Accelerating Multi-agent Reinforcement Learning with Dynamic Co-learning

Dan Garant Bruno Castro da Silva Victor Lesser
School of Computer Science
University of Massachusetts Amherst
{dgarant, bsilva, lesser}@cs.umass.edu

Chongjie Zhang
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
chongjie@csail.mit.edu

January 2014

Abstract

We introduce an approach to adaptively identify opportunities to periodically transfer experiences between agents in large-scale, stochastic, homogeneous, multi-agent systems. This algorithm operates in an on-line, distributed manner, using supervisor-directed transfer, leading to more rapid acquisition of appropriate policies in systems with a large number of cooperating reinforcement learning agents. Our method constructs high-level characterizations of the system—called contexts—and uses them to identify which agents operate under approximately similar dynamics. A set of supervisory agents compute and reason over contextual similarity between agents, identifying candidates for experience sharing, or co-learning. Using a tiered architecture, state, action, and reward tuples are propagated amongst the members of co-learning groups. We demonstrate the effectiveness of this approach on a large-scale distributed task allocation problem with hundreds of co-learning agents operating in an unknown environment with non-stationary neighbors.

1 Introduction

In large-scale multi-agent systems consisting of hundreds to thousands of reinforcement-learning agents, convergence to a near-optimal joint policy, when possible, can require a large number of samples. However, in such settings, there may be groups of agents working on approximately identical local tasks or under approximately similar environmental dynamics. Identification of such groups could prove to be very useful in cooperative domains, giving rise to a number of opportunities to

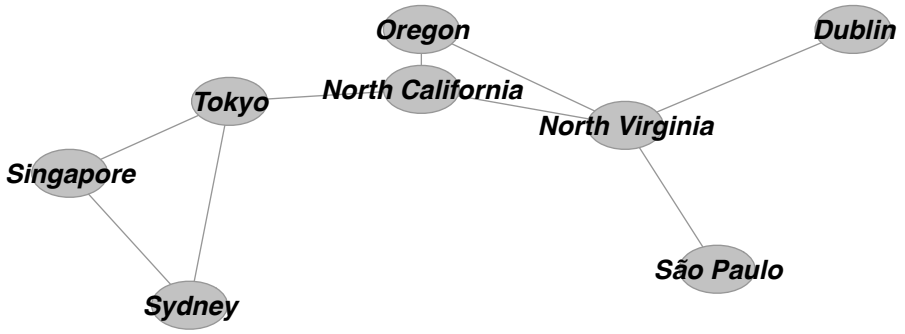

Figure 1: A Small Load Balancing Network

exploit shared information. Information sharing has been studied in single-agent settings in order to generalize knowledge learned for a particular task or environment to novel tasks [13, 5, 1]. Adapting this idea to the multi-agent setting, it is apparent that in addition to transferring experiences across environments or tasks, it is possible to *adapt* experiences for use by concurrently learning agents within a fixed task and environment. This paper focuses on the problem of directly sharing experiences between agents, with emphasis on finding agents within a large community that can form appropriate experience sharing groups.

Consider a distributed load balancing domain, as in Figure 1. This is a cooperative multi-agent system in which agents (depicted as nodes) seek to minimize the joint service time of a set of tasks. A common MARL (multi-agent reinforcement learning) approach in settings such as this consists of decomposing the environment such that each agent is locally autonomous and perhaps capable of observing properties of neighboring agents. Each agent optimizes a value function over a local view of the overall global environment [2, 14, 7, 9]. At the other extreme, attempts could be made to optimize a *single* joint policy directly, but these become intractable as system size grows. In a load balancing network consisting of 1000 agents, each capable of forwarding tasks to three neighboring agents, policies defined over joint actions span a large space with more than 3^{1000} elements. Since the former technique is distributed, the action space is manageable. However, a completely distributed learning process cannot exploit similarities between agents or leveraging knowledge obtained by one node to accelerate the learning of some other agent. In the load balancing domain, the local environment of the "Sydney" node could, for instance, be very similar to that of the "Oregon" node, and it would be valuable for these nodes to communicate and integrate each other's experiences. Attempting to formalize the notion of "similarity" between agents' local environments and communication of experiences among similar agents gives rise to a number of challenges:

- *What information should be transferred between agents?* Taylor and Stone [13] identify several classes of information transfer in single-agent domains. We will discuss these alternatives in the context of multi-agent systems

and introduce a supervisory architecture based on the transfer of low-level experiences taking the form (s, a, s', r).

- *How should we define contextual compatibility and use this to form sharing groups?* Several potential approaches to evaluate the similarity of tasks in single-agent reinforcement learning have been proposed by Carrol & Seppi [1]. We will motivate a similarity measure grounded in comparison of environment dynamics rather than learned policies or Q-values.
- *How should concerns involving scalability be addressed?* As multi-agent systems become larger, it becomes increasingly difficult to construct transition or reward models. As a result, directly operating over model-based information for experience transfer, such as that proposed by Lazaric et al. [5] for single-agent settings, is not practical in large-scale multi-agent settings.
- *How should the architecture handle temporal heterogeneity in transition and reward models?* A principal challenge of multi-agent reinforcement learning is that concurrently learning agents introduce non-stationarity. In such a setting, convergence is not guaranteed. To counteract this, we propose modeling *context* as inherently dynamic, temporal, local characterizations of the system and operating over short-scale time windows during which policies, transition, and reward models are approximately static.

2 Contributions

Our proposed technique accelerates MAS learning in non-stationary situations and does not require augmentation of any agent's state space with additional information about the global environment or modification of the reinforcement learning algorithm employed. Instead, we advocate for the use of *supervisory* agents, differing from a standard agent only in that they are capable of receiving and acting on communications from any subordinate. Supervisors are responsible for identifying *contextually compatible* subordinates, which by construction are those that gather experiences from approximately equivalent state transition and reward models. Our architecture periodically searches for these contextually compatible groups of agents, forms knowledge sharing groups, and propagates learned experiences among members of the group.

Contextual modeling is made possible through the commonly-studied property of interaction sparsity [15]. In many domains such as disaster planning [4, 10] and sensing networks [16], agents interact strongly with only a small group of closely related partners (defined with a metric such as proximity, similarity, or some task-based measure). This leads to promising opportunities to summarize agent interactions in a compact way. In the load balancing example of Figure 1, agents interact most strongly with other agents within some geographic radius. In addition, it may not be necessary to reason about *individual* interactions but rather some aggregate interaction effect over groups of anonymous agents. For instance, it may be unnecessary to model which servers are servicing which tasks but rather aggregate measures, or summaries, of task frequency and difficulty.

This information is likely unobservable by individual agents operating in complete independence, but could be maintained by a regional supervisor.

We evaluate our architecture on a distributed task allocation domain in which agents work to minimize service time for a set of tasks. We use the notion of *context feature fidelity* to measure the discrepancy between a set of context features compactly characterizing the local learning environment and the true state transition & reward models they act as a proxy for. We demonstrate that selection of appropriate context features (discussed in Section 4) can be highly beneficial, allowing for significant performance gains when selected correctly, and introducing volatility in performance when mis-specified. In addition, we justify our supervisory architecture (Section 6) by describing how performance and communication complexity varies with respect to the degree of distribution employed by the system.

3 Multi-agent Reinforcement Learning

Multi-agent decision-making problems are often framed in the context of Markov games. In the most general configuration, these games model n agents, each with a set of allowable actions and operating in an environment with shared state S. A state transition function specifies the conditional probability of existing in an environment state S' at the next step given the *joint* action (an n-tuple describing the actions taken by each agent) and the current shared system state, $P(S'|S, a_1, a_2, \ldots, a_n)$.

Let us decompose the global state S into (potentially overlapping) components representing the portions of the state that agents can *directly* observe and incorporate into their learning. We let s_i denote the observable state of agent i, and let s_e represent all elements of the global state which no agent can observe, that is:

$$s_e = S - \bigcup_{i \in \{1,2,\ldots,n\}} s_i.$$

In large systems, even when each agent can observe all of S (that is, $s_i = S$), it may not be practical for agents to perform learning over this very large state space. In addition, depending on the application domain, physical distance between agents or communication bandwidth can introduce limitations on state observability. For this reason, we focus on problems in which the state observable by an individual agent is insufficient to faithfully reconstruct the state transition model, $P_i(s'_i|S, a_1, a_2, \ldots, a_n)$. Note, however, that there may be sparsity in P_i, allowing an agent to obtain an estimate $\hat{P}_i$ of P_i for which s'_i is conditionally independent of many of the state and action variables of other agents.

In Markov games, each agent i holds a reward function $R_i(r_i|S, a_1, a_2, \ldots, a_n)$, which we represent here as a conditional distribution over rewards for sake of generality. In cooperative environments, individual reward functions may be identical, that is, each agent measures its individual performance in terms of

how well the system performs as a whole. This can again be problematic for large systems of agents, so in this paper we will consider a more general case of *decomposable* reward, found in settings structured as ND-POMPDPs[1] [8] or factored multi-agent MDPs [3]. We assume that rewards can be computed only from a joint action and complete shared state, and that agents receive reward roughly proportional to their independent contribution to the state change.

4 Context Features

In this section we discussion utilizing context features as proxies for compactly characterizing the learning environment of an agent. Carrol & Seppi [1] note the difficulty of constructing reasonable similarity measures for learning environments, given a wide array of possible definitions of agent similarity. For transfer in single-agent reinforcement learning, they propose several techniques including measures derived from similarity in learned policies, MDP distance measures computed over Q-values, or reward function differences. In the multi-agent setting, we are interested in capturing a measure of compatibility in the local learning environment of two agents in a way that does not suffer from any of the following problems:

- **Imbalance in Experience** Attempting to characterize the local learning environment of an agent by comparing policies or Q-values directly requires that the policies under investigation have been constructed with enough samples to be accurate estimates of the optimal policy. Comparing policies and Q-values of agents with inaccurate estimates may bias such similarity measures. We would like to allow formation of mentor/mentee relationships as described by Price and Boutilier [11] (except that we are additionally interested in accounting for non-stationarity arising from transition function dependence).
- **Policy Divergence** Optimal policies for Markov games are not unique [6]. However, supposing two agents working within the same local environment had diverged on the trajectory to an optimal policy, it is still reasonable to share experiences between these agents. When the underlying unobservable environmental model for two agents is identical, they should be able to provide some information to one another, but sharing or comparison of learned values or entire an policy would be problematic due to lack of uniqueness.
- **Latent State Features** While Q-values indirectly describe the transition model experienced by an agent, if potential factors impacting that transition model remain unobserved, two Q-functions are not directly comparable.
- **Disjoint State Visitations** It is possible that two agents could share a common task but have experiences in disjoint regions of the state space. In this case, the Q-values, policies, or reward models are well-estimated in separate regions of the state space, so it may be difficult to compare them

[1] For this work, transition and observation independence is not assumed

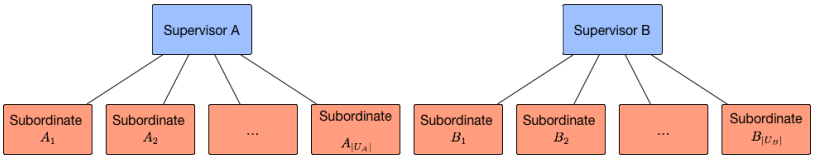

Figure 2: Example Hierarchy Involving 2 Supervisors

directly. However, this could be an excellent opportunity for information sharing in case the agents do transition into new regions of the state space.

To account for these factors, we propose reasoning over the underlying model of a stochastic game as a basis for contextual comparison. In particular, if we can guarantee that agents are working under the same local transition model and they share a common reward function, we know from the homogeneity of the system that the agents are facing the same learning problem. In such a case, the experiences gathered by each compatible agent are interchangeable and opportunities to exchange information between them are numerous. Since estimating these models is impractical in an online manner for large systems, we propose the use of designer-selected context features, designed to provide broad-scope summaries of transition and reward models as experienced from individual agents' perspectives. When agents act without complete knowledge of other agents in the system, state transition and reward models must be approximated as $\hat{P}_i(s_i'|s_i, a_i)$ and $\hat{R}_i(r_i|s_i, a_i)$, respectively. While it could be possible for an individual agent to track and reason over the states and actions of all other agents in the system, thereby removing the non-stationarity, this is not generally possible and becomes increasingly difficult or impossible as the number of agents in the system grows. To counteract this, we propose constructing context features that are compact descriptions of the environmental dynamics experienced by an agent, computable by an agent's supervisor.

4.1 Context Feature Fidelity

Context features are intended to act as an indirect proxy for state transition and reward models. We will show experimentally that when these features are effective, performance improves, and when such features as inappropriate, opportunities for sharing are limited (by Algorithm 2), realizing no benefit over a system which does not leverage contextual similarity. In order to evaluate a set of features, model estimates must be constructed. As we have noted, this may not be possible in an on-line manner. However, in cases where it is possible to build the models in an offline setting, it is possible to evaluate the efficacy of such features in achieving their designed goal of assessing compatibility of transition and reward models.

In particular, we are interested in selecting context features and a distance function over those context features that induces context feature distances which are positively correlated with distances between agents' local state transition

and reward models. When comparing the models of agents i and k,

$$P_i(s_i'|S, a_1, a_2, \ldots, a_n),$$
$$R_i(r_i|S, a_1, a_2, \ldots, a_n),$$
$$P_k(s_k'|S, a_1, a_2, \ldots, a_n), \text{ and}$$
$$R_k(r_k|S, a_1, a_2, \ldots, a_n),$$

it is quite likely that each agent's transition and reward functions will be affected most strongly by a different sets of agents. For instance, agent i might interact strongly with agent 1, so a_1 has importance in P_i and R_i. Perhaps agent k does not interact with agent 1 at all, but has very comparable interactions with agent n, so a_n is important in its model. Thus, the manner we use to "label" agents in each case may be different, but we may be able to draw analogies between the two models. As such, it is necessary to search for a good "correspondence" function, taking the form of a permutation ρ over agents with the restriction that when constructed for agent i, $\rho(i) = i$ (an agent cannot be relabeled in its own local transition model). The quality of a correspondence is characterized as the distance between the component models that it induces. Then, to evaluate context feature fidelity, we can perform a search over all permutations ρ, jointly minimizing the symmetric KL-divergence (denoted SKL) between P_i & P_k, and R_i & R_k, shown in Algorithm 1. This gives us a measure of equality, since when $m = 0$, we have $P_i = P_k$ and $R_i = R_k$, up to labeling [12]. As this divergence measure grows, the two agents become increasingly incompatible.

By relating the divergence measure m to some distance function D^C over context features, we can get a sense for how effective the selected context features are at representing distances between transition and reward models. If C_{i_t} and C_{k_t} are context feature vectors computed from agent i and k's perspective at time t, respectively, then $D^C(C_{i_t}, C_{k_t})$ should be positively correlated with m, derived from the estimated transition and reward models from some window around time t. Context feature fidelity, then, is the correlation of m with values output from D^C, in the range $[-1, 1]$. Values approaching 1 indicate that as D^C distances trend upwards, transition and reward models become increasingly distant/dissimilar as well – exactly the relationship we would like to encode. Values approaching -1 would encode an inverse relationship between D^C distances and the model information, which is undesirable for our purposes. Values approaching 0 indicate that the selected context features are not informative about the true models.

Given a good set of context features, D^C can be used to derive a compatibility measure between agents. When these distances are small, the models currently governing each agent's state transitions are similar. Various selection strategies can be employed to find groups of similar agents given a matrix of pairwise distances. One such technique will be introduced in Section 6.2.

5 Experience Sharing

Typically, sharing experiences between agents in a system is not reasonable, since each agent experiences different state transitions. However, discovery of contextually compatible agent groups $G = \{g_1, g_2, \ldots, g_m\}$ with the D^C measure, allows for reasoning about compatibility of agent experiences. If the agents in G are perfectly contextually compatible, we have

$$P_{g_1} \cong P_{g_2} \cong \cdots \cong P_{g_m} \text{ and } R_{g_1} \cong R_{g_2} \cong \cdots \cong R_{g_m}.$$

Of course, it is unlikely that such a group G would be selected in practices. Context features are approximate measures, and we may be willing to accept agents in G for inclusion even if their transition models are not perfectly identical. As such, agents in G have are approximately equivalent transition models. We will demonstrate that sharing experiences in such circumstances (under imperfect but approximate correspondence of transition models) can be highly beneficial, and is highly dependent on the distances D^C between the agents in G (see Section 7).

Once G has been identified, experiences of its members can be propagated through the group (using a supervisory technique introduced in Section 6.1). This effectively multiplies the number of experiences each agent by $|G|$ (assuming agents gain roughly the same number of experiences). In a non-episodic environment, experiences for agent i can be transferred as a vector of tuples within some time window $[t_0, t_e]$:

$$\Big\langle (s_{i_{t_0}}, a_{i_{t_0}}, r_{i_{t_0}}, s'_{i_{t_0}}), \\ (s_{i_{t_1}}, a_{i_{t_1}}, r_{i_{t_1}}, s'_{i_{t_0}}), \\ \ldots, \\ (s_{i_{t_e}}, a_{i_{t_e}}, r_{i_{t_e}}, s'_{i_{t_0}}) \Big\rangle$$

In episodic environments, it is reasonable to transfer only complete episodes between agents. In this case, agents are responsible for delimiting the tuples comprising each episode and reporting only complete episodes.

We could also consider transferring or merging policies, but this has several disadvantages. In particular, replacing a policy suggests permanence and risks losing information about states which may not be visited uniformly across time. Consider a case agent where agent i experiences transitions from model P_i^1 for some time, then P_i^2, then returns to P_i^1. In addition, imagine another agent k cycling between P_k^1, then P_k^2, and finally P_k^1 again. Further suppose that P_i^2 and P_k^2 are contextually similar, but P_i^1 and P_k^1 are not. In this case, if P_i^1 and P_i^2 have probability mass on different regions of the state space, we would risk losing learning taking place in P_i^1 by deciding to transfer a policy from k during the second time window. Additionally, in domains that may have competitive aspects (such as taxi driving), it may not be practical to impose a policy on an actor. A more agreeable option would be to make experiences available for

Input: R_i, R_k, stochastic reward models for agents i and k, respectively.
P_i, P_k, state transition models for agents i and k, respectively.
$\lambda \in (0, 1]$, a parameter balancing the importance of accuracy in the state transition model with the reward model

Output: m, a measure of context feature fidelity

Compute similarity in reward as symmetric KL divergence for a permutation ρ of agents:

$$D_R(\rho) \leftarrow D_{SKL}\big(R_i(r_i|S_\rho, a_{\rho(1)}, a_{\rho(2)}, \ldots, a_{\rho(n)}), \\ R_k(r_k|S, a_1, a_2, \ldots, a_n)\big)$$

Compute similarity in state transition models as symmetric KL divergence:

$$D_S(\rho) \leftarrow D_{SKL}\big(P_i(s'_i|S_\rho, a_{\rho(1)}, a_{\rho(2)}, \ldots, a_{\rho(n)}), \\ P_k(s'_k|S, a_1, a_2, \ldots, a_n)\big)$$

Find the best permutation ρ according to the balancing parameter λ:

$$m \leftarrow \min_\rho \ \lambda\left(1 - \exp\left(-D_R(\rho)\right)\right) + \\ (1-\lambda)\left(1 - \exp\left(-D_S(\rho)\right)\right)$$

Algorithm 1: Estimated Model Divergence

incorporation into a policy if appropriate and desired, maintaining the autonomy of individual learning.

6 Context-based Learning

In order to enable contextual comparison & experience transfer in a general way, we propose the use of supervisory agents. Each supervisor a is responsible for a set of subordinates U_a. In order to be a candidate for experience sharing, an agent must have a supervisor. Supervisory agents are capable of receiving communications of experiences from all subordinates, and derives context features from the reported experiences by cross-inspection of reports from other subordinates. Given these context features and the previously noted distance function D^C, groups of similar agents can be recovered. The supervisor then relays information amongst the members of these groups, and agents can incorporate the shared experiences into their policy if desired.

6.1 Agent Organization

Illustrated in Figure 2, we view supervision graphically as a forest of agents with maximum tree depth 1. Further, we assume that agents are not capable of communicating directly, relying on a common supervisor to act as a proxy for experience transfer. This arrangement highlights several design decisions. First, supervisory agents are unlikely to have boundless communication bandwidth – there is undoubtedly some limit to the number of agents they can receive communications from (and the size of those messages). It may prove useful to distribute supervisors in manner that spans a physical space most effectively, or in a manner consistent with agent interaction strength [17]. Finally, computing context features and finding similar agents requires a non-trivial amount of computation, typically polynomial in the number of agents (depending on the clustering algorithm employed by Algorithm 2).

A critical aspect of the proposed architecture is selection of other agents with sufficiently small distances in context space to form sharing groups. This requires that a balance be found between the number of supervisors in the system and the number of agents which are candidates for experience sharing. As the number of supervisors grow, the problem becomes increasingly distributed, reducing the requirements imposed on any individual supervisor. In opposition to this, supervisors which oversee larger groups of subordinates are capable of selecting from a larger pool of experience sharing candidates, increasing the likelihood that similar agent groups can be constructed. Additionally, a trade-off is necessary between communication fidelity and bandwidth by allowing for lossy compression of experiences to and from the supervisor. One such lossy compression technique involves the use of linear interpolating functions, and is especially effective when agents infrequently change state. To start, a set of linear regressions can be performed over any continuous state, action (or action probability), and reward variables. These regression equations can be performed in a loss-less manner,

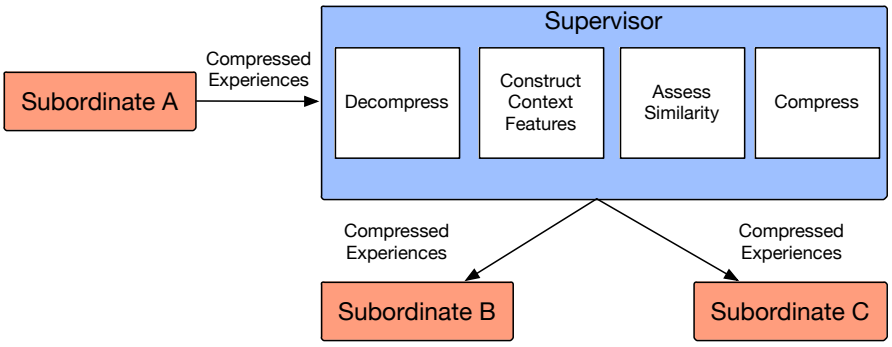

Figure 3: Relaying Experiences through a Supervisor

that is, a sufficient number of lines can be fit to the data to perfectly reconstruct the traces on the original time scale. Alternately, we can allow for interpolation by performing regressions over subsets of the original data. The extent of the compression is tunable by considering more or less sparse data sets for this regression. As shown in Figure 3, communications to and from the supervisor consist of "compressed experiences", which are simply coefficients of regression equations in this formulation. The supervisor is then capable of reconstructing the (perhaps approximated) state, action, and reward tuples forming the agent's experience trace over the report's relevant time window.

6.2 Algorithms

Upon receipt and decompression of an agent observations, supervisors are responsible for construction of context features. The process by which these feature vectors are constructed depends on the features themselves, although if interactions are sparse, we believe that agents' transition models can be explained by experiences from some neighborhood of constant size d. Suppose agent experiences are communicated every K time units, so an agent may report as many as K observations. If a supervisor oversees n subordinates, the task of context feature construction is to transform time-indexed experience vectors

$$O_i = \langle (s, a, s', r)_{t_1}, (s, a, s', r)_{t_2}, \ldots, (s, a, s', r)_{t_K} \rangle$$

for each agent $i \in \{1..n\}$ into a vector of context feature sets

$$V = \langle V_1, V_2, \ldots V_n \rangle .$$

The template we propose to perform this transformation is to use a summarization routine accepting a time k, a reference agent i, and a set of d agents in i's neighborhood, and returning a context feature vector $V_{i_{t_k}}$. Then, the mean of these time-dependent feature vectors is taken to produce a single context feature vector V_i representing a per-agent context summary. If the summarization

process can be performed in constant time, the full vector V can be performed in $\theta(Kn)$ time.

Given the set of context features V, we can compute contextual similarity between agents using Algorithm 2. This process uses a two-stage selection process. First, agents are partitioned into roughly similar groups using a clustering algorithm. Then, within each group, an annealing process is used to select related agents. First, pairwise distances are computed over the agents within the cluster. This distances matrix is then represented as a Boltzmann distribution, and agents are assigned a "selection probability", which can be raised or lowered according to a temperature parameter T. Agents are then selected for sharing according to this probability. Once f has been formed, supervisors relay experiences to agents which may benefit from them. In particular, for agent i, supervisors communicate the compressed experiences for all agents in $f(i)$. Locally, agents decompress these experiences and incorporate them into their policies.

Input: Vector $V = \langle V_1, V_2, \ldots, V_n \rangle$ of feature vectors for agents $A = \{1, 2, \ldots, n\}$, temperature parameter $T \in \mathbb{N}$
Output: A set-valued mapping $f : A \to \mathcal{P}(A)$ describing the (possibly empty) set of agents that the input agent should share with
Cluster V into k groups $C_1, C_2, \ldots, C_k$
for $i \leftarrow \{1, 2, \ldots, k\}$ **do**
  $M \leftarrow$ Pairwise distances D^C over vectors within C_i
  $P \leftarrow \sum_{t \in 1..T} \text{BinomialPMF}\left(\frac{\exp(M)}{\sum \exp(M)}, t\right)$
  for $a \in Agents(C_i)$ **do**
    for $b \in Agents(C_i) \setminus a$ **do**
      Let p be the entry in P corresponding to entry (a, b)
      With probability p, let $f(a) \leftarrow f(a) \cup \{b\}$
    end
  end
end

Algorithm 2: Selection of Sharing Partners

7 Empirical Analysis

We evaluated our architecture on a network-distributed task allocation domain. We can model this domain as a graph $G = \langle V, E \rangle$ where the vertex set V represents the set of agents in the system. Agents maintain a queue of tasks, with the head of this queue representing the task that is currently being worked on. These tasks are annotated with a *service time* s, indicating how many time units it would take to complete each task. After a task has been at the head of an agent's queue for s steps, that task is dequeued and marked as completed. The reward function in this setting is the reciprocal of the average service time over

some recent time window, where service time is measured as the time from task creation to task completion. Tasks are generated by the environment according to pattern unknown to the agents. When a task is created, it is associated with some agent v, and immediately placed in that agent's *routing queue*, which contains tasks that are not actively being worked on because they require some action on the part of v. Agent v has several options available for dealing with the tasks in its routing queue. It may either decide to work on the task itself, adding to the task to v's processing queue, or it may forward the task. An agent may forward a task to any agent that it is connected to. That is, for all $n \in V$ such that $(v, j) \in V$, an agent has action `forward-j` available to them, which adds the task to the routing queue of agent j. Upon taking an action, agents immediately receive a reward signal of $\frac{1}{s}$, where s is the service time of the agent receiving the task. When the task has finally been completed, agents receive a signal propagated back through the routing channel that they can use to update their service time based on how long the forwarded task took to complete.

For this domain, we selected as context features for agent i (1) the rate at which each of i's neighbors receive tasks from the environment, (2) the rate at which each of i's neighbors receive tasks from other agents, and (3) i's load relative to the mean load of i's neighbors. Thus, if i has 4 neighboring agents, i's context feature vector has length 9. By selecting these context features, we impose the requirement that context comparisons can only be performed between two agents if those agents have the same number of neighbors. This is a reasonable requirement, since agents with different neighborhood sizes will have different action spaces, breaking some assumptions of Algorithm 1.

We are interested in characterizing whether our sharing architecture is beneficial and if it can scale. To do this, we constructed a network of 100 agents, imposed a supervisory structures on this network in a manner that minimizes the total network distances among the subordinates of any particular supervisor, and ran a variety of experiments whereby characteristics of the network and the architecture were varied. In particular, we consider varying the task distribution pattern governing the concentration of tasks in particular regions of the network. We also varied the the task rate, or a measure of how *many* tasks arise according to this pattern. Together, these provide a means for adjusting difficulty of a scenario. Then, we vary architectural aspects such as the number of supervisors and examine corresponding changes in performance. Our primary measure of performance is the area under the learning curve (AUC), which is an exponential moving average of the mean task service time over all tasks completed in the network. Before computing the area, we first "lower" this curve onto the x-axis by subtracting the minimum y-value. As a result, the area under the curve will change only minimally once the system has stabilized in performance, which is desirable since we used a fixed run length of 10000 time units, but some easy configurations stabilize in performance very early in this span.

Throughout our experiments, a learning window of $K = 115$ is employed. Smaller values of K lead to more frequent communication, whereas larger values of K decrease the likelihood that agent transition and reward models are static across the K-timestep window. Our experiments focus on cases which are neither

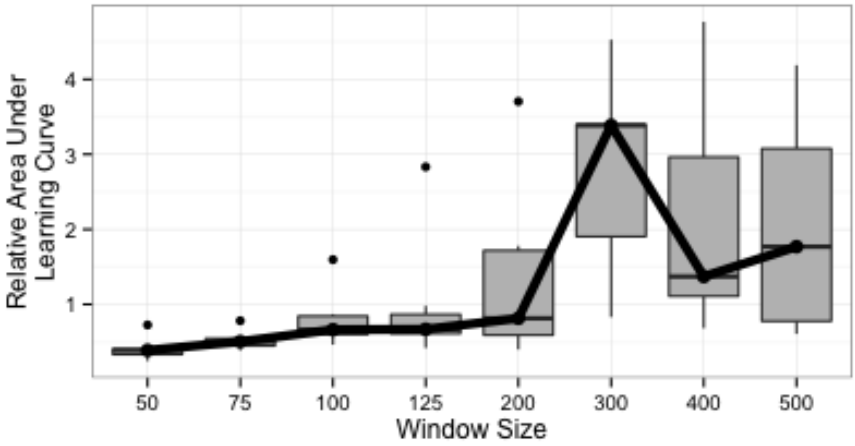

Figure 4: Effect of varying window size K

trivially easy or unrealistically difficult. We characterize the former case as one in which all architectures convergence almost instantly to optimal performance, requiring little or no learning. The latter case is characterized by divergence in performance of the baseline configuration, indicating that agents are unable to learn policies effective enough to handle incoming tasks. This leaves a set of task difficulty settings in which agents learn for a non-negligible amount of time (at least 1000 time units), then converge in performance. We normalize performance of all architectural configurations relative to a *baseline* architecture which does not use supervision or experience sharing, but is otherwise identical in configuration and in the policy learning algorithms employed. Figure 5 illustrates how performance comparisons are carried out given a time series of performance values.

The relative performance of each architecture in simulations with two different task distribution strategies (border-based, center-based) and ten different task distribution rates (ranging from very easy to very difficult) is shown in Figure 7. The "1 Sup" (1-supervisor) configuration is something of a best-case scenario in which we can completely centralize the sharing process. This allows sharing to take place between any pair of agents in the system, and requires roughly half the learning area as the baseline configuration. As we grow the number of supervisors in the system, the problem becomes more tractable, distributed, and realistic to many real domains in which communication cannot practically span the entire breadth of an agent network. In the 9-supervisor configuration, each supervisor need only oversee its 11 closest neighbors, but the required learning area has still been reduced by 25%. As the network size increases, the pool of subordinates in the 9-supervisor configuration increases, leading to additional sharing opportunities. Figure 8 demonstrates this phenomena for 30 simulations

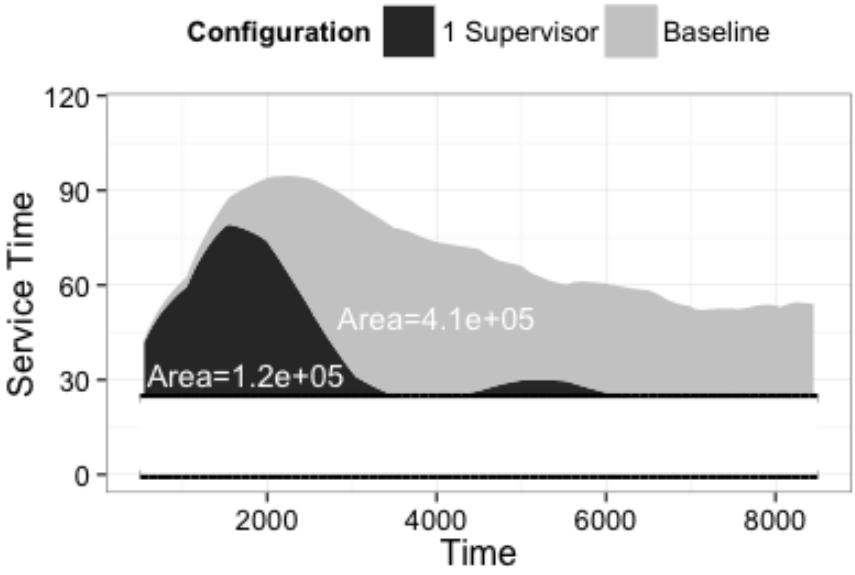

Figure 5: Task Allocation Performance Evaluation. Area under learning curves is computed, then any area below the best convergence point is subtracted from each curve (illustrated as a grey region), penalizing sub-optimal convergence. In this case, the 1-supervisor configuration improves on the baseline area by 70%.

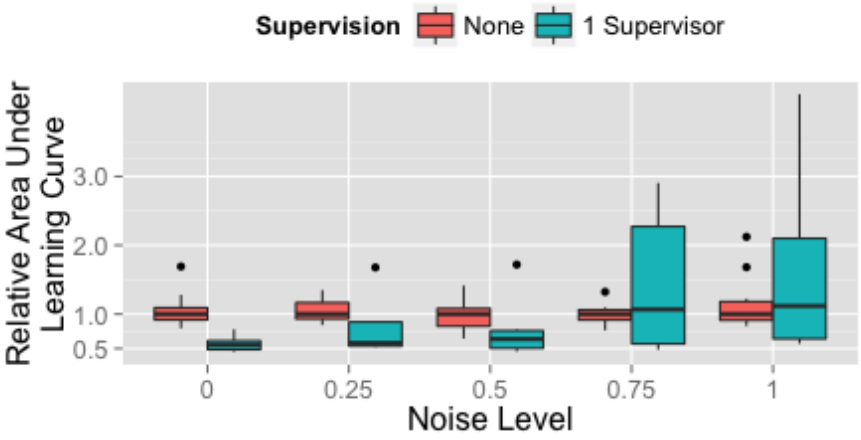

Figure 6: Relative Performance as Context Features are Noised. At noise level 0.75 and 1, the performance of the 1-supervisor case has the same mean as the baseline case, but is much more volatile.

spread across networks of three different network sizes (i.e. 100, 324, and 729 agents).

Of course, any supervisory configuration will require more communication bandwidth than the baseline. However, in this domain at least, agent experiences are easily compressed using the technique described in Section 6.1, with a tunable fidelity. A set of experiments using a single-supervisor architecture demonstrated that the size of messages sent from supervisors to subordinates decreases rapidly with decreased message fidelity. The observed trend indicates that we can vastly reduce the average subordinate-to-supervisor message down to as little as 43 bytes per time unit. A natural concern would be the effect of compression fidelity on performance. We performed a correlation test on this relationship and found no significant trend up to a compression degree of 15, indicating that we can safely compress experiences up to the point where further degrees of compression are ineffective without significantly impacting performance. Another important factor in controlling the amount of communication is the window size K. To characterize the effect of selecting K, we ran 10 trials of the single supervisor and baseline configurations for each of eight window sizes. Figure 4 demonstrates the performance of the single-supervisor configuration relative to the baseline configuration for each window size. As larger window sizes are used, performance degrades, as these windows are unlikely to contain approximately static transition and reward models.

One goal in the development of the sharing methodology is to allow for misspecification of context features without sacrificing performance. Algorithm 2

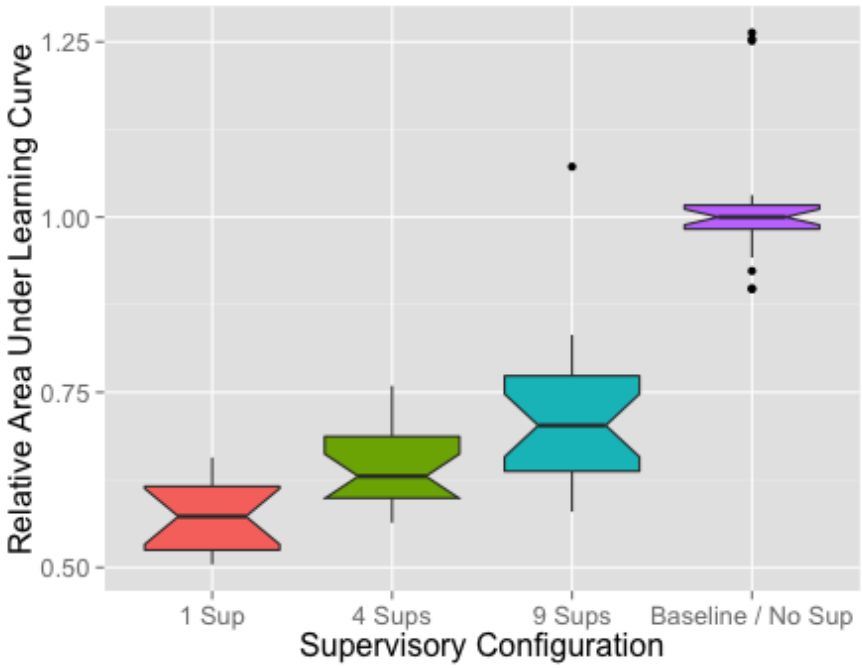

Figure 7: Relative Performance by Supervisory Configuration

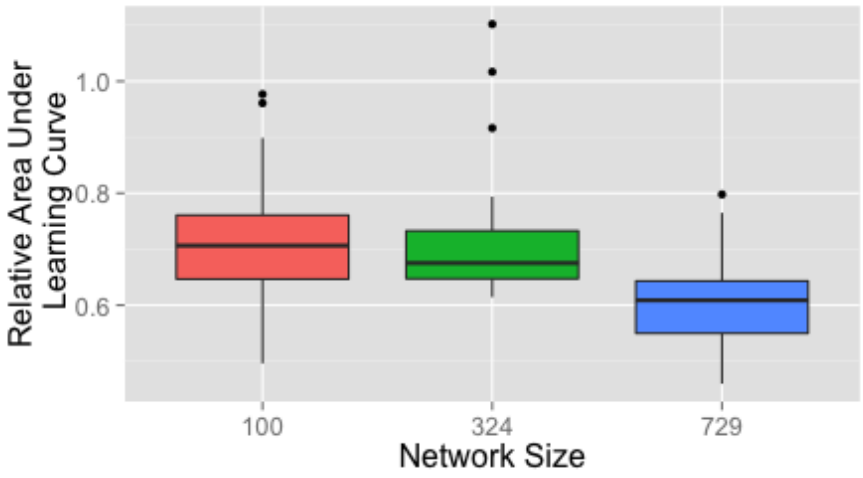

Figure 8: Relative Performance as network size is varied

realizes this goal, identifying sharing opportunities only when context feature distances are significantly close. To evaluate this algorithm, we performed an experiment in which a noise was artificially added to our context features before the sharing partner selection process takes place. This noise degrades the quality of the signal that context features encode, and there is a point at which noise is so strong that features are entirely unhelpful. To accomplish this, we measured the standard deviation of each context feature, and varied noise relative to the largest feature standard deviation. Thus, when the noise level is 1, the standard deviation of the normally-distributed noise term is greater than or equal to the standard deviation of any context feature, effectively eliminating any signal that they encoded. As the noise level approaches 0, this noise term becomes insignificant. Figure 6 demonstrates that at the point that the noise term begins to dominate (approaching 1), the performance of the sharing architecture becomes increasingly volatile, though with mean 1 and no clear skew. As the information encoded in the context features becomes less meaningful, the sharing architecture is equally likely to archieve a 50% reduction in AUC as it is to increase AUC by 100%.

8 Discussion

We have presented an architecture for experience transfer among reinforcement learning agents in large multi-agent systems. Intuitively, when multiple agents work on similar tasks, information transfer can take place. By explicitly transferring experiences, it is possible to bypass issues related to policy divergence, experience imbalance, latent state features, and disjoint state visitations. Further, by transferring (s, a, r, s') tuples, the described technique makes no assumptions about the manner in which individual agents perform policy learning.

In the domain of dynamic task allocation, the proposed experience sharing architecture provides significant improvements over configurations without experience sharing, and becomes increasingly advantageous as system size grows. We have demonstrated that context features can be mis-specified without inducing a systematic degradation in performance.

As future work, we are interested in finding opportunities for automated or informed context feature selection. A simple technique towards this goal could involve an informed search over the space of possible features, using estimated model divergence to guide the search. For model-free learning algorithms, this would require a more efficient implementation of the model divergence estimate. However, by adapting the notion of context features to model-based settings, it may be straightforward to evaluate a measure similar to model divergence directly in an on-line manner, bypassing the need for explicit context features.

9 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1116078.

References

[1] J. L. Carroll and K. Seppi. Task similarity measures for transfer in reinforcement learning task libraries. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 803–808. IEEE, 2005.

[2] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.(JAIR)*, 24:49–79, 2005.

[3] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *NIPS*, volume 1, pages 1523–1530, 2001.

[4] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 739–743. IEEE, 1999.

[5] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551. ACM, 2008.

[6] M. L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[7] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pages 246–253. ACM, 2001.

[8] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI*, volume 5, pages 133–139, 2005.

[9] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009.

[10] F. A. Oliehoek, M. T. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored dec-pomdps. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 517–524. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[11] b. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *JAIR*, 19:569–629, 2003.

[12] A. Rényi. On measures of entropy and information. In *Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pages 547–561, 1961.

[13] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[14] D. Vickrey and D. Koller. Multi-agent algorithms for solving graphical games. In *AAAI/IAAI*, pages 345–351, 2002.

[15] S. J. Witwicki and E. H. Durfee. Influence-based policy abstraction for weakly-coupled dec-pomdps. In *ICAPS*, pages 185–192, 2010.

[16] C. Zhang and V. Lesser. Coordinating Multi-Agent Reinforcement Learning with Limited Communication. In J. Ito and S. Gini, editors, *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, pages 1101–1108, St. Paul, MN, 2013. IFAAMAS.

[17] C. Zhang, V. Lesser, and S. Abdallah. Self-Organization for Coordinating Decentralized Reinforcement Learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 739–746, Toronto, 2010.