward.

Training in this setting, as with most real robot domains, is extremely slow. In our experiments, the robot performs less than two actions every second. Just to perform every action in every state would require over an hour. In addition, training requires human intervention between trials. Despite this discouraging evidence, training was still attempted. After eight hours[2] of training, the robot had completed 550 trials, visited 25,000 states, and only managed 18 successes. The resulting Q table was still very sparse, and performance was expectedly very poor.

Gathering enough real robot data was prohibitive and so it became difficult to directly apply reinforcement learning to real data. Other learning approaches successfully solve this problem by refining a model of the robot. In our case, building a model of each robot is not feasible as we have multiple robots with different hardware characteristics. We followed a different approach. We built a crude simulator of a general robot's actions in the environment. The simulator is an idealistic representation of the world, and uses the same state and action space used in the real robots. It is deterministic with all actions just transitioning to the geometrically expected state. It does not model the effects of the wall on the robot's motion. It also has an extremely simple model of the robot's shape and how it can hit the ball.

Training in this computer simulated model is very fast. After approximately 1500 trials, it had visited 100,000 states, and taken an elapsed time of less than five seconds. The resulting policy is nearly optimal for the model. Of course, the measure of success is performance in the real robots.

This learned policy was then used on the real robots. The quantitative results were very poor, as it never successfully hit the ball in the desired direction. Though qualitatively a couple of things were noticed:

- The robot acts "well" when far from the ball (i.e. it positions itself appropriately to hit the ball.)
- The robot acts "poorly" when near the ball (i.e. it doesn't hit the ball in the correct direction.)
- The robot easily gets caught on the wall.

[1] The rectangle is discretized into 20x13, and the robot's direction is discretized into 8.

[2] This time does not include down time of charging batteries, fixing hardware problems, etc.

These observations are consistent with the simulator's deficiencies, described above.

3 Using a Policy From a Similar Problem

Towards the goal of transferring knowledge from our crude simulator to the real robots, we investigated the more general problem of transferring knowledge between "similar" problems. We will make no attempt to define similarity between problems, and instead appeal to a basic intuition. In the domains we explored similar was restricted to problems with the same transition probabilities, but with different rewards. Though, we expect results to be similar when transition probabilities are different, e.g. when transferring policies from a simulator to real robots.

Two simulated domains were investigated. The first is the simulator used for the robot soccer problem that was just described. In this domain, our similar problem was to alter the desired direction for the robot to strike the ball. The second domain is a discrete grid-navigation problem, presented in [4], and depicted in figure 2. The agent can move in any of the eight grid directions. But, with a probability of 0.1, its action is ignored and it is moved to a random neighbor. The agent receives a reward of 1 for transitions entering the goal state, and a reward of zero for all others. We used a discount factor, γ, of 0.1. In this domain we used different corners for the start and goal states for our similar problems.

All the results shown in this paper are for the grid-navigation problem, since the subproblems in this domain are easy to visualize and the results for the robot soccer problem are identical.

3.1 Refining a Similar Policy

One naive approach to transferring knowledge between two problems is to use the final Q values from a similar problem as the starting Q values for the new problem. Q-Learning is then used normally from these starting values. This approach was tried in both of the simulated domains. The results for the grid-navigation problem are shown in figure 3. Initializing the Q values seems to be a considerable hindrance to learning, and is easily outperformed by simply learning from scratch.

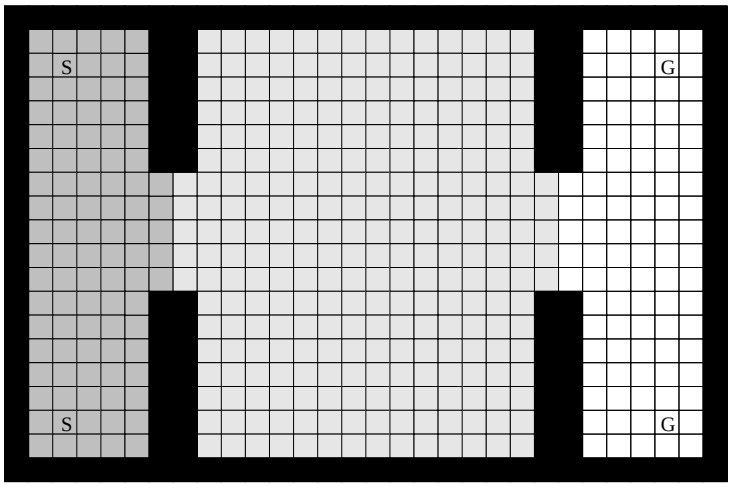

Figure 2: A simple 3 room environment. There are two problems whose start states are labeled 'S' and whose goal state is the state labeled 'G' in the opposite corner. The shading marks two different subproblems.

3.2 Reusing Part of a Similar Policy

Another approach is motivated by the qualitative observations given in section 2. When using the policy, which was learned in the simulator, in the real robots we observed that the policy worked well for some parts of the state but worked poorly in other parts. This lends itself to reusing only parts of a policy that was learned in a similar problem, and holding these parts of the policy fixed during learning. In the next section we'll show a relationship between these subparts and the notion of a subproblem.

This approach was also tried in both simulated domains. For the grid-navigation problem, we tried fixing two different sets of states, which are marked in figure 2 by the shaded regions. The dark region corresponds to fixing the "left room", and the light and dark regions correspond to fixing the "left and middle rooms". The results are given in figure 4.

These results show that fixing a large part of a similar policy is an improvement over learning from scratch when there is very little training. For example, if training were limited to visiting 5000 states, fixing the left and middle rooms is a 600% improvement over learning from scratch. Though, it is obvious from the graph that this improved initial learning is at the cost of converging to the optimal solution. In cases where training data is difficult or expensive to gather, as it is in

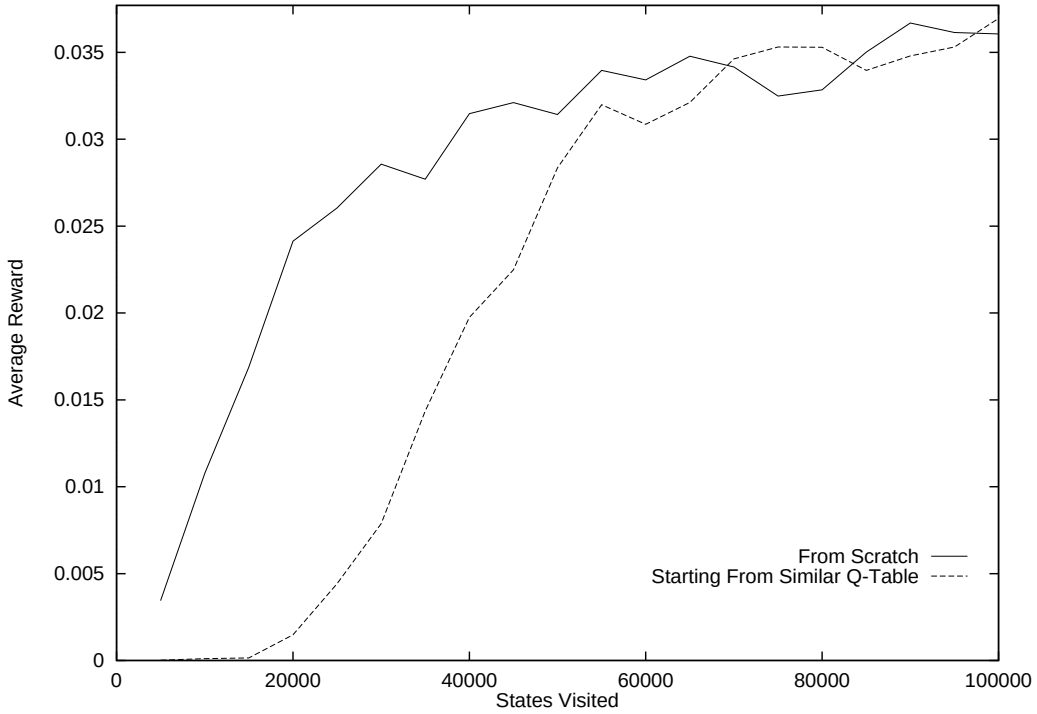

Figure 3: Results from the grid-navigation problem comparing learning from scratch to learning with Q values initialized from a similar problem.

robot learning, it may be beneficial to exploit this trade-off.

The more conservative approach of fixing just a small portion of the similar policy results in a more conservative learning improvement of 300% over learning from scratch. Though it does not appear to sacrifice convergence to optimality. The intuition is that these two problems share the same subproblem of exiting the left room, and therefore the optimal policy over these states is the same for both problems.

4 Bounding the Policy Suboptimality

The above results show that optimality can be sacrificed for faster learning, but it would be good to be able to calculate the loss in optimality, even though the optimal solution is not known. To do this, we will define a notion of a subproblem in reinforcement learning, and then show that the loss in optimality by using that

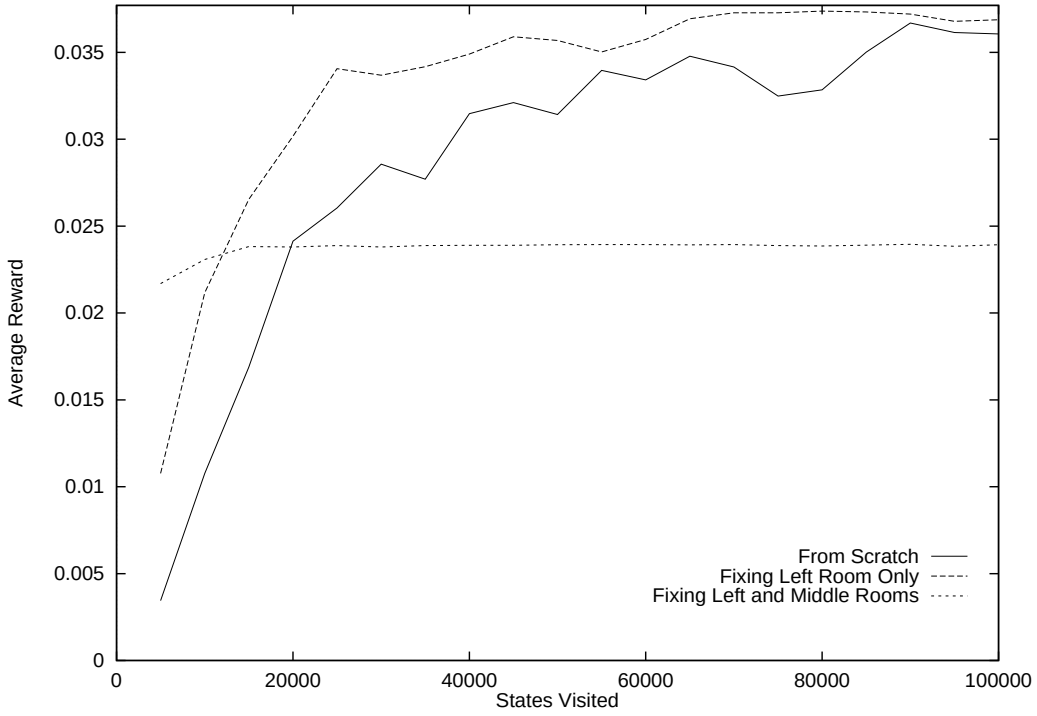

Figure 4: Results from the grid-navigation problem comparing learning from scratch to learning while fixing a portion of a similar policy.

subproblem can be calculated without finding the optimal solution.

4.1 Determining the Bound

The following definitions and theorems assume that $\mathcal{P}$ is a traditional reinforcement learning problem, with a few exceptions. $\mathcal{P}$ must have a set of *goal states* that are absorbing (i.e. there are no transitions from the state.) In addition, reward is only received for transitions into these goal states, all other transitions receive zero reward.

First, we define a measure of the suboptimality of a policy.

Definition 1 *A policy, π, is* α-optimal *if and only if* $\forall s \; V^{\pi}(s) \geq \alpha V^{*}(s)$.

The following two definitions play a crucial role in the bound on suboptimality. The first provides a measure of the equivalence of a set of states. The second provides a measure of the stochasticity of a problem.

Definition 2 *A set of states, S, is* α-equivalent *with respect to π if and only if $\forall s_1, s_2 \in S\ V^\pi(s_1) \geq \alpha V^\pi(s_2)$.*

Definition 3 *Let ρ_s be the probability that following the optimal policy at s will transition to a state, s', where $V^*(s') < V^*(s)$. Let $\rho = \max_s \rho_s$. This captures the stochasticity of the problem, where $\rho = 0$ for a deterministic problem.*

Before we can define the notion of a subproblem, we will need to formally define a boundary of a set of states.

Definition 4 *For a set of states, S, let the* boundary *of S, denoted $T(S)$, be the set of all states not in S with a non-zero transition from a state in S.*

For example, in the 3 room grid-navigation problem, the boundary of the set of states that make up the left room are the five adjacent states in the doorway.

We can now define our notion of a subproblem.

Definition 5 *A pair (S, G) is a* subproblem *of a problem $\mathcal{P}$ if and only if:*

- $S \cap G = \emptyset$.
- $G \subseteq T(S)$.
- *S and $T(S)$ do not contain goal states in $\mathcal{P}$.*

This definition induces a new problem over the states S. The state transitions remain the same as in $\mathcal{P}$, but the reward function is zero for all transitions except those entering states in G, where the reward is one.

There is a relationship between this notion of a subproblem and fixing a portion of a similar policy. By fixing part of the policy we are assuming that the two problems contain a common subproblem over the states that are fixed. So, our bound on the suboptimality of using a subproblem also applies to fixing part of a similar policy.

This definition provides a general notion of a subproblem, but our bound is going to be restricted to subproblems where $G = T(S)$. We can now state our main theorem.

Theorem 1 *Let π_S be an optimal solution to the subproblem $(S, T(S))$ of $\mathcal{P}$. Let π be the optimal policy for $\mathcal{P}$ that uses π_S for the subproblem, i.e. $\forall s \in S\ \pi(s) = \pi_S(s)$. If $T(S)$ is α-equivalent with respect to π, then,*

$$\pi \text{ is } \alpha\beta\text{-optimal for } \mathcal{P},$$

where $\beta = \frac{1-\varrho}{1-\varrho\alpha}$ and $\varrho = \frac{\rho\gamma^2}{1-(1-\rho)\gamma}$.

The ϱ value seems cryptic, but is constructed in the proof and represents the probability of ever reaching a state of lower value, discounted for the distance the state is away. It's derived from the sum,

$$\varrho = \gamma(\rho\gamma + \rho(1-\rho)\gamma^2 + \rho(1-\rho)^2\gamma^3 \ldots),$$

where ρ is the value defined in definition 3 and γ is the reward discount factor.

The complete proof for this theorem is given in the appendix, but some intuition for it is presented here. Consider the boundary state, $t_m \in T(S)$ that maximizes V^*. We can examine all possible paths from t_m, and each must either contain a state in S or not. π is optimal for states not in S, so π achieves the same reward as π^* if it doesn't contain a state in S. If it does, then it also contains a state in $T(S)$. The best case is that this state is t_m, the worst case is it's some other t, which by α-equivalence is within α of t_m. The ratio of these can be shown to be larger than β, which means π is β-optimal for t_m.

We can apply the same reasoning to other states. By examining all possible paths, they either contain a state in S or not. If not, then π does as well as π^*. If so, then they must contain a state in $T(S)$, which is α-equivalent from t_m, which is β-optimal. So all other states are $\alpha\beta$-optimal.

The important thing to notice is that the bound provided by the theorem does not depend on any knowledge of V^* or π^*. The next section below will show how to use this theorem to calculate a bound on the suboptimality of using a subproblem.

Further Extensions. Below are some very interesting corollaries and extensions to our theorem.

Corollary 1 *If $\mathcal{P}$ is deterministic then π is α-optimal for $\mathcal{P}$.*

Corollary 2 *If $T(S)$ is 1.0-equivalent then π is optimal for $\mathcal{P}$.*

These are obtained directly from substituting the appropriate values into the theorem.

Theorem 2 *If the policy π_S is only σ-optimal for the subproblem then the resulting π is $\alpha\beta\sigma$-optimal for $\mathcal{P}$.*

The proof for this follows exactly as for theorem 1, with the additional σ factor. This result allows us to bound the suboptimality of using a policy for subproblem that itself is composed of subproblems.

4.2 Using the Bound

For the three room grid-navigation problem we can use our theoretical results to calculate a lower bound on their suboptimality. We can then verify this bound by our empirical results from section 3.2.

Fixing the Left and Middle Rooms. We can simply follow the computations from theorem 1. From the definition of the problem we know, $\rho = 0.1 * \frac{6}{8} = 0.075$, since when we take a random action only six of the eight moves take us to a state of lower value. After learning by fixing the policy, we examine the values on the five states in $T(S)$ (i.e. the states connecting the middle and right rooms), to get α. Below is the rest of the calculation; in this example the bound provided by the theorem is fairly tight.

$$\rho = 0.075 \quad \varrho = 0.36$$
$$\alpha = 0.71 \quad \beta = 0.86$$

The learned policy must be at least 0.61-optimal.

It is actually 0.65-optimal (see figure 4.)

Fixing the Left Room Only. We can also use corollary 2 to say something about the suboptimality of fixing only the left room. The values on the states in $T(S)$ when fixing this smaller portion of the policy, converge to be 1.0-equivalent. By the corollary, learning with this portion of the policy fixed should still lead to an optimal solution. This fact matches our empirical results from figure 4 and our intuition.

5 Discussion

Finding a similar problem, and deciding what to fix. Our formal analysis and results assume that the similar policies are given, or were acquired in previous learning episodes. However an important question is how to determine what is a similar problem and which parts of its policy to fix.

These two problems have been present in other subareas where reuse of learned knowledge is considered. Of particular related interest to us is the area of plan reuse, as we can view a plan as a policy. For example, when using past plans to solve new problems, the algorithm can be given past similar plans or can retrieve

them according to some measure of similarity. These two cases create different research questions; one involves how to change the plan, and the other involves defining plan similarity. Combining both plan retrieval and plan adaptation is shown to produce interesting results (e.g., [6]).

In our case, it may be possible to learn these subproblems, which is the goal of the SKILLS algorithm [4]. We would have to slightly constrain the state construction of SKILLS to comply to our definition of subproblem. In particular, by our definition of subproblem, states need to be "spatially" adjacent so that boundaries can be identified. This extension is part of our current research agenda.

An alternative approach may be to try multiple subproblems. Since by fixing parts of a subproblem, learning proceeds much faster, it may still be tractable to consider trying a set of subproblems. We can learn fixing each one individually and compute the α value of their boundaries, and using our theorem select the one that is the least suboptimal.

Applying our theoretical results. Finally, the suboptimality bound as given by our theorem could be combined with methods that find structure in reinforcement learning problems. The bound can provide a measure to evaluate the effectiveness of the constructed abstractions or the loss in optimality of using the abstractions [1, 2, 3].

6 Conclusion

In this paper, we introduced and proved a tight bound for the suboptimality of a learned policy that is fixed over a subproblem. By determining the α-value equivalence of the boundary states of the subproblem, we proved that the policy is within a factor $\alpha\beta$ of the optimal policy, where β is a function of how stochastic the problem is and of the reward decay factor. The important fact to note is that the sub-optimality bound does not depend on knowing the ultimate optimal policy.

This formal result has a deep motivation in our work in applying reinforcement learning to real robot learning, especially with multiple, possibly different, robots. In the paper we showed motivating results from our approach to real robot learning. We are currently applying our approach to reuse the policy learned in the simulator in the real robots.

A Proof of Theorem 1

Proof. Let $t_m = \text{argmax}_{t \in T(S)} V^*(t)$. We will first show that π is β-optimal for t_m. From this we will be able to show that for all other states it is $\alpha\beta$-optimal.

We can write $V^*(t_m)$ as,

$$V^*(t_m) = \sum_{\text{all paths}} \Pr(path)\text{Value}(path)$$

We can then split the sum into the paths containing a state in S and those that don't contain a state in S. Let p be the probability a path from t_m, following the optimal policy, contains a state in s. Then,

$$\begin{array}{rll} V^*(t_m) = & (1-p) & E(\text{Value}(path)|S \cap path = \emptyset) + \\ & p & E(\text{Value}(path)|S \cap path \neq \emptyset) \end{array}$$

Let $\nu = (1-p)E(\text{Value}(path)|S \cap path = \emptyset)$. Notice that any path that contains a state in S must also contain a state in $T(S)$. So we can write the second term as a value of that state discounted by the length of the path before the state is reached. Let $\ell(path, t)$ be the number of steps in $path$ until t is reached. So,

$$V^*(t_m) = \nu + \sum_{\substack{paths s^{\pi^*} \text{with} \\ t \in T(S)}} \Pr(path)\gamma^{\ell(path,t)}V^*(t)$$

Since $V^*(t_m) \geq V^*(t)$, we can substitute $V^*(t_m)$ for $V^*(t)$ and pull it outside of the sum. The resulting sum is just the expected discount between t_m and t when following the optimal policy. Let this value be δ. So we get,

$$V^*(t_m) \leq \nu + p\delta V^*(t_m)$$

This recurrence can be written as,

$$\begin{array}{rcl} V^*(t_m) & \leq & \nu \left(\sum_{i=0}^{\infty} (p\delta)^i \right) \\ & \leq & \nu \left(\frac{1}{1-p\delta} \right) \end{array}$$

We can also achieve a similar result for $V^{\pi}(t_m)$. Since π acts optimally for $s \notin S$, then it can achieve the same value for paths not containing $s \in S$. As before, paths containing a state in S, must contain a state in $T(S)$. So we can write,

$$V^{\pi}(t_m) = \nu + \sum_{\substack{paths^{\pi} \text{with} \\ t \in T(S)}} \Pr(path)\gamma^{\ell(path,t)}V^{\pi}(t)$$

By the α-equivalence of $T(S)$ we know that $V^{\pi}(t) \geq \alpha V^{\pi}(t_m)$. So, as before, we can substitute $\alpha V^{\pi}(t_m)$ for $V^{\pi}(t)$ and pull it outside of the sum. The resulting sum is the expected discount between t_m and t when following π. Let this value be δ'. So we get,

$$V^{\pi}(t_m) \geq \nu + p\delta'\alpha V^{\pi}(t_m)$$

Since π is optimal over the subproblem, it will reach a state in $T(S)$ with less expected discount than π^*. Hence, $\delta' \geq \delta$, and we can substitute δ for δ' and maintain the inequality. As above, we can rewrite the recurrence as,

$$\begin{aligned} V^{\pi}(t_m) &\leq \nu\left(\sum_{i=0}^{\infty}(p\delta\alpha)^i\right) \\ &\leq \nu\left(\frac{1}{1-p\delta\alpha}\right) \end{aligned}$$

Now we examine the ratio of the two values,

$$\begin{aligned} \frac{V^{\pi}(t_m)}{V^{*}(t_m)} &\geq \frac{\nu\left(\frac{1}{1-p\delta\alpha}\right)}{\nu\left(\frac{1}{1-p\delta}\right)} \\ &\geq \frac{1/(1-p\delta\alpha)}{1/(1-p\delta)} \\ &\geq \frac{1-p\delta}{1-p\delta\alpha} \end{aligned}$$

The value, $p\delta$, is the probability of returning to a state in S, discounted by the distance to that state and the distance from that state to a state in $T(S)$. Since all states in S have a smaller value than t_m, then we can bound this by the discounted

probability of ever reaching a state with smaller value. But this can be computed from ρ and is the following infinite sum,

$$\gamma(\rho\gamma + \rho(1-\rho)\gamma^2 + \rho(1-\rho)^2\gamma^3 \ldots) = \frac{\rho\gamma^2}{1-(1-\rho)\gamma} = \varrho$$

So, $p\delta \leq \varrho$, hence,

$$\frac{V^\pi(t_m)}{V^*(t_m)} \geq \frac{1-\varrho}{1-\varrho\alpha} \geq \beta$$

We can now show that π is $\alpha\beta$-optimal for any state, s. We can decompose the value of s just as was done for t_m. Let p be the probability that a path from s, following π^*, contains a state in S. Notice that if $s \in S$, then $p = 1$. We let ν and δ represent the same values with respect to s. Then we can bound $V^*(s)$, $V^\pi(s)$, and their ratio as follows:

$$\begin{aligned}
V^*(s) &\leq \nu + p\delta V^*(t_m) \\
V^\pi(s) &\geq \nu + p\delta\alpha V^\pi(t_m) \\
\frac{V^\pi(s)}{V^*(s)} &\geq \frac{\nu + p\delta\alpha V^\pi(t_m)}{\nu + p\delta V^*(t_m)} \\
&\geq \frac{p\delta\alpha V^\pi(t_m)}{p\delta V^*(t_m)} \\
&\geq \alpha\frac{V^\pi(t_m)}{V^*(t_m)} \\
&\geq \alpha\beta
\end{aligned}$$

So, $\forall s V^\pi(s) \geq \alpha\beta V^*(s)$, therefore π is $\alpha\beta$-optimal for $\mathcal{P}$. $\diamond$

References

[1] Craig Boutilier. Macros and subproblems in reinforcement learning. In *personal communication*, 1998.

[2] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[3] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Planning with closed-loop macro actions. In *Working notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems*, 1997.

[4] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press, 1995.

[5] Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim. CMUnited: A team of robotic soccer agents collaborating in an adversarial environment. In Hiroaki Kitano, editor, *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Springer Verlag, Berlin, 1998. A shorter version appeared in the First International Workshop on RoboCup, IJCAI-97, August, 1997.

[6] Manuela M. Veloso. Flexible strategy learning: Analogical replay of problem solving episodes. In *Proceedings of AAAI-94, the Twelfth National Conference on Artificial Intelligence*, pages 595–600, Seattle, WA, August 1994. AAAI Press.