Distributed Value Functions

Jeff Schneider
Carnegie Mellon University
Pittsburgh, PA 15213
Jeff.Schneider@cs.cmu.edu

Weng-Keen Wong
Carnegie Mellon University
Pittsburgh, PA 15213
wkw@cs.cmu.edu

Andrew Moore
Carnegie Mellon University
Pittsburgh, PA 15213
awm@cs.cmu.edu

Martin Riedmiller
University of Karlsruhe
D-76128 Karlsruhe, FRG
riedml@ira.uka.de

Abstract

Many interesting problems, such as power grids, network switches, and traffic flow, that are candidates for solving with reinforcement learning (RL), also have properties that make distributed solutions desirable. We propose an algorithm for distributed reinforcement learning based on distributing the representation of the value function across nodes. Each node in the system only has the ability to sense state locally, choose actions locally, and receive reward locally (the goal of the system is to maximize the sum of the rewards over all nodes and over all time). However each node is allowed to give its neighbors the current estimate of its value function for the states it passes through. We present a value function learning rule, using that information, that allows each node to learn a value function that is an estimate of a weighted sum of future rewards for all the nodes in the network. With this representation, each node can choose actions to improve the performance of the overall system.

We demonstrate our algorithm on the distributed control of a simulated power grid. We compare it against other methods including: use of a global reward signal, nodes that act locally with no communication, and nodes that share rewards (but not value function) information with each other. Our results show that the distributed value function algorithm outperforms the others, and we conclude with an analysis of what problems are best suited for distributed value functions and the new research directions opened up by this work.

1 Introduction

Many interesting problems that are candidates for solving with Reinforcement Learning (RL), also have properties that make distributed solutions desirable. Whenever the state and/or action space is large, a distributed approach to performing the computation is desirable because it makes computational speedups from coarse-grain parallelism possible. In many systems access to sensors and actuators is inherently distributed, thus making a distributed solution method an attractive alternative to implementing a global high bandwidth communication network. Potential applications include control of power grids (or any other distribution of a resource such as water, gas, etc.), automobile traffic control, electronic network routing, and control of robot teams. Fig. 1 provides an intuitive sense of the kinds of applications we envision for our algorithms.

1.1 Related Work

Reinforcement learning and dynamic programming-based methods for optimal control are fairly well understood [14, 4]. By contrast distributed reinforcement learning is a much less mature concept because it is harder to formulate and analyze theoretically. One approach is to assume restricted interaction between the nodes. For example, if the transfer function is such that the dynamics of the local state depend only on the local state and the local control, and the value function can be trivially split into components depending only on local states and controls, then the solution to optimally controlling the entire system is just the composite solution created by solving the individual parts. Of course making that assumption eliminates almost all interesting problems.

A completely segregated approach can be taken with the additional allowance that the choice of controls by one node may restrict the space of controls available to another [12]. The resulting algorithm requires some global information, but does provide a provably optimal solution. Distributed elevator control has been addressed with reinforcement learning [7] . That work has shared, global state and cost information, but the agents act independently. Other work provides a similar framework, but in the context of competing rather than cooperating agents [9]. Homogeneous agents that seek to learn similar value functions can improve learning speed and performance by exchanging learned policies and sensing [15] .

Weiss proposes a "bucket brigade" scheme for credit assignment among cooperating reinforcement learning agents[17]. Rather than each agent acting locally and independently, they communicate globally in an auction to arbitrate which subset of agents will be allowed to "take over the global controls" in each state. Similarly, there are approaches where different behaviors are learned for a system and they are combined with the idea of choosing actions that will accomplish all goals [1]. Similarly, economic models can form the basis of credit assignment methods [3].

Packet routing is a domain for which completely distributed approaches have been taken [5, 6, 13]. In this problem, each node must make a decision about which neighboring node to rout each packet to. The global state space is the list of all packets in the system, their current locations, and their destinations. Each node, however, only has that information for the packets in its own queue. The global cost function is the the average length of time it takes to send packets in the network, but each node only stores and computes the length of time it will take to send packets from itself. As part of the routing algorithm, nodes periodically ask their neighbors how long they predict it will take them to deliver a packet if it is routed through them. Empirical studies have been promising but have also shown problems with ringing, instability, and adapting to changing load conditions.

2 Distributed Value Functions

In the description of our method, we refer to the sample problem diagrammed in fig. 1. The job of the distribution network is to minimize a cost function based on providing resources to the customers. The figure shows the nodes in a chain, but we make no assumptions on the connectivity of the graph. This problem could easily be written as a global reinforcement learning problem with, states X that represent flow rates and resource levels, actions A that represent changes in flow rates, and rewards (or penalties) R that represent satisfaction of customer requests and costs incurred in doing so. A distributed formulation of the problem makes the following changes:

- **State space.** Each node, i, observes some of the flows and resource levels (presumably those it can sense locally), which are represented by X_i. It may be the case that some state variables are observable by more than one, or even all nodes.
- **Action space.** Each node, i, has the ability to choose some of the flow control actions, called A_i. We assume that each action variable from the global formulation is chosen by exactly one node in the distributed formulation.
- **Rewards.** Each node, i, receives a reward presumably based on the satisfaction of its own customers, called R_i. We assume that the sum of the reward functions in the distributed case is equal to the global reward function in the global case ($R = \sum_i R_i$), and that the overall goal in the distributed formulation is to optimize that sum over time.

2.1 Global RL Formulation

The usual approach to solving the resource distribution problem with globally available states, actions, and rewards leads to a Bellman equation:

$$V(x) = \max_{a \in A}(R(x,a) + \gamma \sum_{x' \in X} p(x'|a,x)V(x')) \quad (1)$$

where V is the value function and γ is a temporal discount factor. This equation may be solved using many methods including many reinforcement learning variants. The result is the optimal value function which also represents the expected discounted sum of future rewards under the optimal policy:

$$V^*(x) = E(\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)) \quad (2)$$

Once obtained, the value function may be used to generate optimal control decisions by finding the action, a, that gives the maximum in eq. 1.

2.2 A Distributed RL Formulation

The main purpose of this paper is to introduce and evaluate four algorithms for distributed RL (DRL). In this section we describe each of them. The first two are common in the literature in various forms, while the last two are new.

We start by assuming that each action variable is chosen by a particular node and the reward function is broken up into a sum where each term is received at one node. For simplicity we continue to assume the state is globally observable. The question of local state

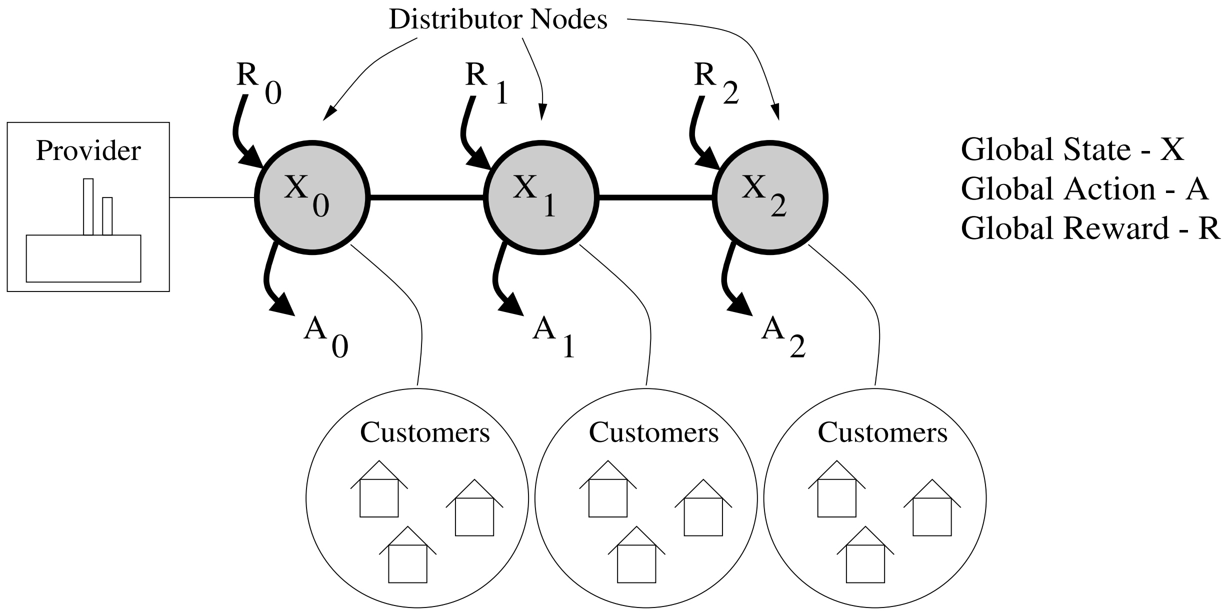

Figure 1: A sample resource distribution network. The shaded circles have a dual meaning in this figure. They represent 1) physical distribution centers in a network with resource providers and customers, and 2) logical nodes in the distributed RL formulation of the problem. They are the points at which state is sensed, actions are taken, and rewards are received.

observability will be addressed later. Our approach is to define a value function for each node, $V_i(x)$, that will be computed and used to choose that node's actions.

2.2.1 Global reward DRL

A first step toward distributed RL (DRL) is to have each node sense, act, and learn locally, but base learning and control decisions on a global reward signal. That results in a Bellman equation of the form:

$$V_i(x) = \max_{a \in A_i}(R(X,A) + \gamma \sum_{x' \in X} p(x'|a,x)V(x')) \quad (3)$$

where A_i represents only the action variables available for node i to choose. The unknown action choices of the other nodes will affect the probabilities in the transfer function, $p()$. X and A refer to the combined global state and action spaces, and R is the single, global reward that is distributed to all nodes. Unfortunately, the probability on the right-hand side of eq. 3 isn't well defined for node i. It depends on what policy the neighboring nodes are following and would be very hard to estimate. An alternative is the Q-learning formulation, which will be described later. Without addressing the issue of how/if a simultaneous solution to the set of these equations can be found, we observe that once it is found, the value function retains its meaning as the expected discounted sum of future rewards under the resulting policy:

$$V_i(x) = E(\sum_{t=0}^{\infty} \gamma^t R(X_t, A_t)) \quad (4)$$

Each node is attempting to learn the same value function ($V_i^* = V^*$), but is hindered by its lack of knowledge about what actions the other nodes will take (and by having only local state available, which will be addressed later). This method is not fully distributed, though, since it depends on the global broadcast of a reward signal.

2.2.2 Local DRL

A naive, but fully distributed, formulation is one where each node acts only to optimize its own rewards and does not communicate with its neighbors. That results in a Bellman-like equation of the form:

$$V_i(x) = \max_{a \in A_i}(R_i(x,a) + \gamma \sum_{x' \in X} p(x'|a,x)V(x')) \quad (5)$$

Rather than assuming a global reward function that is computed (probably based on information transmitted in from each node's sensors to a central location) and broadcast to all nodes, we use a local signal available at each node. It is assumed that this reward function can be computed directly from a node's own local sensors and actions. Again, without addressing the issue of how/if a simultaneous solution to the set of these equations can be found, we observe that the value function retains its meaning as the expected discounted sum of future rewards under the resulting policy:

$$V_i(x) = E(\sum_{t=0}^{\infty} \gamma^t R_i(x_t, a_t)) \tag{6}$$

The catch is that the composite policy is no longer guaranteed to be optimal for the overall system. It is easy to see an example of this in terms of the example in fig. 1. Suppose the provider does not offer enough to satisfy all customers and the penalty for depriving the node 2 customers is greater than that for depriving the node 0 customers. If the resource must flow through the distribution nodes from 0 to 1 to 2, node 0 will act to satisfy its own customers even at the greater expense of starving the others further down. Note that the result is bad even if the penalties are equal, but quadratic.

2.2.3 Distributed reward DRL

One way to have nodes communicate and act to help each other out is to have neighbors exchange information about the immediate rewards they receive. If each considers a weighted average of its local rewards and those of its neighbors we get the following equation for each node's value function:

$$V_i(x) = \max_{a \in A_i}(\sum_j f(i,j) R_j(x, a_j) + \gamma \sum_{x' \in X} p(x'|a,x) V_i(x')) \tag{7}$$

where $f(i,j)$ is a weighting function that determines how strongly node i will weight the immediate rewards of node j in its average. It is assumed that $f(i,j)$ will be zero for all pairs of nodes that are not neighbors, and that $f(i,i)$ will be non-zero. The equation contains the variable a_j, which is the action to be chosen by node j. We do not assume that node i will get to see this action choice, but that node j will report the resulting reward, R_j.

We might hope that this will result in less greedy behavior because if everyone cares about themselves and their neighbors, then good will might propagate throughout the network. If we consider a system with discrete states and the value function is represented in a table, we can see that a solution takes the form:

$$V_i(x) = E(\sum_{t=0}^{\infty} \gamma^t \sum_j f(i,j) R_j(x_t, a_{j,t})) \tag{8}$$

A node that uses this value function to make its policy decisions will act to improve the long term rewards of its neighbors and will make tradeoffs according to the weights given by $f(i,j)$. Unfortunately, it will still not help its non-neighbors, because, as we see in eq. 8, node i still does not have any component of rewards other than its immediate neighbors. Reusing the previous example, node 0 will still act to starve node 2's customers.

2.2.4 Distributed value function DRL

Nodes can act to assist non-neighbors if the value function equation is updated to have the nodes exchange information not about their immediate rewards, but about their value functions:

$$V_i(x) = \max_{a \in A_i}(R_i(x,a) + \gamma \sum_j f(i,j) \sum_{x' \in X} p(x'|a,x) V_j(x')) \tag{9}$$

At this point we have changed the Bellman equation to the point where we can not say much about what properties a solution would have. In a deterministic system such as the example below, we can write a solution in the following form:

$$V_i(x) = \sum_j f'(i,j) R_j(x_t, a_{j,t}) \tag{10}$$

One thing to note is that the weights in the average for eq. 10, $f'(i,j)$, are no longer equal to the weights in eq. 9 as they were previously between eqs. 7 and 8. Pairs of nodes that have zero weight in the former equation, may receive non-zero weights in the latter. In that case, nodes can act to help their non-neighbors as well.

In order to see how weighting of non-neighbors occurs consider a system organized as a 3 node chain where each node uses a simple average of itself and its neighbors. We remove the max from eq. 9 by assuming the nodes already implement a fixed, optimal policy. We further simplify by assuming we have a system with only one state, thereby making it stateless (we'll fix that later). The three value function equations (with the sums in eq. 9 expanded) are:

$$\begin{aligned} V_0(x) &= R_0(x,a) + \gamma(0.50 V_0(x) + 0.50 V_1(x) + 0.00 V_2(x)) \quad (11)\\ V_1(x) &= R_1(x,a) + \gamma(0.33 V_0(x) + 0.33 V_1(x) + 0.33 V_2(x))\\ V_2(x) &= R_2(x,a) + \gamma(0.00 V_0(x) + 0.50 V_1(x) + 0.50 V_2(x)) \end{aligned}$$

Because there is only one state, the transition probabilities and the expected value on the right hand side of eq. 9 are reduced to a single term. The solution to these equations (with $\gamma = 0.9$) is:

$$\begin{aligned} V_0(x) &= 4.0R_0(x,a) + 3.9R_1(x,a) + 2.1R_2(x,a) \quad (12) \\ V_1(x) &= 2.6R_0(x,a) + 4.8R_1(x,a) + 2.6R_2(x,a) \\ V_2(x) &= 2.1R_0(x,a) + 3.9R_1(x,a) + 4.0R_2(x,a) \end{aligned}$$

These equations match eq. 10 with different coefficients for $f'(i,j)$ as mentioned before. Most importantly, we see that the value function for node 0 now includes a positive weight on the rewards received by node 2 even though those two nodes never communicate directly, which was the original intent of this algorithm. In the real case where there are numerous states, eq. 12 becomes a matrix of equations, one for each node-state pair, and the weighted sum of next states returns to each equation. The solution to that set of equations is still just a matter of linear algebra, but the resulting weights on the rewards depend on the transition probabilities between the states.

With this formulation we have effectively created a distributed representation of the value functions over the nodes. Each node is trying to learn a weighted sum of its own expected future rewards and those of the other nodes in the network. If we have chosen our weighting function well (more on this later), we could sum the value functions over all the nodes and the result would be an expected future weighted sum of rewards over all the nodes just as in the global RL approach. However, this is accomplished with access only to local rewards and communication of value function information only between immediate neighbors.

2.3 Local state and the use of Q-Learning

In the previous section, we based our discussions on the use of global state information available at all nodes. In order to have a fully distributed method, it is necessary to also have local state. Intuitively, this is easiest to think about in terms of function approximation. In the case of traditional, global reinforcement learning work, the use of function approximators to represent the value function has become common. Its main purpose is to handle state spaces that are too large to be stored in tabular form, and to speed up learning through generalization in large state spaces. In some cases, the convergence proofs for reinforcement learning have been extended to the use of function approximation [8, 2]. In the case of local state for distributed RL, we can think of it exactly as value function approximation where each node has chosen differing sets of features (their own locally observable state) to be used by their function approximators. Based on that intuition, we speculate that these convergence results may extend to our algorithms, but answering that question is future work. Additionally, the question of how good the chosen function approximator is for a particular problem remains important.

There is a practical problem introduced by the use of local state variables. All of the equations in our derivation have been written strictly in terms of the value function. At first glance, it might seem that this will lead to the use of a kind of value iteration coupled with the learning of system models to solve the problem ([10, 11], for example). This can't be done easily, though, because the use of local state means neighbors can't use state as a common language with which to communicate. It isn't possible for one node to ask another *"If I am in state x and choose action a, what state will that put you in, and what action will you choose?"* because neither of them has any representation of the other's states and actions.

Despite this problem, reward and value functions are a universal language. Constructing a Q-learning [16] algorithm to solve the problem only requires nodes to communicate in these terms. A Q-learning rule that each node can implement to learn the value function in eq. 9 is:

$$\begin{aligned} Q_i(x_i,a_i) &\leftarrow (1-\alpha)Q_i(x_i,a_i) + \qquad (13) \\ &\quad\; \alpha(R_i(x_i,a_i) + \gamma \sum_j f(i,j)V_j(x'_j)) \\ V_i(x_i) &\leftarrow \max_{a \in A_i} Q_i(x_i,a) \end{aligned}$$

Each node performs the updates specified by these equations online. Because the learning is done online (it can also be done off-line using a trace) as the system actually passes through various states, there is no need for neighbors to specify to each other what state or action they've been in or taken. It is only necessary for them to transmit their current estimated value of the state they land in at each iteration.

3 Experimental Results

We tested our algorithm on a simulated power grid problem. We use a D.C. formulation because it simplifies the simulation and keeps the number of state variables smaller. Simulation of a real A.C. power grid requires the use of complex, rather than real, variables. Our simulation uses variable resistors as a means of controlling power flow. Historically, this hasn't been possible in real power grids, but new devices are being developed that will allow it. Our system has the following components (again, refer to fig. 1):

Providers. These are the power generation facilities and are treated as fixed voltage sources.

Customers. These represent cities, which have a desired voltage (or a desired power requirement based on a fixed resistance load).

Distributors. The distributors represent both physical entities and the *nodes* of our distributed RL formulation. This is where the intelligent control decisions must be made. The variables of the control problem are as follows:

Local actions. Each node makes a choice for each power line connected to it. It can request that the resistance on that line be doubled, halved, or left the same. Lines that go between distributors and providers, or distributors and cities, are controlled by the distributor. Any line that goes between two distributor nodes will get two requests on how its resistance should change. The arbitration scheme shown below determines the actual change in resistance. The resistors have 6 possible values and attempts to increase them beyond their maximum value, or decrease them below their minimum value result in them staying the same.

Node 1 request	Node 2 request: Halve	Same	Double
Halve	Halve	Halve	Same
Same	Halve	Same	Double
Double	Same	Double	Double

Local state. A distributor receives state information for each line it has depending on what its connected to:

distributor-distributor: 1) Whether the neighboring voltage is higher or not (2 states), 2) whether the neighboring voltage increased, lowered, or remained the same from the last iteration (3 states), 3) whether the resistor is at its minimum, maximum, or in between (3 states). There are 18 possible states total.

distributor-city: The same as for distributors except that item 2 is whether or not the city needs more voltage. There are 12 possible states total.

distributor-provider: Only items 1 and 3. There are 6 possible states total.

The complete local state space for a node is the cross product of the states for each line it is attached to. For example, a node that connects to one city, one provider, and one other node will sense one of 1296 possible states.

Local reward. The local reward received by each node is based solely on the satisfaction of its local customers (the city it is directly attached to). If the voltage available to them is below the desired level, the node receives a penalty equal to the difference between the two, otherwise the reward signal is zero. Note that this can be a problem when some distributors are connected only to other distributors, but not to any customers. They will have no local reward to base their decisions and learning on. We will see the effects of this in the experiments.

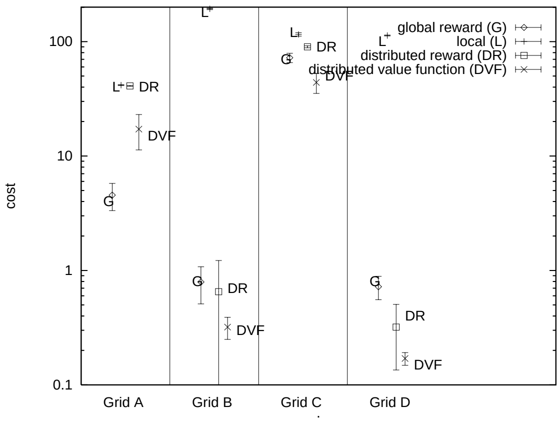

Figure 3: Graph of results on a log scale showing cumulative cost over 60000 iterations in policy evaluation mode.

We ran each distributed RL algorithm on each of the power grids shown in fig. 2. Note that it was not practical to run the traditional global RL method because even the smallest grid has hundreds of millions of states. For each run there were 60000 learning steps, followed by an evaluation phase which did not learn and followed its policy 95% of the time and chose randomly the other 5%. We used $\gamma = 0.95$. $\alpha = 0.1$ initially and is decayed. The exploration rate is started at 1.0 and stays there for the first 30000 steps, then decays. For the distributed reward and distributed value function cases, $f(i, j)$ is chosen to compute a simple average of the node and its neighbors. During the trials, an initial random setting of all the resistors is chosen. The system operates for 300 steps, and then the resistors are changed to new random settings. This loop continues throughout the learning and evaluation phases.

Fig. 3 and table 3 summarize the experimental results. They show average cost (negative reward) per step during the evaluation phase along with their confidence intervals. These are averages over 10 trials and each took about 45 minutes on a 400MHz Pentium II.

The results show distributed value function performing best in every case except grid A. On grid A, global reward is best. We speculate that this is because there are only two cities, and thus only two sources of re-

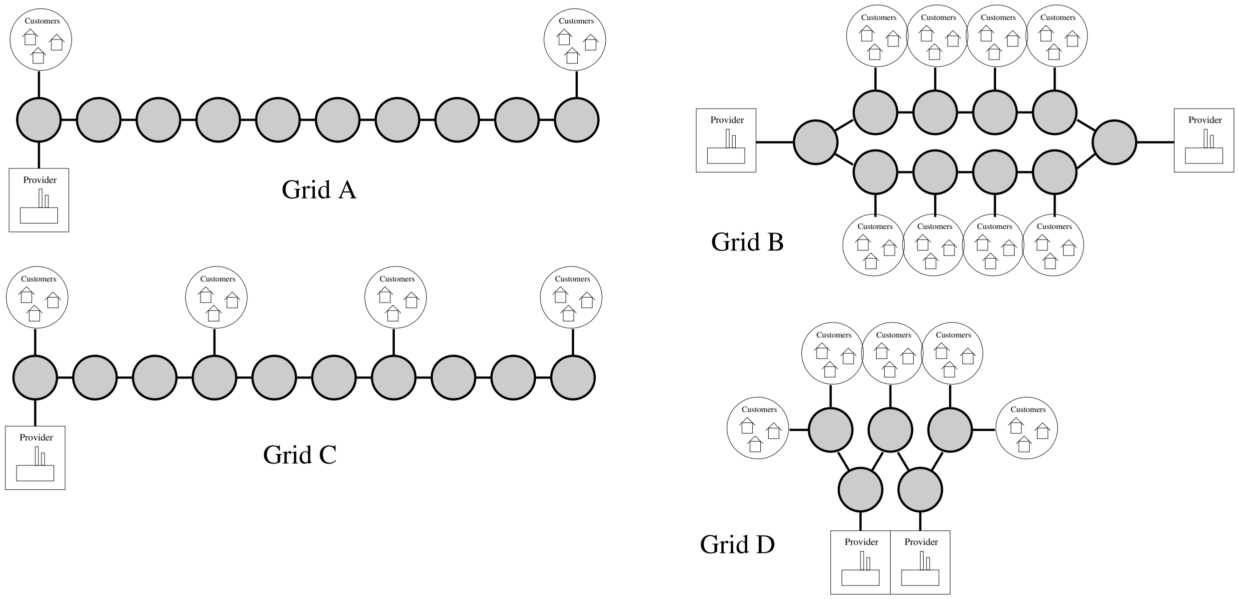

Figure 2: The 4 power grids used in the experiments.

Algorithm	**Grid A**			**Grid B**		
	Rank	Avg cost	95% C.I.	Rank	Avg cost	95% C.I.
Global reward	1	4.55	±1.22	3	0.79	±0.28
Local	4	41.67	±0.39	4	193.33	±3.22
Dist. reward	3	41.00	±0.30	2	0.65	±0.57
Dist. value fn	2	17.17	±5.87	1	0.32	±0.07
	Grid C			**Grid D**		
Global reward	2	72.33	±6.82	3	0.72	±0.17
Local	4	115.50	±3.97	4	113.00	±1.07
Dist. reward	3	90.00	±1.78	2	0.32	±0.19
Dist. value fn	1	44.00	±8.75	1	0.17	±0.02

Table 1: Average reward per step in policy evaluation mode for the 4 DRL algorithms on 4 different power grids.

ward. Lumping them together into a global reward signal doesn't hide as much information as in the other cases. Local DRL performs poorly everywhere, which shows that locally greedy approaches aren't sufficient to solve this problem. The fact that distributed reward performs reasonably, but not as well as distributed value function indicates that it is necessary to consider more than the rewards of immediate neighbors. Local DRL and distributed reward both suffer when there are nodes without cities attached (or in the latter case when no neighboring distributors have cities attached), because they have no source of local reward.

4 Conclusions and Future Work

In summary, we have presented a general algorithm for distributed reinforcement learning that allows nodes to work toward the benefit of other nodes across the system, while only requiring communication with immediate neighbors. Our results show this algorithm performing well on a power grid control application.

A remaining question is how to choose $f(i,j)$ in order to get the best result. In the derivation, we observed that the different algorithms result in similar looking solutions (eqs. 4, 6, 8 and 10). The only difference is in the effective weighting function $f'(i,j)$. Local and global reward offer one specific function each, while distributed reward and distributed value function offer a class of functions depending on the chosen $f(i,j)$. Distributed value function can imitate global reward (using a fully connected graph with uniform weights) and local (zero weights on all neighbors), but it can not imitate distributed reward. Although we did not find any here, there may be cases where distributed reward performs best.

Another question is whether, and under what conditions, the solution to eq. 9 yields an optimal or near-optimal policy for the entire system. It is obvious that it does not for some choices of $f(i,j)$. It might if the graph of nodes is fully connected and all nodes weight all others equally (the global reward case), but this is uninteresting practically because it requires full communication between all nodes. Possible future work is to put bounds on the suboptimality for specific weightings.

An interesting extension to this method is the case where the neighbor relation and the weighting function, $f(i,j)$, change dynamically. In a simple case the structure of the grid may change during the learning process due to external causes. In a more sophisticated version, nodes may have actions that allow them to change who their neighbors are. That could be appropriate when the nodes are mobile agents traveling around occasionally encountering each other, or choosing to work together in teams.

References

[1] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. Coordination of multiple behaviors acquired by vision-based reinforcement learning. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 94)*, pages 917–924, 1994.

[2] L. Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. In *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufman, 1995.

[3] E. Baum. *Manifesto for an Evolutionary Economics of Intelligence*, pages 285–344. Springer-Verlag, 1998.

[4] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[5] J. Boyan and M. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Neural Information Processing Systems*, volume 6, 1994.

[6] S. Choi and D. Yeung. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Neural Information Processing Systems 8*, 1996.

[7] R. Crites and A. Barto. Improving elevator performance using reinforcement learning. In *Neural Information Processing Systems 8*, 1996.

[8] G. Gordon. Stable function approximation in dynamic programming. In *The 12th International Conference on Machine Learning*, 1995.

[9] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[10] J. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *Neural Information Processing Systems 9*, 1996.

[11] J. Schneider, J. Boyan, and A. Moore. Value function based production scheduling. In *International Conference on Machine Learning*, 1998.

[12] S. Singh and D. Cohn. How to dynamically merge markov decision processes. In *Neural Information Processing Systems*, volume 10, 1998.

[13] D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning: A case study in routing dynamic networks. In *Proceedings of IJCAI-97*, 1997.

[14] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[15] M. Tan. Multi-agent reinforcement learning: independent vs. cooperative agents. In *Proceedings of the Tenth International Workshop on Machine Learning*, pages 330–337, 1993.

[16] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[17] G. Weiss. Distributed reinforcement learning. *Robotics and Autonomous Systems*, 15:135–142, 1995.