Fifty-Storms2017: Team Description Paper

Harukazu Igarashi[1], Jun Yamagishi[2], and Masaharu Irikura[3]

Shibaura Institute of Technology, 3-7-5 Toyosu, Koto-ku, Tokyo 135-8548, Japan
{[1]arashi50, [2]al13108, [3]al13013}@shibaura-it.ac.jp

Abstract. Fifty-Storms2017 team participates in the RoboCup Soccer Simulation 2D League 2017. This team is based on the open-version codes of the HELIOS team, agent2d (ver.3.1.1), and exploits the results of reinforcement learning, where a policy gradient method derives appropriate policies for midfielders and forwards. In this learning, rewards are given to soccer agents by humans who are watching the games to modify the policies for passers and receivers.

1 Introduction

From 2005, *Fifty-Storms*, had participated in such domestic competitions as RoboCup Japan Open, changed its underlying base team from *UvA Trilearn* 2003 [1] to the open-version codes of *HELIOS* [2], called *agent2d* in 2008. After adding new dribbling skills and modifications to optimize the abilities of agent2d (ver.1.0.0), Fifty-Storms participated in RoboCup Japan Open 2008 and narrowly lost the championship match to HELIOS.

In the following year, Fifty-Storms 2009 exploited the research results of reinforcement learning for the pass selection problems of midfielders (MFs) and the positioning problems for forwards (FWs) to receive a through pass [3][4]. Fifty-Storms participated in RoboCup 2009 and finished 12th and they finished 17th the next year.

In 2010, a new decision-making algorithm for a player with the ball was added to agent2d (ver.3.0.0). That framework for searching for action sequences is called *chain actions*. Using the latest version (3.1.1) of agent2d, Fifty-Storms 2017 proposes a new state evaluation function for chain-action algorithms and applies reinforcement learning to adjust the weight parameters in the function. In this team, not only players with the ball but also those who will probably receive a pass can use the chain-action algorithm to decide the point at which players should move to receive the ball. This team description paper briefly details the learning method for Fifty-Storms 2017.

2 Decision Making of Agent's behaviors

2.1 Stochastic Policy Using Tree Search

A new online multi-agent planning system was implemented in agent2d (ver.3.0.0). It uses a tree search when a player with the ball makes a decision because a cooperative behavior can be represented as a sequence of kick actions by multiple agents. Actions are generated and stored as nodes of a search tree. A path from the root node to a leaf node represents an action sequence called chain actions. All nodes are evaluated by static functions called *state evaluation functions*.

In the latest version of agent2d, 3.1.1, the best-first search algorithm determines the best path from a search tree. Fig.1 shows an example of an action-state search tree. The numbers under the nodes are the values of the states evaluated by a state evaluation function. In this case, the path from root node S to node S_c is selected as the best path and the agent chooses action c at current state S.

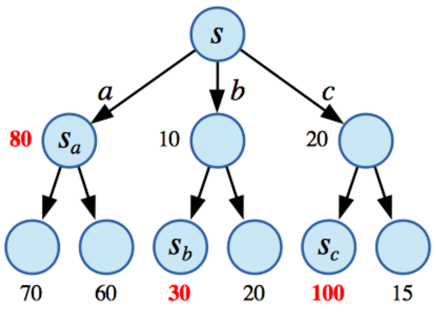

Fig. 1 Example of a search tree

In our model, we use a stochastic policy based on the values of actions. An action's value is defined by the largest value of the nodes under the subtree generated by the action. In this example, action value $E_a(a)$ of action a is given by state value $E_s(S_a)$ of state S_a, i.e., $E_a(a)=E_s(S_a)=80$. In the same way, $E_a(b)=E_s(S_b)=30$ and $E_a(c)=E_s(S_c)=30$. We use the following Boltzmann distribution function as a stochastic policy for an agent to determine its action:

$$\pi\left(a|s;\omega\right) \equiv \frac{e^{E_a(a,s)/T}}{\sum_{x\in A(s)} e^{E_a(x,s)/T}} = \frac{e^{E_s(S_a;\omega)/T}}{\sum_{x\in A(s)} e^{E_s(S_x;\omega)/T}}, \qquad (1)$$

where $E_a(a,s)$ is an action evaluation function of action a at state s and $A(s)$ is a set of feasible actions at s. Symbol ω denotes a set of the parameters included in state evaluation function $E_s(S_a;\omega)$ and T is a parameter called temperature.

2.2 Actions of Players in Possession of Ball

We used nine classes for agent actions that generate edges in the search tree (Table 1). The first eight classes are defined in agent2d. We added a "hold" action class where a player stays at a current position and does nothing.

Table 1 Action classes of players with ball

Acton class	Play
ActGen_StrickCheckPass	direct/lead/through pass
ActGen_Cross	cross pass
ActGen_DirectPass	direct pass
ActGen_ShortDribble	short dribble
ActGen_SelfPass	long dribble
ActGen_SimpleDribble	simple dribble
ActGen_Shoot	shoot
ActGen_hold	hold and stay

2.3 Actions of Receivers

In agent2d, only players in possession of the ball make decisions using the chain-action algorithm. Fifty-Storms 2017 forces players without the ball to use the chain-action algorithm to choose the best position to receive a pass. For the receiver's movement, we define a new action class that consists of "turn" and "dash" actions. Fig. 2 shows an example of the candidates for a receiver's moving action. We consider only search trees whose depths are one for a receiver to choose a moving point in the chain-action algorithm.

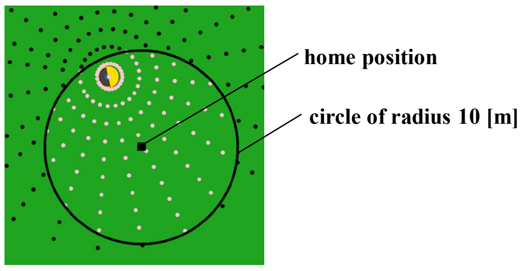

Fig. 2 Candidates of a receiver's moving action

3 Learning by Soccer Agents

3.1 Policy Gradient Approach to Soccer Agents

The RoboCup Simulation 2D League serves as a test bed to learn coordination in multi-agent systems because real robots do not have to be controlled, and learning coordinative behaviors among players can be focused on. As an example of multi-agent learning in a soccer match, Igarashi et al. proposed and applied a policy gradient approach to realize coordination between a kicker and a receiver in direct free kicks [4]. They dealt with learning problems between a kicker and a receiver when a direct free kick is awarded just outside the opponent's penalty area and proposed a function that expressed heuristics to evaluate a candidate target point for effectively sending/receiving a pass and scoring. However, they only addressed the attacking problems of 2v2 (two attackers and two defenders), and their base team [4] was UvA Trilearn 2003. They applied the policy gradient approach to the pass selection problems of MFs and the positioning problems for FWs to receive a through pass and implemented the learning results into Fifty-Storms 2009 [3].

3.2 Characteristics of Policy Gradient Method

A policy gradient method is a kind of reinforcement learning scheme that originated from Williams's REINFORCE algorithm [5]. This method locally increases and maximizes the expected reward per episode by calculating the derivatives of the expected reward function of the parameters included in a stochastic policy function. This method, which has a firm mathematical basis, can be easily applied to many learning problems and used for them even in non-Markov Decision Processes [6][7].

The policy gradient method in refs. [4] and [7] has the following technical characteristics. For autonomous action decisions and the learning of each agent, the policy function for the entire multi-agent system was approximated by the product of each agent's policy function [8], defined by

$$\pi^{\lambda}\left(a^{\lambda}\middle|s^{\lambda},\{\omega_j^{\lambda}\}\right) \equiv \frac{e^{-E^{\lambda}\left(a^{\lambda};s^{\lambda},\{\omega_j^{\lambda}\}\right)/T}}{\sum_a e^{-E^{\lambda}\left(a;s^{\lambda},\{\omega_j^{\lambda}\}\right)/T}}, \tag{2}$$

where a^{λ} is the action of agent λ and s^{λ} is the state perceived by agent λ. Function E^{λ} in Eq. (2) is an energy function of Boltzmann distribution function π^{λ} and an objective function that evaluates an action of agent λ. At the end of each learning episode σ, common reward r (σ) is given to all agents. The derivative of the expectation of reward $E[r]$ for parameter ω^{λ}_j can be calculated to derive the following learning rule on ω^{λ}_j:

$$\Delta\omega_j^{\lambda} = \varepsilon \cdot r(\sigma) \sum_{t=0}^{L(\sigma)-1} e_{\omega_j}^{\lambda}(t) \big/ T, \tag{3}$$

where $L(s)$ is the length of episode σ and ε (>0) is a learning coefficient. e_{ω} are called characteristic eligibilities [5], as shown in Eq. (4), when π_{λ} is given by (2):

$$e_{\omega_j^{\lambda}}(t) \equiv \frac{\partial}{\partial\omega_j^{\lambda}}\pi^{\lambda}\left(a^{\lambda}\middle|s^{\lambda},\{\omega_j^{\lambda}\}\right) = -\frac{1}{T}\left(\frac{\partial E^{\lambda}\left(a^{\lambda};s^{\lambda},\{\omega_j^{\lambda}\}\right)}{\partial\omega_j^{\lambda}} - \left\langle \frac{\partial E^{\lambda}\left(a^{\lambda};s^{\lambda},\{\omega_j^{\lambda}\}\right)}{\partial\omega_j^{\lambda}} \right\rangle_{\pi^{\lambda}}\right), \tag{4}$$

where $\langle X\rangle_{\pi}$ means the expectation of $X(a)$ with respect to stochastic variable a distributed by distribution function π.

4 Learning System in Fifty-Storms 2017

4.1 State Evaluation Function

We replaced an action evaluation function with a state evaluation function in the agent policy in Eq. (1). Characteristic eligibilities e_ω in Eq. (4) are expressed as

$$e_{\omega_j}(t) \equiv \frac{\partial}{\partial \omega_j}\pi\left(a(t)\middle|s(t),\{\omega_j\}\right) = -\frac{1}{T}\left(\frac{\partial E_s\left(S_{a(t)};\{\omega_j\}\right)}{\partial \omega_j} - \left\langle \frac{\partial E_s\left(S_{a(t)};\{\omega_j\}\right)}{\partial \omega_j}\right\rangle_\pi\right), \tag{5}$$

where suffix λ of the agent is omitted for simplicity. State evaluation function $E_s(s)$ is defined by a linear function of terms U_j that evaluate state s. ω_j is a weight parameter multiplied by U_j (s) in $E_s(s)$, i.e., $E_s(s) = -\Sigma_j\ \omega_j U_j\ (s)$. The examples of U_j are terms that evaluate the ball position, the number of safe pass courses to teammates, and the possibilities of shots on the opponent goal if the agent gets the ball at state s. Using U_j (s), characteristic eligibilities e_ω are written:

$$e_{\omega_j}(t) = \frac{1}{T}\left(U_j\left(S_{a(t)}\right) - \sum_{x\in A(S_x)} U_j\left(S_x\right)\pi\left(x\middle|s(t),\{\omega_j\}\right)\right). \tag{6}$$

4.2 Rewards

Fifty-Storms 2017 uses the evaluation of live matches by a soccer audience as rewards in its online reinforcement learning. Fig. 3 shows its learning system. The online-coach agent receives rewards evaluated by subjects who are watching a soccer match and necessary information on state s and the $U_j(s)$ and $\pi(a)$ values from soccer agents to calculate the right side of Eq. (6). The $\Delta\omega$ values in Eq. (3), which are necessary for updating the weight parameters, are sent to the soccer agents by the online coach. It only takes about ten games (instead of thousands) for agents to adequately learn plays to meet a subject's requirements.

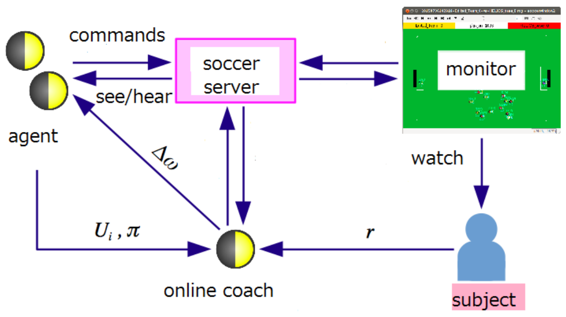

Fig. 3 Overview of learning system

The results of our learning experiments show that the number of goal-scoring patterns after passes increased drastically and the winning rate against agent2d improved from 52.1% to 73.2%. These results support the effectiveness of our proposed state evaluation function and the learning algorithm for receiver agents to determine their positions.

5 Summary

In this team description paper, we outlined the characteristics of our Fifty-Storms team that participates in RoboCup 2D Soccer Simulation League 2017. This team is based on an open version of the HELIOS team, agent2d (ver.3.1.1), to which modifications were added to enhance the offensive abilities of the forwards and the defenders by hand coding. In addition, we exploited the results of online reinforcement learning, where a policy gradient method derived appropriate policies for receivers as well as other players with the ball.

References

1. UvA Trilearn 2003, http://staff.science.uva.nl/~jellekok/robocup/2003/
2. HELIOS's URL site, http://rctools.sourceforge.jp/pukiwiki/ (in Japanese)
3. Igarashi, H., Fukuoka, H., and Ishihara, S.: Learning of Soccer Player Agents Using a Policy Gradient Method: Pass Selection, In: Proceedings of International MultiConference of Engineers and Computer Scientists (IMECS) 2010, Vol. I, pp. 31-35 (2010).
4. Igarashi, H., Nakamura, K., and Ishihara, S.: Learning of Soccer Player Agents Using a Policy Gradient Method: Coordination between Kicker and Receiver during Free Kicks. In: 2008 International Joint Conference on Neural Networks (IJCNN 2008), Paper No. NN0040, pp. 46-52 (2008).
5. Williams, R. J.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning, vol. 8, pp. 229-256 (1992).
6. Igarashi, H., Ishihara, S., and Kimura, M.: Reinforcement Learning in Non-Markov Decision Processes-Statistical Properties of Characteristic Eligibility. IEICE Transactions on Information and Systems, vol. J90-D, no. 9, pp. 2271-2280 (2007, in Japanese). This paper was translated into English and included in The Research Reports of Shibaura Institute of Technology, Natural Sciences and Engineering, vol. 52, no. 2, pp. 1-7 (2008). ISSN 0386-3115.
7. Ishihara, S. and Igarashi, H.: Applying the Policy Gradient Method to Behavior Learning in Multiagent Systems: The Pursuit Problem. Systems and Computers in Japan, vol. 37, no. 10, pp. 101-109 (2006).
8. Peshkin, L., Kim, K. E., Meuleau, N., and Kaelbling, L. P.: Learning to Cooperate via Policy Search. In Proc. of 16th Conference on Uncertainty in Artificial Intelligence (UAI2000), pp. 489-496 (2000).