Improving Multi-Agent Learning through Automated Supervisory Policy Adaptation

Chongjie Zhang
Computer Science Department
University of Massachusetts Amherst

Sherief Abdullah
Institute of Informatics
British University in Dubai

Victor Lesser
Computer Science Department
University of Massachusetts Amherst
UMass Computer Science Technical Report #08-03

January 29, 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. Experimental results show that our framework increases both the likelihood and the speed of 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, 5] 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 Such latency can result in neighborhood information being outdated, hence leading to mutually inconsistent views among agents.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 [13], 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 slowly converge 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 [5].

Two paradigms have been studied to speed up the learning process. The first paradigm is to reduce the policy search space. For example, the TPOL-RL [12] reduced the state space by mapping states onto a limited number of action-dependent features. The hierarchical multi-agent reinforcement learning [6] 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 [14] 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) [8] 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 did not discuss how heuristics were constructed.

This paper presents a supervision framework, called Automated Supervisory Policy Adaptation (ASPA), to accelerate the learning. ASPA follows the second paradigm that uses heuristics to guide the policy search. The main contribution of ASPA 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., ReDVaLeR [3], WoLF-GIGA [4], 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 ASPA 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 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. The generation of broader views is based on the assumption that agents will voluntarily share their state information. Our supervision mechanism also

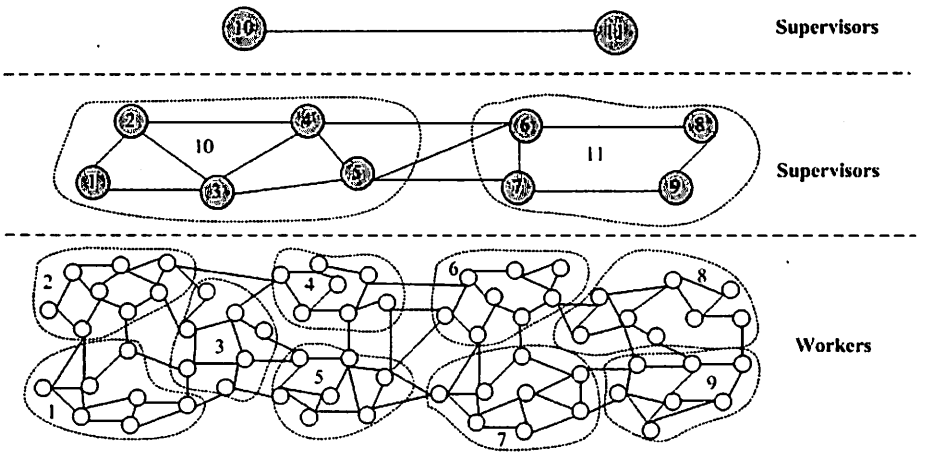


Figure 1: An organization structure for multi-level supervision

implicitly assume the original multi-agent system can be formed into a nearly decomposable hierarchy [10] of at least one level. For clarity, this paper limits the discussion to the case where learning only happens in the bottom level, but ASPA does not restrict how many levels and what levels use learning.

In order to evaluate ASPA, a simulated distributed task allocation application (DTAP) adopted from the work [2] has been constructed. We chose one representative unsupervised MARL algorithm, WPL [1], and compared its performance with and without ASPA. Simulation results show that ASPA incorporated with some simple domain knowledge not only dramatically speeds up the convergence rate, but also increases the likelihood of convergence when an unsupervised MARL algorithm fails to converge. ASPA also shows robustness when not all supervising agents work properly.

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. ASPA is then empirically evaluated on DTAP. Finally, we concludes this work and discusses some future work.

2 Organizational Supervision

The supervision mechanism commonly exists in human organizations (e.g., enterprises and governments), whose purpose is to run an organization effectively and efficiently to fulfill the organization goals. Typically, supervision involves gathering information, making decisions, and providing directions to regulate and coordinate actions of organization members. The practical effectiveness of the supervision in human organizations, especially in large organizations, inspired us to introduce a similar mechanism into multi-agent systems to improve the efficiency of MARL algorithms.

To add a supervision mechanism to a MAS with an overlay structure, ASPA 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. 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 organization. [1] While, in this paper, our discussion focuses on the situation where each agent belongs to only one cluster, ASPA is also suitable when clusters overlap.

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, *report*, *suggestion*, and *rule*, are used in ASPA. 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 (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. A *rule* is defined as a tuple $< c, F >$, where

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

A *suggestion* is defined as a tuple $< c, A, d >$, 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.

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

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.

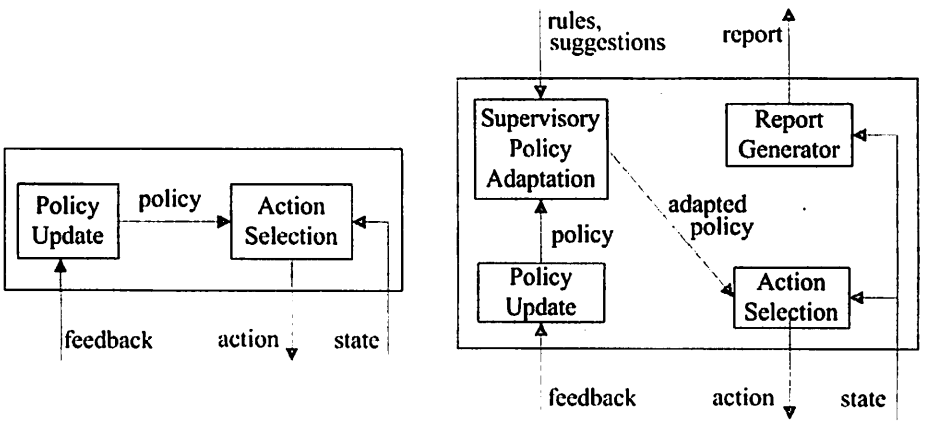
Rules are "hard" constraints on subordinates' behavior. In contrast, suggestions are "soft" constraints and allow a supervisor to express its preference for subordinates' behavior. In our example use, a suggestion have a condition matching all states. A supervisor knows that the system performance benefits 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.

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* 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 [11], 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



(a) (b)

Figure 2: Unsupervised MARL vs. Supervised MARL with ASPA

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 [7]. The use of the special reward affects both policy update and action selection. In a multi-agent context, special rewards may generate a policy that is undesirable in that they may distract from the main goal, which is supported by the normal reward.

In contrast, ASPA directly biases the action selection without changing the policy update process. Hence ASPA's effect on the final learned policy is transient (can be turned off at any time), while reward shaping has a permanent effect. As shown in Figure 2 (b), ASPA' supervisory policy adaptation integrates rules and suggestions into the learned policy and then outputs an adapted policy. This adapted policy is intended to control exploration and does not directly affect the learned policy. The report generator summarizes the states that the agent has experienced and sends this abstracted state to its supervisor.

As described previously, 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 , ASPA uses the strategy that the lower the probability of a state-action pair, the greater the effect a positive suggestion has on it 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, 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.

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

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

supervisory policy adaptation:

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

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] In the case where no clusters overlap (no suggestions are in conflict), an agent only considers the strongest suggestion, either positive or negative. This definition is also applicable to the case where an agent belongs to multiple clusters and may receive conflicting suggestions. Conflicting suggestions are integrated by summing the degrees of the strongest positive suggestion and the strongest negative suggestion.

As similarly defined in the work [9], 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 DTAP, 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.

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.

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 $\pi^A(s,a) = \pi(s,a) + \eta(s) * deg(s,a) * (1 - \pi(s,a)) = \pi(s,a) + (1 - \pi(s,a)) * (\pi^*(s,a) - \pi(s,a))/(1 - \pi(s,a)) = \pi^*(s,a)$.

Therefore, with the supervisor's suggestion, the action choice of this subordinate is based on the optimal policy. □

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

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.

Note that our normalization 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.

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

To evaluate ASPA, 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 Poisson 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 Implementation

We assume agents in the overlay network are clustered using Manhattan distance. No two clusters overlap. The agent closest to the center of each cluster is elected as the supervisor. Supervisors also play the worker role. We assume that there is a routing algorithm in the network that allows direct communication between subordinates and their supervisors and between adjacent supervisors. Workers use the Weighted Policy Learner (WPL) algorithm [1] to learn task allocation policies. Note that ASPA does not depend on a specific MARL and the only requirement is that the MARL can learn

Algorithm 1: WPL: Weighted Policy Learner

```
begin
    r ← the reward for action a at state s
    update Q-value table using < s, a, r >
    r̄ ← average reward = Σ_{a∈A} π(s,a)Q(s,a)
    foreach action a ∈ A do
        Δ(a) ← Q(s,a) − r̄
        if Δ(a) > 0 then Δ(a) ← Δ(a)(1 − π(a))
        else Δ(a) ← Δ(a)(π(a))
    end
    π ← limit(π + ζΔ)
end
```

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 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 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

[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 π_i^{AC}
  foreach request $\langle i, T_i \rangle \in MSGS$ **do**
    choose and execute an action a based on π_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

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

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.

5.2 Results & Discussions

We have tested ASPA in the DTAP simulation mentioned above, where four 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 improvement 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 median time of the selected points.

ASPA 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 ASPA 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.

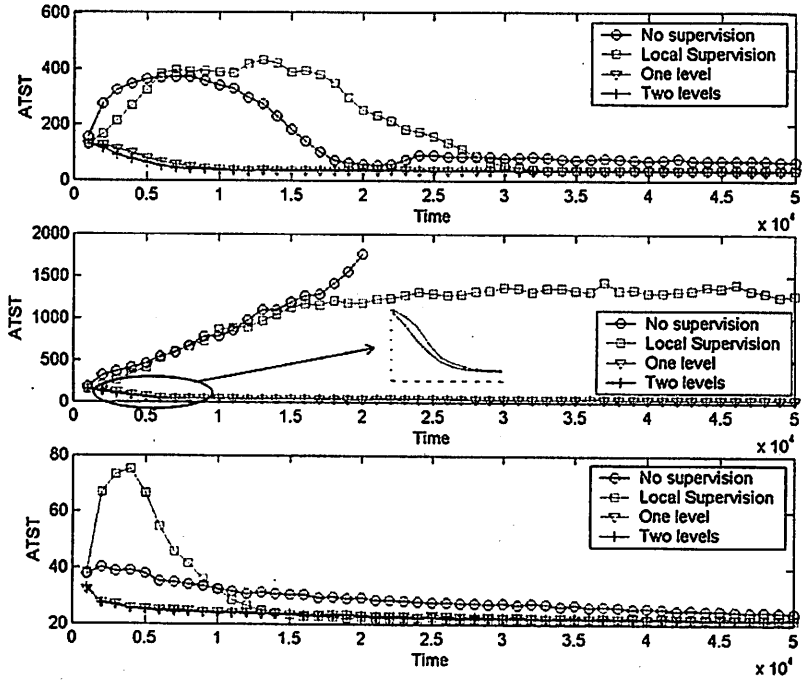


Figure 3: ATST for different structures, boundary load: top, center load: medium, uniform load: bottom

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.

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 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

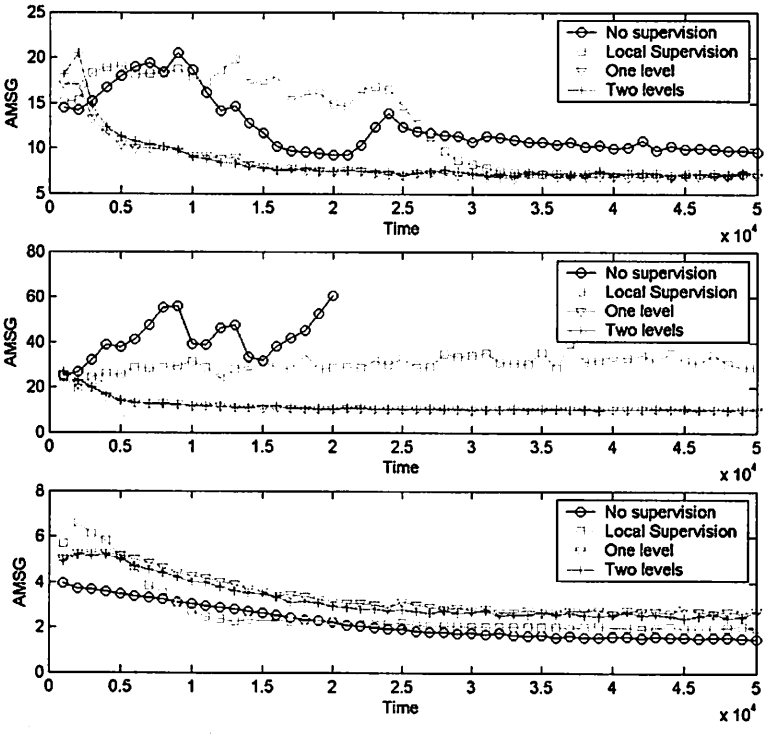


Figure 4: AMSG for different structures, boundary load: top, center load: medium, uniform load: bottom

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.

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

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

To show the robustness of the multi-level supervision, we evaluated ASPA 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 shows the performance of one-level supervision with difference fp values. It can be seen that ASPA 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

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

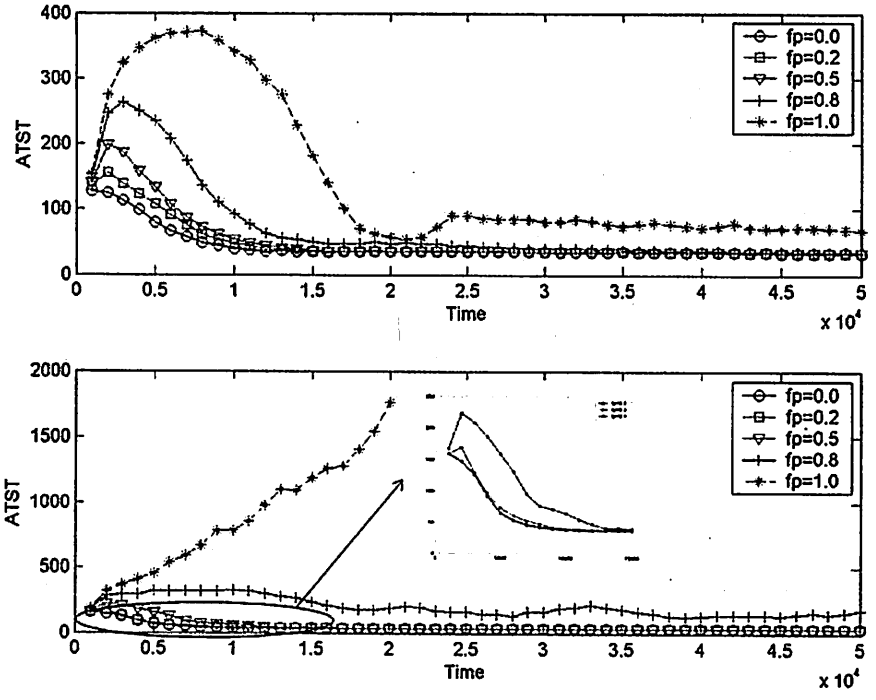


Figure 5: Performance with different failure probabilities of supervisors, boundary load: top, center load: bottom

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.

6 Conclusion

This work presents ASPA, a scalable and robust supervision framework, that enables efficient learning in large-scale multi-agent systems. In ASPA, 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 a simplified distributed task allocation problem show that ASPA significantly accelerates the learning process and reduces the communication overhead per task 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 *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[2] Sherief Abdallah and Victor Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2007.

[3] Bikramjit Banerjee and Jing Peng. Performance bounded reinforcement learning in strategic interactions. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 2–7. AAAI Press / The MIT Press, 2004.

[4] Michael Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17 (NIPS)*, pages 209–216, 2005.

[5] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

[6] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 246–253, Montreal, Canada, 2001. ACM Press.

[7] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287. Morgan Kaufmann, San Francisco, CA, 1999.

[8] Anna H. R. Costa Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro. Heuristic selection of actions in multiagent reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007.

[9] Michael T. Rosenstein and Andrew G. Barto. *Supervised Actor-critic Reinforcement Learning*, pages 359–380. Learning and Approximate Dynamic Programming: Scaling Up to the Real World. John Wiley and Sons, New York, j. si, a. barto, w. powell, and d. wunsch edition, 2004.

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

[11] 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.

[12] Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In *Proceedings of the Third International Conference on Autonomous Agents*, pages 206–212. ACM Press, 1999.

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

[14] P. Tangamchit, J. Dolan, and P. Khosla. Learning-based task allocation in decentralized multirobot systems, 2000.

[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 Tom Fawcett and Nina Mishra, editors, *ICML*, pages 928–936. AAAI Press, 2003.