An Hybrid Approach for Multiagent Learning Problem and Comparative Results

Utku Şirin

Computer Engineering Department
Middle East Technical University
Ankara, Turkey
usirin@ceng.metu.edu.tr

Keywords:multiagent planning, multiagent learning, q-learning,markov decision process, stochastic planning

Abstract

In this paper, we have studied the multiagent planning and learning problem. We have implemented a multiagent planning algorithm based on Factored Markov Decision Processes (MDPs), in which each agent runs its own value iteration algorithm and tries to maximize its own action-value function. We have also implemented three different multiagent learning algorithms. The first and second ones are known methods from the literature, namely Coordinated Reinforcement Learning *[7] and* Sparse Cooperative Q-Learning *[9], whereas, the third one is a novel, Hybrid approach combining the first and second methods together. It applies the two algorithms concurrently to different agents in the system based on the coordinations between the agents. All of the three learning algorithms is based on agent based decompositions, but uses different types of action-value update semantics. We have done several experiments on single problem instance, namely System Administrator, which defines a network of several machines and aims to maximize number of successfully completed jobs by the system. We have done experiments on three different network topologies, simple star topology, 4-Clique topology and combination of star and*

4-*Clique topology. Results show that Factored MDP approach fails to finish for all network topologies except the star, due to high number of in-degree per node. Among the learning approaches, SparseQ algorithm always perform the best in terms of number of successfully completed jobs, with respect to the Coordinated RL and Hybrid Learning approaches. Results also show that none of the learning algorithms suffers from time constraint, the longest measured running time is about 45 seconds for system of* 34 *agents.*

1 Introduction

A multiagent system can be defined as a collection of agents that are trying to reach to a common/personal goal possibly by interacting each other [13]. Building a multi-agent system provides us to understand and manage the system in various ways. However, it is hard to obtain single, uniform model for all multiagent problems due to their highly domain specific natures.

There are important and successful single agent, centralized models and algorithms for examining single agent systems. Markov Decision Problem formulation is one of the important planning frameworks for stochastic single agent planning problems. Applying well-known algorithms, such as value iteration or policy iteration we can get an optimal policy for the system. An agent trying to reach to a goal cell in a maze may be an example of this kind of problems [12]. If the model of the environment, the state transition probabilities and reward function, is not known by the agent, on the other hand, there are also various learning algorithms. Here, the agent tries to learn the optimal policy on-line. It makes an action to the environment and get a reward based on the action. That leads to learn the series of actions that would maximize the reward and return the optimal policy. The same maze problem for a single agent can easily be solved using this online learning approach, too.

Since a multiagent system can be viewed as a huge single agent system, the models and algorithms for single agent systems are always valid for multiagent systems, also. The aim is to get an optimal policy for the system that maximizes the total reward. However, there are critical points that should be

handled. First of all, enumerating all possible joint states and joint actions leads to exponential size of states and actions. Applying classical value iteration or Q-learning algorithm will definitely fail due to exponential size of state and action space. Besides, it is very common for a multiagent system that most of the joint states and joint actions is unnecessary since it will not be visited as a joint state nor fired as a joint action. So, it is not required to consider all possible joint state and joint action values. We need only to know the *required* states and actions that will affect our solution to the problem. Hence, while designing a multiagent system, the key point is to *manage the locality of agents*. Since the management of locality highly depends on the domain, there are many points that should be handled carefully for a multiagent system design.

Very first point for modeling a multiagent system is that agents may represent cooperative or competitive behavior. Modeling completely differs depending of this choice. Another point about multiagent system design is transitions between the states of the systems. As it is intractable to enumerate all possible system states we have to provide a way to consider only the states that are *meaningful* for the system and agents. Another point that may differ with respect to the problem is the reward mechanism. The reward may depend on the total system state or differs for each single agent or depend on only some subset of agents for each agent. The coordination between agents is also another problem as different types of coordination may lead to different type of solutions.

Hence, in this study, we have implemented different coordinative multiagent planning and learning algorithms and tried to understand the semantics behind the algorithms based on experimental results. The first algorithm we have implemented is a multiagent planning algorithm. It is a variant of the factored MDP approach presented in [6]. The second algorithm we have studied is the one presented in [7], namely *Coordinated Reinforcement Learning*. It is a Q-learning algorithm adapted for multiagent systems. The third algorithms is also a variation of Q-learning algorithm. It is presented in [9] and applying different type of decomposition on Q-learning algorithm providing pure local iterations for each agent. The last one, on the other hand, is a novel and Hybrid algorithm combining both *Coordinated RL* and *SparseQ Learning* algorithms. It tries to subsume beneficial sides of both algorithm. We have

implemented all of the algorithms on one problem instance, *System Administrator* [5], for different size and types of network topologies. System Administrator problem defines a network machines. Each machine corresponds to an agent. The machines are affecting their neighbors and the aim is to maximize successfully completed jobs by the system of agents.

2 Background

There are several multiagent planning and learning approaches.[4] assumes that agents have capability of making an estimation about the states of other agents. This is done distributing the state information for each agent. Based on the estimation, agents trying to act optimally. [1] identifies multiagent planning problem as a decomposed MDP problem. They decompose the transition function and assumes each agent's state transition is independent of all of the other agents. As the transitions are decomposed they divide the system for different MDP problems for each agent and find formulas for each of them separately first. Then they tie the total reward function on different policies of other agents and trying to improve total reward by checking all possible solutions for each separated MDPs. [2] also uses multiagent MDPs but tries to define multiagent coordination problem from a game-theoretic approach and formulate it as a Nash Equilibrium problem. The critical part is that they assume all agents know the structure of the game and compute the joint MDP to find the optimal value of each state. This assumption is unrealistic in the sense of the agents that may not the actual environment and optimal value of each state. [11], on the other hand, uses direct policy search for finding optimal policy for all agents. It uses gradient-descent policy search algorithm for cooperative multiagent planning problem. They assume each agent act independently but gains same global reward and try to combine local optimal solutions for each agent to get the global optimal solution. They also relate their algorithm to Nash Equilibrium for obtaining local optimal solutions. [8] implements different decomposition schema for multiagent planning using MDPs. First one is to run value iteration at each agent separately, but use global reward in the update of value function for each agent. The second one uses local reward for each agent and runs value iteration for each agent separately. In the third and fourth ones, they distribute the local reward to the neighbors for

each agent and expected next state value to the neighbors for each agent in a weighted way, respectively. In all algorithms, they assume each agent is represented by a state variable and the next time step value for a state variable only depends on its previous state variable and action variable. In that sense they are not accounting the coordination between agents explicitly but trying to combine the local solutions in a proper way. [14], on the other hand, draws a general framework for multiagent learning problem. They refer to the term COIN (Collective Intelligence) and try to explain the principles of COIN. It is said that COIN framework should represent *subworlds, constraint-alignment, subworld-factored* principles. Subworlds principle is to partition the agents so that each subworld contains only agents affecting each other. Each subworld has its own total utility. Constraint-alignment is that different subworlds do not affect each other and subworld-factored principle is that local utility of a subworld cannot decrease the global utility of the whole system.

There are also several application domains for multiagent systems. For example [1] used Mars Rovers problem for their algorithm. Periodically, rovers are exploring the space and bringing the scientific data from the environment to the control central in which they always communicate. [11] implements a soccer game including two learning agents and one opponent agent with a fixed policy. There are also network rouing [3] and sensor networks domains [10] for applications of multiagent systems.

The algorithms that we will use in this work is presented in [6, 7, 9], one is a multiagent planning algorithm for Factored MDPs and the three of them are multiagent learning algorithms using active-learning techniques. We will implement both local and global reward mechanism and use Dynamic Decision Networks for state transitions between state variables. As an application domain we will implement all of the algorithms on System Administrator problem [5], in which each agent is associated with a machine and machines are connected in a specific way. Machines are loaded by several jobs with some probability and at each time, agents have to make a decision on rebooting the machine or not. The ultimate goal is to maximize the successfully completed jobs.

3 Problem Description

In this project we will solve the System Admin Problem presented in [5] for testing the algorithms. The specifications of the problem as following:

1. The problem consists of a network of n machines connected in one of the following topologies: chain, ring, star, ring-of-rings, or start-and-ring.
2. Each machines are associated with and agent A_j
3. Each agent has two actions: rebooting the machine or not.
4. Rebooting make the status of the machine as good but losing any running jobs
5. States of machines are described by two variables, status S_j and load L_j for agent j
6. Status variable may have three values as {good, faulty, dead}
7. Load variable may have three values as {idle, loaded, process successful}
8. New jobs arrive with probability of 0.5 on idle machines and make them loaded
9. A machine both loaded and good may finish its job successfully with 0.5 probability
10. Faulty machines can execute jobs but it takes longer times to terminate and jobs end with 0.25 probability
11. A dead machine is not able to execute jobs remains dead until it is rebooted
12. Each machine receives a reward of +1 for each job completed successfully, otherwise it receives 0 reward
13. Machines fail stochastically and switch status from good to faulty and faulty to dead
14. A dead machine increases the probability that its neighbors will become faulty as 0.5 and eventually dies

15. At the beginning all machines are in good status and the load is set to 'idle'

16. The aim is to maximize the number of successfully completed jobs

4 Models and Algorithms

In this section, we will explain the models and algorithms that we have used for our study. We will explain 4 different algorithms that are solving multiagent planning and learning problem from different aspects. The first one is based on value iteration for Factored MDPs and the second, the third and the fourth ones are variations of distributed Q-learning algorithms. All of the methods assume the global action-value function is the summation of local action-value functions, the global reward function is the summation of local reward functions and each agent tries to maximize its own local action-value function.

4.1 Multiagent planning for factored MDPs

The multiagent planning algorithm we will show is based on factored MDPs [6]. In this algorithm, each agent is connected to some other agents in a specific way. The state of an individual agent may only be affected by the state of the agent itself, the states of the agents that are connected to the agent and the action that the agent has fired. Hence, the exponential joint state and action space is reduced to the number of coordinations in the system. Each agent is represented as a state variable and each action is represented as an action variable. The state transition of each agent is expressed by the state and action variables that the agent depends on.

The dependency between state and action variables are represented with a *Dynamic Decision Network (DDN)*. This network provides us to build the model of the environment. That is, the parameters of the local state transition functions and parameters of the local reward functions. Figure 1 presents a DDN for star topology of three agents. Here, we can see that the value of state variable X_2 at time $t+1$ only depends on the value of X_1, X_2 and A_2 at time t. We also define the reward mechanism by the DDN. Each state variable is associated with one reward value connected to itself implying that the local reward

function of each agent is only affected by the state of the agent itself. The last thing that we will extract from the DDN is the next state expected value function, h_i, for each agent. Here, it is shown that next state expected value of each agent may depend on only its own state variable as for h_1, or it may also depend on a subset of state variables as for h_2 and h_3.

After defining the DDN and the model details, each agent runs its own value iteration as in the Algorithm 1. Note that, the optimal joint action is achieved by applying *Variable Elimination* technique to the local action-value functions in order to get the optimal policy for the system [6]. Note also that the operator $p(i)$ in Algorithm 1 represents the parents of agent i defined by the DDN.

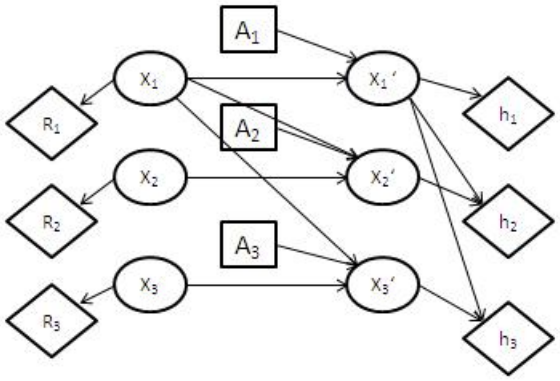

Figure 1. Dynamic Decision Network for star topology

4.2 Coordinated Reinforcement Learning

The first learning algorithm we present is *Coordinated Reinforcement Learning* [7]. Here, it is assumed that the action-value function of each agent is associated with global reward and global temporal difference. That is, the total immediate reward obtained by the system is fed to all agents as if they are running as a single agent system. Moreover, each agent grows by global temporal difference corresponding to the global reward mechanism. The algorithm is shown in 2.

Here, one important point is how to get the values of $Q(X, A)$, $max_{A'}Q(S', A')$ and $r(X)$ that are using the joint state and action values. For the first one, we just apply simple message passing schema

Algorithm 1: Value Iteration for Factored MDPs

while *the policy is not good enough* **do**
  foreach i *in Agents* **do**
    foreach x_i *in LocalStatesOfAgent* i **do**
      foreach a_i *in LocalActionsOfAgent* i **do**
        $Q_i(x_i, p(i), a_i) = r_i(x_i) + 0.95 * \sum_{x'_i} \sum_{u'_i \in p(i)} Tr(x_i, p(i), a_i, x'_i) * h_i(u'_i, x'_i)$
      end
      $h_i(p(i), x_i) = max_{a_i} Q(x_i, p(i), a_i)$
    end
  end
  $policy = variableElimination()$
end

and pass current state, x_i information between all agents. That will give current joint state, X, and will be used for information of parents for each agent. Using the information of parents, each agent finds its own Q_i value and then pass this again as a message to the system resulting in a global Q value for joint state and action. For the second one, on the other hand, we follow almost same procedure. Instead of passing current state, x_i values, agents send the next state, x'_i values to each other. By using this joint next state information, each agent returns its maximized local Q_i value which results in global Q value for joint next state X'. The global immediate reward, $r(X)$ can also be obtained by passing local reward values between all agents and summing them up.

4.3 Sparse Cooperative Q-Learning

The second algorithm we show is Sparse Cooperative Q-Learning algorithm [9]. It uses very similar decomposition schema to one in Section 4.2, however, it updates local Q_i values based on local reward value and local temporal difference instead of using global reward and temporal difference. The algorithm

Algorithm 2: Coordinated Reinforcement Learning

foreach *Episod* **do**

  foreach *i in Agents* **do**

    $Q_i(x_i, p(i), a_i) = Q_i(x_i, p(i), a_i) + \alpha * [r(X) + \gamma * max_{A'} Q(X', A') - Q(X, A)]$

  end

end

is shown in 3.

Algorithm 3: Sparse Cooperative Q-Learning Algorithm

foreach *Episod* **do**

  foreach *i in Agents* **do**

    $Q_i(x_i, p(i), a_i) = Q_i(x_i, p(i), a_i) + \alpha * [r_i(x_i) + \gamma * max_{a'_i}(Q_i(x'_i, p(x'_i), a'_i)) - Q_i(x_i, p(i), a_i)]$

  end

end

Here, the information of parents is obtained just like in 4.2, by applying simple message passing schema. One important point here is that the agents are assumed to have nothing with the other system variables in terms of reward. That makes the update rule purely local and makes the agents affect only themselves as the system runs.

4.4 Hybrid Learning

The last algorithm that we will present is a novel, Hybrid multiagent learning algorithm that is able to combine different multiagent learning methods together. Hybrid approach exploits the locality principle in multiagent learning algorithms. Since each agent keeps its own local Q_i action-value function, the method that is used to update the Q_i's may differ for different agents. Hence, Hybrid approach applies different learning algorithms to different agents in the system with respect to the system definition.

Here, we have chosen the presented two multiagent algorithms, namely Coordinated RL and SparseQ

Learning and apply them as following. Given the coordination graph for a multiagent system, the nodes, corresponding to agents, having relatively large number of in-degree will use the update rule in CoordRL algorithm and the nodes having relatively small number of in-degree will use the update rule in SparseQ Learning algorithm. The algorithm is presented in 4. The motivation behind this combination method is that CoordRL algorithm uses the global reward and temporal difference and will be more effective for densely connected nodes; SparseQ Learning algorithm, on the other hand, uses local reward and temporal difference and will be more effective for sparsely connected graphs. If the coordination graph defining the network between the agents is composed of different types of subgraphs, we may run different types of multiagent learning algorithm for different types of subgraphs.

Algorithm 4: Hybrid Learning Algorithm

foreach *Episod* **do**

  foreach *i in Agents* **do**

    if *inDegree(i)* $\geq$ *threshold* **then**

      $Q_i(x_i, p(i), a_i) = Q_i(x_i, p(i), a_i) + \alpha * [r(X) + \gamma * max_{A'} Q(X', A') - Q(X, A)]$

    end

    else

      $Q_i(x_i, p(i), a_i) = Q_i(x_i, p(i), a_i) + \alpha * [r_i(x_i) + \gamma * max_{a'_i}(Q_i(x'_i, p(x'_i), a'_i)) - Q_i(x_i, p(i), a_i)]$

    end

  end

end

5 Experimental Results

In this section, we will describe the experiments that we have done to understand the effecitivity of algorithms explained in section 4. We have done all of the experiments on single problem instance, System Administrator [5], with three different topologies.

The three different topologies are shown in Figure 2. There are star topology, 4-Clique topology and combination of star and 4-Clique topology. The reason for choosing these three topologies is that we want to have one sparse, one dense and one combined topology for being able to test presented algorithms fairly. For a star of three agents and 4-Clique topologies we have applied Factored MDP, Coordinated RL and SparseQ Learning algorithms. For three different combined topologies having 7, 10, 13, 19 and 34 agents, we have applied Coordinated RL, SparseQ Learning and Hybrid Learning algorithms. All combined topologies contains one 4-Clique topology and one star topology including 3, 6, 9, 15 and 30 agents respectively. In order to compare the algorithms we have measured the running time of the learning, and number of successfully completed jobs per agent for 100 runs of the optimal policy. For the parameter setting, we have chosen $\alpha = 0.2$ and $\gamma = 0.9$. Note that we have used ϵ-greedy approach for all Q-learning algorithms and chosen $\epsilon = 0.3$. We have applied 15000 iterations for each learning algorithm. Note also that all experiments are repeated 10 times and the average of the 10 results are reported in order to reduce the probabilistic nature of the results.

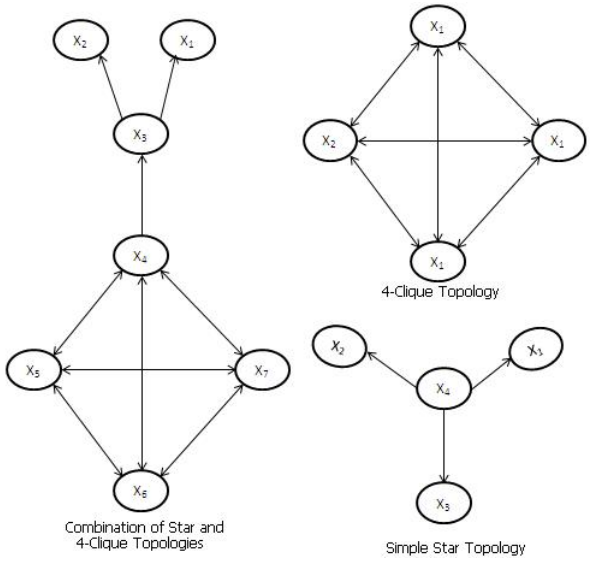

Figure 2. Topologies

In Figure 3, the results for the topology of star of three agents is shown. The first set of bars corresponds to time results and second ones corresponds to the number of successfully completed jobs. It can be seen

that Factored MDP solution achieves best result in terms of number of successfully completed jobs. However, SparseQ algorithm has also very similar results to the Factored MDP solution. Beside running time of SparseQ algorithm is much less than factored MDP solution. Coordinated RL, on the other hand, has the lowest running time and achieved reasonably good score for this simple topology.

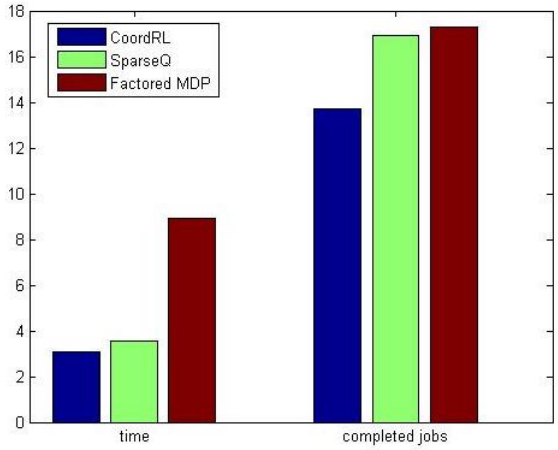

Figure 3. Simple Star Topology Results

Figure 4 shows the results for 4-Clique topology. Factored MDP solution failed to solve this topology due to high value of in-degree per node. In star topology, the in-degree is at most 1; in 4-Clique topology, however, it is 3. Since we have 9 state value per agent, and 2 action values, we should have $18^4 * 9^4$ many calculations for each agent. The latter 9^4 is for the all possible combination of next state values of parents. The results for the learning algorithms show that Coordinated RL has given worse results than the previous previous topology in terms of number of successfully completed jobs. The SparseQ, whereas, gave almost same result with the previous topology. This is due to the fact that SparseQ does not depend on the size or connectedness of topology, it only makes each agent run its own learning function without being affected by the other parts of the network.

For the combined network, we have done two different types of experiments. The first one is same as the previous experiments and applied to a combined network comprising of a star topology of 3 agents and a 4-Clique topology. While applying Hybrid Learning algorithm to a combined network, the

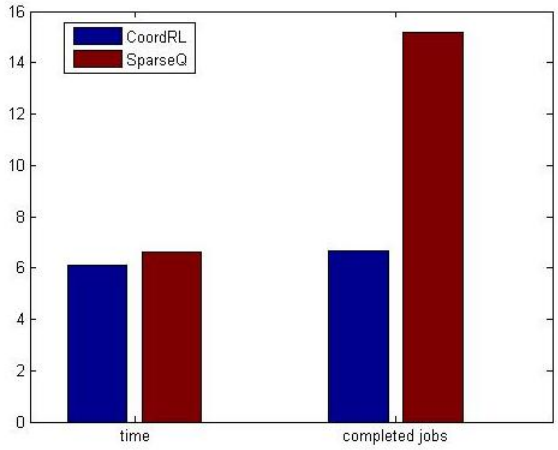

Figure 4. 4**-Clique Topology**

agents that belong to star topology updates their Q_i values by using SparseQ algorithm, whereas, the agents that belong to Clique topology updates their Q_i values by Coordinated RL algorithm. Thereby, we used different learning approach in one problem setup. Figure 5 shows the results. Here, SparseQ algorithm still performs best and has given almost same number of complete jobs with the previous experiments, which validates the stability of SparseQ algorithm. The Hybrid Learning approach, on the other hand, results in somewhere between SparseQ and CoordiantedRL. This is understandable since it applies SparseQ algorithm to 3 agents of the system and Coordinated RL to 4 agents of the system. The lowest running time belongs to Coordinated RL algorithm although the difference is not significance.

The second experiment on the combined network is about the number of agents in the system. We have increased the number of agents in the star part of the combined topology and check the results of three learning algorithms. The results are shown in Figure 6. Still the number of successfully completed jobs per agent is almost same for SparseQ algorithm and Coordinated RL fails with respect to the other algorithms. For the Hybrid algorithm, the result increases as the number of agents in star topology increases. This is because the sparsity of the graph is increasing and the SparseQ part of the Hybrid algorithm becomes dominant. For the topology having 15 agents in its star part, the different is not very much, indeed.

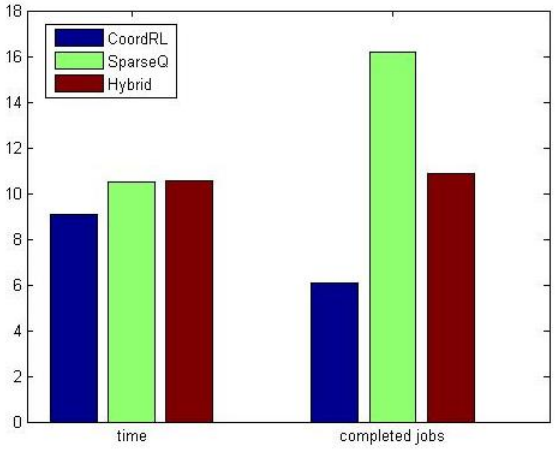

Figure 5. Combined Topology

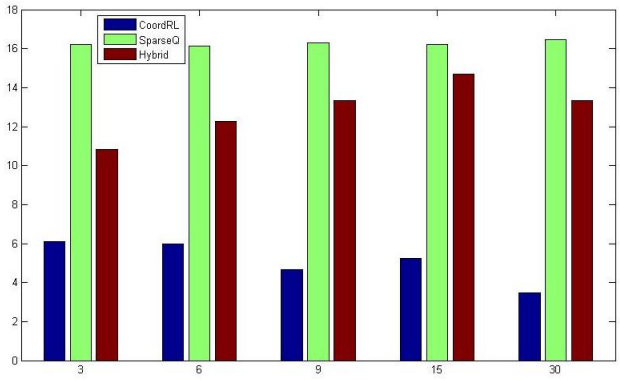

Figure 6. Combined Topology for Different Number Of Agents

6 Conclusion and Future Work

In this work, we have compared several multiagent planning and learning algorithms. Firstly, we explained the Factored MDP solution to multiagent planning problem. Then, we have shown the Coordinated Reinforcement Learning and Sparse Cooperative Q-Learning algorithms for multiagent learning problem. Then, we have introduced a new Hybrid Learning approach that is combining both Coordinated RL and SparseQ learning algorithms. It applies Coordinated RL to the dense graphs and SparseQ Learning to sparse graphs. The algorithms are tested on single problem instance, System Administrator, with three different topologies. The experimental results showed Factored MDP solution did not finish except the star topology due to high number of in-degree per node in the topologies. Results also showed that SparseQ learning algorithm always perform best among the learning algorithms. Although the Hybrid approach always produces scores between Coordinated RL and SparseQ, as the sparsity of the graph increases, it gets close to the SparseQ.

As future work, we may be interested in trying different combinations of learning algorithms for different types of topologies. We believe that the idea about hybridization of different multiagent learning algorithms is very attractive. Hence, we will try to enhance the idea by using different types of decomposition schema and experimental setups.

References

[1] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 41–48, Melbourne, Australia, July 2003. ACM Press.

[2] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, TARK '96, pages 195–210, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[3] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6*, pages 671–678. Morgan Kaufmann, 1994.

[4] Partha Sarathi Dutta, Nick R. Jennings, and Luc Moreau. Cooperative information sharing to improve distributed learning in multi-agent systems. *CoRR*, abs/1109.5712, 2011.

[5] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored mdps. In *In IJCAI*, pages 673–682, 2001.

[6] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *In NIPS-14*, pages 1523–1530. The MIT Press, 2001.

[7] Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 227–234, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[8] Andrew Moore Martin Riedmiller Jeff Schneider, Weng-Keen Wong. Distributed value functions. In *Proceedings of the 16th International Conference on Machine Learning*, pages 371–378. Morgan Kaufmann, San Francisco, CA, 1999.

[9] Jelle R. Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *JOURNAL OF MACHINE LEARNING RESEARCH*, 7:1789–1828, 2006.

[10] V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed Sensor Networks: A Multiagent Perspective (Edited book)*, volume 9. Kluwer Academic Publishers, May 2003.

[11] Leonid Peshkin, Kee eung Kim, Leslie Kaelbling, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. In *In UAI*, pages 489–496. Morgan Kaufmann, 2000.

[12] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[13] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[14] David H. Wolpert, Kevin R. Wheeler, and Kagan Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the third annual conference on Autonomous Agents*, AGENTS '99, pages 77–83, New York, NY, USA, 1999. ACM.