Distributed Reinforcement Learning for a Traffic Engineering Application

Mark D. Pendrith*
DaimlerChrysler Research & Technology Center
1510 Page Mill Rd
Palo Alto, CA 94304, USA

E-mail: pendrith@ieee.org

*Author's current address/affiliation: SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, USA.

ABSTRACT

In this paper, we report on novel reinforcement learning techniques applied to a real-world application. The problem domain, a traffic engineering application, is formulated as a distributed reinforcement learning problem, where the returns of many agents are simultaneously updating a single shared policy. Learning occurs off-line in a traffic simulator, which allows us to retrieve and exploit good transient policies even in the presence of instabilities in the learning. We introduce two new algorithms developed for this situation, one which is a value function based, and one that employs a direct policy evaluation approach. While the latter is theoretically better motivated in several ways than the former, we find both perform comparably well in this domain and for the formulation we use.

1 INTRODUCTION

In reinforcement learning (RL), the problem is most commonly formulated to involve finding a good policy for a single agent situated in some environment. Although several workers have investigated multi-agent reinforcement learning for a small number of agents (typically less than 10), less work has been done in multi-agent RL involving much larger numbers of agents. In this paper, we investigate a formulation of the RL problem that involves integrating the experiences of a large number of agents sharing a partially observable environment in order to learn a common observation-based policy that seeks to optimize a group metric of performance rather than the rewards for any particular individual agent.

Agents 2000 Barcelona Spain

In this formulation, each agent shares a common policy, and so we have a homogeneous population of agents where specialization will not occur, as may be the case when each agent learns an individual policy (e.g., Mataric 1996). The "state" of the world, from each agent's point of view, may involve some other nearby agents in the immediate vicinity, but not others that are beyond sensing range. Since sensing the true state of the world would involve knowing about the situation of every agent in it, the environment is only partially observable to each agent in it.

At each time-step, there is a single global reinforcement signal. (In the problem domain we consider, this is actually an aggregate of the immediate reward of each individual agent, but it need not be so.) Considered as an optimization problem, we are searching over the space of possible shared policies in order to find the policy that results in group behaviour that maximizes a particular global objective function.

2 THE PROBLEM DOMAIN

The motivation for the distributed RL formulation described above may be clearer once we describe the particular problem domain that has led to this work. The problem domain is a traffic engineering problem, in which a population of cars, each with its own desired travelling speed, share a freeway network. We suppose a subpopulation of these cars are equipped with a short-range ($\sim$ 100m) radar capability to detect the relative speeds and distances of cars immediately ahead, behind and around them.[1] This sensor-equipped subpopulation of cars can potentially provide services to the driver that are unavailable to the drivers of cars without this sensing ability; one such application is a lane-change advisory system.

The purpose of a lane-change advisory system is to recommend to the driver a lane change to the left or right on

[1]This radar capability, used as part of an advanced cruise-control system, is currently available as optional equipment on high-end Mercedes-Benz vehicles; we expect it to become available from other manufacturers on a wider range of car models in the near future.

the basis of the sensed pattern of nearby traffic at any given point in time. A simple reactive scheme that maps advice directly from sensory inputs would seem to be well-motivated. Such a system could conceivably be evaluated using different metrics. We will be attempting to optimize a group metric in order to maximize overall utilization of the freeway as a shared resource.

3 PROBLEM FORMULATION

In this domain, we formulate the problem so the goal to optimize is a per time-step average reward, rather than a future discounted reward metric. More specifically, we are interested in minimizing the average per-step normalised global loss value

$$\frac{\sum_{i=0}^{n-1}[v_d(i) - v_a(i)]}{n} \quad (1)$$

where $v_d(i)$ is the desired speed of the i^{th} car, and $v_a(i)$ is the actual speed of the i^{th} car.[2] We normalise the total loss by dividing by n, the the number of cars in the simulation on a particular time-step. This yields a per-car average loss at each time-step, which is a simple but natural economic utility metric.

We use two learning algorithms, the first being a Monte Carlo style of algorithm that attempts to maximize the average reward directly, and the second being a Q-learning style of algorithm that uses discounting. Although there are arguments that suggest that this indirect approach of attempting to find a good average reward policy via learning on the basis of discounted rewards shouldn't work very well (see, for example, Schwartz (1993) for some discussion), in practice we find it often does anyway.[3] To add an extra dimension to this study, we thought it would be interesting to compare the two approaches directly for this problem, since it was reasonably straightforward to devise a distributed variant of the standard Q-learning algorithm suitable for this domain. The representation we chose, as described in the next section, was selected to allow us to do this comparison in direct a way as possible.

4 REPRESENTATION

The view of the world from the perspective of each car (agent) is described in an eight-dimensional feature vector, which encodes information about the distance and relative speeds of other cars immediately in front, to the side and behind. (See Figure 1.)

AL	AC	AR
CL	△	CR
BL	BC	BR

Figure 1: The eight features encoding an agent's state representation arranged as a grid. △ represents the position of the agent. The three grid positions in the top row hold integer values that encode information about cars ahead in the left (AL), current (AC) and right (AR) lanes respectively. The three grid positions in the bottom row encode information about cars behind in the same three lanes. The two grid positions CL (clear left) and CR (clear right) each hold a bit indicating whether a lane change in that direction is legal at that time.

The first three dimensions of the feature vector encode the pattern of cars in front of the agent, in the lane immediately to the left, the current lane, and the lane immediately to the right, respectively. Each of these first three features hold integer ranking values in the range zero to three:

- **Zero** indicates that the lane is clear; that is, that there is no car in that lane within radar range, or that the nearest car in front in that lane is currently travelling faster than the agent's desired travelling speed.
- **One** indicates that of the cars immediately ahead that are travelling less than the agent's desired speed, this car is fastest.
- **Two** indicates a car that slower.
- **Three** indicates a car that is slower still.

If two slow cars are travelling at the same speed, then they will be given the same speed ranking according to this scheme.

The last three dimensions of the feature vector encode the pattern of cars behind the agent, similar to the above scheme except in this case zero indicates that there is no car within radar range, or that the nearest car in that lane is travelling *slower* than the agent's *current* speed. One indicates the slowest car behind that is travelling faster than the agent's current speed, etc.

Finally, the remaining two dimension correspond to bits encoding whether a lane change to the left or a lane change to the right is currently a valid recommendation. If there is not a safe gap in front and behind, a lane change is considered illegal.

We also experimented with a reduced version of the state representation which only involves the first five features (ef-

[2] The actual speed of a vehicle may reach but will never exceed its desired speed, at least not in the scenario simulated, and so $v_d(i) - v_a(i)$ never becomes negative.

[3] From a theoretical viewpoint, this is actually not very surprising, since it is not difficult to show that for any (unichain) MDP, there will be a critical discount factor γ_c such that an optimal discounted policy for a discount factor $\gamma \in [\gamma_c, 1)$ will also represent a gain optimal (i.e. optimal simple average reward) policy. The trick is guess what a good choice of γ is for any particular problem formulation; too high a value will slow convergence, too low a value will fall below the critical discount factor.

fectively ignoring cars behind). It enabled us to measure the advantage gained by cars being able to look behind them.[4]

The representation used to map sensory inputs to an observation for each agent is a simplified version of the representation used in (Moriarty & Langley, 1998). We chose to simplify the representation for a number of reasons. Firstly, the relatively small state space made a table-lookup representation of the Q-function possible, which obviated the need for using a function approximator such as the ANN that was used in (Moriarty & Langley, 1998). This simplified matters conceptually, as RL with function approximation still has many theoretical and practical problems, and the table-lookup scheme allowed for a "neutral" representation to more fairly compare the behaviour of the learning algorithms we will be testing in these experiments. Secondly, the "ranking" representation allowed us to fairly compare the performance of learnt policies with that of a hand-crafted, psychologically plausible benchmark policy, since our standard "selfish drone"[5] policy (Moriarty & Langley, 1998) can be written straightforwardly as a deterministic policy given this input representation.[6]

5 THE ALGORITHMS

The first new algorithm we describe (see Figure 2) is a Monte Carlo-based method that searches the space of deterministic policies directly, without representing the value function. From a dynamic programming perspective, it is conceptually closer to policy iteration than to value iteration.

Policy iteration is a policy improvement method that starts with an arbitrary deterministic policy for a diven Markov decision process, and generates a better policy by calculating what is the best single improvement in policy available for each state individually, then combining all these individual changes in policy to generate a successor policy. The process is reapplied in turn to each successive policy. When a policy is encountered such that no improvement in policy for any single state is possible, then the method halts; such a policy can be shown to be optimal (Puterman, 1994).

The new algorithm, which we will refer to as APPIA, might be considered to perform approximate piecewise policy iteration, where the the possible policy changes for each state are evaluated by Monte Carlo estimation, rather than, for example, by matrix calculations. By "piecewise", we mean that the policy for each state is changed one at a time, rather than in parallel as in standard policy iteration. The policy improvement theorem guarantees that this approach still results in effective hill-climbing through the policy space. Moreover, if these individual change were cached and applied together at the end of one pass through all the states, the evolution through the policy space would be identical to that of classical policy iteration,[7] which is known to be an efficient method to arrive at an optimal policy for infinite-horizon Markov decision problems (Puterman, 1994).

The second algorithm we describe is a variant of Watkins' well-known Q-learning algorithms (Watkins, 1989; Watkins & Dayan, 1992).

In standard Q-learning, Q-value estimates are updated after each time-step based on the transition from state s to successor state s' by the RL agent, after selecting action a from state s. For each time-step there is one state transition and one action to deal with in order to update the Q-value estimates.

In the distributed variant of Q-learning we describe, which we will refer to as algorithm DQL (see Figure 3), we have potentially as many different state transitions to deal with per time-step as there are agents.

We deal with this straightforwardly by taking an average backup value for a state/action pair $\langle s, a \rangle$ over all agents that selected action a from state s at the last time-step. We note that the successor state for these agents may not be the same in a domain where state transition probabilities are not deterministic. We also note that the backup values may not be the same even for all agents that make the transition to the same successor state, even though all agents share the same immediate reward. This is because not all agents will necessarily have the same actions available from the same state at any given time; for example, they cannot initiate another lane change until the current lane change has been completed. In this case the Q_{max} component of the backup value is calculated over the actions that are valid for a particular agent to select at the next time-step.

One of the theoretically interesting things about this team-learning approach is that as the number of agents $n \rightarrow \infty$, the sample distribution of the successor state used for the backups for each state/action pair approaches the real underlying distribution, reducing one source of variance in the backup values. So theoretically, a massively distributed backup could accurately model the transition probabilities in as little as one time-step. This provides some motivation for combining learning a transition model with a method like this, and comparing it to a simple model-free approach as we have described here. This remains as potential future work.

[4] We note that in the current Mercedes-Benz models that employ these radar sensors, they are only configured to "see" forward.

[5] The "selfish drone" rule is simply to change to the lane that has lowest integer ranking in the top three grid positions, i.e. always greedily change what appears to be the fastest moving lane in front. Only legal lane changes are considered (the two "lane change clear" bits are respected.) If there is a tie between changing right and changing left, left lane changes win. If the current lane ties as the lowest integer, no lane change is initiated.

[6] We note the smaller version of the representation is still powerful enough to represent the "selfish drone" policy.

[7] We assume the Monte Carlo estimation method is sufficiently accurate to identify the best one-state change in policy at each step. In practice, we would only have an arbitrarily high level of confidence, depending on how much sampling we are willing to perform. This is why we characterise the method as a form of approximate piecewise policy iteration.

```
algorithm APPIA() /* Monte Carlo-based approximate piecewise policy iteration */
begin
  set arbitrary initial policy; set all states as eligible
  global_policy_value = best_policy_value = Monte_Carlo(num_monte_carlo_steps);
  while (not all states ineligible) do
    policy_value = Monte_Carlo(num_monte_carlo_steps);
    global_policy_value = α * global_policy_value + (1 − α) * policy_value;
    /* update current policy value estimate with exponentially weighted
       moving average, α ∈ [0, 1]. */
    state = select_next_eligible_state();
    best_action = cur_policy = state.policy;
    best_policy_value = global_policy_value;
    for (a ∈ valid actions for state) do
      if (a ≠ cur_policy) then
        state.policy = a;
        policy_value = Monte_Carlo(num_monte_carlo_steps);
        if (policy_value > best_policy_value) then
          best_action = a;
          best_policy_value = policy_value;
          if (policy_value > global_policy_value) then
            global_policy_value = policy_value;
          endif
        endif
      endif
    endfor
    if (cur_policy ≠ best_action) then /* suggests a policy change */
      state.policy = best_action;
      set all states as eligible
    else
      state.policy = cur_policy;
    endif
    state.eligible = False; /* set this state as ineligible */
  endwhile
end
```

Figure 2: The approximate piecewise policy iteration algorithm (APPIA). The selection of the next eligible state is stochastic, but weighted according to the total number of times that state has been visited in Monte Carlo trials so that selection is biased towards the most frequently visited states. This simple selection heuristic sped up learning in this particular domain; however, any selection strategy that guarantees that all eligible states will eventually be selected will suffice.

```
algorithm DQL(double global_reward) /* multi-agent distributed Q-learning algorithm */
begin
  for (s ∈ all states) do
    for (a ∈ all actions) do
      qtable[s][a].count = 0;
      qtable[s][a].r_sigma = 0.0;
    endfor
  endfor
  for (agent ∈ all agents) do
    new_state = agent.state;
    last_state = agent.prev_state;
    if (last_state ≠ Nil) then /* state initially Nil */
      last_action = agent.prev_action;
      qtable[last_state][last_action].r_sigma += q_max(agent,new_state);
      qtable[last_state][last_action].count++;
    endif
  endfor
  for (s ∈ all states) do
    for (a ∈ all actions) do
      if (qtable[s][a].count > 0) then
        qtable[s][a].value = (1 – β)qtable[s][a].value +
          β(global_reward + γ(qtable[s][a].r_sigma/qtable[s][a].count));
          /* β is the stepsize parameter, γ is the discount factor */
      endif
    endfor
  endfor
end
```

Figure 3: The distributed Q-learning algorithm (DQL). The value of the q_max() function depends on the agent as well as the successor state in this domain, because not all agents will have the same actions available to them from the same state in the same time-step. In this case, we return the maximum Q-value estimate over the valid actions available to the agent in that state at that time-step.

6 EXPERIMENTAL METHODOLOGY

For this traffic engineering application, we chose to write a purpose-built traffic simulator, and learn the desired reactive policies off-line. As in the elevator scheduling application of Crites & Barto (1996), off-line reinforcement learning using a simulator has many methodological advantages for this particular problem domain.

First and foremost it means that with an omniscient viewpoint we can actually measure directly the loss metric we are trying to minimise (see Equation 1). We also have the advantage of being able run the simulator faster than real time, making many long learning trials feasible. Finally, we don't need to instrument a fleet of real cars to learn a policy, and, even if we wanted to, it would be unclear that the drivers would be happy with the consequences of active exploration until the final policy was learnt. The development plan is to learn appropriate policies off-line using a simulator of appropriate fidelity, and then testing the candidate policies in real vehicles with learning turned off. As part of a product, the final policy would be integrated into an intelligent cruise control system which provides lane changing advice, possibly along with other telematic services such as route planning, etc.

The specification for the traffic simulation follows that described in Moriarty & Langley (1998). A three-lane stretch of freeway 13.3 miles long is populated by 200 cars, half of which follow the fixed "selfish drone" policy, and the other half are learning cars that follow the current learnt policy, modulo active exploration decisions.[8] The end of the stretch of road joins to the start, creating an endless loop of freeway. The cars are created each with a set desired speed drawn from a truncated Gaussian distribution, with a mean desired speed of 60 mph. All the cars share low level collision avoidance strategies that governs acceleration, deceleration and legality of lane changes. The only strategy that the drones and learning cars will differ in is in lane selection.

The results in Moriarty & Langley (1998) were derived from a simple cellular automata style of simulator that allowed for instantaneous lane changes, and did not otherwise attempt to model the spatial extension of the cars realistically. For the results we present in this paper, we use a simulator that is more realistic in this regard; cars now have a continuous spatial extension, and take several seconds to complete a lane change. Further, while a lane change is in progress, both lanes are unavailable for another car to occupy. This was felt necessary to more realistically simulate the congestion consequences of excessive lane changing. In this way it was not necessary to add an arbitrarily penalty for lane changing in the reward function, as described in Moriarty & Langley (1998); any negative consequences stemming from excessive lane changing will automatically be reflected in the global loss metric.

[8]For DQL, a simple epsilon-greedy active exploration policy was followed.

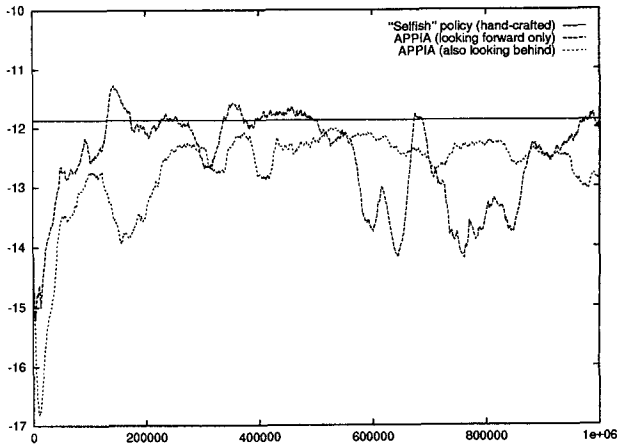

Figure 4: Learning curves for the APPIA algorithm, plotted against the hand-crafted "selfish drone" policy.

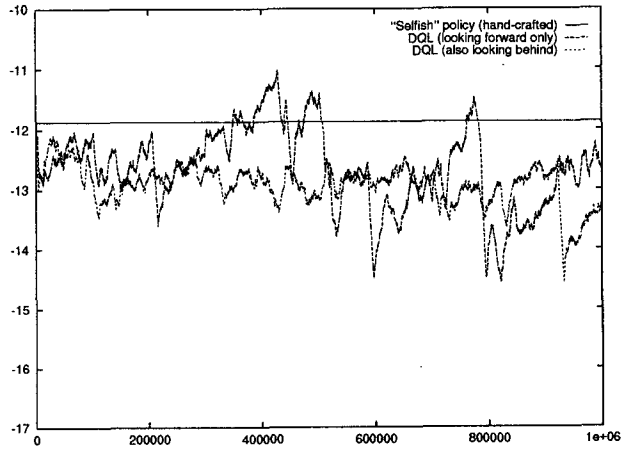

Figure 5: Learning curves for the DQL algorithm, plotted against the hand-crafted "selfish drone" policy.

7 EXPERIMENTAL RESULTS

The selfish drone policy was subjected to a long term evaluation (1,000,000 simulation time-steps using this as a fixed policy) over several runs. These consistently resulted in a per-step average reward of -11.9 (significant to three figures) per agent; we can interpret this directly as each agent, on average, travels at about 11.9 mph below its desired speed. This baseline of performance is represented as a straight line in both Figures 4 and 5, which show the learning curves for runs of the APPIA and DQL algorithms respectively.

We found that in this case the hand-crafted policy was not easy to beat; for the problem formulation and representation we are using, we now have reason to believe it is actually close to being an optimal policy, which was not obvious. Nevertheless, both algorithms did find policies that were marginally superior, in the order of 3-5% better.

Closer inspection of the learning curves for the APPIA

and DQL algorithms (Figures 4 and 5) reveal that for both algorithms, learning with the simpler "look ahead only" representation yielded marginally better results for the trials, which was also not expected. The best policies were extracted from the learning trials using the simpler representation for both algorithms. The policy corresponding to APPIA at timestep 139,900 for the trial plotted turned out to evaluate at 4.2% better than the hand-crafted "selfish drone" policy with a long-term average per-step reward of -11.4. (To get this value, the candidate policy was evaluated on a further, fixed-policy run over 1,000,000 time-steps, in the same manner as the evaluation of the hand-crafted policy as described above.) The policy corresponding to DQL at timestep 428,000 evaluated at -11.6, a 2.5% improvement over the hand-crafted policy.

We also note that for both APPIA and DQL, while the extended "look behind" representation did not result in discovering better policies, they did seem to effect more stable learning. We note the "crises" of temporarily losing a good policy that both learning algorithms suffer periodically; the effect is much more pronounced for both when the more compact representation is used.

This phenomenon might be an effect of increased state aliasing in the compact representation; interestingly, although dealiasing the states by introducing additonal features makes the learning more stable, these additional features seemed not to be useful in learning a better policy. Indeed, it appears that there is a trade-off between learning stability and the quality of policy learnt in this domain, which as far as we know is an effect that has not been reported previously in the RL literature.[9]

Because we are learning off-line in simulation, these instabilities do not represent the problem they would in an on-line setting. When learning off-line, asymptotic convergence of learning is not necessary for discovering a good policy, since we can potentially extract a good policy from any stage of the learning process. Indeed, both the learnt policies that beat the hand-crafted reference policy were extracted by simply inspecting the respective learning curves of Figures 4 and 5, and noticing where the learning algorithms were doing well, and finding out what the policy had been at stage. This might seem a simple or even obvious methodology for off-line reinforcement learning, but it also seems worth mentioning.[10]

Finally, these learning "crises" are very reminiscent of observed learning performance in a walking robot domain which also used a compact but heavily state-aliased representation (Pendrith & Ryan, 1997). In this situation, an automatically adjusting step size parameter algorithm (ccBeta) was used to help stabilise learning. We anticipate such an approach could potentially be a useful extension of the DQL algorithm.

[9] Although there may be a connection to recent work by Gomes, Selman & Kautz (1998) investigating the role of randomization in search.

[10] Of course, this process could be automated by caching the best policies as is routinely the practice in, for example, simulated annealing; but a direct inspection of a learning curve in itself can be enlightening, so the manual approach is not necessarily without its advantages.

8 RELATED WORK

The problem domain was first described by Moriarty & Langley (1998), in which some early results using Moriarty's SANE reinforcement learning scheme were reported. SANE is a unconventional and somewhat elaborate RL scheme that incorporates a genetic search over the space of possible hidden-layer neurons for an ANN. The work described in the present paper was motivated in part to better understand some of the results from this earlier study.

McCallum (1995) studied RL in the context of an agent learning a driving policy in a simulated freeway domain. This work formulated the problem in terms of classical selfish[11] single agent RL. Using an evolutionary method, Sukthankar, Baluja & Hancock (1997) also investigated learning of tactical driving skills for a single agent.

Of the work in multi-agent RL, one paradigm has involved learning individual policies for each of the agents, and the challenge has been to see if co-operation can be learnt (e.g., Gordon (1993), Mataric (1996, 1997)). For learning a single shared control policy among multiple agents, the work of Crites & Barto (1996) in the elevator control domain is most well-known.[12] The main difference between their paradigm and the one we are presenting in this paper is that they were including the states of all other agents in their state description. This is feasible when the number of agents if fixed and small, as was the case for three elevator cars working together, but this approach would not scale to a large and open ended population, as we require for this domain.

Wolpert et al. (1999) have investigated the problem of designing large decentralized multi-agent systems in the context of their COIN (collective intelligence) paradigm, which assumes a team of RL agents. The Ant-Q algorithm by Dorigo (1996) should also be mentioned, as it is also an example of many RL agents contributing to learning a single policy, but in the Ant-Q paradigm the sub-policy of the "ants" were part of the larger policy, and so they were not homogeneous agents in the sense described above.

Stone and Veloso (1997) have written a survey of multi-agent systems literature from a machine learning point of view, with a useful taxonomy of system and problem types.

[11] Hence the term "New York driving" applied to the learning problem. The reference to "New York" describes a driving philosophy, rather than to the actual scenario.

[12] Actually, the authors experimented with learning a shared policy and learning individual policies, and found it did not make a significant difference in the final performance; this result was attributed to the symmetry of the problem from the point of view of each agent (i.e., each of the three elevator cars).

9 CONCLUSIONS

Traffic engineering, which to date has largely relied on traditional operations research techniques like classical dynamic programming (e.g. Hall (1995), Ramaswamy et el. (1997)), offers a rich a challenging set of problems for which both single and multi-agent reinforcement learning methods might be useful. Some of these problem domains are naturally cast as on-line learning problems, while others, such as the one we describe in this paper, lend themselves to an off-line learning formulation.

By experimenting with two new algorithms developed for this application, we have shown that successful distributed multi-agent reinforcement learning, as formulated in this paper, is possible using both a direct policy-space approach and a value-function learning approach. Interestingly, in spite of some theoretical considerations, neither approach was obviously superior to the other for learning in this domain. In both cases, the distributed multi-agent reinforcement learning algorithms found policies that outperformed the benchmark hand-crafted policy, even though the magnitude of improvement was relatively small in absolute terms.

We have observed that in an off-line learning setting, learning instabilities are not as important as they will be in on-line reinforcement learning; good transient policies more important. Interestingly, for this domain and problem formulation, the best transient policies were observed using the representation resulting in the least stable learning curves. The apparent trade-offs at work here are worth further investigation and attempts at characterisation.

References

Crites, R., & Barto, A. (1996). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8* Cambridge, MA. MIT Press.

Dorigo, M. (1996). A study of some properties of Ant-Q. In H.-M. Voigt, W. Ebeling, I. R., & Schwefel, H.-S. (Eds.), *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, pp. 656–665 Berlin. Springer-Verlag.

Gomes, C. P., Selman, B., & Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 431–437. MIT Press.

Gordon, D., & Subramanian, D. (1993). A multistrategy learning scheme for agent knowledge acquisition. *Informatica, 17*, 331–344.

Hall, R. (1995). Lontitudinal and lateral throughput on an idealiized highway. *Transportation Science, 29*, 118–127.

Mataric, M. J. (1996). Reinforcement learning in the multi-robot domain. *Autonomous robotics, 4*(1), 73–83.

Mataric, M. J. (1997). Using communication to reduce locality in multi-robot learning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 643–648.

McCallum, A. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, Department of Computer Science, University of Rochester.

Moriarty, D. E., & Langley, P. (1998). Learning cooperative lane selection strategies for highways. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 684–691.

Pendrith, M., & Ryan, M. (1997). Estimator variance in reinforcement learning: Theoretical problems and practical solutions. In *On-line Search: Collected Papers from the 1997 Workshop. AAAI Technical Report WS-97-10*, pp. 81–88. AAAI Press.

Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: John Wiley & Sons.

Ramaswamy, D., Medanic, J., Perkins, W., & Benekohal, R. (1997). Lane assignment on automated highway systems. *IEEE Transactions on Vehicular Technology, 46*, 755–769.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Machine Learning: Proceedings of the Tenth International Conference* San Mateo, CA. Morgan Kaufmann.

Stone, P., & Veloso, M. (1997). Multiagent systems: A survey from a machine learning perspective. Tech. rep. CMU-CS-97-193, Carnegie Mellon University.

Sukthankar, R., Baluja, S., & Hancock, J. (1997). Evolving an intelligent vehicle for tactical reasoning in traffic. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Watkins, C. (1989). Learning from delayed rewards. PhD Thesis, King's College, Cambridge.

Watkins, C., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning, 8*, 279–292.

Wolpert, D. H., Wheeler, K. R., & Tumer, K. (1999). General principles of learning-based multi-agent systems. In *Proceedings of the Third Annual Conference on Autonomous Agents*. ACM Press.