In *Proceedings of the Seventeenth International Conference on Machine Learning* (ICML 2000), pp. 935-942, 2000.

TPOT-RL Applied to Network Routing

Peter Stone PSTONE@RESEARCH.ATT.COM
AT&T Labs — Research, 180 Park Ave. Room A273, Florham Park, NJ 07932 USA

Abstract

Team-partitioned, opaque-transition reinforcement learning (TPOT-RL) is a distributed reinforcement learning technique that allows a team of independent agents to learn a collaborative task. TPOT-RL was first successfully applied to simulated robotic soccer (Stone & Veloso, 1999). This paper demonstrates that TPOT-RL is general enough to apply to a completely different domain, namely network packet routing. Empirical results in an abstract network routing simulator indicate that agents situated at individual nodes can learn to efficiently route packets through a network that exhibits changing traffic patterns, based on locally observable sensations.

1. Introduction

Team-partitioned, opaque-transition reinforcement learning (TPOT-RL) is a distributed reinforcement learning technique that allows a team of independent agents to learn a collaborative task. The individual agents make their decisions based on locally observable, action-dependent features and update their value functions without knowledge of state transitions following their actions.

TPOT-RL was introduced for learning team strategies in the simulated robotic soccer domain (Stone & Veloso, 1999). This domain posed a challenging learning environment due to its enormous state space, limited training examples available, dynamically changing conditions (opponents), and *opaque transitions*: agents are unable to track state transitions subsequent to taking actions. TPOT-RL enabled multiple agents to simultaneously learn a distributed collaborative team policy by aggressively abstracting the state space. The agents learned a distributed passing and shooting policy that was empirically effective.

The purpose of this paper is to demonstrate TPOT-RL's potential as a generally applicable multiagent learning technique. In order to do so, we implement and test TPOT-RL on a completely different multiagent learning task, namely network packet routing. Like robotic soccer, network routing can be formulated as a distributed learning task with opaque transitions and dynamically changing conditions.

The remainder of the paper is organized as follows. Section 2 gives an overview of TPOT-RL. Section 3 introduces the packet routing simulator used in our experiments. Section 4 lays out our application of TPOT-RL to this domain. Section 5 details our empirical results and Section 6 concludes.

2. TPOT-RL

TPOT-RL learns a value function that maps state-action pairs to expected rewards. TPOT-RL includes three main adaptations to the standard RL paradigm:

- The value function is *partitioned among the team*, with each agent only learning for states from which it can act. All agents are trained simultaneously with a decreasing exploration rate.

- *Action-dependent features* (defined below) are used to produce an aggressively generalized feature space, which is used as the input representation for learning. While other RL approaches aggregate states to reduce the size of the learning task (e.g., McCallum, 1996), action-dependent features enable the creation of a particularly small but informative feature space for learning.

- Agents learn despite the fact that state transitions subsequent to their actions are opaque to them. No agent is in full control of the team's goal achievement. A domain is *opaque-transition* if the learning agents do not know the complete and relevant state information that guides their teammates' actions, or even when they can act.

Formally, a policy is a mapping from a state space S to an action space A such that the agent using that policy executes action a whenever in state s. Designed to work in real-world domains with far too many states to handle individually, TPOT-RL exploits action-dependent features to create a small feature space V. V is used as a component of the input representation of the learned value function $Q : V \times A \mapsto \mathbb{R}$.

In short, the policy's mapping from S to A in TPOT-RL can be thought of as a 3-step process:

State generalization: The state s is generalized to a feature vector v using the state generalization function $f : S \mapsto V$.

Value function learning: The feature vector v is used as an input to the learned value function $Q : V \times A \mapsto \mathbb{R}$, which estimates the expected reward for taking each possible action.

Action selection: An action a is chosen for execution and its long-term, observed reward is used to further update Q.

State generalization, value function learning, and action selection in TPOT-RL are further specified in Sections 2.1, 2.2, and 2.3 respectively.

2.1 State Generalization

$f : S \mapsto V$ maps the current state of the world, s, to the feature vector used for learning, v. f relies on a unique approach to constructing V. Rather than discretizing the various dimensions of S, it uses an *action-dependent* feature function.

The action-dependent feature function

$$e : S \times A \mapsto U$$

evaluates each possible action $a_i \in A$ based on s. U is a discrete set of features reflecting expected short-term effects of actions. Unlike Q, e does not produce the expected long-term reward of taking an action; rather, it classifies the likely short-term effects of the action. In the soccer implementation, e was a learned pass-completion predictor that compressed an enormous state space into a single feature useful for leaning the long-term effects of actions (Stone & Veloso, 1999).

In the multiagent scenario, other than one output of e for each action, the feature space V also involves one coarse component that partitions the state space S among the agents. The partition function

$$P : S \mapsto M$$

breaks the state space into $|M|$ disjoint partitions to be divided among the teammates, with $|M| \geq m$ where m is the number of teammates. Partitioning is possible in domains in which specialization is possible. For example, in the robotic soccer implementation, agents learned how to act from a particular geographic location on the field. In particular, if the set of possible actions in state s is $A = \{a_0, a_1, \ldots, a_{n-1}\}$, then

$$\begin{aligned} f(s) &= \langle e(s, a_0), e(s, a_1), \ldots, e(s, a_{n-1}), P(s)\rangle, \text{ and} \\ V &= U^{|A|} \times M. \end{aligned}$$

Thus, $|V| = |U|^{|A|} * |M|$. Since the goal of constructing V is to create a feature space that is smaller than the original state space, the ranges of the action-dependent feature function and partition function, U and M respectively, are ideally as small as they can be without abstracting away the useful information for learning.

This state generalization process reduces the complexity of the learning task by constructing a small feature space V which partitions S into $|M|$ regions. Each agent needs to learn how to act only within its own partition(s). Nevertheless, for large A, the feature space can be too large for learning, especially with limited training examples. Our particular action-dependent formulation allows us to reduce the effective size of the feature space in the value-function-learning step. Choosing features for state generalization is generally a hard problem. While TPOT-RL does not specify the function e, our work illustrates effective choices of e.

2.2 Value Function Learning

As we have seen, TPOT-RL uses action-dependent features. When using action-dependent features, we can assume (heuristically) that the expected long-term reward for taking action a_i depends only on the feature value related to action a_i. That is,

$$Q(f(s), a_i) = Q(f(s'), a_i) \tag{1}$$

whenever $e(s, a_i) = e(s', a_i)$ and $P(s) = P(s')$. Recall that

$$\begin{aligned} f(s) &= \langle e(s, a_1), \ \ldots, e(s, a_{n-1}), \ P(s)\rangle \\ f(s') &= \langle e(s', a_1), \ldots, e(s', a_{n-1}), P(s')\rangle \end{aligned}$$

Another way of stating this same assumption is that $Q(f(s), a_i)$ depends upon $e(s, a_i)$ and is independent of $e(s, a_j)$ for all $j \neq i$.

Without this assumption, since there are $|A|$ actions possible for each element in V, the value function Q has $|V| * |A| = |U|^{|A|} * |M| * |A|$ independent values. Under this assumption, however, the Q-table has at most $|U|^1 * |M| * |A|$ entries: for each action possible from state s, only one of the $|A|$ action-dependent feature values $e(s, a_i)$ comprising $f(s)$ is relevant. Therefore, even with only a small number of training examples available, we can treat the value function Q as a lookup-table without the need for any complex function approximation. To be precise, Q stores one value for every possible combination of $a \in A$, $e(s, a) \in U$, and $P(s) \in M$.

After taking an action a while in state s with $f(s) = v$, an agent receives reward r and uses it to update $Q(v, a)$ from its previous value $Q(v, a)$ in the Monte Carlo style as follows:

$$Q(v, a) = Q(v, a) + \alpha(r - Q(v, a)) \tag{2}$$

where α is the learning rate. The reward r is derived from the observable environmental characteristics—those that are captured in S—over a maximum number of time steps t_{lim} after the action is taken. The reward function

$$R : S^{t_{lim}} \mapsto \mathbb{R}$$

returns a value at some time no further than t_{lim} in the future. The reward is discounted based on the amount of time between acting and receiving the reward.[1] During that time, other teammates (or opponents, if any) can act in the environment and affect the action's outcome, but the agent may not be able to observe these actions. In practice, the range of R is $[-Q_{max}, Q_{max}]$ where Q_{max} is the reward for immediate goal achievement.

Notice from Equation 2 that in TPOT-RL, the updated action value depends only on the stored action value in the same feature-state (i.e. element of V) as opposed to chains of learned values. That is, when updating $Q(v, a)$ TPOT-RL does not reference any $Q(v', a')$ such that $v \neq v'$ or $a \neq a'$. However, the update in Equation 2 is not strictly Monte Carlo since r may not reflect the result of the entire trial.

The reward function, including t_{lim} and Q_{max}, is domain-dependent. One possible type of reward function is based entirely upon reaching the ultimate goal (Monte Carlo). In this case, an agent charts the actual (long-term) results of its policy. However if goal achievement is infrequent, a reward function based on intermediate reinforcement may be needed. The latter was the case in the robotic soccer domain.

2.3 Action Selection

As in all RL techniques, the issue of exploration vs. exploitation is important for TPOT-RL. Particularly since the target concept can shift due to teammates learning and changing their policies, or due to changes in policies of opponents (if any), it is important for agents to gather information about the value of actions that are currently considered sub-optimal by the value function. Any standard exploration heuristic, such as the randomized Boltzmann exploration strategy (Kaelbling et al., 1996), could be used.

[1]This discounting replaces γ that is often used in RL paradigms.

2.4 Summary

In order to apply TPOT-RL to particular learning problems, as we do in Section 4, the following functions and variables must be specified within the domain:

- The action-dependent feature function e and its range U.
- The partition function P and its range M.
- The reward function R including variables Q_{max} and t_{lim}.
- The learning rate α.

3. Network Routing

TPOT-RL has previously been applied to learn effective behaviors in one team-partitioned, opaque-transition domain, namely simulated robotic soccer (Stone & Veloso, 1999). We identify network routing as another team-partitioned, opaque transition domain. We use a modified version of a publicly available packet routing simulator (Boyan & Littman, 1994).

In this simulator, a network at time t consists of:

- A set of *nodes* $N = \{n_0, \ldots, n_{m-1}\}, |N| = m$. Each node n_i consists of a queue of packets $K_{n_i} \subseteq K$, $|K_{n_i}| = k_{t,n_i}$ at time t. As packets are introduced into and removed from the queue, K_{n_i}, and consequently k_{t,n_i}, changes over time.

- A set of *links* $L \subseteq \{(n_i, n_j) | n_i, n_j \in N\}$ connecting pairs of nodes. From any node n_i, $L_{n_i} \subseteq N$ is the set of links from n_i: $L_{n_i} = \{n \in N | (n, n_i) \in L\}$. $|L_{n_i}| = l_{n_i}$. All links are bidirectional: $(n_i, n_j) \in L \rightarrow (n_j, n_i) \in L$.

- A set of *packets* $K = \{k_0, \ldots, k_{z_t - 1}\}, |K| = z_t$ at time t. Each packet k_i is introduced at a source node $k_{i_{source}} \in N$ and travels towards its destination node $k_{i_{dest}} \in N$. The packet also stores the time at which it left its source, $k_{i_{stime}}$, and when it arrives, the time at which it reaches its destination $k_{i_{dtime}}$. $k_{i_{path}} \subseteq L$ is an ordered list of links along which k_i has traveled from $k_{i_{source}}$ to its current position along with the times at which it has traversed each link. As packets are introduced into and removed from the network, K, and consequently z_t, changes over time.

- A *node capacity* C_{node} indicating the maximum number of packets allowed in a node's packet queue: $\forall i, k_{t,n_i} \le C_{node}$.
- A *network capacity* C_{net} indicating the number of packets that can be active at one time in the network: $z_t \le C_{net}$.

Two other parameters affecting the simulation are t_l which is the time it takes a packet to traverse a link and t_n which is the time it takes for a node to process one packet. If at time t, packet k enters the queue K_{n_i} at node n_i, it will stay there for $k_{t,n_i} t_n$ seconds. t_l, t_n, C_{node}, and, C_{net} all apply uniformly in the network.

The packet routing problem can be viewed as a multiagent collaborative problem by modeling each node as having an independent agent which makes the routing decisions at that node. The agents act as a team as they try to cooperate in sending packets through the network as efficiently as possible. The domain is team-partitioned since each agent learns only a policy at its own node: the function P partitions the state space based, in part, on the node at which each agent is situated. Prior research indicates that distributed network control is advantageous even in high-speed networks (such as ATM networks) in which centralized control is possible (Horikawa et al., 1996).

Network routing is opaque-transition because agents cannot see a packet's route after sending it. Agents' actions are chained, with each agent able to affect which agent will act *next*, but with no control beyond that. Agents get no short-term reward for their actions and cannot track the transitions in the environment.

Nonetheless, the team of agents can learn to effectively route packets by observing locally observable state information (network traffic). The action-dependent feature function e in our implementation of TPOT-RL for network routing provides useful local information that correlates with the long-term reward: e returns the amount of recent network traffic on the links leading from an agent's node.

Another feature of network routing is that the world changes dynamically in a manner beyond the team's control: the distribution of packets introduced into the network can change.

4. TPOT-RL for Network Routing

4.1 State Generalization

As defined in Section 2.1, the function $f : S \mapsto V$ generalizes the state space based on two components: an action-dependent feature function $e : S \times A \mapsto U$ and a coarse partitioning function $P : S \mapsto M$. Given a state $s \in S$ from which the agent at node n_i is faced with the decision of routing packet k_j:

$$\begin{aligned} M &= N \times N \quad (|M| = m^2) \\ P(s) &= (n_i, k_{j_{dest}}) \end{aligned}$$

Using this partitioning function P, the agent at node n_i learns to act only in the cases that $P(s) = (n_i, k_{j_{dest}})$ for some j: the space is partitioned evenly among the m agents, with each getting m partitions.

The action-dependent feature function $e : S \times A \mapsto U$ is defined as follows. A is the set of actions available and is represented in terms of the nodes to which a packet can be sent. Thus $A = N$. However, the agent at node n_i may only use the actions in $L_{n_i} \subseteq N$—the set of links from node n_i. The elements of

$$U = \{high, low\}$$

reflect the network activity over a particular link in the last `activity_window` time units. An agent can store the link activity along all of the links from its node since it is either the sender or the recipient of all packets sent along these links.

Define $\tau(s, a, \texttt{activity\_window})$ as the number of packets sent along the link corresponding to action a in the last `activity_window` simulated seconds divided by `activity_window`. Then if action a is the act of sending a packet along link l,

$$e(s,a) = \begin{cases} high & \text{if } \tau(s,a,\texttt{activity\_window}) \ge C \\ low & \text{if } \tau(s,a,\texttt{activity\_window}) < C \end{cases}$$

We use `activity_window` = 100 and $C = .5$. Notice that e is an action-dependent function since it depends on the proposed action of sending a packet along link l. It is also based entirely upon local information available to agents that maintain internal state, collecting traffic statistics over time.

This state generalization reduces a huge state space to the point that agents can store Q-values in a lookup table. In our implementation, the entire state space in our experiments has more than 10^{3000} states (Stone, 2000). However, with $|U| = 2$, $|A| \le 3$, and $|M| = m^2$ the total number of Q-values for the team to learn is at most $6m^2$, with each agent learning no more than $6m$ values. In our experiments reported below, $m = 12$. Therefore each agent must learn only 72 Q-values.

4.2 Value Function Learning

As per equation 2, agents learn $Q(v, a)$—the value of taking action a when in a state s such that $f(s) = v$—by receiving a reward r via a reward function R. In this case, if v indicates that node n is trying to send a packet k on its way to node k_{dest} ($P(s) = (n, k_{dest})$), then $Q(v, a)$ is meant to estimate the time that it will take for the packet to reach node k_{dest}. Thus agents aim to take actions that will lead to *minimal* r.

R is almost entirely based on the actual time that the packet k takes to travel from node n_i to k_{dest}. When k successfully arrives at k_{dest}, the agent at node k_{dest} can examine the times at which it left each node along its path from k_{source} as stored in k_{path}. From this information, it can deduce the time taken from each node along the path given the action taken at that node. Then periodically, every `update_interval` seconds, the nodes in the network update each other on the long-term results of their actions.

Thus, after the agent at node n_i sends a packet k_i along link l at time $t_{k_i,l}$, the agent at node n_i receives reward r equal to the time it took for the packet to eventually reach its destination $k_{i_{dest}}$: $r = k_{i_{dtime}} - t_{k_i,l}$. In this case, the goal of each node is to minimize its reward r (which can therefore be thought of as a "cost"). Notice that this formulation of the reward function R is entirely goal oriented—in general, there is no opportunity for the agent at node n_i to observe a packet's progress on the way to its destination. Thus, this implementation of TPOT-RL uses a Monte Carlo learning strategy.

However, there *is* one exception. Especially at the early stages of learning when actions are mostly random, a packet often returns to a node from whence it came at some interval t later than it last left the node. In this situation, the agent at that node infers that the previous action a taken on this packet was ineffective and generates an intermediate reward $r = Q(v, a) + t$, thus increasing the cost estimate $Q(v, a)$.

To put a bound on the timing of rewards, r is bounded by Q_{max}. That is, if a node takes longer than Q_{max} to arrive, $r = Q_{max}$. A node can assume that the packet did not arrive in this time if it has not heard about its arrival after $Q_{max}+$ `update_interval` simulated seconds. Therefore, $t_{lim} = Q_{max}+$ `update_interval`. In our experiments, we use `update_interval` $= 10$, $Q_{max} = 2000$, and $t_{lim} = 210$.

Finally, after the agent at node n takes action a on a packet k destined for node k_{dest} and receives reward r, $Q((e(s, a), (n, k_{dest})), a)$ is updated according to equation 2 with learning rate $\alpha = .02$. Thus, even though we average all reward values achieved as a result of taking an action in a given state, each new example accounts for 2% of the updated Q-value: rewards achieved further in the past are weighted less heavily.

4.3 Action Selection

In our network routing implementation of TPOT-RL, we use optimistic initialization and no deliberate exploration: Q-values are all initialized to low values (0 or the shortest path length between nodes) and agents always choose the action with the lowest Q-value. Thus, each action is tried at least once. We find the total exploitation strategy to be effective, presumably because as unsuccessful actions are repeated, their costs increase due to network congestion, thus causing agents to try the other alternatives periodically. Should exploration become necessary, we could easily switch to a probabilistic action-selection strategy, as in the previous TPOT-RL application.

5. Empirical Results

5.1 Experimental Setup

All of our experiments use a network architecture (node and link patterns) as shown in Figure 1. The nodes are numbered for reference in the text. Packets are injected into the network at random intervals according to a Poisson distribution at an average rate of 3 per simulated second. We create three different traffic patterns within this network by controlling the distributions of sources and destinations of injected packets. Our traffic patterns are controlled by two variables: p_6, the probability that a new packet is destined for node 6; and f_s, the frequency with which the traffic pattern switches (i.e. number of simulated seconds between pattern switches). In our experiments, we use $p_6 = .25$ and $f_s = 10{,}000$. The three traffic patterns we define are:

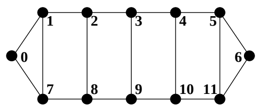

Figure 1: The network architecture used for our experiments. The nodes are numbered for reference in the text.

Top-heavy: With probability p_6, the injected packet has node 6 as its destination and a random source; with probability $1 - p_6$, the injected packet has source and destination chosen randomly (without replacement) from the set of nodes {1,2,3,4,5}.

Bottom-heavy: With probability p_6, the injected packet has node 6 as its destination and a random source; with probability $1 - p_6$, the injected packet has source and destination chosen randomly from the set of nodes {7,8,9,10,11}.

Switching: Every f_s simulated seconds, the traffic pattern switches between the top-heavy and bottom-heavy patterns.[2]

In all of our experiments, we use $C_{node} = C_{net} = 1000$, $t_n = t_l = 1.0$. We test several different packet routing strategies under the different traffic patterns defined above. We define the strategies in terms of what the agent at node n does when trying to route packet k to its destination k_{dest}. It must choose from among the possible links in L_n.

Shortest (SHRT): k is sent along the link that would get it to k_{dest} in the fewest number of hops. Shortest paths are precomputed and stored based on the network topology. If more than one link would lead along paths of the same shortest length, one such link is chosen randomly.

Hand-coded (HAND): We designed a policy to work well with the top-heavy traffic pattern.

- When $k_{dest} = 6$, if $n \in \{7, 8, 9, 10, 11\}$, k is sent along the shortest path towards node 6 (along the bottom of the network in Figure 1). Otherwise, ($n \in \{0, 1, 2, 3, 4, 5\}$) k is sent to a node in the set $\{6, 7, 8, 9, 10, 11\}$ from where it can then continue along the shortest path to node 6. Note that in all cases k can be sent to a node in this set in one hop.
- When $k_{dest} \in \{1, 2, 3, 4, 5\}$, k is sent along the shortest path (along the top of the network).
- Similarly, when $k_{dest} \in \{7, 8, 9, 10, 11\}$, k is sent along the shortest path (along the bottom of the network).

We expect this policy to do fairly well with the top-heavy traffic pattern since the traffic is distributed fairly evenly among the top and bottom portions of the network. Packets headed for node 6 ($p_6 = 25\%$ of the packets) use the bottom, while other packets use the top of the network.

Q-routing (QROUT): This strategy is the Q-routing algorithm (Littman & Boyan, 1993). We use the developers' exact implementation for testing. We use the default learning rate of $\eta = 0.7$. Q-routing is not the best known approach in this domain. We compare against it as another learning approach which performs respectably and whose implementation by its creators is available in the simulator we use. Q-routing has the same action space as TPOT-RL, and the same inputs except for the action-dependent feature τ.[3]

TPOT-RL: This strategy is the one described in detail in Sections 4.1–4.3.

Stored Q-routing and TPOT-RL policies can be loaded and used exclusively with no further learning allowed. We use this technique for testing purposes.

It should be noted that Q-routing and TPOT-RL operate under slightly different assumptions. TPOT-RL relies on transmission of packet arrival time from the eventual destination node to all other nodes that routed the packet (TD(1) or Monte Carlo); in Q-routing, nodes only get "expected-time-to-go" information from neighboring nodes (TD(0)). The feedback packets traverse one link each in Q-routing, while they traverse multiple links in TPOT-RL. However, Q-routing requires such information to be transmitted after each packet is sent, while in TPOT-RL, the feedback can be bundled and transmitted arbitrarily infrequently (by adjusting `update_interval`) or opportunistically when network load is low, allowing for a tradeoff between communication overhead and rate of learning. Thus each approach has its advantages and disadvantages in terms of overhead.

5.2 Experiments

This section presents empirical results verifying the effectiveness of TPOT-RL in the network routing domain. First we present comparisons of the different routing strategies in a top-heavy traffic pattern. Then we describe the effects of testing the resulting policies on the bottom-heavy traffic pattern. Finally, we present our results from training policies on the switching traffic and testing their generalization across the three traffic patterns.

5.2.1 Top-Heavy Traffic

First, we tested the four different routing strategies under top-heavy network traffic conditions. We chart both the average delivery time of the packets and the average number of hops per packet. Results are tabulated over intervals of 100 simulated seconds (see Figure 2). In this and all other TPOT-RL runs, unless specified otherwise, $|U| = 1$: under constant conditions, TPOT-RL can be effective even without the help

[2] Such traffic patterns might be observed due to the opposite day/night cycles in Asia and the U.S.

[3] We tried augmenting Q-routing's feature space to include τ so that it could learn from the state as TPOT-RL. Doing so caused Q-routing's performance to degrade slightly. So we did not augment Q-routing's state space for the experiments reported in this paper.

of action-dependent features. Figure 2 shows averages over 10 runs with error bars.

By definition, SHRT produces the minimum possible hops per packet, but it exhibits a continual increase in average delivery time as the network fills up and the nodes' packet queues lengthen. SHRT is not shown in the right graph in Figure 2 since it quickly goes off the scale, ending up at over 250 seconds. HAND also outperforms TPOT-RL in terms of number of hops, but in terms of the real measure of interest—delivery time—TPOT-RL and HAND do not perform significantly differently. TPOT-RL achieves its performance by sending some packets along longer, less congested paths. The remainder of the results in this section are presented in terms of average delivery time.

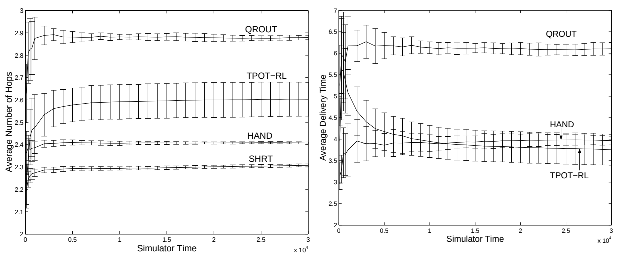

Figure 2: Average number of hops (left) and delivery time (right) in a network with top-heavy traffic.

5.2.2 Bottom-Heavy Traffic

Figure 3 (left) illustrates the results of running policies designed or trained for top-heavy traffic under bottom-heavy conditions. Results represent averages over 100 runs with error bars shown. The solid bars show average traffic delivery time for SHRT, HAND, QROUT, and TPOT-RL under top-heavy traffic.[4] These numbers are the same as the end-results shown in Figure 2. As noticed above, the HAND, QROUT, and TPOT-RL strategies all perform well, having been designed or trained for these conditions.

However, when the resulting policies are tested under bottom-heavy traffic conditions, none of them perform well. In these runs (hollow bars), the policies trained for the top-heavy case are used with no additional learning. Of course the SHRT and HAND policies, which are not learned, are constant throughout.

[4]QROUT and TPOT-RL exhibit similar results (6.0 and 3.9 average delivery time respectively) under uniform traffic conditions. Since it was specifically designed for top-heavy traffic, HAND does significantly worse (470). As would be expected, SHRT does the best under these conditions (3.3).

5.2.3 Switching Traffic

While we would not expect a policy trained exclusively under top-heavy network traffic conditions to generalize to bottom-heavy conditions, we would like a policy trained under switching conditions, since it includes periods of both other traffic patterns, to generalize across all three. Figure 3 (right) shows the results of HAND, QROUT, and TPOT-RL under all three traffic patterns. Here, TPOT-RL is run with $|U| = 2$: agents can implement different policies under different local traffic conditions. Note that Q-routing always routes packets based solely on their destinations.

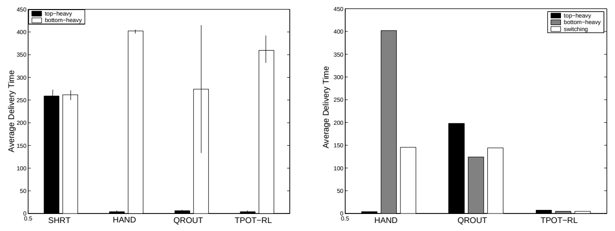

Figure 3: Fixed policies running in different traffic patterns. (left) The QROUT and TPOT-RL policies are trained under top-heavy conditions. (right) They are trained under switching conditions.

In all the learning cases, a trial consists of first training a policy under switching network traffic for 300,000 simulated seconds and then fixing the policy for testing. Results are averaged over 1000 trials.

Figure 3 (right) clearly shows the advantage of TPOT-RL. Since HAND is designed explicitly for top-heavy traffic, it fails as expected in other conditions. Similarly, QROUT is forced to adapt to one traffic pattern at the expense of the other. Some of the time, it optimizes its performance for top-heavy conditions, causing its results to look like those of HAND. Other times, it optimizes for bottom-heavy conditions, leading to exactly the opposite performances under top and bottom-heavy conditions. Therefore, error bars on this graph would go completely off the scale.

On the other hand, TPOT-RL takes advantage of its action-dependent feature function to find correlations between local information and long-term reward. In this case, TPOT-RL is able to perform well under all traffic conditions with a single policy.

6. Discussion and Conclusion

TPOT-RL applies in domains in which there are multiple agents organized in a team; there are opaque state transitions; there are too many states and/or not enough training examples for traditional RL techniques; and the target concept is non-stationary.

It relies on a long-range reward signal and action-dependent features. TPOT-RL was has been applied to robotic soccer and now to network routing. Other domains to which TPOT-RL could potentially be applied are information networks, distributed logistics, and rescue missions. In all of these team-partitioned, opaque-transition domains, a team of agents works together towards a common goal, but each individual agent only executes a portion of the actions along the path to the goal. Agents can control their own destinies only intermittently and at irregular intervals.

The most challenging aspect of environments for which TPOT-RL is designed is the opaque-transition characteristic. Q-routing requires that agents have or learn at least an approximate model of the state transitions resulting from its actions. Even the partially observable Markov decision process (POMDP) (Kaelbling et al., 1996) framework, which is designed for problems with hidden state, relies on agents having some knowledge of state transitions: POMDPs assume that agents know when the system has transitioned to a new state and a new action can be taken. Thus reward can be passed back through the state-action trajectory.

It has been previously shown that undiscounted Monte Carlo credit assignment (actual return) works in non-Markov decision processes when using "direct" RL, but not using TD or discounted rewards (Pendrith & McGarity, 1998). TPOT-RL takes advantage of this fact, although it does not in general use full trajectories as the source of reward: in the robotic soccer implementation, reward was based on the change in the environment over a limited time period.

The experiments in this paper are conducted in an abstract network routing simulator using a single, simple network topology. In addition, the algorithm against which TPOT-RL is compared is not the best known approach in this domain. As such, we do not present TPOT-RL as an industrial-strength network router, and we do not claim that TPOT-RL dominates any other algorithm under all conditions. Rather, this paper demonstrates that TPOT-RL can achieve respectable results in a domain different from its original application domain and therefore warrants further investigation as a general multiagent learning technique.

Aside from Q-routing, another approach to network routing in the same simulator is inspired by an ant metaphor (Subramanian et al., 1997). Ants crawl backwards over the network to discover link costs and shortest paths. This system adds an overhead cost of sending the ant packets through the network.

Similarly inspired by the ant metaphor, AntNet (Caro & Dorigo, 1998) agents traverse a routing network and write information at the nodes reflecting their experience of the current network status. Within the framework presented in that work, TPOT-RL is a distributed, adaptive, non-minimal (i.e. packets do not necessarily always go along minimal cost paths), and optimal (i.e. the objective is to optimize the entire network's performance as opposed to any individual packet's traversal time) routing algorithm.

In addition to testing TPOT-RL in more complex routing scenarios, future work includes applying TPOT-RL to still more distributed, opaque-transition domains such as information networks, distributed logistics, and rescue missions.

References

Boyan, J. A., & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances In Neural Information Processing Systems 6*. Morgan Kaufmann Publishers.

Caro, G. D., & Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, *9*, 317–365.

Horikawa, K., Aida, M., & Sugawara, T. (1996). Traffic control scheme under the communication delay of high-speed networks. *Proceedings of the Second International Conference on Multi-Agent Systems* (pp. 111–117). Menlo Park, California: AAAI Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Littman, M., & Boyan, J. (1993). *A distributed reinforcement learning scheme for network routing* (Technical Report CMU-CS-93-165). Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Pendrith, M. D., & McGarity, M. J. (1998). An analysis of direct reinforcement learning in non-markovian domains. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 421–429). Morgan Kaufmann.

Stone, P. (2000). *Layered learning in multiagent systems: A winning approach to robotic soccer*. MIT Press.

Stone, P., & Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. *Proceedings of the Third Annual Conference on Autonomous Agents* (pp. 206–212). ACM Press.

Subramanian, D., Druschel, P., & Chen, J. (1997). Ants and reinforcement learning: A case study in routing in dynamic networks. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann.