Temporal Difference Learning in Network Routing

Reid Broadbent Casey T. Deccio
Mark Clement
Computer Science Department
Brigham Young University
Provo, UT 84602
rb48@email.byu.edu, casey@byu.edu, clement@cs.byu.edu

Abstract

Efficient network routing is essential to the success of applications on the Internet. In this paper we summarize work that has been done to apply Q-learning—a machine learning paradigm—to the problem of network routing. We propose an extension to this work, that of using an algorithm called $TD(\lambda)$ to speed up the process of learning optimal network routes. Employing a simple ad hoc simulator, we show how this approach can be applied to network topologies of various sizes and structures. We compare the behavior of the TD-Routing approach with a previously published method called CDRQ-Routing. Our results show that proper use of TD-Routing can significantly reduce routing policy convergence time.

1 Introduction

As the Internet grows and reaches a larger population of users, network routing is a practical problem to which machine learning can be applied. Intelligent and efficient mechanisms are needed to maintain connectivity in dynamic environments and in case of network failure. Traditional routing algorithms [3, 6] have relied on non-learning approaches, and these have generally sufficed. As topologies become increasingly more complex and diverse, intelligent algorithms are needed to learn the shortest paths to other nodes. Other research has been produced [2, 4, 8], showing that machine learning techniques can be successfully applied to the task of choosing routes in a computer network.

In this paper, we outline several ways that machine learning has been applied to the field of network routing and describe an extension to these approaches. In section 3, we describe Q-Routing—a routing technique based on the Q-learning paradigm—and several variations of it. Section 4 describes the machine learning concept of $TD(\lambda)$ and its network routing counterpart, TD-Routing. In section 5 we discuss the simulation environment and the methods we used to validate our work. We describe the algorithm comparison experiments in section 6 and present their results in section 7. Section 8 contains our conclusions, and the possibilities for future work are found in section 9.

2 Related Work

In his paper entitled *Distributed Neural Network Routing* [4], Menke applied the technology of neural networks to the problem of net-

work routing. He concluded that neural networks learn network routing paths as well as traditional methods such as OSPF [6], and in some cases could do so more quickly. This is a demonstration of a successful application of machine learning techniques to network routing.

A Ph.D. dissertation by Peter Stone [8] showed another application of a machine-learning paradigm to network routing. Stone's major focus was learning in a multi-agent systems environment, and his dissertation concentrated mainly on the problem of learning control strategies for robot soccer. He also included a section on how his method can be applied to network routing, as a demonstration of the algorithm's generality.

Kumar and Miikkulainen [2] wrote a paper on Q-Routing and three variations on it, called CQ-Routing (confidence-based Q-Routing), DRQ-Routing (dual-reinforcement Q-Routing) and CDRQ-Routing, a combination of the previous two. Their conclusion was that the combination approach, CDRQ-Routing, significantly outperformed the other variations as well as the original Q-Routing. The research presented in this paper is largely an extension to their work.

3 Routing approaches using Q-learning

This section briefly describes the concepts of using Q-learning for network routing, a concept known as *Q-Routing*. The following are descriptions of the Q-learning paradigm and three variations of the Q-learning paradigm, presented by Kumar and Miikkulainen [2].

3.1 Q-Routing

Using the Q-Routing algorithm, each network router r maintains a table of values $Q_r(h, d)$ estimating the cost to send a packet to a destination d through an immediate neighbor h. In routing a packet, r selects its neighbor h with the shortest estimated cost to d. Router r learns by receiving information back from the neighbor h and updating its table accordingly. After h receives the packet, it will return a message to r with its estimate of the remaining cost to send the packet to its destination: $Q_h(\hat{z}, d) = \min_{z \in N(h)} Q_h(z, d)$, where $N(n)$ denotes the set of neighbors of node n. Router r updates its table with this rule:

$$\Delta Q_r(h, d) = \eta_f (Q_h(\hat{z}, d) + c_{h \leftarrow r} - Q_r(h, d)) \quad (1)$$

where $c_{h \leftarrow r}$ is the cost of the link from r to h and η_f is the learning rate for forward exploration.

3.2 CQ-Routing

An extension to Q-Routing is *confidence-based Q-Routing* (CQ-Routing). This algorithm makes use of the idea that a Q-value that has not been updated for a long time is less reliable than a more-recently-updated one. A *confidence value* (C-value) $C_r(h, d) \in [0, 1]$ is assigned to each Q-value $Q_r(h, d)$. A C-value close to 1 indicates that the Q-value is a good predictor of network state, and one close to 0 is highly random. C-values corresponding to direct links between nodes are set to 1, because it is assumed that these costs are known exactly. Others are initialized to 0.

C-values are used to determine the learning rate for updates to Q-values. Both the C-value at sending router r and the one received from the neighbor h (which is sent along with the update for the Q-value) are used in calculating that learning rate:

$$\eta_f = \max(C_h(m, d), 1 - C_r(h, d)) \quad (2)$$

In this way, the learning rate is high if either the old C-value $C_r(h, d)$ is low or the one associated with the update, $C_h(m, d)$, is high.

All C-values are decayed at each time step by multiplying by a factor $\lambda_C \in (0, 1)$. (Kumar [2] refers to this factor as λ, but herein it will be denoted λ_C to distinguish it from the λ used in the $TD(\lambda)$ algorithm.)

3.3 DRQ-Routing

Dual-reinforcement Q-Routing (DRQ-Routing) is another extension to the Q-Routing paradigm. This extension takes advantage of the information that a packet has learned on the journey from its source s to the current router r, where the packet resides in mid path. When r sends the packet to its neighbor h, it includes in the packet the cost estimate of sending a packet from r to s, that is $Q_r(\hat{z}, s) = \min_{z \in N(r)} Q_r(z, s)$. With this new estimate and a learning rate for backward exploration, η_b, router h is able to update its estimate of the cost to send a packet to node s through its neighbor r:

$$\Delta Q_h(r, s) = \eta_b(Q_r(\hat{z}, s) + c_{r \leftarrow h} - Q_h(r, s)) \quad (3)$$

3.4 CDRQ-Routing

CDRQ-Routing is simply the combination of the two previous approaches, CQ-Routing and DRQ-Routing. Updates are made in both the forward and backward directions, and for each update there are associated C-values which are used in calculating both learning rates η_f and η_b. Kumar and Miikkulainen [2] found this combination approach to perform better than any of the previously described algorithms, demonstrating that the two extensions had independent and complementary contributions to the algorithm's performance.

4 TD-Routing

We present in this paper an extension to Q-Routing that makes use of a variation of Q-learning called $TD(\lambda)$ or *temporal difference* learning [5]. In traditional Q-learning, when an agent goes through one time step of learning, it only updates the Q-value associated with the single state transition that it just experienced. With $TD(\lambda)$, however, multiple Q-values can be updated at each time step. The Q-values that have an appreciable change are the ones that occurred in the recent past. Each update is discounted by a factor λ, which is a real value chosen between 0 and 1, for each time step that it is removed from the present. For example, the transition that occurred one time step before the most recent one will have its Q-value updated like the most recent one, except the magnitude of the update will be multiplied by λ, making it somewhat smaller. The Q-value for the transition two time steps before the most recent one is discounted by λ^2, and so on. Traditional Q-learning is actually a special case of $TD(\lambda)$, with $\lambda = 0$.

We have applied the $TD(\lambda)$ algorithm to Q-Routing in a mechanism called *TD-Routing*. It would be impractical to update Q-values arbitrarily far back in time (since that would require sending an arbitrarily large number of packets with update information), though in theory this is what $TD(\lambda)$ does. A close approximation is to update a fixed number of transitions in the past. When $\lambda < 1$, successive powers of λ will approach zero. More than a few time steps in the past, an update will usually be discounted so much that the update can be ignored without a noticeable impact (The exception to this would be when λ is close to or equal to 1). In a network routing context, this approximation to the $TD(\lambda)$ approach corresponds to sending Q-value update packets out to neighbor nodes for a fixed number of hops, whenever

an update is made to Q-Routing information due to the sending of a normal data packet.

5 Simulation Environment

We designed a simulator in which we could compare the behavior of $TD(\lambda)$ to several of the Q-Routing algorithms described in prior research [2]. The simulator accepts as input a file containing a description of an arbitrary-sized network, including the number of the nodes, the links connecting pairs of nodes, and the cost associated with each link.

The simulator was designed to be similar to those used in prior research [2]. Each network node has an unbounded FIFO queue. At every time step, each node takes the packet in the front of its queue, and according to the destination of that packet and the current routing algorithm and Q-value estimates, it forwards the packet to one of its neighbors. Also at each time step, one or more data packets are introduced into the network, with a random source node, a random destination node, and a random first hop chosen from the neighbors of the source node. The number of data packets introduced per time step is called the *network load*. Smaller packets with Q-value and C-value update information are sent in the same time step as the data packets to which they are related. This simplification is based on the assumption that these update packets will be small compared to normal data packets and will not have a large effect on network traffic. It is also assumed that when a router receives such an update packet, it will process and respond to it immediately instead of putting the response packet or packets (if any) in its queue.

5.1 Validation

We used Dijkstra's algorithm, as described in [7], to provide validation for our approach.

Dijkstra's algorithm, when run on the whole network, terminates in $O(n^3)$ time, where n is the number of nodes in the network. In an actual network, however, each node is only concerned with the costs and next hops for routes beginning at that node, meaning that each node only has a task of $O(n^2)$ complexity. This is still fairly expensive when n is large. This fact, combined with the need for each node know the state of every link in the network in order to run it, makes Dijkstra's algorithm impractical for use with very large networks. However, it does guarantee to return the shortest paths between all pairs of nodes, and this makes it a good tool for comparison with other routing algorithms such as the one proposed in this paper.

The routing algorithms tested in this paper each maintain a set of Q-values, defined for each triple of nodes (r, h, d) such that h is an immediate neighbor of r, representing the estimated total cost when sending a packet from a router r out on a particular outgoing link for which the next hop is h, given the packet's destination node d. These Q-values can be used to define a policy, which is a function mapping a pair of nodes (a router r and destination d) to an outgoing link h. This is performed by simply choosing the outgoing link (labeled by its next hop h) at a given router r for which the Q-value associated with that link and a given destination node d, $Q_r(h, d)$, is a minimum. Or more formally, denoting the set of all the nodes in the network as N, the policy $\pi : N \times N \rightarrow N$ is defined as:

$$\pi(r, d) = \arg \min_{h \in N(r)} Q_r(h, d) \qquad (4)$$

Dijkstra's algorithm also defines such a policy: it gives the next hop that should be taken

from any given router to any given destination node, to achieve the shortest path. It is the comparison with this Dijkstra policy that we use to define the performance metric. For each node pair (r, d) for which $r \neq d$, we compare the policy given from the Q-values with the Dijkstra policy. The percentage of times that the two policies matched is then recorded.

It is also possible to define a metric that would measure how accurately the $TD(\lambda)$ algorithm approximated the true costs of forwarding a packet on each link. This can be performed by finding the global difference between the Q-value estimates and the Dijkstra costs for each node pair (r, d). A total sum-squared error metric can be calculated by summing the square of these differences. The costs calculated by Dijkstra's algorithm are not used for validation of $TD(\lambda)$, since the only influence that the Q-values have on the actual routing decisions made by nodes in the network was the choice of which outgoing link to use. The policy generated from the Q-values in the performance metric is used, and not the Q-values themselves.

In some of the topologies used in this paper (described below), there are cases where multiple optimal paths exist between a pair of nodes. In these cases, a concern may arise about detecting the convergence of $TD(\lambda)$ through simple comparison of the generated Q-Routing policy with the Dijkstra solution. This problem was averted by making both our implementation of Dijkstra's algorithm and the policy generator favor lower-numbered nodes when choosing a next hop for a path. Thus, when there are multiple optimal solutions, there is a consistent method for tie breaking that allows the comparison to be valid.

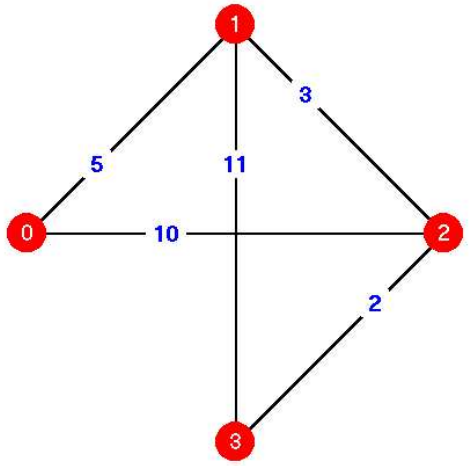

Figure 1: A simple network topology of four nodes

5.2 Network Topologies

Ten different network topologies were created. The first and simplest topology, shown in Figure 1, was used in prior research to demonstrate how Dijkstra's algorithm works [7]. The second topology, shown in Figure 2, was used by Tseng and Garzon [9]. The third topology is nearly identical to Figure 2, except that the links connected to the central node have increased costs to simulate congestion in the network [9]. (Menke also used these two topologies [4].) The fourth topology, called the "6x6 Irregular Grid," came from the Kumar and Miikkulainen paper [2]. It has 36 nodes, with all links having unit cost, as shown in Figure 3.

The next three topologies are the hypercubes of dimensionality 3, 4 and 5. These topologies have 8, 16 and 32 nodes, respectively, and there is a direct link with unit cost between any two nodes for which the numerical identifiers of the two nodes have binary representations that differ in exactly one bit (for example, nodes 5 and 7 are directly linked because their binary representations, 101 and 111, differ in exactly one bit).

To generate the final three topologies, we used the Waxman model, as explained by

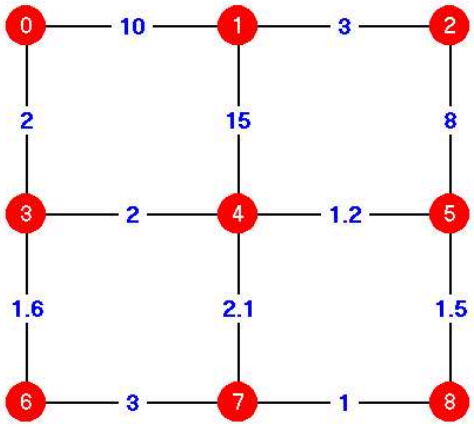

Figure 2: A network topology of nine nodes

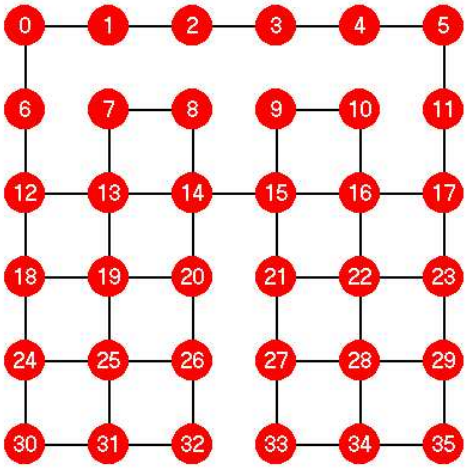

Figure 3: The "6x6 Irregular Grid" topology

Tseng [9]. This model states that the probability of an edge existing between two nodes u and v should be

$$P(u,v) = \alpha e^{-d/\beta L}, \alpha, \beta \in (0,1] \qquad (5)$$

where d is the Euclidean distance between u and v, and L is the maximum distance between any two nodes in the network. The parameters α and β can be adjusted to vary the richness of the network connections; increasing either will increase the total number of links in the network. We created 4x4, 5x5 and 6x6 topologies using this model, using an α and β that produced approximately $(n \log_2 n)/2$ links, where n is the number of nodes. (The hypercube networks have exactly that number of links, since there is a link to each node for every dimension of the hypercube, and there are $\log_2 n$ dimensions.) We found that if there were fewer than $(n \log_2 n)/2$ links, the topologies generated tended to be disconnected. To each link in these Waxman topologies, we assigned, as a cost, the Euclidean distance between the two nodes it connected.

Calvert [1] points out that when using the Waxman method, there is no guarantee that every node will be reachable from every other node. To ensure that the topologies created using this and the other methods were connected, we ran Dijkstra's algorithm on each network topology. A distance value of $infinity$ between any two nodes signaled a partitioned network, and in those cases the topology was discarded and regenerated, after adjusting any necessary parameters. Table 1 summarizes the important information about each of the topologies.

Table 1: Topologies used in the experiments

Topology	*Nodes*	*Links*
Simple	4	5
Tseng #1	9	12
Tseng #2	9	12
6x6 Irregular Grid	36	50
3-D hypercube	8	12
4-D hypercube	16	32
5-D hypercube	32	80
Waxman 4x4	16	29
Waxman 5x5	25	59
Waxman 6x6	36	95

6 Experiments

Simulations for Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing were implemented in order to produce comparative

results. Six variations on the $TD(\lambda)$ extension to Q-Routing were also explored. The six variations differed in the number of "generations" of neighbors that received an update packet for each data packet. For example, for the two-generation case, an update packet was sent from the receiving router to the sending router, and the sending router sent a similar update packet (based on its updated Q-value estimate) to each of its neighbors. The strength of the update for the second-generation neighbors was discounted by the parameter λ_{TD} (not to be confused with the λ_C used in CQ-Routing). For a third generation, the discount would be $(\lambda_{TD})^2$, etc. TD-Routing was implemented for 2, 3 and 4 generations. These are labeled in Figure 4 as *TD2-Routing*, *TD3-Routing* and *TD4-Routing*, respectively. The idea of dual-reinforcement updates from DRQ-Routing was implemented with three variants of TD-Routing that sent both forward- and backward-exploration updates. These are labeled *DRTD2-Routing*, *DRTD3-Routing* and *DRTD4-Routing*.

Parameter values specified in prior research [2] were used for all experiments: $\eta_f = 0.85$; $\eta_b = 0.95$; $\lambda_C = 0.95$ (CQ-Routing) and $\lambda_C = 0.90$ (CDRQ-Routing). A value of 0.7 was used for λ_{TD}. CDRQ-Routing and each of the six TD-Routing algorithms were run on all ten topologies, with a network load of 3.0 packets/time step. For each experiment, the simulation was executed ten times. Each execution counted the number of time steps it took to match the Dijkstra policy by 95% or better. The average of the ten runs is reported. Figures 4 and 5 show the results.

7 Results

In a majority of the topologies (6 out of 10), the simplest version of TD-Routing, (labeled TD2-Routing), outperformed CDRQ-Routing. However, in the four cases that TD2-Routing was worse, it was significantly worse. This suggests that in these cases, performance can be improved more by the use of C-values than by the propagation of forward exploration through a second generation. Interestingly, those four cases are the three hypercubes and the 6x6 irregular grid—the topologies for which all links have unit cost! In these networks, there is no difference in cost between any two direct links, and this seems to make a two-generation propagation of information less helpful than a confidence value, which can implicitly carry information from several previous hops as learning progresses.

The other variants of TD-Routing consistently outperformed CDRQ-Routing, with the exception of DRTD2-Routing, in the same four cases as mentioned above. The fastest convergence comes with the DRTD4-Routing algorithm. This shows that a greater amount of communication between nodes causes the routing algorithm to converge in fewer iterations. But we should be very cautious about asserting that this is the best routing algorithm to use in an actual network, because of our simplifying assumption that the effect of update packets on network traffic is negligible. These update packets would indeed be very small, requiring only the identity of two network nodes (next hop and destination), a Q-value and a generation number. These could have a smaller header and work at a higher network layer than normal data packets, since they would only be exchanged between routers. But as the number of generations is increased, there is exponential growth in the number of these packets that would need to be sent. In the topologies we used, each node has approximately $log_2 n$ immediate neighbors, where n is the number of nodes in the network. If the number of generations is g, then TDg-Routing sends $(log_2 n)^{g-1}$ update packets for each

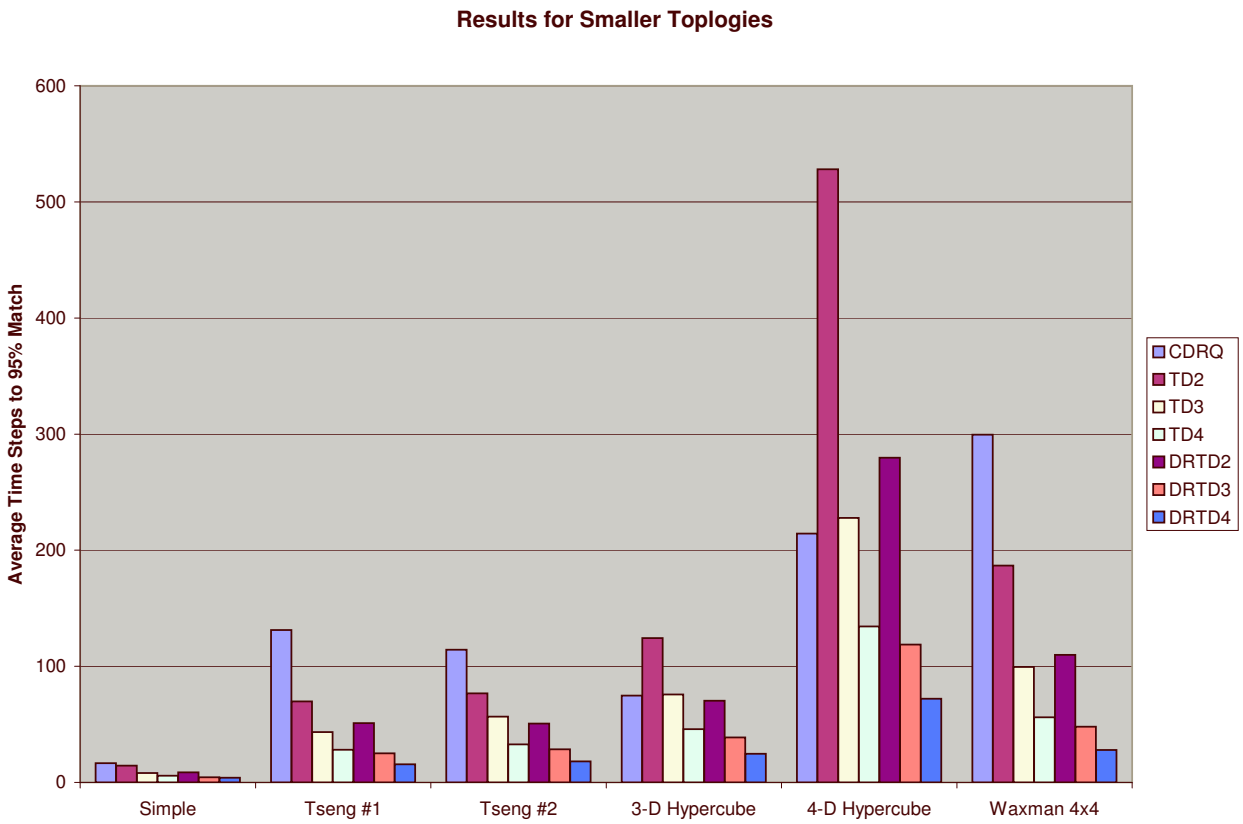

Figure 4: Comparison of routing algorithms for the six smaller topologies

hop of a normal data packet, and DRTDg-Routing generates $2(log_2 n)^{g-1}$ update packets per hop. The relative sizes of an average data packet and an update packet could be used to make an informed decision about the actual effects of the update overhead.

8 Conclusion

In this paper an algorithm called TD-Routing is presented, which makes use of the machine-learning algorithm $TD(\lambda)$ and applies it to the problem of network routing. Several variants of this algorithm are compared with CDRQ-Routing, and TD-Routing is shown to be competitive. We have demonstrated that an increase in communication between routers leads to faster convergence in terms of simulation time steps, though the analogy to actual networks becomes weaker as the simulated communication increases. Further analysis, including knowledge of average data packet size and its comparison to update packet size would be important before attempting to implement this algorithm on a physical network.

9 Future Work

One significant improvement to this work would be to implement it on a more universally accepted network simulator, such as *ns* [10]. This would provide a more solid validation of these results. Using *ns* would provide a better simulation of scheduling and queues at each node, and allow the removal of some of the assumptions used in the experiments. For example, the effect of the Q-value update packets on network traffic could be simulated, rather than assuming it to be negligible.

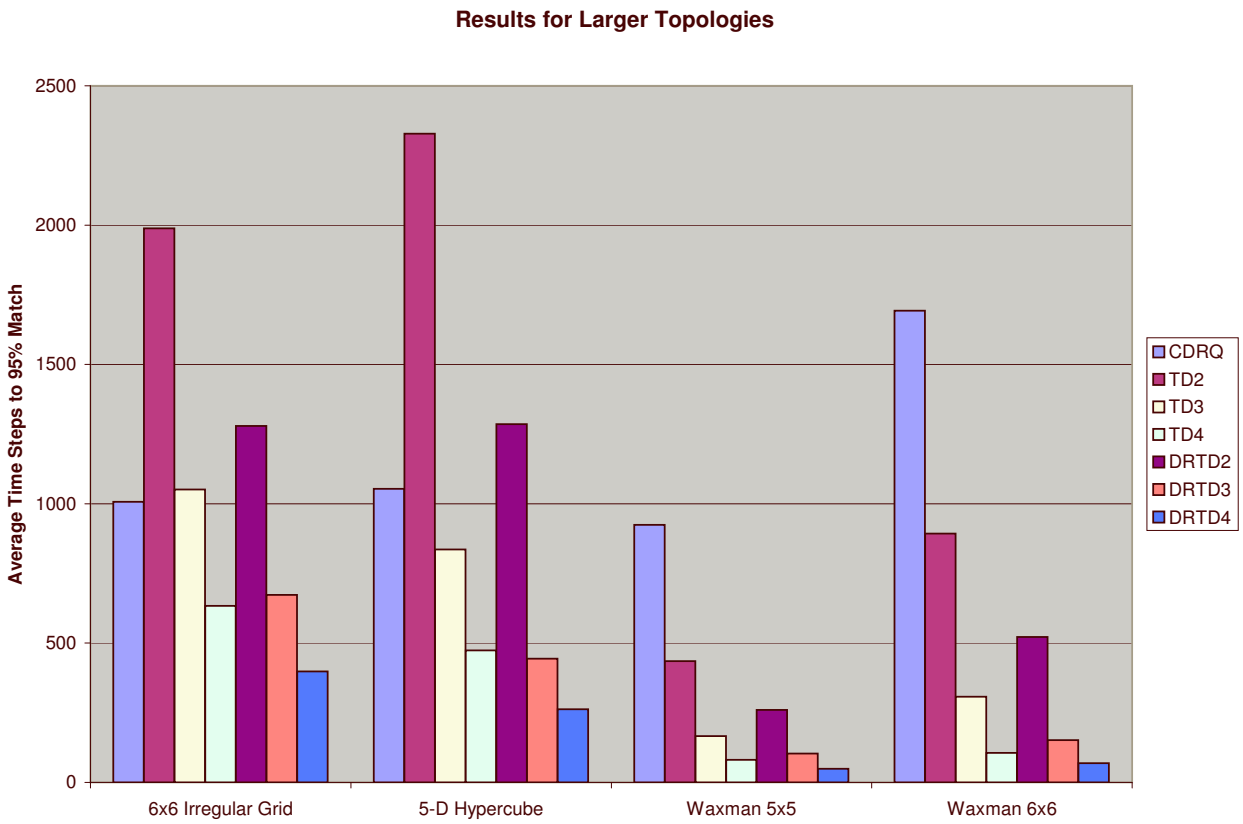

Figure 5: Comparison of routing algorithms for the four larger topologies

Another possible improvement would be to consider queue lengths at intermediate network nodes in calculating Q-values. The queue length could be combined in some way with the link cost to determine a combined cost metric of each direct link. This could be simulated using a simulator such as *ns*.

Yet another improvement would be to use a more robust internetwork modeling scheme for topology generation, like the three-level hierarchy proposed in [1]. This would give greater insight into the applicability of this routing algorithm in larger routing domains, such as company LANs, WANs or even the Internet as a whole.

Finally, it might be interesting to measure the accuracy of TD-Routing's estimations of cost. Instead of simply comparing the Dijkstra policy with that produced by choosing the minimum Q-value for a packet, one could calculate a total sum-squared error for the routing costs estimated by the algorithm versus those calculated with Dijkstra, and measure how long it takes for that error to diminish to an acceptable level.

References

[1] Ken Calvert, Matt Doar, and Ellen Zegura. Modeling internet topology. *IEEE Communications Magazine*, Jun 1997.

[2] Shailesh Kumar and Risto Miikkulainen. Confidence based dual reinforcement Q-Routing: An adaptive online network routing algorithm. 16th International Joint Conference on Artificial Intelligence, 1999.

[3] G. Malkin. RIP version 2. RFC 2453, Nov 1998.

[4] Joshua Menke. Distributed neural network routing. Technical Report NCL-2001-1, Department of Computer Science, Brigham Young University, Provo, UT, 2001.

[5] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Companies, Inc., Boston, Massachusetts, 1997.

[6] J. Moy. OSPF version 2. RFC 2328, Apr 1998.

[7] Larry Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[8] Peter Stone. *Layered Learning in Multi-Agent Systems.* PhD thesis, Carnegie Mellon University School of Computer Science, 1997.

[9] Chiu-Che Tseng and Max Garzon. Hybrid distributed adaptive neural router. Proceedings of the ANNIE, 1998.

[10] The VINT Project. *The ns Manual*, Mar 2003.