Planning with Multiple Agents

Sabina Kruk
Seminar on Autonomous Learning Systems
Department of Computer Science
TU Darmstadt
sabina.kruk.stud@tu-darmstadt.de

Abstract

In this paper focus is set on models of decentralized cooperative multi-agent systems. These models are used in settings where multiple agents work together, possibly on different tasks to fulfill a bigger goal. Agents try to maximize their joint reward while having their own local observation functions which can differ according to their information about the world and their resources. Real-world examples of such planning are mentioned and theoretical models providing formal solutions to different types of decentralized multi-agent planning situations are introduced. The models differ in their level of agents' observability of the world as well as communication between them. It is invesigated how this has effect on the models complexity and how it can be sometimes reduced by making alterations to the model. Last but not least, optimal and approximate algorithms providing solutions to the models are regarded and their advantages and drawbacks investigated.

1 Introduction

1.1 Motivation

There are many models for planning under uncertainty and algorithms that solve these planning problems, but most of them concentrate on single-agent cases. Many processes nowadays require multiple agents to work together, for example in achieving a certain goal. Such processes are part of applications used in the military, networking, industry, space research, logistics, games and many more. In this paper the focus is set on multi-agent planning under uncertainty, where each of the agents possibly has different tasks to fulfill, as well as resources and information about the world. Each agent has its own separate observations, but the agents must coordinate their actions to optimize the joint reward. Such problems are referred to as cooperative decentralized decision problems. They stand in contrary to the centralized ones, where the knowledge about the world and the decision making authority is located in a single unit/agent.

In the next section, we illustrate the difference between centralized and decentralized planning problems through an example firstly introduced by Kaelbling et. al. [3] in a centralized version and then extended to a decentralized one by Nair et. al. [4].

1.2 Tiger problem

Imagine an agent standing in front of two closed doors. Behind one of them is a tiger and behind the other a treasure. If the agent opens the door with the tiger, it receives a penalty. But the agent has another option; it can listen for the tiger to gain some information about the location of the danger instead of opening one of the doors. But listening has a cost and is not entirely accurate, because probability of hearing the tiger at a specific door if it actually is behind that door is less than 1. After

the agent opens a door and receives a reward or penalty, the game is over. This kind of partially observable problem is solved using the Partially Observable Markov Decision Process (POMDP) model where instead of planning in the space of physical world states, you plan in the space of what you might know about these states. These abstract states are called belief states.

In a decentralized version of this problem with multiple agents, we have two agents who also have only partial knowledge about the environment. In each step, each agent can independently either listen or open one of the doors. If one of the agents opens the door with the treasure behind it, they both get the reward. If either agent opens the door with the tiger, a penalty is received. The crucial issue is that the agents cannot communicate with each other. They have to come up with a joint policy for listening and deciding what door to open. It is of huge importance that the agents coordinate their actions because if both of them open the tiger door at the same time, their penalty is not as high. After a door is opened and the agents receive a reward/penalty, the problem starts over again.

1.3 Real-world examples

A real-world example is a system called "Mobile Communications amongst Heterogeneous Agents" (MoCHA) [5] which is a mobile communication network where every user creates a personalized agent. The personalized agents locate and communicate with each other to plan schedules and collaborative tasks with personalized agents of the other users. This requires a decentralized planning model. where a personal agent created by the user plans and schedules collaborative tasks with personalized agents of the other users.

Another example is a multi-access broadcast channel where two agents control a message channel. To avoid a collision, only one message at a time can be sent. The agents want to maximize the global throughput of the channel and the only decision they can make is to send a message or not. There is a global reward 1 if the message was sent successfully and a penalty/no reward 0 if not. Every agent has observed information at each step, including their own message buffer, information about a possible collision or a possible successful message broadcast. The problem here is that the observations of possible collisions are noisy i.e. the agents can only have uncertain beliefs about the outcome of their actions.

Last but not least, in many cases the agents have isolated missions they need to fulfill and must optimize their own objectives independent of the other agents. On the other hand, they might have to complete a task from time to time on which they have to work together and here the optimal planning depends on all agents' actions. A good example of such a system is that of controlling the operation of multiple exploration rovers, such as those used by NASA to explore the surface of Mars. Each rover is assigned a separate region that it is supposed to explore. Occasionally they have to solve a problem together, for example make a 3D image of an area. This is possible only by combining information obtained from all the rovers. Sometimes he rovers can communicate with a central control unit but it is not always possible.

2 Classification criteria of multi-agent planning models

There are many decision-theoretic models for decentralized multi-agent planning and they can be categorized in different ways. An important criterion in decentralized multi-agent decision models is their level of decentralization, determining the extent to which the agents have influence on each other and how dependent they are on each other. The level of decentralization can be described by the type of communication between agents and their level of observability. The models marked in the table are the ones considered in the next chapters.

2.1 Observability

One differentiates between full observability, joint full observability and partial observability. Full observability implies that each agents partial view (local observation) is sufficient to uniquely determine the global state of the system. Joint full observability implies that the tuple of observations made by all the agents uniquely determines the current global state of the world. Partial observability refers to a case where all agents observations together still do not determine the global state.

Table 1: Decentralization levels

	Full observ.	Joint full observ.	Partial observ.
No comm.	*MMDP* [1]	*DEC-MDP* [1], [2]	MTDP [1], *DEC-POMDP* [1], I-POMDP [1]
General comm.	MMDP	DEC-MDP	COM-MTDP [1], DEC-POMDP-COM [1]
Free comm.	MMDP	MMDP	MPOMDP

2.2 Communication

Communication can be free, general or there can be no communication at all. In this paper communication refers to agents simply communicating their observations with each other. Agents with free communication share all observations with each other any time they want and it does not cost them anything. In the case of general communication, at least one communication action has to have a cost.

3 Theoretic models

In this chapter different models according to their level of decentralization are introduced. All of the models mentioned in this paper are a modification of a general one. This general model is called DEC-POMDP and its assumptions are a fundament for all others.

DEC-POMDP (decentralized partially observable Markov decision process) In this model agents cannot communicate with each other. It is defined as a tuple $< I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T >$, where:

- I is a finite set of agents indexed $1, ..., n$
- S is a finite set of states, with distinguished initial state s_0.
- A_i is a finite set of actions available to agent i and $\vec{A} = \otimes_{i \in I} A_i$ is the set of joint actions, where $\vec{a} =< a_1, ..., a_n >$
- $P : S \times \vec{A} \to \Delta S$ is a Markovian transition function. $P(s^{'}|s, \vec{a})$ denotes the probability that after taking joint action $\vec{a}$ in state s a transition to state $s^{'}$ occurs.
- Ω_i is a finite set of observations available to agent i and $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ is the set of joint observations, where $\vec{o} =< o_1, ..., o_n >$ denotes a joint observation.
- $O : \vec{A} \times S \to \Delta\vec{\Omega}$ is an observation function. $O(\vec{o}|\vec{a}, s^{'})$ denotes the probability of observing joint observation $\vec{o}$ given that joint action $\vec{a}$ was taken and led to state $s^{'}$. Here $s^{'} \in S, \vec{a} \in \vec{A}, \vec{o} \in \vec{\Omega}$.
- $R : \vec{A} \times S \to \Re$ is a reward function. $R(\vec{a}, s^{'})$ denotes the reward obtained after joint action $\vec{a}$ was taken and a state transition to $s^{'}$ occurred.
- For a finite-horizon problem, the agents act for a fixed number of steps, which is called the horizon and denoted by T.

Local policy for a DEC-POMDP A local policy for agent i, δ_i , is a mapping from local histories of observations $\overline{o} = o_{i_1}, ..., o_{i_t}$ pver Ω_i to actions in $A_i, \delta_i : \Omega_i^* \to A_i$.

Joint policy for a DEC-POMDP A joint policy, $\delta =< \delta_1, ..., \delta_n >$, is a tuple of local policies, one for each agent.

Solving a DEC-POMDP Finding a joint policy that maximizes the expected total reward. Such a policy can be found by maximizing its value. This is obtained by considering every possible sequence of rewards, multiplying it by the probability that it will happen and then averaging. It is nothing but the expected sum of future rewards. We use expectation (E) and not simply sum of future rewards, because the transition from state to state is not deterministic.

The value of a joint policy δ for a finite-horizon DEC-POMDP with initial state s_0 is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{T-1} R(\vec{a_t}, s_t)|s_0, \delta\right]$$

.

The models illustrated in the next section, can be seen as special cases of the general DEC-POMDP model.

3.1 Partial observability

3.1.1 No communication

MTDP (multi-agent team decision problem)

In 2002, Pynadath and Tambe [7] presented the MTDP framework, which is very similar to the DEC-POMDP framework. It differs only in the fact that it assumes perfect recall for each agent i.e. the agent does not "forget" any decisions made in the past. At any time, it has also access to every piece of information it ever received. This includes its local observations as well as local observations it received from other agents, if it was able to communicate with them.

To describe the idea, a space of belief states is introduced. B_i denotes the set of possible belief states for agent i. Each agent $i \in \alpha$ forms an abstract belief state $b_i^t \in B_i$ about the world. A belief state of an agent with perfect recall incorporates its current observation as well as all the information it ever received from other agents.

Since the MTDP model assumes no communication, the belief state is based solemnly on its observations seen through time t. This leads to an alteration in the definition of policies. Instead of mapping an agent's observations to actions, as it was done in the DEC-POMDP, the observations are replaced by the agent's belief states:

Domain-level policy for an MTDP The set of possible domain-level policies in a MTDP is defined as the set of all possible mappings from belief states to action, $\pi_{i_A} : B_i \rightarrow A_i$.
(A_i is a set of actions available to agent i, just as in the DEC-POMDP).

Joint-domain-level policy for an MTDP A joint domain-level policy for an MTDP,
$\pi_{\alpha_A} =< \pi_{1_A}, ..., \pi_{n_A} >$, is a tuple of domain-level policies, one for each agent.

Solving an MTDP Finding a joint policy that maximizes the expected global reward.

The function preforming the mapping is called the state estimator function.

I-POMDP (interactive POMDP)

The last model to mention in this category. It is an alternative to the DEC-POMDP with an important difference. In MTDP and DEC-POMDP agents have only beliefs about their own local state. I-POMDP, on the contrary, is based on so called explicit belief representation. This means agents maintain a belief about their own local states and about the other agents (their states, their policies etc.), as well.

To model the richer belief of the agents, an interactive state space for each one is used. A belief over an interactive state subsumes the belief over the underlying state of the environment, as well as the belief over the other agents.

In order to understand how such an interactive state space is created, the definition of an agent's model has to be introduced. Agents can have models of other agents. We differentiate between subintentional and intentional models. An intentional model of an agent j, IM_j provides information about its beliefs i.e. how agent j maps possible histories of observations to distributions of actions. On the contrary, its subintentional model SM_j merely makes other agents who have it, aware of its existence. Every agent j has a set of possible models M_j which consists of the subintentional SM_j and intentional models IM_j. An agent having access to the model $m_j \in M_j$ is not only aware that agent j is somewhere out there, but can also a possible belief about the agent.

Now we can use the models of agent SM_j to form the interactive set state IS_i of agent i: $IS_i = S \times M_j$, where S is the set of states of the physical environment and M_j is the set of possible models

of agent j. Put together, agent i's belief is now a probability over states of the environment and the models of the other agent, agent j: $b_i \in \Delta(IS_i) \equiv b_i \in \Delta(S \times M_j)$.

3.1.2 General communication

One can extend the DEC-POMDP and MTDP to models by allowing communication. The agents need to have conventions about how to interpret the communication messages and how to combine this information with their own local information.

It is assumed in both models that the agents simply communicate their observations. Now, after performing an action, all agents send a message to the other agents with the observation they made at that point. In case of general communication, each such message has a specified cost. Costs for communication actions can be defined either implicitly by the reward function or explicitly by a separate cost function.

DEC-POMDP-COM (decentralized partially observable Markov decision process with general communication)

In the case of DEC-POMDP-COM, there is an overall distinction between communication actions and regular actions, and i.e. they also have separate cost functions and policies. The total reward function incorporates the reward for communication and regular actions.

Σ is the alphabet of communication messages. $\sigma_i \in \Sigma$ is an atomic message sent by agent i, and $\vec{\sigma} =< \sigma_1, ..., \sigma_n >$ is a tuple of all messages sent by the agents in one time step, a joint message.

$C_\Sigma : \Sigma \rightarrow \Re$ is the message cost function.

Joint policy for a DEC-POMDP-COM A joint policy $\delta =< \delta_1, ..., \delta_n >$ is a tuple of local policies, one for each agent, where each δ_i is composed of the communication and action policies for agent i.

Solving a DEC-POMDP-COM Finding a joint policy that maximizes the expected total reward.

COM-MTDP (communicative multi-agent team decision problem)

In a COM-MTDP the communication actions are separated from the normal actions, but the cost of communicating is incorporated in the reward function.

Σ_α is a set of combined communication messages, defined by $\Sigma_\alpha \equiv \Pi_{i \in \alpha} \Sigma_i$, where $\{\Sigma_i\}_{i \in \alpha}$ is a set of possible messages for agent i.

There is only one reward function now, that directly incorporates the cost of communication.

The policies are defined as follows:

Communication policy for a COM-MTDP A communication policy for a COM-MTDP is a mapping from the extended belief state space to communication messages, i.e. $\pi_{i\Sigma} : B_i \rightarrow \Sigma_i$

Joint communication policy for a COM-MTDP A joint communication policy for a COM-MTDP is a tuple of communication policies, one for each agent.

Joint policy for a COM-MTDP A joint policy for a COM-MTDP is a pair consisting of a joint domain-level policy and a joint communication policy, $< \pi_{\alpha\Sigma}, \pi_{\alpha A} >$.

Solving a COM-MTD Finding a joint policy that maximizes the expected total reward.

3.1.3 Equivalence and Complexity Properties

It has been proven that a non-deterministic Turing machine can solve any instance of a DEC-POMDP in at most exponential time. This means that DEC-POMDP is NEXT-complete.

One can determine the complexity of the other models (except I-POMDP) by showing that they are equivalent, where equivalence is defined as follows:

Equivalence of models: Two models are called equivalent if their corresponding decision problems are complete for the same complexity class.

Instead of converting the models to decision problems and then finding a proof for the completeness, it can be simply shown that the two obtained decision problems are reducible to each other in some specified time.

Here is an example of showing that DEC-POMDP and MTDP are equivalent. The only syntactical difference is the estimator function in MTDP, which maps observations to belief states. In a MTDP, agents remember all local observation received from other agents i.e. the resulting belief states in the MTDP are simply lists of observations. If we look at the estimator function in this simplified way, we can see that it implicitly exists in a DEC-POMDP, too.

It can be proved that DEC-POMDP, MTDP, DEC-POMDP-COM and MTDP-COM are all reducible to each other after redefining the models without changing their semantics. This means they are all NEXT-complete.

Unfortunately, I-POMDP is likely to be at least as hard to solve as DEC-POMDP, even though it expresses more information about the world. It has a problem of nesting beliefs. A second agents belief might also include a belief over the first agents belief and so on. This makes finding optimal solutions very complicated.

3.2 Joint full observability

This category includes a model called DEC-MDP (decentralized Markov decision process) which is simply a DEC-POMDP with joint full observability. Each agent still has only partial observability of the world, but the joint observations of all the agents provide full information about the world.

DEC-MDP is NEXT-complete, but the complexity can be reduced by making some variations to the model and changing its decentralization level.

3.2.1 Factored MDPs

One of these variations is factoring the world state into $n + 1$ components $S = S_0 \times S_1 \times ... \times S_n$ where n is the number of agents involved. The s_0 state describes the worlds external features and the other ones describe the local state of agent i. Making such a discrimination guarantees the local state of agent i to be only dependent on its private state features and on the external features.

In the next step, the agents transitions and observations are made independent from those of the other agents. i.e. the agent is able to determine the local state given its local observation. This changes the models decentralization level and makes the factored DEC-MDP locally fully observable.

In addition, the model can be made reward-independent i.e. the overall reward is composed of a function of the local reward functions, each of which depends only on the local state and local action of one of the agents. Maximizing each of the local reward functions individually is sufficient to maximize the overall reward function.

Taken into account that the model is transition, observation and reward independent, it can be decomposed into n independent local MDPs and making the problem P-complete.

If one of those independencies is missing though, it is not that easy to reduce the complexity.

3.2.2 Problematic sub-classes of Factored MDPs

An important subclass of a factored, locally fully observable DEC-MDP is one with no reward independence. The reward function has a specific form in such cases.

It consists of two different components. The first is a set of local reward functions that are independent. The second component is an additional global reward that depends on the actions of all the agents. Such a model is NP-complete and can be applied to settings where the whole system gets a reward after multiple agents have successfully completed a task.

A good example of such a system is the mentioned rover mission on Mars. The rovers have independent transitions and observations. They have isolated tasks to fulfill for which they get their own independent reward but sometimes they have to work together for example when building a 3D model of an area, where every agent can only take pictures of one part. Completing a task that involves joint planning requires a separate reward that depends on the actions of all the agents.

3.3 Full observability

MMDP (multi-agent Markov decision process is a DEC-POMDP with full observability)

An MMDP is a straightforward extension of the completely observable MDP model for multiple agents where a single action is replaced by a vector of actions. The complexity is though the lowest possible, P-complete.

The mapping from states to actions in the transition function of the DEC-POMDP becomes a mapping from observations to actions, since we assume that the agents can now observe all available information.

What is interesting about this model is that it can be achieved by different combinations of observability and communication between agents. Full observability always guarantees a MMDP problem but so does joint full observability combined with free communication. Allowing the agents in a DEC-MDP to freely communicate with each other whenever they want without any costs for their communication messages implies they achieve full observability. They can obtain any information that is available, simply by communicating with each other. This reduces the planning problem to the solution of a MMDP.

4 Optimal solutions

A policy of agent i can be represented as a policy tree $q_i \in Q_i$. It is a decision tree, where nodes are labeled with actions and arcs are labeled with observations. The total number of possible policy trees Q_i for agent i is the number of possible ways to assign different combinations of actions. This means that the tree grows exponentially. In addition, the problem is also exponential in the number of agents, as the number of joint policies is the product of the number of possible policies for all agents. Different algorithms take different approaches in order to reduce the exponential blowup.

4.1 MAA^* a heuristic search algorithm for DEC-POMDPs

4.1.1 Approach

The algorithm finds an optimal joint policy for a DEC-POMDP. It is based on the search algorithm A^* and searches through the space of possible joint policies. Heuristics are used to make it more effective. A following representation is used:

q_i^t is a depth-t policy tree for agent i and $\delta^t = (q_1^t, ..., q_n^t)$ is a policy vector of trees, one for each agent. T-depth means that the agents have been acting for a t fixed number of steps. $V(s_0, d)$ denotes the expected value of executing policy vector d from state s_0. Finding the optimal policy is thus identical to finding a policy vector that maximizes $V(s_0, d)$.

The algorithm works as follows: A policy vector of trees is built, where nodes of the tree at level t correspond to partial solutions of the problem for all the agents, namely policy vectors obtained from t steps. The algorithm searches through this space of policy vectors, but not all nodes at every level are fully expanded. Instead, a heuristic function is used to evaluate the leaf nodes of the search tree. The node with the highest heuristic estimate is expanded in each step.

The figure below illustrates the algorithm for 2 agents with horizon 3. Each of them can have one of the two observations $\{o_1, o_2\}$ and choose one of two actions $\{a, b\}$.

Suppose that the horizon 2 policy for agent 1, δ_1^2 is described by the corresponding policy tree q_1^2. This means that in the first step, agent 1 should pick action a. In the second step, if it has observed the observation o_1 it should choose action a, otherwise b. The analog mechanism applies to agent 2. Its horizon 2 policy is reflected by the policy tree q_2^2. For both agents, these are only one of the possible policies for horizon 2. This makes the policy tree vector δ^2, where it is simply a vector of the two chosen individual policies, also only one of the possible solutions to this 2-agent problem. Assuming the aim is to find a solution for horizon 3, the horizon 2 policy vector trees need to be expanded. This is where the heuristic comes into the picture. Only the nodes, alias horizon 2 policy vectors, with the best heuristic estimate will be chosen for expansion to horizon 3. In this figure, this node happens to be the described policy vector tree. After making the optimal node choice, the

agents' individual horizon 2 policy trees are expanded. This leads to all possible horizon 3 policy vector trees that could be obtained from the chosen horizon 2 node.

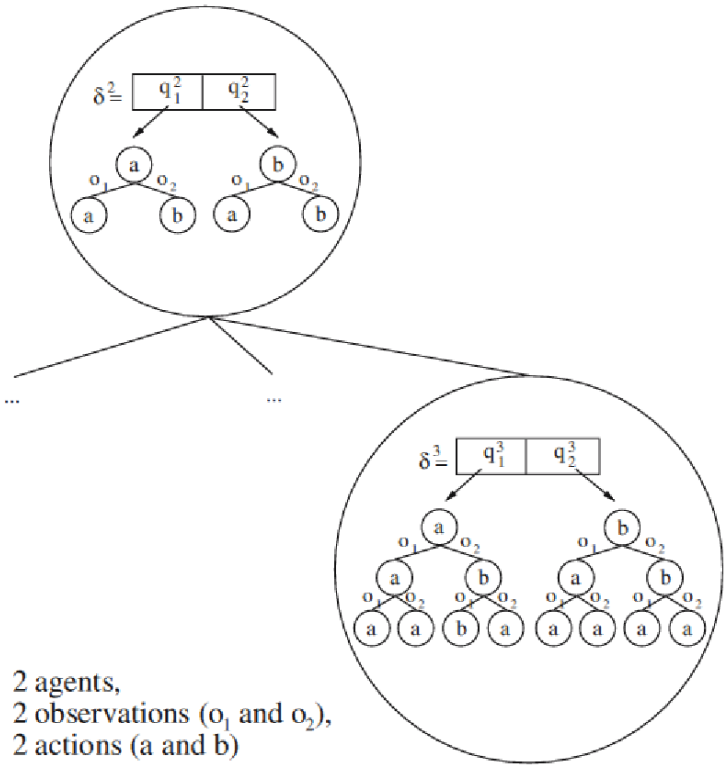

Figure 1: A section of the multi-agent A* search tree, showing a horizon 2 policy vector with one of its expanded horizon 3 child nodes (courtesy of Daniel Szer)

4.1.2 Effectiveness

So far, the MAA^* algorithm runs out of time relatively fast when computing higher horizons.

4.2 Dynamic Programming for DEC-POMDPs

4.2.1 Approach

The main idea of dynamic programming for DEC-POMDPs is to incrementally search through the space of possible policies, pruning dominated policies as early as possible in the construction process to avoid the exponential blowup.

(Dominated policy trees) A policy tree $q_j \in Q^t$ with corresponding value vector $v_j \in V^t$ is considered dominated if for all $b \in B$ there exists a $v_k \in V^t \backslash v_j$ such that $b \cdot v_k \geq b \cdot v_j$.

This can be described as follows: B is here a state set that consists of all possible beliefs about the current state. q_j is a policy tree for agent i. Each such policy tree has a corresponding value vector v_j made up of all rewards received while pursuing the policy corresponding to that particular tree. Such a tree is considered dominated if for all possible beliefs about the current state, there is a value corresponding to that belief state and the pursued policy, such that it is higher or equal to any other policy.

4.2.2 Effectiveness

Unfortunately, the dynamic programming algorithm runs out of memory after the 4th iteration due to the rapid growth in the number of policies trees.

5 Approximate solutions: Sub-solution for Multi-agent Planning with Factored MDPs

5.1 Motivation

This section illustrates a near optimal solution to a multi-agent planning problem using the notion of factored MDPs. The main idea is to avoid the exponential blowup in the state and action space that you achieve when you search through a joint policy tree to find the best one.

5.2 Approach

5.2.1 Factoring

In case of a single MDP, solving it means finding a policy that maximizes the value function. In this approach, we factor the joint value function of all agents into local value functions. They can be approximated as a linear combination which can be solved efficiently by a single linear program.

The local value function for each agent is the sum of an immediate reward and a value that they expect to receive one step in the future.

The key of this approach is to notice that the action, current and future states of an agent depend only on a small collection of variables (states, actions), namely only the ones belonging to a small set of agents. This means that the action space as well as state space can be factored into local state/action spaces that depend only on a small subset of variables. We thereby avoid the exponential blowup in the state and action space. This reduces the complexity of finding the immediate reward and value expected to receive one step in the future, thus simplifying the calculation of a local value function.

5.2.2 Approximating value functions

Here is an example of how a joint value function for such a factored MDP can be optimized to a form in which it can be solved by the linear program. We can imagine this as follows:

A collection of agents is presented where each agent i chooses an action A_i. Each agent i has a local Q function Q_i, which represents its local contribution to the total utility function. The joint value function is denoted $Q = \sum Q_i$ and depends on the agents' actions. The agents have to work together and coordinate their actions. This can be represented using a coordination graph. There is a node for each agent and an edge between two agents if they must directly coordinate their actions to optimize some particular Q_i. The task is now to select a joint action that will maximize the value function Q. The key idea is that, rather than summing all functions Q_i and then doing the maximization, we maximize over the action variables a_i one at a time.

Suppose that the joint Q function that is to be maximized looks as follows:

$$Q = Q_1(a_1, a_2) + Q_2(a_2, a_4) + Q_3(a_1, a_3) + Q_4(a_3, a_4)$$

where $Q_1(a_1, a_2)$ implies that the Q function of agent 1 depends on its own actions as well as the actions of agent 2. This leads to this coordination graph:

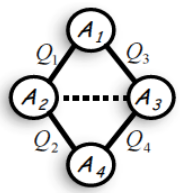

Figure 2: Coordination graph for a 4-agent problem

1. The function Q can be optimized by maximizing over one of the a_i 's at a time. This means that only summands involving a_i participate in the maximization. The optimization for agent 4 looks as follows. To optimize A_4, functions Q_3 and Q_1 are irrelevant. This leaves: $max_{(a_1,a_2,a_3)}Q_1(a_1,a_2) + Q_3(a_1,a_3) + max_{(a4)}[Q_2(a_2,a_4) + Q_4(a_3,a_4)]$
2. Agent 4 can summarize the value that it brings to the system in the different circumstances using a new function $e_4(a_2,a_3)$, having one fewer agent and whose value at the point a_2, a_3 is the value of the internal max expression. The problem reduces to computing: $max_{(a_1,a_2,a_3)}Q_1(a_1,a_2) + Q_3(a_1,a_3) + e_4(a_2,a_3)$.
3. Next, agent 3 makes its decision, giving: $max_{(a_1,a_2)}Q_1(a_1,a_2) + e_3(a_1,a_2)$, where $e_3(a_1,a_2) = max_{(a3)}[Q_3(a_1,a_3) + e_1(a_2,a_3)]$.
4. Agent 2 now makes its decision, giving: $e_2(a_1) = max_{(a_2)}[Q_1(a_1,a_2) + e_3(a_1,a_2)]$.
5. Agent 1 can now simply choose the action a_1 that maximizes $e_1 = max_{(a_1)}e_2(a_1)$.

The maximizing set of actions can be recovered by performing the entire process in reverse: The maximizing choice for e_1 selects the action a_1^* for agent 1. To fulfill its commitment to agent 1, agent 2 must choose the value a_2^* which maximizes $e_2(a_1^*)$. This, in turn forces agent 3 and then agent 4 to select their actions appropriately. The cost of computing this maximization is linear and overcomes the exponential blowup in the state and action spaces. This leads to the conclusion that the algorithm's complexity is linear as well. The size of the linear program used to solve the maximization problem is dependent on the width of the corresponding coordination graph used by the agents to negotiate their action selection. The wider the graph, the more dependencies there are between the agents and the single Q_i functions depend on more a_i's.

6 Conclusion

As one can easily see, the models used to solve a multi-agent planning problem depend strongly on the level of observability and communication between agents i.e. level of decentralization.

These models have different complexities which are inherently dependent on the level of decentralization, as well. In case of models with full observability, it does not matter if we change the type of communication between agents because they can already observe all the available information. The problem occurs when the observability is limited. If the agents are able to communicate freely without any cost, the problem can be described by a model belonging to a lower complexity class. An example is the DEC-MDP, which becomes an MMDP given free communication. We have seen that there are optimal algorithms for solving decentralized planning problems, but most of them run out of time or memory for bigger horizons, more agents etc. This leads to the idea to create approximate algorithms that provide sub-optimal solutions, but function for larger settings. Such an example is the introduced algorithm for factored MDPs. Another idea for approximate solution is to combine optimal algorithms to create approximate ones, that function for larger settings. An example is combining the heuristic approach of MAA* to identify relevant belief sates and dynamic programming, that can then evaluate the policy trees and select the best joint policy.

The aim to solve a multi-agent decision problem is to reduce the complexity as much as possible and look for approximate algorithms that provide solutions that are accurate enough for the situation but do not fail on slightly bigger settings.

References

[1] Sven Seuken and Shlomo Zilberstein (2008). Formal Models and Algorithms for Decentralized Decision Making Under Uncertainty. In Journal of Autonomous Agents and Multi-Agent Systems, 17:2, pp. 190-250

[2] Carlos Guestrin, Daphne Koller and Ronald Parr. Multiagent Planning with Factored MDPs; In Advances in Neural Information Processing Systems (NIPS-14), pp. 1523 - 1530, Vancouver, Canada, December 2001.

[3] Kaelbling, L. P., Littmann, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(2), 99134.

[4] Nair, R., Pynadath, D., Yokoo, M., Tambe, M., and Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI) (pp. 705711). Acapulco, Mexico, August 2003.

[5] Carnegie Mellon University, Robotics Institute, Projects. MoCHA

[6] Multiagent Planning: A Survey of Research and Applications. Brad Clement, Artificial Intelligence Group, Jet Propulsion Laboratory, California Institute of Technology. Keith Decker, Dept. of Computer and Information Sciences, University of Delaware

[7] Pynadath, D. V., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. Journal of Artificial Intelligence Research (JAIR), 16, 389423.