Distributed decision making of robotic agents

Jelle R. Kok Nikos Vlassis

Informatics Institute, Faculty of Science, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{jellekok,vlassis}@science.uva.nl

Keywords: Multiagent systems, distributed decision making, coordination, robot soccer

Abstract

In order to make a distributed decision in a group of agents, each agent has to take into account the actions of the agents on which it depends. In a dynamic environment, these dependencies may change rapidly as a result of the continuously changing state. Coordination graphs offer a scalable solution to the problem of multiagent coordination by decomposing the problem into smaller subproblems. In this paper, we show how coordination graphs can be applied to dynamic and continuous domains by assigning roles to the agents and then coordinating the different roles. Moreover, we show how an agent can predict the action of the other agents, making communication unnecessary. Finally, we will demonstrate this method on the RoboCup soccer simulation domain.

1 Introduction

A distributed system is defined as a collection of independent computers that appear to the users of the system as a single computer [10]. To accomplish this, the distributed computers transparently distribute the resources among the different machines using a shared communication protocol and hide all irrelevant system-dependent details from the user. One of the most fundamental problems in this domain is to reach consensus among the different agents, for example, whether the different components in a distributed database system either all commit or all abort a transaction.

Multiagent systems (MAS) can be regarded as specialized distributed systems consisting of a collection of agents[1] that coexist in an environment, interact (explicitly or implicitly) with each other, and try to optimize a performance measure. Research in MAS focuses on behavior management issues (e.g. coordination of behaviors) in such systems. Although the individual capabilities of an agent in MAS are often less complex than in distributed systems, the overall (complex) behavior of the system is derived from the interaction of the different agents. These agents could not only have entirely different designs, but also their own set of actions and their own (possible conflicting) interests and goals.

In our case, we are interested in fully cooperative MAS in which all agents share a common goal. A key aspect in such systems is the problem of *coordination*: how do the individual agents reach consensus about the joint action to take in order to successfully achieve the common goal.

In principle game theoretic techniques can be applied to solve the coordination problem [7]. The difficulty with this approach is that the joint action space is exponential in the number of agents. For practical situations, involving many agents, modeling an n-person game therefore becomes intractable. However, the particular structure of the coordination problem can often be exploited to reduce its complexity.

A recent approach involves the use of a *coordination graph (CG)* [4]. In this graph, each node represents an agent, and edges between nodes indicate that the corresponding agents have to coordinate their actions. In a context-specific CG [5] the topology of the graph is dynamically updated based on the current context.

In this paper we describe a framework for distributed decision making in dynamic and continuous domains using coordination graphs. We assume a group of robotic agents that are embedded in a continuous domain and are able to per-

[1] We define an agent as anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators [8]. This definition applies to humans, robotic agents, but also to software agents.

ceive their surroundings with sensors. In order to apply the coordination graph algorithm, we appropriately 'discretize' the continuous state by assigning *roles* to the agents [9] and then, instead of coordinating the different agents, coordinate the different roles. Furthermore, we will describe a method that allows an agent to efficiently predict the optimal action of its neighboring agents, making communication unnecessary. Finally, we work out an extensive example that applies coordination graphs to robot soccer.

The setup of the paper is as follows. In Section 2 we review the coordination problem from a game-theoretic point of view, and in Section 3 we explain the concept of a coordination graph. In Section 4 we describe our framework to coordinate agents in a continuous dynamic environment using roles and without using communication. These two methods are applied to the RoboCup soccer simulation domain in Section 5. Finally, we give our conclusions and discuss possible further extensions in Section 6.

2 The coordination problem

In order to place the coordination problem in a broader context, we first review it from a game theoretic point of view. A strategic game [7] is a tuple $(n, A_{1..n}, R_{1..n})$ where n is the number of agents, A_i is the set of actions of agent i and R_i is the payoff function for agent i. This payoff function maps the selected joint action $A = A_1 \times ... \times A_n$ to a real value: $R_i(A) \rightarrow \mathbb{R}$. Each agent selects an action from its action set, and then receives a payoff based on the actions selected by all agents. The goal of the agents is to select, via their individual decisions, the most profitable joint action. In the remainder of this paper, we are interested in fully cooperative strategic games, so-called coordination games, in which all agents share the same payoff function $R_1 = \ldots = R_n = R$.

A Nash equilibrium defines a joint action $a^* \in A$ with the property that for every agent i holds $R_i(a_i^*, a_{-i}^*) \geq R_i(a_i, a_{-i}^*)$ for all $a_i \in A_i$, where a_{-i} is the joint action for all agents excluding agent i. Such an equilibrium joint action is a steady state from which no agent can profitably deviate given the actions of the other agents. Formally, the coordination problem can be seen as the problem of selecting one out of many Nash equilibria in a coordination game.

Several different methods exist to solve a coordination game, for example by using communication, learning, or by imposing social conventions [1]. In these cases it is assumed that the Nash equilibria can be found and coordination is then the problem of agreeing on a single equilibrium. However, the number of joint actions grows exponentially with the number of agents, making it infeasible to determine all equilibria in the case of many agents. This calls for methods that first reduce the action space before solving the coordination problem. One such approach, explained next, is based on the use of a coordination graph that captures local coordination requirements between agents.

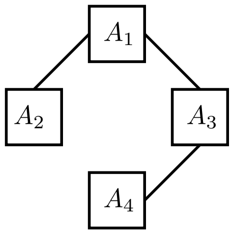
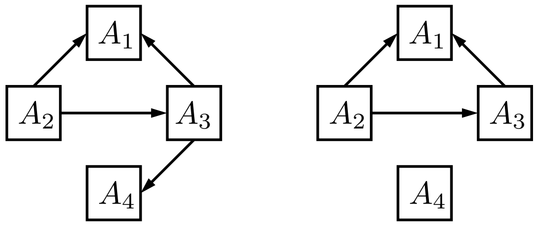


Figure 1: An example coordination graph for a 4-agent problem.

3 Coordination graphs

A coordination graph (CG) represents the coordination requirements of a system [4]. A node in the graph represents an agent, while an edge in the graph defines a (possibly directed) dependency between two agents. Only interconnected agents have to coordinate their actions at any particular instance. Figure 1 shows a possible CG for a 4-agent problem. In this example, A_2 has to coordinate with A_1, A_4 has to coordinate with A_3, A_3 has to coordinate with both A_4 and A_1, and A_1 has to coordinate with both A_2 and A_3. The key idea of CGs is that when the global payoff function can be decomposed as a sum of local payoff functions, the global coordination problem can be replaced by a number of easier local coordination problems. The agents can then find the joint optimal action by using an efficient variable elimination algorithm in combination with a message passing scheme [4].

The algorithm assumes that each agent knows its neighbors in the graph (but not necessarily their payoff function which might depend on other agents). Each agent is 'eliminated' from the graph by solving a local optimization problem that involves only this agent and its neighbors: the agent collects from its neighbors all relevant payoff functions, then optimizes its decision conditionally on its neighbors' decisions, and communicates the resulting 'conditional' payoff function back to its neighbors. A next agent is selected and the process continues. When all agents have been eliminated, each agent communicates its decision to its neighbors in the reverse elimination



A_1	$\langle a_1 \wedge \overline{a_3} \wedge x \ : 4\rangle$	$\langle a_1 \wedge \overline{a_3} \ : 4\rangle$
	$\langle a_1 \wedge \overline{a_2} \wedge x \ : 5\rangle$	$\langle a_1 \wedge \overline{a_2} \ : 5\rangle$
A_2	$\langle \overline{a_2} \wedge x \ : 2\rangle$	$\langle \overline{a_2} \ : 2\rangle$
A_3	$\langle a_3 \wedge a_2 \wedge x \ : 5\rangle$	$\langle a_3 \wedge a_2 \ : 5\rangle$
A_4	$\langle a_3 \wedge a_4 \wedge \overline{x} \ : 10\rangle$	

Figure 2: Initial coordination graph (left) and graph after conditioning on the context $x = true$ (right).

order in order for them to fix their strategy.

The local payoff functions can be matrix-based [4] or rule-based [5]. In the latter case the payoff rules are defined using 'value rules', which specify how an agent's payoff depends on the current context. The context being defined as a propositional rule over the state variables and the actions of the agent's neighbors. These rules can be regarded as a sparse representation of the complete payoff matrices.

As an example, consider the case where two persons have to coordinate their actions to enter a narrow door. We can describe this situation using the following value rule:

$$\begin{aligned} \langle p1 \quad ; \quad & \text{in-front-of-same-door}(A_1, A_2) \quad \wedge \\ & a_1 = \text{enterDoor} \quad \wedge \\ & a_2 = \text{enterDoor} : -50\rangle \end{aligned}$$

This rule indicates that when the two agents are located in front of the same door and both select the same action (entering the door), the global payoff value will be reduced by 50. When the state is not consistent with the above rule (the agents are not located in front of the same door), the rule does not apply and the agents do not have to coordinate their actions. By conditioning on the current state the agents can discard all irrelevant rules, and as a consequence the CG is dynamically updated and simplified. Each agent thus only needs to observe that part of the state mentioned in its value rules.

For a more extensive example, see Figure 2. Beneath the left graph we list all value rules (defined over binary action and context variables) together with the agent the rule applies to. The coordination dependencies between the agents are represented by directed edges, where each (child) agent has an incoming edge from the (parent) agent that affects its decision. After the agents observe the current state, $x = true$, the last rule does not apply anymore and can be removed. As a consequence, the optimal joint action is independent of the action of A_4 and the edges to A_4 can be deleted from the graph as is shown in the right graph of Figure 2.

After the agents have conditioned on the state variables, the agents are one by one eliminated from the graph. Let us assume that we first eliminate A_3 in the above example. After collecting from its neighbors all rules that involve A_3, A_3 has to maximize over the rules $\langle a_1 \wedge \overline{a_3} : 4\rangle\langle a_3 \wedge a_2 : 5\rangle$. For all possible actions of A_1 and A_2, A_3 determines its best response and then distributes this conditional strategy, in this case equal to $\langle a_2 : 5\rangle\langle a_1 \wedge \overline{a_2} : 4\rangle$, to its parent A_2. After this step, A_3 has no children in the coordination graph anymore and can be eliminated. The procedure then continues and after A_2 has distributed its conditional strategy $\langle a_1 : 11\rangle\langle \overline{a_1} : 5\rangle$ to A_1, A_2 is also eliminated. Finally, A_1 is the last agent left and fixes its action to a_1. Now a second pass in the reverse order is performed, where each agent distributes its strategy to its neighbors, who can then determine their final strategy. In this case, this will result in the optimal joint action, $\{a_1, \overline{a_2}, \overline{a_3}\}$ and a corresponding global payoff of 11.

It is not difficult to see that the outcome of this algorithm is independent of the elimination order and the distribution of the rules, and will always result in an optimal joint action.

A limitation of this approach however, is that it is based on propositional rules and therefore only applies to discrete domains. In our case, we are interested in agents that are embedded in continuous domains. Next, we show how we extend this framework to continuous dynamic environments.

4 Dynamic continuous domains

We are interested in problems that involve multiple agents that are embedded in a continuous domain, have sensors with which they can observe their surroundings, and need to coordinate their actions. As a main example we will use the RoboCup simulation soccer domain (see [3] and references therein) in which a team of eleven agents have to fulfill a common goal (scoring more goals than their opponent). Depending on the current situation, certain agents on the field have to coordinate their actions, for example the agent that controls the ball must decide to which nearby agent to pass, etc. Such dependencies can be modeled by a CG that satisfies the following requirements: (i) its connectivity should be dynamically updated based on the current (continuous) state, (ii) it should be sparse in order to keep the dependencies and the associated local coordination problems as simple as possible.

4.1 Context-specificity using roles

Conditioning on a context that is defined over a continuous domain is difficult in the original rule-based CG representation. A way to 'discretize' the context is by assigning *roles* to agents [9]. Roles are a natural way of introducing domain prior knowledge to a multiagent problem and provide a flexible solution to the problem of distributing the global task of a team among its members. In the soccer domain for instance one can easily identify several roles ranging from 'active' or 'passive' depending on whether an agent is in control of the ball or not, to more specialized ones like 'striker', 'defender', 'goalkeeper', etc.

Given a particular local situation, each agent is assigned a role that is computed based on a role assignment function that is common knowledge among agents. The set of roles is finite and ordered, so the most 'important' role is assigned to an agent first, followed by the second most important role, etc. By construction, the same role can be assigned to more than one agent, but each agent is assigned only a single role. Environment-dependent 'potential' functions can be used to determine how appropriate an agent is for a particular role given the current context. Such functions can for example be based on the spatial relationships between the agents or the specific capabilities of an agent. For more details on the assignment of roles to agents see [9] and our experiments later on.

Such an assignment of roles provides a natural way to parametrize a coordination structure over a continuous domain. The intuition is that, instead of directly coordinating the agents in a particular situation, we assign roles to the agents based on this situation and subsequently try to 'coordinate' the set of roles. For example, a priori rules can exist that specify which roles should be coordinated and how. See Section 5 for examples.

The roles can be regarded as an abstraction of the continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. Moreover, roles can reduce the action space of the agents by 'locking out' specific actions. For example, the role of the goalkeeper does not include the action 'score', and in a 'passive' role the action 'shoot' is deactivated. Such a reduction of the action space can offer computational savings, but more importantly it can facilitate the solution of a local coordination game by restricting the joint action space to a subspace that contains only one Nash equilibrium.

4.2 Non-communicating agents

Variable elimination in a CG requires that each agent first receives the payoff functions of its neighboring agents, and after computing its optimal conditional strategy it communicates a new payoff function back to its neighbors. Similarly, in the reverse process each agent needs to communicate its decision to its neighbors in order to reach a coordinated joint action. The elimination order is a priori defined and is common knowledge among the agents.

In may practical dynamic situations, the agents may not be able to communicate with all neighbors (or have the time to finalize all communication before selecting an action) due to failures or time constraints. However, when communication is unavailable the variable elimination algorithm can still be used if we further impose the requirement that the payoff function of an agent i is common knowledge among all agents that are *reachable* from i in the CG. Since only agents that are reachable in the CG need to coordinate their actions, the above requirement in fact frees agents from having to communicate their local payoff functions during optimization.

Moreover, in the noncommunicative case the elimination order neither has to be fixed in advance nor has to be common knowledge among all agents as in [4], but each agent is free to choose *any* elimination order, for example, one that allows the agent to quickly compute its own optimal action. This is possible because a particular elimination order affects only the speed of the algorithm and not the computed joint action.

In summary, each agent i maintains a pool of payoff functions, corresponding to all payoff functions of the agents in its subgraph. Starting from itself, agent i keeps eliminating agents using an appropriate elimination order, until it computes its own optimal action unconditionally on the actions of the others. For each eliminated agent j, the newly generated payoff functions are introduced into the pool of payoff functions of agent i and the process continues. In the worst case, agent i needs to eliminate all agents $j \neq i$, for j reachable from i. Note that, although each agent computes its own action in a different way (during optimization the pool will look different for different agents), the resulting joint action will always be the optimal one.

In terms of complexity, the computational costs for each individual agent are clearly increased to compensate for the unavailable communication. Instead of only optimizing for its own action, in the worst case each agent has to calculate the action of every other agent in the subgraph. The computational cost per agent increases thus linearly with the number of new payoff functions generated during the elimination procedure. Communication, however, is not used anymore which allows for a speedup of the complete algorithm since these extra individual computations may now run in parallel. This is in contrast to the original CG approach where computations need to be performed sequentially.

5 Experiments

We have implemented the described framework in our simulation robot soccer team UvA Trilearn [3] to improve upon the ball passing between teammates. The RoboCup soccer server [2] provides a fully distributed dynamic multiagent domain with both teammates and adversaries. It models many real-world complexities such as noise in object movement, noisy sensors and actuators, limited physical ability and restricted communication.

In case of a non-coordinating pass, a teammate moves to the interception point only after he has observed a change in the ball velocity (indicating someone has passed the ball) and computes that he is the fastest teammate to the ball. Before the ball changes velocity, he has no notion of the fact that he will soon receive the ball and thus does not coordinate with the passing player.

To accomplish coordination, all agents are first dynamically assigned a role based on the current continuous state. Thereafter, these roles are coordinated by performing the variable elimination algorithm using predefined value rules that make use of the available actions and context variables. Next, we will describe in more detail how we use coordination graphs in order to coordinate the passer and the receiver, but also the receiver with the second receiver, that is, the player who will be passed to by the first receiver.

First, we have implemented a role assignment function that assigns the roles *interceptor*, *passer*, *receiver*, and *passive* among the agents using the continuous state information. The assignment of roles can be computed directly from the current state information. For instance, the fastest player to the ball will be assigned the interceptor role when he is not able to kick the ball and will be assigned the passer role when he can kick the ball. All receiver roles are given to the

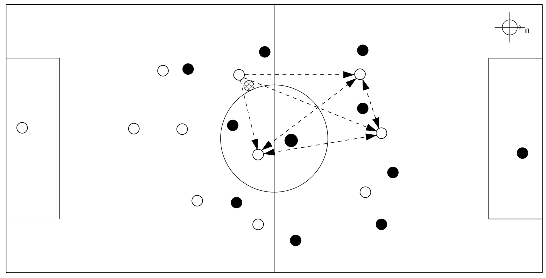


Figure 3: A situation involving one passer and three possible receivers. The other agents are passive.

agents that are inside a predefined range of the ball position. The rest of the players are passive. A common situation is depicted in Figure 3, where the agent with the ball has the passer role, the three players that are in range of the passer are given the receiver role and the other players are passive. This assignment of roles defines the structure of the coordination graph: all interceptors, passers, and receivers are connected. Note that this assignment changes dynamically as the state of the world (and thus the assignment of roles) changes.

As long as the structure of the graph is known, the connected agents can coordinate their actions. Each agent can choose from the following set of actions:

- $passTo(i, dir)$: pass the ball to agent i. The position to which is shot lies at a fixed distance from agent i in the direction $dir \in D$ with $D = \{center, n, nw, w, sw, s, se, e, ne\}$[2].
- $moveTo(dir)$: move in the direction $dir \in D$.
- $dribble(dir)$: move with the ball in the kickable range of the agent in direction $dir \in D$.
- *score*: try to score by shooting towards the best point on the opponent goal line [6].
- *clearBall*: clear the ball to the opponent side by shooting the ball in the bisector of the largest angle between the opponent defenders.
- *moveToStratPos*: move to your strategic position (a position based on the home position of the agent and the current position of the ball).

We also defined state variables that extract important (high-level) information from the world

[2] 'North' is directed towards the opponent goal and 'center' corresponds to a pass directly to the current agent position.

state. The first is is-pass-blocked(i, j, dir) that indicates whether a pass from agent i to the position in direction dir of agent j is blocked by an opponent or not. In this case no opponents are located within a cone from the passing player to this position. The second is is-in-front-of-goal(j) that indicates whether the agent j is located in front of the opponent goal and the last, is-empty-space(i, dir), indicates that there are no opponents in direction dir of agent i.

Finally, we can define the complete strategy of the team by means of value rules which specify the contribution to the global payoff in a specific context. These value rules are specified for each player i and make use of the above defined actions and context variables[3].

$$
\begin{array}{lcl}
\langle p_1^{passer} & ; & \text{has-role-receiver}(j) \quad \wedge \\
 & & \neg\text{isPassBlocked}(i, j, dir) \quad \wedge \\
 & & a_i = \text{passTo}(j, dir) \quad \wedge \\
 & & a_j = \text{moveTo}(dir) : u(j, dir)\rangle \ \forall j \neq i \\
\langle p_2^{passer} & ; & \text{is-empty-space}(i, \text{n}) \quad \wedge \\
 & & a_i = \text{dribble(n)} : 30\rangle \\
\langle p_3^{passer} & ; & a_i = \text{clearBall} : 10\rangle \\
\langle p_4^{passer} & ; & \text{is-in-front-of-goal}(i) \quad \wedge \\
 & & a_i = \text{score} : 100\rangle \\
\langle p_5^{receiver} & ; & \text{has-role-interceptor}(j) \quad \wedge \\
 & & \neg\text{isPassBlocked}(j, i, dir) \quad \wedge \\
 & & a_i = \text{moveTo}(dir) : u(i, dir)\rangle \quad \forall j \neq i \\
\langle p_6^{receiver} & ; & \text{has-role-passer}(j) \quad \wedge \\
 & & \text{has-role-receiver}(k) \quad \wedge \\
 & & \neg\text{isPassBlocked}(k, i, dir) \quad \wedge \\
 & & a_j = \text{passTo}(k, dir2) \quad \wedge \\
 & & a_k = \text{moveTo}(dir2) \quad \wedge \\
 & & a_i = \text{moveTo}(dir) : u(i, dir)\rangle \ \forall j, k \neq i \\
\langle p_7^{receiver} & ; & \text{moveToStratPos} : 10\rangle \\
\langle p_8^{intercep.} & ; & \text{intercept} : 100\rangle \\
\langle p_9^{passive} & ; & \text{moveToStratPos} : 10\rangle
\end{array}
$$

The first rule p_1 indicates that a passer can shoot the ball to the relative direction dir of the player j in case the pass is not blocked by an opponent and the receiver will move in that direction. The value that is contributed to the global payoff is returned by u and depends on the position where the receiving agent j will receive the pass (the closer to the opponent goal the better). The next three rules indicate the other individual options for the passer: dribbling (we only allow forward dribbling), clearing the ball and scoring. Using the same principle as the first rule, we can also create more advanced dependencies. For example, rule p_5 indicates the situation where a receiver already moves to the position it expects the current interceptor to pass the ball to when it reaches the ball. Rule p_6 indicates that a receiver can already move to a position, it will expect the receiver of another pass to shoot the ball to. Rule p_7 is the situation where a receiving player moves to its strategic position on the field. This action is only executed when it is not able to coordinate with one of the other agents, since it contributes only a small value to the global payoff value. Finally, rules p_8 and p_9 contain the single action option for respectively an interceptor (intercept the ball) or a passive player (move to its strategic position).

[3]Note that we enumerate all rules using variables. The complete list of value rules is the combination of all possible instantiations of these variables. Although not specified, $dir \in D$.

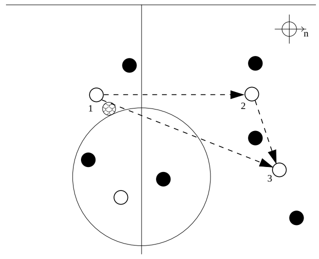


Figure 4: The coordination graph at Fig. 3 after conditioning on the state variables. The passer (agent 1) decides to pass the ball to the first receiver (agent 2), while the second receiver (agent 3) moves to a good position for the first receiver to pass the ball to.

With the above rules, we illustrate that even with a small set of rules a complete (although simple) team strategy can be specified that makes explicit use of coordination. Furthermore, the rules are easily interpretable which makes it possible to add prior knowledge into the problem. Another advantage is that the rules are very flexible: existing rules can directly be added or removed. This makes it possible to change the complete strategy of the team when playing different kinds of opponents.

The above rules contain a lot of context-dependencies represented in the state variables. In Figure 3 we simplified the coordination graph by conditioning on the roles, if we now condition further on the specific context variables, we get the graph depicted in Figure 4, corresponding to the following value rules (we assume for simplicity that only the context variables ¬isPassBlocked(1, 2, s) and ¬isPassBlocked(2, 3, nw) are true):

$$
\begin{array}{rcl}
A_1 : \langle p_1^{passer} & ; & a_1 = \text{passTo}(2, \text{s}) \quad \wedge \\
 & & a_2 = \text{moveTo(s)} : 50\rangle \\
\langle p_2^{passer} & ; & a_1 = \text{dribble(n)} : 30\rangle \\
\langle p_3^{passer} & ; & a_1 = \text{clearBall} : 10\rangle \\
A_2 : \langle p_7^{receiver} & ; & a_2 = \text{moveToStratPos} : 10\rangle \\
A_3 : \langle p_6^{receiver} & ; & a_1 = \text{passTo}(2, dir) \quad \wedge \\
 & & a_2 = \text{moveTo}(dir) \quad \wedge \\
 & & a_3 = \text{moveTo(nw)} : 30\rangle \\
\langle p_7^{receiver} & ; & a_3 = \text{moveToStratPos} : 10\rangle
\end{array}
$$

Now the variable elimination algorithm can be performed. Each agent is eliminated from the graph by maximizing its local payoff. In the case that agent 1 is eliminated first, it gathers all value rules that contain a_1 and distributes its conditional strategy

$$
\begin{array}{rcl}
\langle p_1^{passer} & ; & a_2 = \text{moveTo(s)} \quad \wedge \\
 & & a_3 = \text{moveTo(nw)} : 80\rangle \\
\langle p_1^{passer} & ; & a_2 = \text{moveTo(s)} \quad \wedge \\
 & & a_3 = \neg\text{moveTo(nw)} : 50\rangle \\
\langle p_1^{passer} & ; & a_2 = \neg\text{moveTo(s)} : 30\rangle
\end{array}
$$

to its parents. After agent 2 and 3 have also fixed their strategy, agent 1 will perform passTo(2, s), agent 2 will execute moveTo(s) to intercept the pass and agent 3 will perform moveTo(nw) to intercept a possible future pass of agent 2. In case for some unpredicted reason the first pass fails, the graph will automatically be updated and correspond to the new situation.

To test this approach, we played games against ourselves, with one team using explicit coordination and the other team without using any coordination at all during passing. The latter case was modeled by deleting the rules, p_5 and p_6 from the list of value rules and removing the condition $a_j = \text{moveTo}(dir)$ from the first value rule to indicate that it is not necessary for the receiver to anticipate the pass.

Table 1 shows the results over the course of 10 full-length games. Each player can take 6000 decisions during each match, so the coordination algorithm was executed 60.000 times by each player. The strategy for the team was completely specified by the value rules above. These rules are too general to create very sophisticated scoring possibilities and therefore the difference in goal difference is rather small. The actual coordination, however, was used to improve the passing between the players and therefore we were more interested in detailed statistics about the passing percentages. It turned out that the successful passing percentage over these 10 matches was 82.72% (total of 1450 passes) for the team with the CG and 64.62% (total of 1411 passes) for the team without. These percentages indicate that due to the better coordination of the teammates, fewer mistakes were made when the ball was passed between teammates.

Table 1: Results of 10 games against ourselves, with and without coordination in passing.

	With	Without
Wins	5	2
Draws	3	3
Losses	2	5
Avg. score	0.9 (± 1.19)	0.2 (± 0.42)
Passing %	82.72 (± 2.06)	64.62 (± 2.17)

6 Conclusions and future work

We showed how coordination graphs can be successfully applied to cases where a group of robotic agents are embedded in a dynamic and continuous domain and take decisions in a distributed fashion.. We assigned roles in order to abstract from the continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. We also showed that we can dispense with communication if additional assumptions about common knowledge are introduced.

Currently, we assume that each agent observes that part of the state that affects its local decisions and its role assignment. As future work, we would like to apply the same framework to domains where the agents do not observe all required state information. Possible solutions would be to make the action of the agent dependent on its current state information (i.e., by actively looking towards relevant hidden parts of the state space) or to derive missing state information based on the observed actions from agents (and thus to deduce why the agent is performing its action).

We also want to investigate situations in which agents break down or start acting maliciously. In the current algorithm we neglect the possibility of agent failures. One obvious approach is to remove all value rules involving an agent that appears to have failed (this can either be observed from the agent's sensors or derived from the fact that no messages were received within a specific time interval) and solve the coordination problem without this agent. The returned payoff may be less than the situation in which that agent was available but it is the highest possible payoff possible considering that agent has failed.

Furthermore, we are interested in applying reinforcement learning techniques to a continuous-domain CG in order to learn the payoff functions in an automatic way. Finally, from an application point of view we want to apply the CG model further to the simulation RoboCup, such that the agents also coordinate during other actions than passing, like organizing the defense or obstructing opponent passes.

Acknowledgements

This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414.

References

[1] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Conf. on Theoretical Aspects of Rationality and Knowledge*, 1996.

[2] M. Chen, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin. RoboCup Soccer Server for Soccer Server Version 7.07 and later, 2002. At http://sserver.sourceforge.net/.

[3] R. de Boer and J. R. Kok. The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team. Master's thesis, University of Amsterdam, The Netherlands, Feb. 2002.

[4] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems (NIPS-14)*, Canada, Dec. 2001.

[5] C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, Edmonton, Canada, July 2002.

[6] J. R. Kok, R. de Boer, N. Vlassis, and F. Groen. Towards an optimal scoring policy for simulated soccer agents. In G. Kaminka, P. Lima, and R. Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, pages 292–299, Fukuoka, Japan, 2002. Springer-Verlag.

[7] M. J. Osborne and A. Rubinstein. *A course in game theory*. The MIT Press, Cambridge, MA, 1994.

[8] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[9] M. T. J. Spaan, N. Vlassis, and F. C. A. Groen. High level coordination of agents based on multiagent Markov decision processes with roles. In A. Saffiotti, editor, *IROS'02 Workshop on Cooperative Robotics*, Lausanne, Switzerland, Oct. 2002.

[10] A. S. Tanenbaum. *Distributed Operating Systems*. Prentice Hall, 1995.

[11] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons, Chichester, England, feb 2002.