MASPA: Multi-Agent Automated Supervisory Policy Adaptation

Chongjie Zhang
Computer Science Department
University of Massachusetts Amherst

Sherief Abdallah
Institute of Informatics
British University in Dubai

Victor Lesser
Computer Science Department
University of Massachusetts Amherst

UMass Computer Science Technical Report #08-03

July 21, 2008

Abstract

Multi-Agent Reinforcement Learning (MARL) algorithms suffer from slow convergence and even divergence, especially in large-scale systems. In this work, we develop a supervision framework to speed up the convergence of MARL algorithms in a network of agents. Our framework defines a multi-level organizational structure for automated supervision and a communication protocol for exchanging information between lower-level agents and higher-level supervising agents. The abstracted states of lower-level agents travel upwards so that higher-level supervising agents generate a broader view of the state of the network. This broader view is used in creating supervisory information which is passed down the hierarchy. The supervisory policy adaptation then integrates supervisory information into existing MARL algorithms, guiding agents' exploration of their state-action space. The generality of our framework is verified by its applications on different domains (i.e., distributed task allocation and network routing) with different MARL algorithms. Experimental results show that our framework improves both the speed and likelihood of MARL convergence.

1 Introduction

A central challenge in multi-agent systems (MAS) research is to design distributed coordination mechanisms to agents that have only partial views of the whole system to generate efficient solutions to complex, distributed problems. To effectively coordinate their actions, agents need estimate the unobserved states of the system and adapt their actions to the dynamics of the environment. Multi-agent reinforcement learning (MARL) techniques have been extensively explored in such setting.

To scale up, previous research [2, 16, 4] has distributed the learning and restricted each agent to using the information received only from its immediate neighbors to update its estimates of the world states (i.e., Q-values for state-action pairs). However, this constraint results in long latency to propagate the state information to agents further away. Such latency can result in neighborhood information being outdated, hence leading to mutually inconsistent views among agents. In addition, updating local estimates using information only from immediate neighbors can potentially suffer from the ”Count-to-Infinity” problem [12], where agent A's estimate of the world state is calculated from agent B's estimate, which is calculated from from agent A's estimate. Therefore, such limited view for each agent and the non-stationarity of the environment (all agents are simultaneously learning their own policies) causes MARLs to converge slowly and even diverge. Furthermore, the slowness of MARL convergence is worsened by the large policy search space. Each agent's policy not only includes its local state and actions but also some characteristics of the states and actions of its neighboring agents [2], or the state size of each agent may be proportional to the size of the system [4].

Two paradigms have been studied to speed up the learning process. The first paradigm is to reduce the policy search space. For example, the TPOT-RL [11] reduced the state space by mapping states onto a limited number of action-dependent features. The hierarchical multi-agent reinforcement learning [5] used the explicit task structure to restrict the space of policies, where each agent learned joint abstract action-values by communicating with each other only the state of high-level subtasks. The second paradigm is to use heuristics to guide the policy search. The work [13] used both local and global heuristics to accelerate the learning process in a decentralized multirobot system. The local heuristic used only the local information and the global heuristic used the information that was shared and required to be exactly the same among robots. The Heuristically Accelerated Minimax-Q (HAMMQ) [3] incorporated heuristics into the Minimax-Q algorithm to speed up its convergence rate, which shared the convergence property with Minimax-Q. HAMMQ was intended for use only in a two-agent configuration and further the authors had no discussion how heuristics were constructed.

This paper presents a supervision framework, called Multi-Agent Automated Supervisory Policy Adaptation (MASPA), to accelerate the learning. MASPA follows the second paradigm that uses heuristics to guide the policy search. The main contribution of MASPA is that it defines a decentralized hierarchical supervision mechanism to automate the generation of heuristics (also called supervisory information) and uses a supervisory policy adaptation that integrates heuristics into existing unsupervised MARL algorithms (e.g., GIGA [17], WPL [1], etc.) in a generic manner to speed up their convergence. The supervision mechanism is defined by a multi-level supervision organization (a meta-organization built on top of the agents' overlay network) and a communication protocol for exchanging information between lower-level agents and higher-level supervising agents.

The key idea of MASPA is as follows. Each level in the supervision organization is an overlay network in itself. For example, Figure 1 shows a three-level supervision organizational structure. The abstracted states of lower-level agents travel upwards so that higher-level supervising agents can generate a broader view of the state of the network. This broader view comes from not only information about the states of lower-level agents but also information from neighboring supervising agents. In turn, this broader view results in creating supervisory information which is passed down the hierarchy. The supervisory information guides the learning of agents in collectively

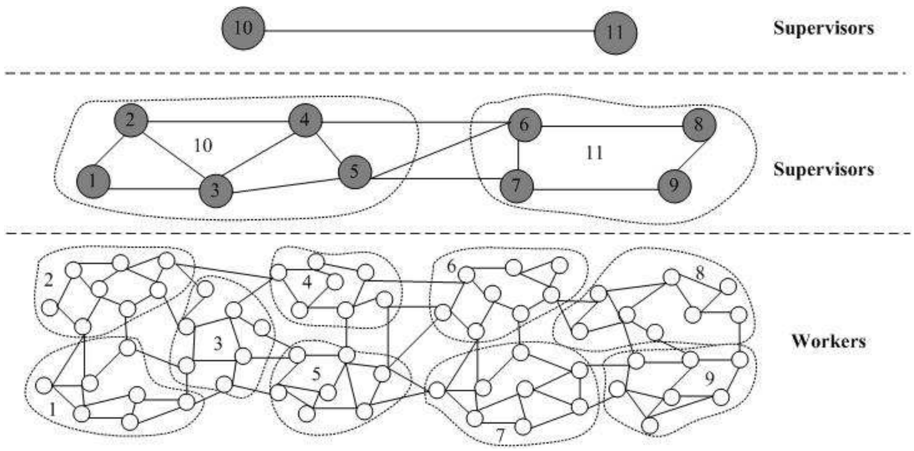

Figure 1: An organization structure for multi-level supervision

exploring their state-action spaces more efficiently, and consequently results in faster convergence. To provide up-to-date supervisory information, the process above is periodically repeated.

Our approach has a hierarchy of control and data abstraction, which is conceptually different from existing hierarchical multi-agent learning algorithms that uses a hierarchy of task abstraction. The generality of MASPA is verified by its applications in different domains (i.e., distributed task allocation and network routing) with different MARL algorithms. Experimental results show that it not only dramatically speeds up the rate of MARL convergence, but also increases its likelihood of convergence. MASPA also shows robustness when not all supervising agents work properly. To our knowledge, it is the first work that demonstrates that if appropriately exploited a more global view of network state significantly improves MARL performance.

MASPA assumes agents will voluntarily share their state information. It also implicitly assumes the original multi-agent system can be formed into a nearly decomposable hierarchy [9] of at least one level. This assumption implies that If agents in the original MAS are far apart in spatial terms, their behaviors are also far apart in causal terms. For example, in Figure 1, knowing detail information about agents in cluster 6 will not affect much behaviors of agents in cluster 1. For clarity, this paper limits the discussion to the case where learning only happens at the bottom level and supervising agents use some heuristics to make decisions, but MASPA does not restrict the opportunity for supervising agents to learn their supervision policies.

The rest of the paper is organized as follows. First, we present a multi-level organizational structure used by the supervision mechanism. Then a communication protocol is defined for agents at different levels. After that, we describe the supervisory policy adaptation that integrates supervisory information into MARL algorithms. MASPA is then empirically evaluated on DTAP and network routing problem. Finally, we concludes this work and discusses some future work.

2 Organizational Supervision

Supervision mechanisms commonly exist in human organizations, such as enterprises and governments. The purpose of these mechanisms is to run an organization effectively and efficiently to fulfill the organization goals. Supervision involves gathering information, making decisions, and providing directions to regulate and coordinate actions of organization members. The practical effectiveness of supervision mechanisms in human organizations, especially in large organizations, inspired us to introduce a similar mechanism into multi-agent systems in order to improve the efficiency of MARL algorithms.

To add a supervision mechanism to a MAS with an overlay structure, MASPA adopts a multi-level, clustered organizational structure. Agents in the original overlay network, called workers, are clustered based on some measure (e.g., geographical distance). Each cluster is supervised by one agent, called the supervisor, and its member agents are called subordinates (note that subordinates at the lowest level are workers). The supervisor role can be played by a dedicated agent or one of the workers. If the number of supervisors is large, a group of higher-level supervisors can be added, and so on, forming a multi-level supervision structure.[1] In this paper, our discussion focuses on the situation where each agent belongs to only one cluster.

Two supervisors at the same level are adjacent if and only if at least one subordinate of one supervisor is adjacent to at least one subordinate of the other. Communication links, which can be physical or logical, exist between adjacent workers, adjacent supervisors, and subordinates and their supervisors. Figure 1 shows a three-level organizational structure. The bottom level is the overlay network of workers which forms 9 clusters. A shaded circle represents a supervisor, which is responsible for a corresponding cluster. Note that links between subordinates and their supervisors are omitted in this figure.

3 Communication Protocol

Three types of communication messages are used in MASPA: *report*, *suggestion*, and *rule*. A worker's report passes its activity data upwards to provide its supervisor with a broader view. A supervisor's report aggregates the information of reports from its subordinates. A supervisor sends its report to its adjacent supervisors at the same level in addition to its immediate supervisor (if any). The supervisor's view is based on not only the agents that it supervises (directly or indirectly) but also its neighboring supervisors. This peer-supervisor communication allows each supervisor to make rational local decisions when directions from its immediate supervisor are unavailable. To prevent supervisors from being overwhelmed and reduce the communication overhead in the network, the information is summarized (or abstracted) in reports. Furthermore, reports are only sent periodically.

Based upon this information, a supervisor employs its expertise, integrates directions from its superordinate supervisor, and provides supervisory information to its subordinates. As in human organizations, rules and suggestions are used to transmit supervisory information. We define a *rule* as a tuple $\langle c, F \rangle$, where

- c: a condition specifying a set of satisfied states
- F: a set of forbidden actions for states specified by c

[1] The top supervision level can have multiple supervisors.

A *suggestion* is defined as a tuple $\langle c, A, d \rangle$, where

- c: a condition specifying a set of satisfied states.
- A: a set of actions
- d: the suggestion degree, whose range is $[-1, 1]$.

A suggestion with a negative degree, called a *negative suggestion*, urges a subordinate not to do the specified actions. In contrast, a suggestion with a positive degree, called a *positive suggestion*, encourages a subordinate to do the specified action. The greater the absolute value of the suggestion degree, the stronger the impact of the suggestion on the supervised agent.

Each rule contains a condition specifying states where it can be applied. Subordinates are required to obey rules from their supervisors. Due to their imperativeness, correct rules greatly improve the system efficiency, while incorrect rules can lead to inefficient policies. Therefore, a supervisor requires domain knowledge, in addition to information from its subordinates, to make rules that have a positive impact on the organizational performance.

Rules are "hard" constraints on subordinates' behavior. In contrast, suggestions are "soft" constraints and allow a supervisor to express its preference for subordinates' behavior. In our example use, a suggestion have a condition matching all states. A supervisor knows that the system performance benefits from a subordinate doing a particular action more frequently. However, due to limited domain knowledge or limited information about the subordinate's local policy and surrounding environment, a supervisor can only suggest the subordinate to do more of that action without telling when it should and when it should not. Therefore, a subordinate does not rigidly adopt suggestions. The effect of a suggestion on a subordinate's local decision making may vary, depending on its current policy and state. A supervisor will refine or cancel rules and suggestions as new or updated information from its subordinates become available.

A set of rules are in conflict if they forbid all possible actions on some state(s). Two suggestions are in conflict if one is positive and the other is negative and they share some state(s) and action(s). A rule conflicts with a suggestion if a state-action pair is forbidden by the rule but is encouraged by the suggestion. In our supervision mechanism, we assume each supervisor is rational and will not generate rules and suggestions that are in conflict. However, in a multi-level supervision structure, a supervisor's local decision may conflict with its superordinate direction. Rules have higher priority than suggestions. There are several strategies for resolving conflicts between rules or between suggestions, such as always taking its superordinate or local rule, stochastically selecting a rule, or requesting additional information to make a decision. The strategy choice depends on the application domain. Note that it may not always be wise to select the superordinate decision, because, although the superordinate supervisor has a broader view, its decision is based on abstracted information. Our strategy for resolving conflicts picks the most constraining rule and combines suggestions by summing the degrees of the strongest positive suggestion and the strongest negative suggestion.

The supervisory organization defined above is robust, scalable, and immune to single-point failures, because agents at each level are fully-distributed and able to make local decision without the supervision of higher-level agents. Meanwhile, the supervision mechanism allows subordinates to utilize a more global view through rules and suggestions from their supervisors in making more informed local decisions.

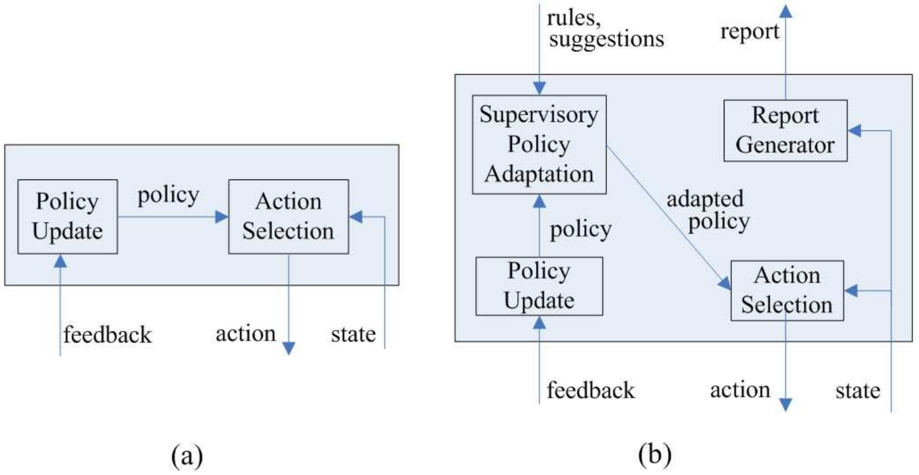

Figure 2: Unsupervised MARL vs. Supervised MARL with MASPA

4 Supervisory Policy Adaptation

Using MARL, each agent gradually improves its action policy as it interacts with other agents and the environment. A *pure* policy deterministically chooses one action for each state. A *mixed or stochastic* policy specifies a probability distribution over the available actions for each state. Both can be represented as a function $\pi(s, a)$, which specifies the probability that an agent will execute action a at state s. As argued in [10], mixed policies can work better than pure policies in partially observable environments, if both are limited to act based on the current percept. Due to partial observability, most MARL algorithms are designed to learn mixed policies. The rest of this section shows how mixed policy MARL algorithms can take advantage of higher-level information specified by rules and suggestions to speed up convergence.

As shown in Figure 2 (a), a typical unsupervised MARL algorithm contains two components: policy (or action-value function) update and action selection based on the learned policy. One common method to speed up learning is to supply an agent with additional reward to encourage some particular actions [6], which is called reward shaping. This use of the special reward affects both policy update and action selection. In a single-agent setting, there are potential function forms of reward shaping [6] that leave the optimal policy/value-function unchanged. However, due to the non-stationary learning environment in a multi-agent setting, reward shaping may generate a policy that is undesirable in that they may distract from the main goal, which is supported by the normal reward.

MASPA directly biases the action selection for exploration without changing the policy update process. As shown in Figure 2 (b), MASPA' supervisory policy adaptation integrates rules and suggestions into the policy learned by an unsupervised MARL algorithm and then outputs an adapted policy. This adapted policy is intended to control exploration. Our integration assumes policies learned by an unsupervised MARL are stochastic. The report generator summarizes the states that the agent has experienced and sends this abstracted state to its supervisor.

Let R and G be the rule set and suggestion set, respectively, that a worker received and π be its learned policy. We define $R(s, a) = \{r \in R|$ state s satisfies the condition

$r.c$ and $a \in r.F\}$[2] and $G(s,a) = \{g \in G|$ state s satisfies the condition $g.c$ and $a \in g.A\}$. Then the adapted policy π^A for the action selection is generated by the supervisory policy adaptation:

$$\pi^A(s,a) = \begin{cases} 0 & \text{if } R(s,a) \neq \emptyset \\ \pi(s,a) * (1 + \eta(s) * \; deg(s,a)) & \text{else if } deg(s,a) \leq 0 \\ \pi(s,a) + (1 - \pi(s,a)) * \; \eta(s) * deg(s,a) & \text{else if } deg(s,a) > 0 \end{cases}$$

where $\eta(s)$ is a state-dependent function ranging from $[0,1]$, and the function $deg(s,a)$ returns the degree of the satisfied suggestion. One assumption is that a MARL's state contains enough information for checking whether a rule (or suggestion) is satisfied or not.

We define the function $deg(s,a) = max(\{g.d > 0|g \in G(s,a)\}) + min(\{g.d < 0|g \in G(s,a)\})$.[3] With this definition, a subordinate only considers the strongest suggestions, either positive or negative. Conflicting suggestions are integrated by summing the degrees of the strongest positive one and the strongest negative one. Note that this function ensures the degree ranging in $[0,1]$.

As similarly defined in the work [8], the function $\eta(s)$ determines the receptivity for suggestions and allows the agent to selectively accept suggestions based on its current state. For instance, if an agent becomes more confident in the effectiveness of its local policy on state s because it has more experience with it, then $\eta(s)$ decreases as learning progresses. In our experiments, we set $\eta(s) = k/(k + visits(s))$ where k is a constant and $visits(s)$ returns the number of visits on the state s.

With the supervisory policy adaptation, a rule explicitly specifies undesirable actions for some states and is used to prune the state-action space. Suggestions, on the other hand, are used to bias agent exploration. To integrate suggestions into MARL , MASPA uses the strategy that the lower the probability of a state-action pair, the greater the effect a positive suggestion has on the pair and the less the effect a negative suggestion has on it. The underlying idea is intuitive. If the agent's local policy already agrees with the supervisor's suggestions, as indicated by the policy having high (or low) probabilities for state-action pairs from the positive (or negative) suggestions, it is going to change its local policy very little (if at all); otherwise, the agent follows the supervisor's suggestions and makes a more significant change to its local policy.

To normalize π^A such that it sums to 1 for each state, the ***limit*** function from GIGA [17] is applied with minor modifications so that every action is explored with minimum probability ϵ:

$$\pi^A = limit(\pi^A) = argmin_{x:valid(x)}|\pi^A - x|$$

i.e., $limit(\pi^A)$ returns a valid policy that is closest to π^A.

Our normalization also implicitly solves the issue of rules in conflict. If a set of rules forbids all actions on a state, then the probability of each action is set to 0. After normalization, the probabilities of all actions are equal, that is, the action choice becomes completely random. This strategy is reasonable when the agent does not know the consequence of violating each rule.

Although a supervisor provides directions to its subordinates via rules and suggestions as defined above, instead of explicit policies, the following proposition holds with the integration developed above.

[2] We use "." as a projection operator. For example, $r.c$ returns the rule condition of rule r.

[3] If $G(s,a)$ is empty, then $deg(s,a) = 0$.

Proposition 1. *If a supervisor knows the optimal policy for each subordinate and each subordinate completely trusts the supervisor's suggestions (that is, $\eta(s) = 1$, for all state s), then the supervisor can force each subordinate to execute the optimal policy via the adapted policy π^A.*

Proof. The supervisor requires each subordinate to send its local policy to it via report messages. Consider an arbitrary subordinate. Let π^* be the optimal policy of this subordinate, and π be its current local policy. To force this subordinate to execute the optimal policy, for each state-action pair (s, a), the supervisor sends a suggestion $< s, \{a\}, d >$ to this subordinate, where the suggestion degree d is defined as following:

Case 1 if $\pi^*(s, a) \leq \pi(s, a)$, then $d = \pi^*(s, a)/\pi(s, a) - 1$. Since $-1 \leq d \leq 0$, then $\pi^A(s, a) = \pi(s, a)*(1+\eta(s)*deg(s, a)) = \pi(s, a)*(1+\pi^*(s, a)/\pi(s, a)-1) = \pi^*(s, a)$.

Case 2 if $\pi^*(s, a) > \pi(s, a)$, then $d = (\pi^*(s, a) - \pi(s, a))/(1 - \pi(s, a))$. Since $0 < d \leq 1$, then $\pi^A(s, a) = \pi(s, a) + \eta(s) * deg(s, a) * (1 - \pi(s, a)) = \pi(s, a) + (1 - \pi(s, a)) * (\pi^*(s, a) - \pi(s, a))/(1 - \pi(s, a)) = \pi^*(s, a)$.

Note that the proof does not need to use this *limit* function, because the resulting π^A is already valid. Therefore, with the supervisor's suggestion, the action choice of this subordinate is based on the optimal policy. □

Based upon the mechanism developed above for integrating suggestions and rules into the learning process, both MARL and the organization supervision mechanism can affect each other. Rules and suggestions provide bias for the action choice during exploration and speed up the learning process. In turn, workers improve their performance through learning and provide supervisors with new information to refine rules and suggestions. Due to the pruning effect of rules, supervisors need to have a mechanism to detect if a rule is overconstraining and then to refine the rule to allow workers to properly explore the environment.

5 Experimental Results

We have tested MASPA in both distributed task allocation problem (DTAP) and network routing problem. In the following experiments, we manually cluster agents in the overlay network using Manhattan distance. The agent closest to the center of each cluster is elected as the supervisor. Supervisors also play the worker role. We assume that there are links that allows direct communication between subordinates and their supervisors and between adjacent supervisors.

5.1 Distributed Task Allocation Problem

To evaluate MASPA, we extended the simulator of a simplified DTAP [2] to incorporate Poisson task arrival and exponential task service time. In the DTAP, agents are organized in an overlay network. Agent i executes tasks with rate ω_i work units per time unit and receives tasks from the environment with rate λ_i tasks per time unit, where tasks' work units are under a exponential distribution with mean μ_i. At each time unit, an agent makes a decision for each task received during this time unit whether to execute the task locally or send it to a neighboring agent for processing. A task to be executed locally will be added to the local queue with unlimited queue length, where tasks are executed on a first-come-first-serve basis. Agents interact via communication messages and communication delay between two agents is proportional to the Manhattan distance between them, one time unit per distance unit (each agent has a physical location). The main goal of DTAP is to minimize the total service time of all tasks, averaged by the number of tasks, $ATST = \frac{\sum_{T \in \bar{T}_\tau} TST(T)}{|\bar{T}_\tau|}$, where $\bar{T}_\tau$ is the set of tasks received during a time period τ and $TST(T)$ is the total service time that task T spends in the system, which includes the routing time in the network, waiting time in the local queue, and execution time.

5.1.1 Implementation

Algorithm 1: WPL: Weighted Policy Learner

```
begin
    $r \leftarrow$ the reward for action $a$ at state $s$
    update Q-value table using $< s, a, r >$
    $\bar{r} \leftarrow$ average reward $= \sum_{a \in A} \pi(s,a)Q(s,a)$
    foreach action $a \in A$ do
        $\Delta(a) \leftarrow Q(s,a) - \bar{r}$
        if $\Delta(a) > 0$ then $\Delta(a) \leftarrow \Delta(a)(1 - \pi(a))$
        else $\Delta(a) \leftarrow \Delta(a)(\pi(a))$
    end
    $\pi \leftarrow limit(\pi + \zeta\Delta)$
end
```

Workers use the Weighted Policy Learner (WPL) algorithm [1] to learn task allocation policies. Note that MASPA does not depend on a specific MARL and the only requirement is that the MARL can learn mixed policies. Algorithm 1 describes the policy update rule of WPL. WPL is a gradient ascent algorithm which is based on the following strategy: learn fastest when the policy gradient Δ changes its direction and gradually slow down learning if the gradient remains in the same direction. A worker's state is defined by a tuple $\langle l_c, \beta, \tilde{S}_1, \ldots, \tilde{S}_n \rangle$, where l_c is the current work load (or total work units) in the local queue, β is the rate of incoming task requests, and $\tilde{S}_i$ is the expected service time of a task if forwarded to neighbor i. The state space is continuous and is dynamically discretized with the maximum and minimum values of each vector component, which are updated periodically during the learning. The reward for forwarding a task to neighbor i is $-\tilde{S}_i$.

Algorithm 2 shows the decision process that takes place at each worker on every cycle. There are three types of messages generated by a worker: *result*, *request*, and *report*. A *result* message $\langle i, T, t \rangle$ indicates that task T is completed at time t after being sent to neighbor i, and is used to calculate $TST(T)$ and update $\tilde{S}_i$ in the current state with the following equation (adopted from Q-learning [15]): $\tilde{S}_i = \alpha * \tilde{S}_i + (1 - \alpha) * TST(T)$, where α is the decay rate. A result message for a task will be passed back to all agents on the task routing path.[4] A *request* message $\langle i, T_i \rangle$ indicates a request from neighbor i to execute task T. A *report* $\langle i, l, n, \tau \rangle$ is generated by agent i that consists of the average work load l of the workers over a period time τ and the number

[4] The state update mechanism proposed in [2] can reduce the number of messages. This paper mainly focuses on the supervision mechanism and the use of this feedback mechanism can help eliminate other potential factors that affect the system performance.

Algorithm 2: Worker's Decision Making Algorithm

```
begin
    $n \leftarrow$ the identity of the worker
    $t_c \leftarrow$ the current time
    if a task $T$ in the local queue is done then
        send result $\langle n, T, t_c \rangle$ to the $T$'s sender
    end
    $MSGS \leftarrow$ messages received in this cycle
    foreach result $\langle i, T, t \rangle \in MSGS$ do
        update the current state $s$
        if $T$ is received from agent $j$ ($j \neq n$) then
            send result message $\langle n, T, t \rangle$ to $j$
        end
    end
    Use rules and suggestions from $MSGS$ to update the integrated policy $\pi_i^{AC}$
    foreach request $\langle i, T_i \rangle \in MSGS$ do
        choose and execute an action $a$ based on $\pi_i^{AC}$
        update the current state $s$
        $learn(s, a)$
    end
    collect work load information in the local queue
    if $t_c$ is a reporting time then
        generate and send a report to its supervisor
    end
end
```

of workers n (which is 1 in a worker report). It is possible that more information, such as average utilization and task arriving rates, can be added to allow supervisors to make more informed decisions. An agent sends a report to its supervisor every τ time period.

Algorithm 3: Supervisor's Decision Making Algorithm

```
begin
    $sr$ keeps the latest rule received from its superordinate supervisor
    $ss$ keeps the latest suggestions received from its superordinate supervisor
    if all subordinates' reports for the current period are received then
        generate an aggregated report $rep$
        add $rep$ to repList
        $r \leftarrow generateRule(repList)$
        $r \leftarrow combine(r, sr)$
        $distribute(r)$
        send $rep$ to all peer supervisors and its superordinate supervisor
    end
    if all peer supervisors' reports for the current period are received then
        $hc \leftarrow$ clusters with higher average load
        $lc \leftarrow$ clusters with lower average load
        foreach cluster $c \in hc$ do
            $sendNegativeSuggestions(c, ss)$
        end
        stochastically choose one cluster $c$ in $lc$ and
        $sendPositiveSuggestion(c, ss)$
    end
end
```

Algorithm 3 shows the decision process that takes place at each supervisor on every cycle. Three types of messages are generated by supervisors: *report*, *rule*, and *suggestion*. The creation of both reports and rules are based on subordinates' reports. Let $reps$ be the set of reports from subordinates. A supervisor's report sp aggregates data in subordinates' reports, where $sp.n = \sum_{r \in reps} r.n$, $sp.l = \sum_{r \in reps}(r.l * r.n)/sp.n$ and $sp.\tau = r.\tau$ for an $r \in reps$.

We define one rule for DTAP, called *load limit rule* $\langle limit \rangle$, that specifies, for all states whose work load exceeds $limit$, a worker should not add a new task to the local queue. The *limit* is set with the information about the average load within a cluster, so this rule helps balance load within the cluster. On the other hand, since the worker's state contains the load information, this rule can reduce the state-action space for the MARL exploration. To generate a stable and accurate rule, a supervisor keeps its own aggregated reports in $repList$, a fixed-length list. The function $generateRule(repList)$ returns a load limit rule $\langle limit \rangle$, where $limit = \sum_{r \in repList} r.l/m$ and m is the size of $repList$. The function $combine(r, rs)$ chooses a more constrained rule (i.e., with a lower load limit) between the local rule r and the superordinate rule sr. The function $distribute(r)$ sends rule r to subordinates. In order to avoid excessive sending of rule messages, a supervisor sends a new rule iff the difference of load limits between the new rule and the current rule exceeds a certain threshold.

The load limit rule forbids adding a task to the local queue only if the current load is already greater than the limit. Therefore, it is possible that the work load in local queue of a worker is greater than the load limit. For example, suppose agents in a cluster do not receive external tasks by themselves. Initially, the cluster has few tasks forwarded from its neighboring clusters and thus has a very low average load (e.g., 3), from which a rule is generated. As time goes on, more and more tasks are forwarded to the cluster and each agent is more likely to add tasks to its local queue, whose load is close to 3. As a result, each local load is frequently greater than the load limit and the average load of the cluster increases. From report messages, the supervisor can detect that the current rule is over-constraining and generate a less constraining rule with a higher load limit.

We utilize suggestions to balance the load across clusters. The creation of suggestions is based on reports from peer supervisors. Let rep_i and rep_k be the report of supervisor i and its neighboring supervisor k respectively, c_i and c_k be the cluster supervised by supervisor i and k respectively, m_i be the number of subordinates of cluster c_i that are adjacent to cluster c_k, and com_cost_k be the communication cost between supervisor i and supervisor k, which can be estimated from the communication between them.

If $rep_k.l - rep_i.l > 0$, supervisor i considers cluster c_k having a higher load. Function $sendNegativeSuggestions(c_k, ss)$ creates a negative suggestion with degree $nd = (rep_i.l/rep_k.l$ $-1)/m_i$ to discourage forwarding tasks to cluster c_k, combining it with the matching suggestion (if exists) in ss from its superordinate supervisor, and then sending the combined suggestion to subordinates adjacent to cluster c. Two suggestions match if they share the same action set (i.e., both local decision and superordinate decision suggest forwarding tasks to cluster c_k) and some state(s). Our combination strategy is that if the degrees of two matching suggestions have the same sign, the integrated suggestion uses the degree of the stronger suggestion; otherwise, it uses the sum of two degrees.

If $diff_k = rep_i.l - rep_k.l - com\_cost_k > 0$, then cluster c_i considers cluster c_k has a lower average load. In order to avoid "hot spot" problems, supervisor i probabilistically selects one from the set of neighboring clusters with lower load, where the probability of selecting cluster c_k is given by $Pr(k) = \frac{diff_k}{\sum_{n \in neighbors(i)} diff_n}$, where the function $neighbors(i)$ returns all neighboring clusters of supervisor i. Function $sendPositiveSuggestion(c_k, ss)$ does the same thing as $sendNegativeSuggestions(c_k, ss)$ except that it sends a positive suggestion with degree $pd = diff_k/rep_i.l/m_i$. To strengthen the effect of suggestions, if a suggestion with degree d is sent to subordinate j and its neighbor n doesn't receive the same suggestion, then a suggestion with degree ξd and action j is sent to n, where ξ is the suggestion decay rate. To reduce network overhead, a suggestion with degree less than a threshold (e.g., 0.05) will not be sent to subordinates.

5.1.2 Results & Discussions

We have tested MASPA in the DTAP simulation mentioned above, where three measurements are evaluated: the average total service time (ATST), the average number of messages (AMSG) per task, and the time of convergence (TOC). ATST indicates the overall system performance, which can reflect the effectiveness of learning and supervision mechanism and can also be used to verify system stability (convergence) by showing monotonic decrease in ATST as agents gain more experiences. AMSG shows the overall communication overhead for finishing one task. To calculate TOC, we take 10 sequential ATST values and then calculate the ratio of those values' deviation to their mean. If the ratio is less than a threshold (we use 0.025), then we consider the system stable. TOC is the start time of the selected points.

MASPA does not pose any constraint on the network structure. However, as mentioned, we do implicitly assume the system is nearly-decomposable with a hierarchy of at least one level. For clarity, Experiments were conducted using uniform two-dimension grid networks of agents with different sizes: 6x6, 10x10, and 27x27, all of which show similar results. But as the size of the system increases, the MASPA impact on the system performance becomes greater. For brevity, we only present here the results for the 27x27 grid. In each simulation run, ATST and AMSG are computed every 1000 time units to measure the progress of the system performance. Results are then averaged over 10 simulation runs and the variance is computed across the runs. All agents use WPL with learning rate 0.001. Workers send reports to their supervisors every 500 time units. Our experiments use the parameter $\eta(s) = 1000/(1000+visits(s))$ and the suggestion decay rate $\xi = 0.5$.

For simplicity, we assume that all agents have the same execution rate, $\forall i : \omega_i = 1$, and that tasks are not decomposable. The service time of tasks are under a Poisson distribution with mean $\mu = 10$. We tested three patterns of task arrival rates over the 27x27 grid:

Boundary Load where the 200 outermost agents receive tasks with $\lambda = 0.33$ and other agents receive no tasks from the external environment.

Center Load where the 121 agents in the centric 11x11 grid receive tasks with $\lambda = 0.5$ and other agents receive no tasks from the external environment.

Uniform Load where all 729 agents receive tasks with $\lambda = 0.09$.

We compared four structures: *no supervision, local supervision, one-level supervision, and two-level supervision*. In the *local supervision* structure, agents are their own supervisors. With this structure, each agent gains a view only about itself and its neighbors, which is not much different from its view in the organization without supervision. So we use the *local supervision* structure to evaluate whether domain knowledge combined with a limited view, which is used to create rules and suggestions, still improves the system performance. In contrast, the performance of the two following structures with supervision show the benefits of having a broader view combined with domain knowledge. The *one-level supervision* structure has 81 clusters, each of which is a 3x3 grid and the agent at each cluster center is elected as the supervisor. The *two-level supervision* structure forms from the *one-level supervision* structure by grouping 81 supervisors into 9 clusters, each of which is a 3x3 grid. The supervision structures with three or more levels did not show further improvement over the two-level supervision in our DTAP experiments. This is because a wide-range task transfer causes a long routing time which offsets the reduction of the queuing time in each agent.

Figure 3 plots the trend of ATST for different structures as agents learn. As expected, systems with *one-level supervision* or *two-level supervision* converge much faster than that without supervision. The system with *two-level supervision* performs better than with *one-level supervision*, because bottom-level supervisors create more accurate rules and suggestions for workers by combining local decisions with superordinate decisions which are based on a broader view. But as the system stabilizes, the system load tends to be smoothly distributed among the agents and the broader view of higher-level supervisors does not provide more information than that of lower-level supervisors. Therefore *two-level supervision* and *one-level supervision* show almost the same performance after stabilization.

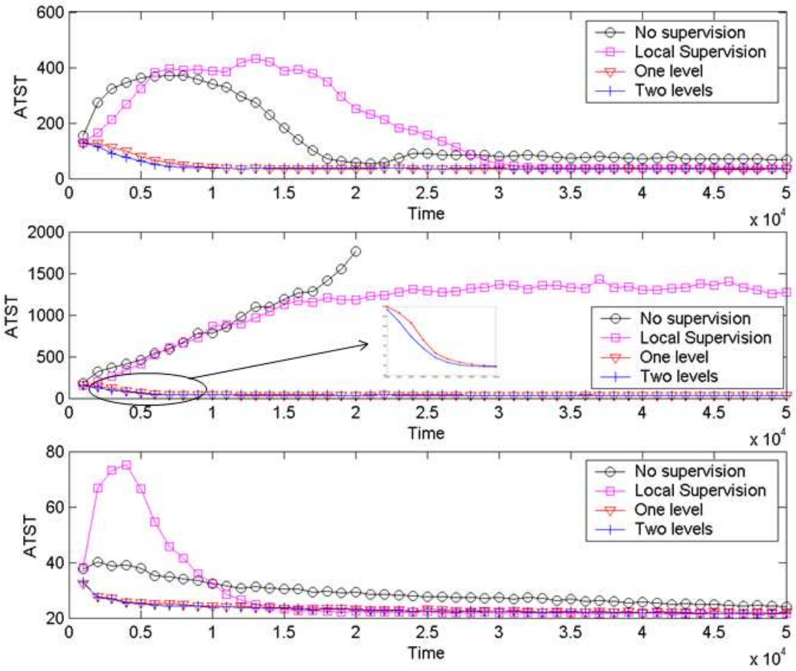

Figure 3: ATST for different structures, boundary load: top, center load: medium, uniform load: bottom

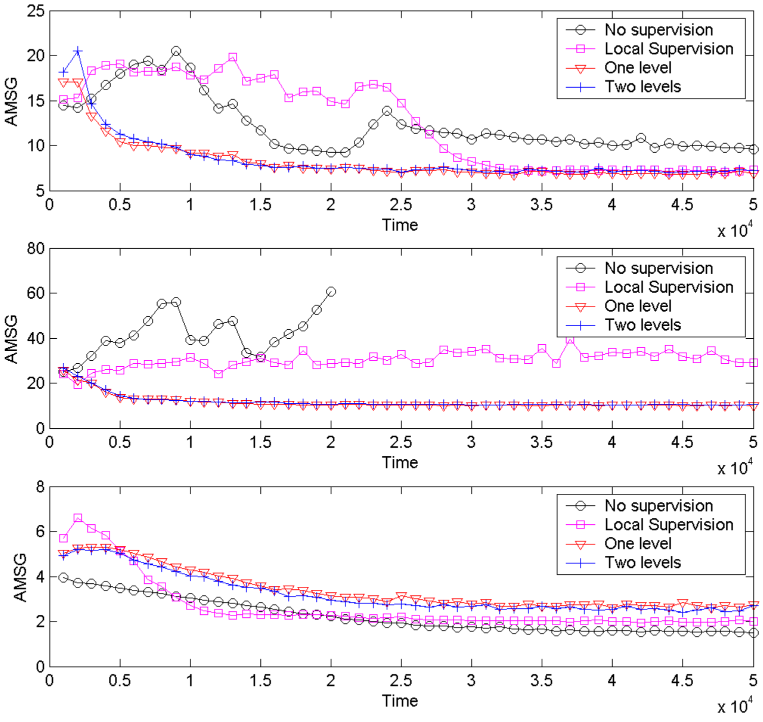

Figure 4: AMSG for different structures, boundary load: top, center load: medium, uniform load: bottom

Interestingly, *local supervision* improves its performance only after a certain period of time, and at an early stage, it may even decrease system performance. With *local supervision*, each worker is a supervisor, so a supervisor's suggestion is based only on the load information of its immediate neighboring workers, which can be incorrect at early stages. For example, worker A with a high load has two neighbors worker B, with a low load, and worker C, with a high load. As a result, worker A will create a positive suggestion to itself to send more tasks to worker B and a negative suggestion to send less tasks to worker C. In fact, all other neighbors of worker B have a very high load and all other neighbors of work C have a very low load. Misleading suggestions based on these incorrect information cause oscillation in worker policies and severely degrade the normal learning process, resulting in a decreased performance. However, as time passes, each agent learns a better policy; meanwhile, $\eta(s)$ decreases and suggestions have a less impact on the action choice. On the other hand, the load limit rule, based on its own load history, can reduce the exploration space, resulting in faster convergence.

Under the boundary-load and uniform-load pattern, all systems show monotonic decrease in ATST after a certain period of time, which indicates the stability (convergence) of these systems. However, under the center-load pattern, the system without supervision crashes and runs out of the computing resources before showing signs of convergence. This happens because, using random exploration, agents in the inner layer do not learn and propagate quickly enough knowledge that agents in the outside layer are light-loaded. As a result, more and more tasks loop and reside in the center 11x11 grid where agents receive external tasks. This makes the system load severely unbalanced and the system capability not well utilized. In contrast, the supervisory information guides and coordinates the exploration of agents and allows them to learn quickly to effectively route tasks.

Under the uniform-load pattern, the system load is actually not evenly distributed, with a higher load around the center and a lower load on the boundary, but the load difference is not as significant as that under boundary-load and center-load patterns. Therefore supervision with a broader view improves the performance, though not as significantly.

Figure 4 illustrates the communication overhead for different structures. Initially, the system without supervision has lower AMSG. This is because supervision mechanism increases the communication overhead for sending reports, rules and suggestions and its encouragement of exploration at the early stage also increases the number of *request* and *result* messages. However, under the boundary-load and center-load patterns, the supervision mechanism leads workers to learn how to route tasks effectively to balance the load much more quickly, which dramatically reduces the number of *request* and *result* messages. As a result, these systems with supervision mechanism obtain lower AMSG after a short period, as shown in the Figure 4. Under the uniform-load pattern, the system does not benefit enough from supervision mechanism to offset the communication overhead caused by the supervision mechanism.

Table 1, Table 2, and Table 3 show the different measures after agents have learned for 100000 time units. Here ATST and AMSG are measured for each supervision structure at their own convergence time point. Although, in some case, *two-level supervision* has a slightly higher ATST than that of *one-level supervision* at their own convergence time points, it has actually a lower or almost same ATST *one-level supervision* when evaluated at the same time point.

Supervision	ATST	AMSG	TOC
No	60.75 ± 1.10	8.80 ± 0.22	61000
Local	37.44 ± 0.51	7.27 ± 0.08	37000
One-level	35.38 ± 0.64	7.39 ± 0.24	16000
Two-level	35.96 ± 0.62	7.56 ± 0.17	14000

Table 1: Performance of different structures with boundary load

Supervision	ATST	AMSG	TOC
No	N/A	N/A	N/A
Local	1328 ± 33	32.89 ± 3.15	30000
One-level	36.95 ± 0.45	10.24 ± 0.17	14000
Two-level	37.12 ± 0.81	11.07 ± 0.45	12000

Table 2: Performance of different structures with center load

To show the robustness of the multi-level supervision, we evaluated MASPA when not all supervisors worked properly. Each supervisor has a probability *fp* of failing to function during a report period. We assume that, when its supervisor fail to function, a worker will use rules and suggestions last received from its supervisor. We tested both one-level supervision and two-level supervision and they showed similar results.

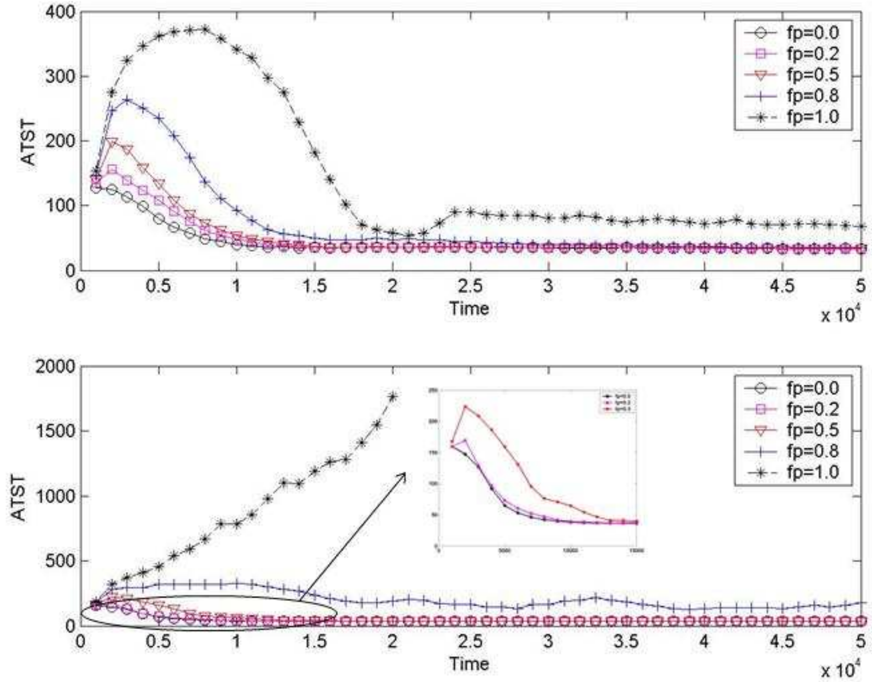

Figure 5: Performance with different failure probabilities of supervisors, boundary load: top, center load: bottom

Supervision	ATST	AMSG	TOC
No	28.57 ± 0.68	1.89 ± 0.13	21000
Local	22.36 ± 0.42	2.17 ± 0.08	19000
One-level	24.46 ± 0.61	3.83 ± 0.38	9000
Two-level	24.34 ± 0.59	3.75 ± 0.41	8000

Table 3: Performance of different structures with uniform load

Figure 5 shows the performance of one-level supervision with difference *fp* values. It can be seen that MASPA still improves the learning in a reasonable degree when each supervisor works properly with only one half probability[5].

In our simulation, we observed that supervisory information corresponding to coarse-grained control tend to be more helpful than that corresponding to fine-grained control in improving the system performance. Moreover, fine-grained may even decrease system performance. Coarse-grained control considers and operates on the whole cluster as one entity, while fine-grained control operates on individual cluster members. "Moving more tasks from my cluster to one of neighboring clusters" and "balancing the load within the cluster" are examples of coarse-grained control . "Moving more tasks from a high-loaded agent to a low-loaded agent along the shortest path" is an example of fine-grained control. One explanation for this observation is that supervisory information corresponding to coarse-grained control results in more coordination among agents' exploration, speeding up the learning convergence. In contrast, in our simulation, due to lack of detailed information of each cluster member, fine-grained control for some individual members is not able to fully evaluate the impact on and from other agents. As a result, the fine-grained control may interfere with the normal learning process of other agents and the dynamics of other agents may degrade the fine-grained control.

We have explored different values of cluster size and found that system performance decreases with cluster size that are either too small or too large . This is because, with too small a cluster size, supervisors do not collect enough information to create correct rules and suggestions. With too large a cluster size, they are not able to create rules and suggestions that are suitable for every subordinate. Therefore, there is a trade-off for the cluster size.

Similarly, there is a trade-off for the length of the report period. A too short report period causes a large variance of the abstracted state (also increases communication overhead) and results in oscillating suggestions and rules. A too long report period causes the supervisory information received by workers to be out-dated and as a result, decreases the convergence rate.

5.2 Network Routing

We also evaluate our framework using a network routing simulator adopted from Boyan and Littman [4]. It is a discrete time simulator of communication networks with various topologies. A communication network consists of a homogeneous set of nodes (or agents) and links between them. Packets are periodically introduced into the network under a Poisson distribution with a random origin and destination. No packets have the same agent as their origin and destination. When a packet arrives at an agent, the agent puts it into the local FIFO (first in first out) queue. At each time, an agent makes its routing decision to forward the top packet in the queue to one of its neighbors. Once a

[5] This result may compromise when the task arriving pattern is changing continuously

packet reaches its destination, it is removed from the network. In our experiments, we set the time cost of sending a packet down a link as a unit cost. So the delivery time of packet consists of its transmission cost and its waiting time in queues. The main goal of a network routing algorithm is to minimize the Average Delivery Time (ADT) of all packets.

5.2.1 Implementation

Algorithm 4: Policy Gradient Descent (PGD) Algorithm

```
begin
    $r \leftarrow$ the cost for action $a$ at state $s$
    update Q-value table using $< s, a, r >$
    $t \leftarrow$ summed cost $= \sum_{a \in A} Q(s, a)$
    $\bar{r} \leftarrow$ average cost $= \sum_{a \in A} \pi(s, a) Q(s, a)$
    foreach action $a \in A$ do
        $\Delta(s, a) \leftarrow \zeta(Q(s, a) - \bar{r})/t$
    end
    $\pi(s) \leftarrow limit(\pi(s) - \Delta(s))$
end
```

To minimize the time cost of delivering packets, each agent uses a Policy Gradient Descent(PGD) algorithm to learn its routing policies. Algorithm 4 describes its policy update rule, where ζ is the policy learning rate. PGD learns stochastic policies, but, unlike multi-agent OLPOMPD [14] and GAPS [7], PGD does not require a global reward signal. The state s is defined by the destination of the packet that an agent is forwarding. We define $Q_x(s, a)$ as the estimated time that an agent x takes to deliver a packet to the destination s through its neighbor a, including any time that the packet would have to spend in the agent x's queue. Upon sending a packet to a, x immediately gets back a's estimate for the time remaining in the trip, namely

$$q_a = \min_{a \in \text{neighbors of} y} Q_y(s, a)$$

Then the "cost signal" $r(s, a)$ for forwarding a packet with destination s to its neighbor a is $q_a + w + t$, where w is the waiting time of the packet in x's queue and t is the transmission time between agent x and a. The Q-learning algorithm is used to update x's estimate:

$$Q_x(s, a) = (1 - \alpha) * Q_x(s, a) + \alpha * r$$

where α is a learning rate (usually 0.5 in our experiments). With updated Q-values, the PGD algorithm revises its policy.

The MASPA implementation in the network routing is similar to that in DTAP, and the main difference is the way of generating MASPA messages. In the network routing problem, we do not use rules. A worker's report contains a vector $\langle t_1, t_2, \ldots, t_m \rangle$, where t_i is the average estimated time that a worker takes to deliver a packet to destination agents in cluster i. A supervisor aggregates its subordinates' reports and generates its own report by averaging their estimated delivery time to any destination cluster.

Let t_i^d be the estimated time for cluster i to deliver a packet to destinations cluster d. Let N_c be the neighbor set of cluster c. For each cluster d, if $t_c^d > t_n^d$, where n is a neighboring cluster of c, then the supervisor of cluster c provides positive suggestions

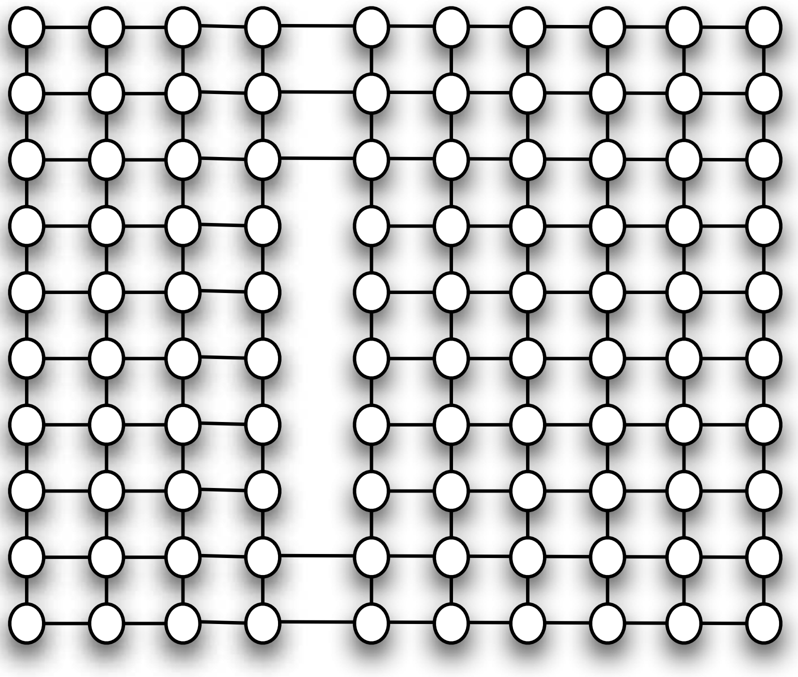

Figure 6: The 10 x 10 grid toplogy

to its boundary members to encourage forwarding packets with destinations in cluster d to cluster n. The positive suggestion degree is calculated:

$$deg = \frac{t_c^d - t_n^d}{\sum_{t_c^d > t_i^d, i \in N} (t_c^d - t_i^d)}$$

If $t_c^d < t_n^d$, then supervisor c sends out negative suggestions, whose degree is

$$deg = \frac{t_c^d - t_n^d}{\sum_{t_c^d < t_i^d, i \in N} (t_i^d - t_c^d)}$$

Similar to our implementation for DTAP, non-boundary members receive suggestions, but with decayed degrees.

5.2.2 Results & Discussions

We have tested the PGD algorithm with and without MASPA on several network topologies with various number of nodes, all of which show similar results. For brevity, we concentrate on the result analysis for the 10 x 10 grid network pictured in Figure 6. The Q-routing [4] algorithm is used as baseline, which learns deterministic policies. . Two measurements are evaluated: the average delivery time (ADT) and the time of convergence (TOC). The ADT is computed every 1000 time units. To calculate TOC,

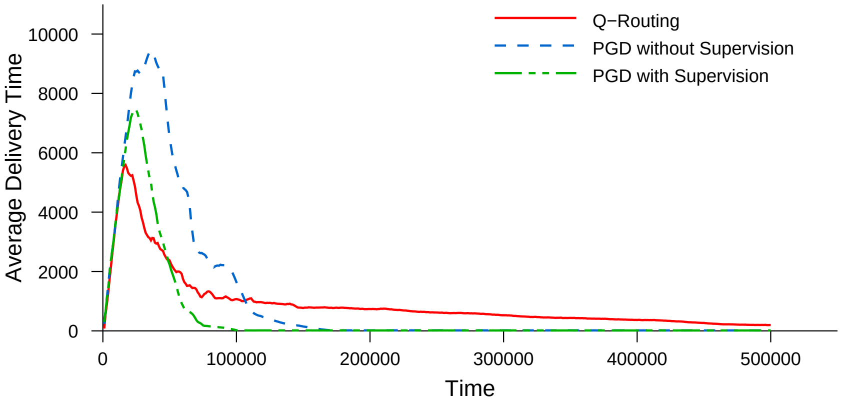

Figure 7: Performance under network load = 7.0

we take 50 sequential ADT values and then calculate the ratio of those values' deviation to their mean. If their mean is less than the maximum expected ADT (we use 300) and the ratio is less than a threshold (we use 0.05), then we consider the system stable. TOC is the start time of the selected points.

Results are then averaged over 10 simulation runs. All agents use the PGD algorithm with a learning rate $\zeta = 0.1$. Workers send reports to their supervisors every 500 time units. Our experiments use the parameter $\eta(s) = 20000/(20000 + visits(s))$ and the suggestion decay rate $\xi = 0.5$.

Figure 7 shows the performance trend as agents learn under network load= 7.0. All three algorithms, after initial periods of inefficiency during which they randomly explore the environment, gradually improve their performance and stabilize. At the very early period, MASPA does not improve the performance much. This is because, due to almost complete random exploration, subordinates do not provide accurate environment information to their supervisor, which may result in some improper suggestions. As information accuracy increases, MASPA properly biases the policy search of the PGD algorithm and speeds up the convergence. Due to policy oscillation, Q-routing shows slow convergence.

Figure 8 shows the TOC of three algorithms under various network loads. As expected, MASPA consistently speeds up the convergence of the PGD algorithm. The higher the network load, the greater the speed improvement. For example, when load ≥ 5.5, MASPA decreases the TOC by around 40% or more. Under low network loads, optimal policies usually follows shortest paths, so they are deterministic. The PGD algorithms use gradient update and gradually converge to deterministic policies, slower than Q-routing that directly learns deterministic policies. However, under high loads, where optimal policies are usually stochastic, the Q-routing policies show oscillation during the learning and the PGD algorithm with MASPA converges faster to stochastic policies.

Figure 9 shows the ADT at the convergence time point under various network loads. Under low loads, as both PGD algorithms with and without MASPA converge deterministic policies, they shows almost the same performance. Due to random exploration with some probability, they performs slightly worse than Q-routing. However, under high loads, MASPA improves the PGD performance. For example, when load ≥ 6.5,

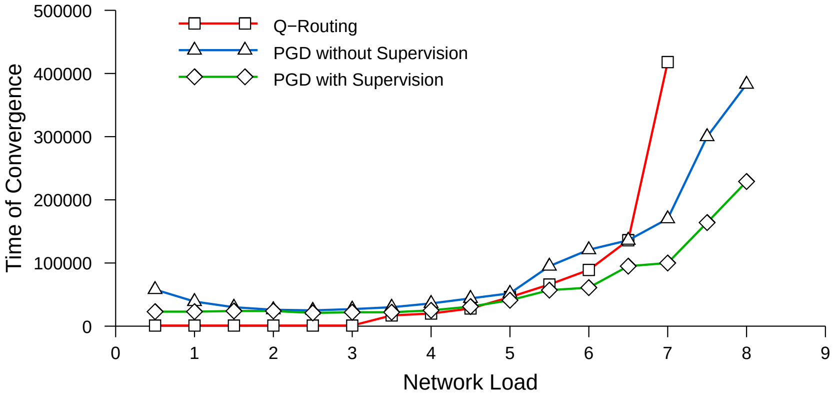

Figure 8: Time of Convergence at various loads

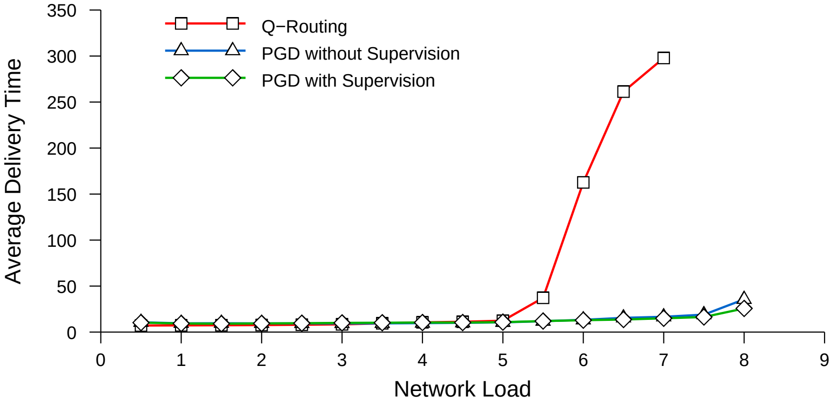

Figure 9: Delivery time at various loads

MASPA decreases the ADT by at least 10%, and when load $= 8.0$, MASPA reduces the ADT by around 30%. As both PGD algorithms converge to stochastic policies, which allows agents to simultaneously exploit multiple paths to deliver packets to a single destination, they perform much better than Q-routing under high loads.

6 Conclusion

This work presents MASPA, a scalable and robust supervision framework, that enables efficient learning in large-scale multi-agent systems. In MASPA, the automated supervision mechanism fuses activity information of lower-level agents and generates supervisory information that guides and coordinates agents' learning process. This supervision mechanism continuously interacts with the learning process. Simulation results obtained in two different application domains verify the generality of MASPA and demonstrate that MASPA significantly accelerates the learning process and reduces the communication overhead due to the earlier convergence.

Future work includes providing a distributed algorithm for forming supervision organizations (addressing agent clustering and supervisor election). The supervision mechanism generates a broader view which potentially benefits the restructuring process in the underlying network. Thus, another future direction is to explore an adaptive reorganization algorithm that exploits information from the supervision mechanism. In this work, learning only takes place in workers' decision making. It would be interesting to allow workers to learn how to integrate rules and suggestions and supervisors to learn how to make rules and provide suggestions.

References

[1] Sherief Abdallah and Victor Lesser. Learning the task allocation game. In *AAMAS'06*, 2006.

[2] Sherief Abdallah and Victor Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *AAMAS'07*, 2007.

[3] Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. R. Costa. Heuristic selection of actions in multiagent reinforcement learning. In *IJCAI'07*, Hyderabad, India, 2007.

[4] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *NIPS'94*, volume 6, pages 671–678, 1994.

[5] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Autonomous Agents'01*, pages 246–253, 2001.

[6] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *ICML'99*, pages 278–287, 1999.

[7] Leonid Peshkin and Virginia Savova. Reinforcement learning for adaptive routing. In *International Joint Conference on Neural Networks (IJCNN)*, 2002.

[8] Michael T. Rosenstein and Andrew G. Barto. Supervised actor-critic reinforcement learning. In J. Si, A. Barto, W. Powell, and D. Wunsch, editors, *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, pages 359–380. John Wiley and Sons, 2004.

[9] H. A. Simon. Nearly-decomposable systems. In *The Sciences of the Artificial*, pages 99–103. MIT Press, 1969.

[10] Satinder P. Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[11] Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In *Autonomous Agents'99*, pages 206–212, 1999.

[12] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, New York, 4th edition edition, 2003.

[13] P. Tangamchit, J. Dolan, and P. Khosla. Learning-based task allocation in decentralized multirobot systems. In *DARS'00*, pages 381–390, 2000.

[14] Nigel Tao, Jonathan Baxter, and Lex Weaver. A multi-agent policy-gradient approach to network routing. In *ICML '01*, pages 553–560, 2001.

[15] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[16] Haizheng Zhang and Victor Lesser. A reinforcement learning based distributed search algorithm for hierarchical content sharing systems. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2007.

[17] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML'03*, pages 928–936, 2003.