Multi-robot decision making using coordination graphs

Jelle R. Kok Matthijs T. J. Spaan Nikos Vlassis
Intelligent Autonomous Systems Group, Informatics Institute
Faculty of Science, University of Amsterdam, The Netherlands
{jellekok,mtjspaan,vlassis}@science.uva.nl

Abstract

Within a group of cooperating agents the decision making of an individual agent depends on the actions of the other agents. In dynamic environments, these dependencies will change rapidly as a result of the continuously changing state. Via a context-specific decomposition of the problem into smaller subproblems, coordination graphs offer scalable solutions to the problem of multiagent decision making. We will apply coordination graphs to the continuous domain by assigning roles to the agents and then coordinating the different roles. Finally, we will demonstrate this method in the RoboCup soccer simulation domain.

1 Introduction

A multiagent (multi-robot) system is a collection of agents that coexist in an environment, interact (explicitly or implicitly) with each other, and try to optimize a performance measure. Research in multiagent systems aims at providing principles for the construction of complex systems containing multiple independent agents and focuses on behavior management issues (e.g. coordination of behaviors) in such systems.

In our case, we are interested in fully cooperative multiagent systems in which all agents share a common goal. A key aspect in such systems is the problem of *coordination*: the process that ensures that the individual decisions of the agents result in jointly optimal decisions for the group.

In principle game theoretic techniques can be applied to solve the coordination problem [7]. The problem 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 games 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 an edge indicates 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 will describe a framework to coordinate multiple robots using coordination graphs. We assume a group of robotic agents that are embedded in a continuous and dynamic domain and are able to perceive their surroundings with sensors. The continuous nature of the state space makes the direct application of context-specific CGs difficult. To alleviate the problem, we propose a discretization of the state by assigning *roles* to the agents, and subsequently apply the CG-based method to the derived set of roles.

It turns out that such an approach offers additional benefits: the set of roles allows for the definition of natural coordination rules that exploit prior knowledge about the domain. This greatly simplifies the modeling and the solution of the problem at hand.

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 CG. In Section 4 we will describe our framework to coordinate agents in a continuous dynamic environment using roles, followed by an extensive example using 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 will 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}

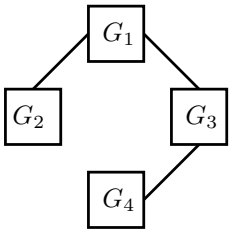


Figure 1: *An example coordination graph for a 4-agent problem.*

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 [1], for example by using communication, learning, or by imposing social conventions. If we assume that with these methods the Nash equilibria can be found, coordination becomes the problem of selecting the same 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 size of the joint 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.

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 (possible 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, G_2 has to coordinate with G_1, G_4 has to coordinate with G_3, G_3 has to coordinate with both G_4 and G_1, and G_1 has to coordinate with both G_2 and G_3. 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 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. Next, we will define this more formally.

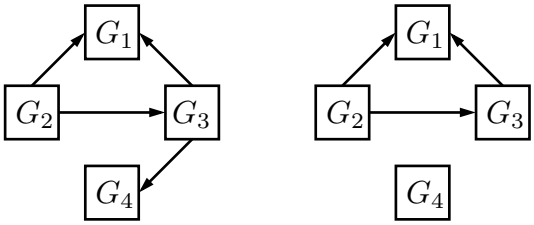
Let $G_1, \ldots, G_n$ be a group of agents, where each agent G_j has to choose an action $a_j \in A_j$ and let X be a set of discrete state variables. The context c is then an element from the set of all possible combinations of the state and action variables, $c \in C \subseteq X \cup A$. A value rule $\langle p\,;\, c : v \rangle$ is a function $p : C \rightarrow \mathbb{R}$ such that $p(x, a) = v$ when c is consistent with the current context and 0 otherwise.

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:

$$\begin{aligned} \langle p1 \quad ; \quad & \text{in-front-of-same-door}(G_1, G_2) \quad \wedge \\ & a_1 = \text{enterDoor} \quad \wedge \\ & a_2 = \text{enterDoor} : -50 \rangle \end{aligned}$$

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 (and 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 all value rules, defined over binary action and context variables, are depicted 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



G_1	$\langle a_1 \wedge \overline{a_3} \wedge x \quad : 4\rangle$	$\langle a_1 \wedge \overline{a_3} \quad : 4\rangle$
	$\langle a_1 \wedge \overline{a_2} \wedge x \quad : 5\rangle$	$\langle a_1 \wedge \overline{a_2} \quad : 5\rangle$
G_2	$\langle \overline{a_2} \wedge x \quad : 2\rangle$	$\langle \overline{a_2} \quad : 2\rangle$
G_3	$\langle a_3 \wedge a_2 \wedge x \quad : 5\rangle$	$\langle a_3 \wedge a_2 \quad : 5\rangle$
G_4	$\langle a_3 \wedge a_4 \wedge \overline{x} \quad : 10\rangle$	

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

action is independent of the action of G_4 and the edge to G_4 can be deleted from the graph as 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 G_3 in the above example. After collecting all relevant rules, G_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 G_1 and G_2, G_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 G_2. After this step, G_3 has no children in the coordination graph anymore and is eliminated. The procedure then continues and after G_2 has distributed its conditional strategy $\langle a_1 : 11\rangle\langle \overline{a_1} : 5\rangle$ to G_1, it is also eliminated. Finally, G_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 parents, who then determine their final strategy. This results in the optimal joint action, $\{a_1, \overline{a_2}, \overline{a_3}\}$ and a global payoff of 11.

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 [4].

A limitation of this approach is that it is based on propositional rules and therefore only applies to discrete domains. However, we are interested in robots that are embedded in continuous domains. Next, we will show how to utilize this framework in continuous dynamic environments.

4 Dynamic continuous environments

We are interested in problems that involve multiple robots 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.

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 [8]. 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. For details on the assignment of roles to agents see [8].

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. A priori rules exist that specify which roles should be coordinated and how. In Section 5 we give a detailed example from robot soccer.

The roles can be regarded as an abstraction of a continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. Furthermore, 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.

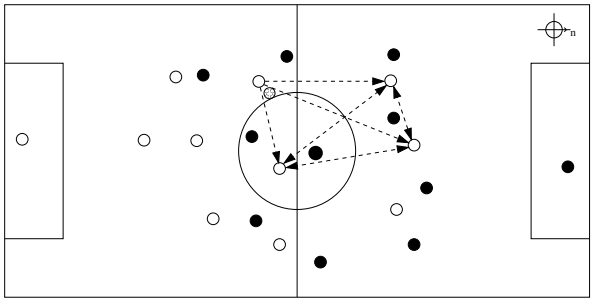


Figure 3: *A situation involving one passer and three possible receivers. The other agents are passive.*

5 Experiments

We have implemented this 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 multi-robot 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.

The RoboCup soccer simulation does not allow agents to communicate with more than one agent at the same time, which makes it impossible to apply the original variable elimination algorithm. Therefore, we have decided to make the state fully observable to all agents. This makes communication superfluous, since each agent can model the complete variable elimination algorithm by itself (see [6] for details). This has no effect on the outcome of the algorithm.

In the non-coordinating case a teammate moves to the interception point only after he has observed a change in the ball velocity (after someone has passed the ball) and concludes 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 does not coordinate with the passing player.

To accomplish coordination, all agents are first dynamically assigned a role based on the current state. Next, these roles are coordinated by performing the variable elimination algorithm using predefined value rules that make use of the available actions and context variables. Hereafter, 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 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 changes.

Now all connected agents have to coordinate their actions. For this, each agent can select one of the following actions:

- $passTo(i, dir)$: pass the ball to a position with a fixed distance from agent i in the direction $dir \in D = \{center, n, nw, w, sw, s, se, e, ne\}$[1].
- $moveTo(dir)$: move in the direction $dir \in D$.
- $dribble(dir)$: move with ball in direction $dir \in D$.
- $score$: try to score in the opponent goal.
- $clearBall$: shoot the ball hard between the opponent defenders to the opponent side.
- $moveToStratPos$: move to agent's strategic position (based on home and current ball position).

We also defined state variables that extract important (high-level) information from the world 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[2].

$$\begin{aligned}\langle p_1^{passer} \quad ; \quad & \text{has-role-receiver}(j) \quad \wedge \\ & \neg\text{isPassBlocked}(i, j, dir) \quad \wedge \\ & a_i = \text{passTo}(j, dir) \quad \wedge \\ & a_j = \text{moveTo}(dir) : u(j, dir)\rangle \ \forall j \neq i\end{aligned}$$

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

[2] Note that we enumerate all rules using variables. The complete list of value rules is the combination of all possible instantiations of these variables. In all rules, $dir \in D$.

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

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 $u(j)$ 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, 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 describes 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 has only a small 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).

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 $\neg$isPassBlocked$(1, 2, \text{s})$ and $\neg$isPassBlocked$(2, 3, \text{nw})$ are true):

$$
\begin{aligned}
G_1 : \langle p_1^{passer} \quad ; \quad & a_1 = \text{passTo}(2, \text{s}) \quad \wedge \\
& a_2 = \text{moveTo(s)} : 50\rangle \\
\langle p_2^{passer} \quad ; \quad & a_1 = \text{dribble(n)} : 30\rangle \\
\langle p_3^{passer} \quad ; \quad & a_1 = \text{clearBall} : 10\rangle \\
G_2 : \langle p_7^{receiver} \quad ; \quad & a_2 = \text{moveToStratPos} : 10\rangle \\
G_3 : \langle p_6^{receiver} \quad ; \quad & a_1 = \text{passTo}(2, dir) \quad \wedge \\
& a_2 = \text{moveTo}(dir) \quad \wedge \\
& a_3 = \text{moveTo(nw)} : 30\rangle \\
\langle p_7^{receiver} \quad ; \quad & a_3 = \text{moveToStratPos} : 10\rangle
\end{aligned}
$$

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

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

to its parents. After agent 2 and 3 have also fixed their strategy, agent 1 will perform passTo$(2, \text{s})$, 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, 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

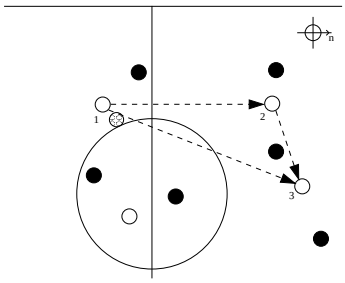


Figure 4: *The coordination graph at Fig. 3 after conditioning on the state variables. The passer (agent 1) decides to pass the ball to the first receiver (agent 2), while the second receiver (agent 3) moves to a good position for the first receiver to pass the ball to.*

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)

executed 60.000 times by each player. The strategy for the team was completely specified by the value rules above. These rules are not specific enough to create good scoring probabilities 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.

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. 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.

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). Second, 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 *Proc. 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 14*. The MIT Press, 2002.

[5] C. Guestrin, S. Venkataraman, and D. Koller. Context-specific multiagent coordination and planning with factored MDPs. In *AAAI 8th Nation. Conf. on Artificial Intelligence*, Edmonton, Canada, July 2002.

[6] J. R. Kok, M. T. J. Spaan, and N. Vlassis. An approach to noncommunicative multiagent coordination in continuous domains. In M. Wiering, editor, *Benelearn 2002: Proceedings of the Twelfth Belgian-Dutch Conference on Machine Learning*, pages 46–52, Utrecht, The Netherlands, Dec. 2002.

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

[8] 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.