Bridging the Gap between AI Planning and Simulation 2D: A DEC-POMDP Perspective

Feng Wu, Aijun Bai, and Xiao-Ping Chen

Multi-Agent Systems Lab,
School of Computer Science and Technology,
University of Science and Technology of China
{wufeng, baj}@mail.ustc.edu.cn, xpchen@ustc.edu.cn

Abstract. We propose a novel solution to bridge the gap between AI planning and Simulation 2D. The basic idea is to formulate the soccer problem as a DEC-POMDP and solve it using sample-based approaches. A middleware is introduced to communicate with the planner and the soccer server. The goal of this work is to further boost the research of AI planning using Simulation 2D as a standard testbed.

1 Introduction

As one of oldest leagues in RoboCup, Simulation 2D has achieved great successes and inspired many researchers all over the world to engage themselves in this game each year [3]. Hundreds of research articles based on Simulation 2D have been published [1]. Comparing to other leagues in RoboCup, the key advantage of Simulation 2D is the abstraction made, which relieves the researchers from having to handle low-level robot problems such as object recognition, communications, and hardware issues [2]. The abstraction enables researchers to focus on higher level concepts such as cooperation and learning. One important goal of Simulation 2D is to serve as a testbed for AI and boost the research in related fields. It has been more than a decade since the first competition in 1997. However, the gap between AI planning and Simulation 2D is still relatively large.

Generally, planning is the process of creating a sequence of actions that achieve some desired goals. Typical planning problems are characterized by a structured state space, a set of possible actions, a description of the effects of each action, and an objective function. A planner takes inputs of the model and produces a sequence of actions that lead from the initial state to a state meeting the goals. In known environments, a complete plan can be found offline and evaluated prior to execution. In unknown domains, the plan may need to be revised online. State-of-the-art techniques for planning include dynamic programming, combinatorial optimization, reinforcement learning, etc.

Stone *et al.* [7] have done a lot of impressive work on applying reinforcement learning methods to Simulation 2D. Their approaches learn higher-level decisions in a keepaway subtask using episodic SMDP Sarsa(λ) with linear tile-coding

[1] http://www.cs.utexas.edu/~pstone/tmp/sim-league-research.pdf

function approximation. More precisely, their robots lean individually when to hold the ball and when to pass to a teammate. The key challenges present in their work are the large state space, hidden and uncertain state, multiple independent agents learning simultaneously and long and variable delays in the effects of actions. Most recently, they extended their work to a more general task named half field offense [4]. On the same reinforcement learning track, Riedmiller *et al.* [5] have developed several effect techniques for learning mainly lower-level skills in Simulation 2D.

In this paper, we propose an alternative idea for automated planning in Simulation 2D. It extends the most recent approach in DEC-POMDPs to learn the control rules (or policies) by directly interacting with the simulator. Given the huge state space of Simulation 2D, it is more scalable to do planning with only state samples and construct policies using abstracted actions and observations. A middleware is implemented to communicate between the planner and the soccer server. Basic protocols are designed to enable the planner working with universal models such as DEC-POMDPs. Furthermore, the middleware can benefit from the existing team codes for implementing the low-level acting and sensing functions. With the planner and the middleware, a robot soccer team can compete with other teams on the standard platform–the soccer server–used in RoboCup.

The remainder of the paper is organized as follows. Firstly, we will introduce the basic background knowledge. Then, we propose the ideas on modeling and planning in Simulation 2D with DEC-POMDPs as well as the implementation of the middleware. Finally, we summarize the paper with conclusions.

2 Background

In this section, we briefly introduce the backgrounds, namely Simulation 2D and the DEC-POMDP model. We assume that readers from the RoboCup community already have sufficient knowledge on Simulation 2D. For DEC-POMDPs, we only describe basic concepts of the model but refer [6] for more details.

2.1 Simulation 2D

The match of Simulation 2D is based on a simulator called the soccer server in a client-server style. The server is a physical soccer simulation system and each client is a separate process that communicates with the server via network sockets. A team can have up to 12 clients including 11 players (10 fielders plus 1 goalie) and a coach. Each client sends requests to the server regarding the actions it wants to perform (e.g. kick the ball, turn, run, etc). The server receives those messages, handles the requests, and updates the environment accordingly. Then, the server provides all players with sensory information (e.g. visual data regarding the position of objects on the field, or data about the player's resources like stamina or speed). The game is played with discrete time intervals (or cycles). A detailed description of Simulation 2D can be found in Chen *et al.* [2].

Fig. 1. Snapshot of Simulation 2D

2.2 The DEC-POMDP Model

The Markov Decision Process (MDP) and its partially observable counterpart (POMDP) have proved very useful for planning and learning under uncertainty. The Decentralized POMDP (DEC-POMDP) [1] offers a natural extension of these frameworks for cooperative multi-agent settings. We use DEC-POMDPs to model Simulation 2D by assuming that the opponents are always executing fixed policies and can be treated as parts of the environments. Hence we can narrow down our focus on the cooperation and collaboration of a single team.

Definition 1 (DEC-POMDP). *A decentralized partially observable Markov decision process can be defined as a tuple* $\langle I, S, \{A_i\}, \{\Omega_i\}, P, O, R, b^0, T\rangle$*, where*

- I *is a collection of agents, identified by* $i \in \{1 \dots m\}$*, and* T *is the time horizon of the problem.*
- S *is a finite state set and* $b^0 \in \Delta(S)$ *is the initial belief state, i.e. a probability distribution over states.*
- A_i *is an action set for agent* i*. We denote by* $\boldsymbol{a} = \langle a_1, a_2, \cdots, a_m\rangle$ *a joint action where* $a_i \in A_i$ *and* $\boldsymbol{A} = \times_{i\in I} A_i$ *is the joint action set.*
- Ω_i *is an observation set for agent* i*. Similarly* $\boldsymbol{o} = \langle o_1, o_2, \cdots, o_m\rangle$ *is a joint observation where* $o_i \in \Omega_i$ *and* $\boldsymbol{\Omega} = \times_{i\in I}\Omega_i$ *is the joint observation set.*
- $P : S \times \boldsymbol{A} \to \Delta(S)$ *is the state transition function and* $P(s'|s, \boldsymbol{a})$ *denotes the probability of the next state* s' *when the agents take joint action* $\boldsymbol{a}$ *in state* s*.*
- $O : S \times \boldsymbol{A} \to \Delta(\boldsymbol{\Omega})$ *is an observation function and* $O(\boldsymbol{o}|s', \boldsymbol{a})$ *denotes the probability of observing* $\boldsymbol{o}$ *after taking joint action* $\boldsymbol{a}$ *with outcome state* s'*.*
- $R : S \times \boldsymbol{A} \to \mathcal{R}$ *is a reward function and* $R(s, \boldsymbol{a})$ *is the immediate reward after agents take* $\boldsymbol{a}$ *in state* s*.*

In a DEC-POMDP, each agent $i \in I$ executes an action a_i based on its policy at each time step t. Thus a joint action $\boldsymbol{a}$ of all the agents is performed, followed by a state transition of the environment and an identical joint reward obtained by the team. Then agent i receives its private observation o_i from the

Algorithm 1: Example of Control Rules

```
if has-ball then
    if can-shoot then
        shoot-goal
    else if should-pass then
        pass-ball(best-agent)
    else
        dribble-ball
else if is-fast-to-ball then
    intercept-ball
else
    go-formation
```

environment and updates its policy for the next execution cycle. The goal of each agent is to choose a policy that maximizes the accumulated reward of the team over the horizon T, i.e. $\sum_{t=1}^{T} E[R(t)|b^0]$.

Generally, a policy q_i is a mapping from agent i's observation history to an action a_i and a joint policy $\boldsymbol{q} = \langle q_1, q_2, \cdots, q_m \rangle$ is a vector of policies, one for each agent. The value of a fixed joint policy $\boldsymbol{q}$ at state s can be computed recursively by the Bellman equation: $V(s, \boldsymbol{q}) = R(s, \boldsymbol{a}) + \sum_{s', \boldsymbol{o}} P(s'|s, \boldsymbol{a}) O(\boldsymbol{o}|s', \boldsymbol{a}) V(s', \boldsymbol{q_o})$, where $\boldsymbol{a}$ is the joint action specified by $\boldsymbol{q}$, and $\boldsymbol{q_o}$ is the joint sub-policy of $\boldsymbol{q}$ after observing $\boldsymbol{o}$. Given a state distribution $b \in \Delta(s)$, the value of a joint policy $\boldsymbol{q}$ can be computed by $V(b, \boldsymbol{q}) = \sum_{s \in S} b(s) V(s, \boldsymbol{q})$.

3 Modeling and Planning

The soccer server operates in discrete time steps, $t = 0, 1, 2, \cdots$, each representing 100 micro seconds of simulated time. The DEC-POMDP model directly maps to the discrete-time episode of the soccer server and the planning is to decide what to do at each time step. In principle, it is possible to do planning both for the offense scenario and the defense scenario. In this paper, we take the offense task as an example to present our ideas. In the following parts, we use the words *agent* and *robot* interchangeably to denote a player in the team.

3.1 Modeling Simulation 2D

In order to incorporate domain knowledge and simplify the planning task, the robots choose actions not from the simulator's primitive actions such as `dash` and `kick`, but from higher level actions constructed from skills used in our WrightEagle team. Depending on whether the robot possessing the ball or not, the high level actions include:

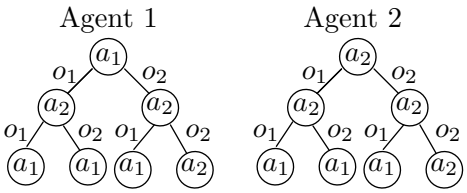

Fig. 2. Example of policy trees

- `pass-ball(k)`: The robot directly kicks the ball to the k-th teammate in a queue. One simplest way to maintain the queue is based on the distance of each teammate to the ball. Hence the k-th teammate in the queue is the k-th closest to the ball, where $k = 2, 3, \cdots, m$.
- `dribble-ball`: The robot dribble towards the goal as fast as possible. A direction is chosen to maximize the dribbling distance while avoid the defense players in the path.
- `shoot-goal`: The robot kicks the ball towards the goal in the direction with the consideration of the goalie, other defenders and the goalposts.
- `intercept-ball`: The robot dashes directly towards the interception point to get the ball. The interception point is calculated based on the dynamic information of the ball.
- `go-formation`: The robot dashes towards a position in order to maintain a predefined formation of the team.

Given the high level skills above, the action set of each agent i in the DEC-POMDP model could be $A_i = \{$ `pass-ball(k)`, `dribble-ball`, `shoot-goal`, `intercept-ball`, `go-formation` $\}$. It is straightforward to include other high level actions in this model, e.g. other types of passing or dribbling behaviors. Using these actions, it is straightforward for researchers to manually write control rules for the team. A naive example is shown in 1.

In DEC-POMDPs, the control rules (a.k.a the policies) are often represented as decision trees, where the nodes are actions to be taken and the branches are observations received from the environment as shown in Fig. 2. In the above example, the observations are the combinations of the following information:

- `has-ball`: The robot is possessing the ball.
- `is-fast-to-ball`: The robot is the fastest agents to get the ball.
- `can-shoot`: The robot is able to score by shooting the ball to the goal.
- `should-pass`: The robot should pass the ball to one of its teammates instead of dribbling by itself.

By considering merely boolean variables, agent i's observation set is $\Omega_i = \{$`has-ball` $\wedge$ `is-fast-to-ball` $\wedge$ `can-shoot` $\wedge$ `should-pass` $\}$ and the observation size is $|\Omega_i| = 2^4$. Note that the observation variables have interdependence, e.g. `can-shoot` and `should-pass` can be `true` if and only if the variable `has-ball` is `true`. Therefore the structure of the observation space

Algorithm 2:

```
Generate default policies for each agent.
Generate a set of potential scenarios for each step.
for t = T to 0 do
    foreach decision node do
        b ← Choose t-step scenarios by their distribution.
        repeat
            foreach agent i ∈ I do
                Keep the other agents' policies fixed.
                Estimate the policy parameters by simulation.
                Improve agent i's policy based on the estimate.
        until no improvement in all agents' policies
return the improved policies for all agents.
```

can be exploited to further simplify the model. Rather than boolean variables, `can-shoot` and `should-pass` can be real values in practice to represent the chance to score or the reward of passing instead of dribbling. Obviously, this will complicate the model by making the observation space continual.

In Simulation 2D, the soccer server sends each robot its own sensory information at each time step. The real system state maintained by the server is hidden but partially observable to the robots. Besides, the state space of Simulation 2D is continual with hundreds of dimensions including the positions and velocities of all the players and the ball. All the dynamics of states are simulated by physical rules. There are no closed forms of the transition or observation function such as P and O in DEC-POMDPs. These present major challenges for planning and learning in Simulation 2D. The reinforcement learning approaches assume the local estimate maintained by each robot is accurate enough to represent the underlying state and learn the corresponding Q-value with function approximation methods [7, 4]. In this paper, we propose an alternative solution using the recent approach for DEC-POMDPs [9], which can learn the policies by directly interacting with the simulator. The main advantage of this approach is that it is not necessary to explicitly represent the states when computing the policies.

3.2 Planning in DEC-POMDPs

When a complete model of the domain is available, DEC-POMDPs can be solved using a wide range of optimal or approximate algorithms [6]. However, existing DEC-POMDP algorithms assume that a complete model of the domain is known. This assumption does not hold in Simulation 2D, the dynamics of which includes complex physical systems. Incomplete domain knowledge is often addressed by reinforcement learning algorithms. Monte-Carlo methods are useful learning techniques that allow agents to choose actions based on experience. These methods require no prior knowledge of the dynamics, as long as sample trajectories can be generated online or using a simulator of the environment.

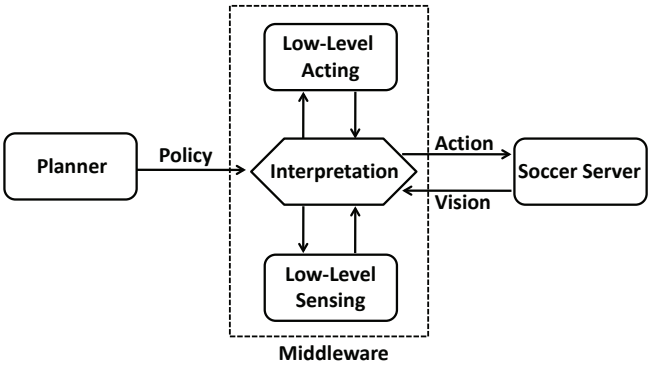

Fig. 3. The Architecture of the Middleware

Most recently, Wu *et al.* introduced the decentralized rollout sampling policy iteration (DecRSPI) algorithm [9] for finite-horizon DEC-POMDPs. It computes a set of cooperative policies using Monte-Carlo methods, without having an explicit representation of the dynamics of the underlying system. DecRSPI first samples a set of reachable belief states based on some heuristic policies. Then it computes a joint policy for each belief state based on the rollout estimations. DecRSPI learns policies from experience obtained by merely interacting with the environment. The learned policies are totally decentralized without any assumption about communication or global observability.

In Algorithm 2, we outline the basic processes of DecRSPI. The default policies of each agent can be initialized manually or even randomly. To generate the set of potential scenarios, one choice is to sample from the game logs by running certain heuristic policies. The actions and observations are high level concepts abstracted from the server's primitives. DecRSPI provides a potential framework for planning in Simulation 2D. Comparing to reinforcement learning approaches [7], the advantage of DecRSPI is that it can learn decentralized policies with merely the sampled state set.

3.3 Implementing the Middleware

The actions and observations described above are, in themselves, quite abstract, and each of them depends on possibly more than one primitive functions of the soccer server. To achieve this level of abstraction in Simulation 2D, it is necessary to implement a middleware as the bridge between the soccer server and the agents. The basic functions of the middleware are twofold: (1) It receives the information from the server, processes it and outputs a set of observations to the agents; (2) It takes actions from the agents, interprets them, and sends a series of primitive commands to the server. The benefit of the middleware is to bridge the gap between the low-level function of the soccer server and the high-level representation of the planner.

To implement the middleware, we borrow ideas from RL-Glue [8], a language-independent software for reinforcement learning experiments. The architecture of the middleware is shown in Fig. 3. We develop protocols for communication between the planner and the middleware. The protocols describe the input and outcome of the planner, i.e. the abstracted actions and observations. In the middleware, the messages received from the server are processed by the low-level sensing component and the high-level actions of the planner are interpreted by the low-level acting component. The low-level components can be implemented by other existing team sources such as WrightEagleBase. It is free for the users to develop their own implementation of the low-level programs and link them to the middleware.

Similar to RL-Glue, the middleware can be used either in internal or external mode. In internal mode, all the components are linked into a single program, and their communication is through function calls as long as all of them are written exclusively in C/C++. In external mode, the planner, the low-level acting and sensing components are into separate programs. Each program connects to the interpreter program, and all communication is over TCP/IP socket connections. External mode allows these programs to be written in any programming language that supports socket communication such as C/C++, Java, Python, etc. Each mode has its strengths and weaknesses. Internal mode has less overhead while external mode is more flexible and portable.

4 Conclusion

In this paper, we proposed a novel solution for planning in Simulation 2D. The middleware enables planners to interact with the soccer server. The protocols used by the middleware formulate the messages of the soccer server as abstracted actions and observations. The users of the middleware can design their own implementations of low level functions and utilize the resources of existing team codes. It is hopeful that the team developed over the middleware can compete with other teams using the standard soccer server in Simulation 2D. The planner can lend itself the power of the existing techniques in the literature and further boost the research in the AI community.

References

1. D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 32–37, 2000.
2. M. Chen, K. Dorer, E. Foroughi, F. Heintz, Z. Huangy, S. Kapetanakis, K. Kostiadis, J. Kummeneje, J. Murray, I. Noda, O. Obst, P. Riley, T. Steeffens, Y. Wang, and X. Yin. *Users Manual: RoboCup Soccer Server*, 2003.
3. T. Gabel and M. Riedmiller. On progress in RoboCup: The simulation league showcase. In *RoboCup 2010: Robot Soccer World Cup XIV*, 2010.

4. S. Kalyanakrishnan, Y. Liu, and P. Stone. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup 2006: Robot Soccer World Cup X*, 2006.
5. M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2007.
6. S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.
7. P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(2):165–188, 2005.
8. B. Tanner and A. White. RL-Glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10(Sep):2133–2136, 2009.
9. F. Wu, S. Zilberstein, and X. Chen. Rollout sampling policy iteration for decentralized pomdps. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 666–673, 2010.