FUT-K Team Description Paper 2017

Tomohiro Iwanaga, Kosuke Onda, and Teruya Yamanishi

Department of Management Information Science, Fukui University of Technology
Gakuen, Fukui 910–8505, Japan

Abstract. This paper describes our contents of the agent programs and any position of the agent. The problem of how to assign the agents to the position is very important to dominate the play. We improved the Hungarian method to including a priority on the position. The new method improved 13% for time necessary to the assignment compared with conventional Hungarian method.

Key words: Assignment problem, Priority group, Hungarian Algorithm, Nearest neighbor algorithm

1 Introduction

FUT-K that is mainly composed of undergraduate students of Fukui University of Technology in Japan has been organized since fall 2007. At the beginning of inauguration, we have participated in two leagues, namely one is RoboCup Soccer 3D Simulation, and another RoboCup Soccer Mixed Reality. Since the mixed reality league was withdrawn, we are concentrating operations on 3D simulation league at present.

The purposes of our team are to grow knowledge and experience of the computer language and the information science through applying themselves to RoboCup Soccer. Though almost members of our team are unskilled at programming yet, we believe that now our team is developing with getting advice from other teams.

We made six appearances in the world competition from RoboCup 2009 in Graz to RoboCup 2016 in Leipzig, and could get to a lot of things about soccer strategies and techniques of the movements for humanoid robot as the 3D soccer agent from these competitions.

In this paper, we introduce our activities for developing the 3D soccer agent of this year as follows:

- Contents of the overall program code on the agent,
- Algorithm on assigning agents to the position.

The details are explained in the following sections.

2 Program Contents

The program of our agents consists of three classes, namely normal class, singleton class, and static class[1]. The normal class has no constraints on the generation of instance or on the way of access from that class. It includes Agent class, Dribble class, Effector class, Walk class, and so on. The singleton class confines only one instance as the generation of instance, but does not restrict the access from other classes. There are AgentState class, Drawing class, FieldState class, Parse class, and so on. Finally, the static class collects only static functions like Debug class, KalmanFilter class, and Tool class. Also, we use three well-known libraries, namely Eigen, TinyXML-2, and libcmaes.

3 Algorithm on assigning agents to the position

In the problem of how to assign an agent to a position, it is necessary to avoid the collision between agents with movement and to minimize the amount of moving distance of agents. So, we investigate the assignment problem.

Let us consider to the n coordinates of targets for n current coordinates of agents. This problem is so-called "complete bipartite graph", where the number of each vertices is equal in Fig. 1. For soccer agents, it is taken as n =10 except a keeper. We attempt to simulate the assignment problem by using three algorithms:

1. Nearest neighbor algorithm,
2. Hungarian algorithm,
3. Hungarian algorithm with priority group.

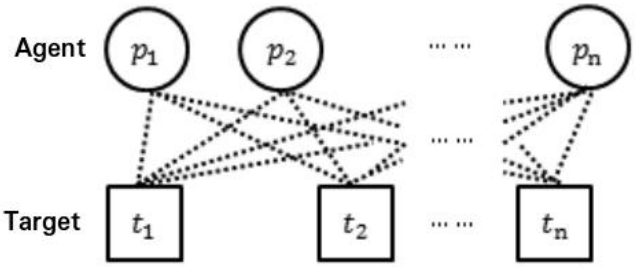

Fig. 1 Complete bipartite graph with n =10 if case of soccer agents is considered.

3.1 Nearest neighbor algorithm

In this algorithm, the distance between the coordinate of some target and current coordinate of the agent is calculated in turn, and the agent with the shortest distance is arranged in some target. The calculation cost is a $O(n^2)$, and the assignment has a dependence of initial calculation order on the coordinate of some target and current coordinate of the agent. Also, it has possibilities of a collision between agents, and of a distance becoming long.

3.2 Hungarian algorithm

This algorithm finds out the minimum combination of cost sums in equal weighted complete bipartite graphs with equal vertices on both sides[2]. We put a square of the distance between the coordinate of some target and current coordinate of the agent as an evaluation value rather than just a distance in order to make the dependence of the evaluation value strong. The calculation cost is a $O(n \cdot n!)$, and it spends much time rather than the Nearest neighbor algorithm.

3.3 Hungarian algorithm with a priority

The calculation cost of the Hungarian algorithm for the full search is so large even n =10. For the soccer game, it is not always necessary to estimate the assignment on the full search. It is a few positions around the ball or around offence agents of foe. So we improve the Hungarian

algorithm to one with a priority target. Each target is divided the position in the same priority group, and the assignment is calculated from the Hungarian method for each of the groups. Let us consider targets n with the coordinate T, and the targets are divided into M groups:

$$\begin{aligned} G_t &= \{T_1, T_2, \cdots, T_{M-1}, T_M\}, \\ T_i &= \{t_{s_i+1}, t_{s_i+2}, \cdots, t_{s_i+m_i-1}, t_{s_i+m_i}\}, \\ m &= \{m_1, m_2, \cdots, m_{M-1}, m_M\}, \\ s_i &= \sum_{j=1}^{i-1} m_j, \ n = \sum_{i=1}^{M} m_i, \end{aligned} \tag{1}$$

where G_t and m represent an entire set and a number of coordinates in each group, respectively. For example, we can write

$$\begin{aligned} G_t &= \{T_1, T_2, T_3\}, \\ T_1 &= \{t_1, t_2\}, \\ T_2 &= \{t_3, t_4\}, \\ T_3 &= \{t_5\}, \end{aligned} \tag{2}$$

if n = 5, M = 3, and m = {2, 2, 1}. The pseudocode can be written as follows:

```
# Target indices is stored in assignedAgentPos array.

Initialise isAssigned array to false
for (i ← 1 to M)
        μ ← Calculate average coordinate of Ti
        for (j ← 1 to n)
                agentIndex[j] = j
                if (isAssigned[j] = false)
                        targetDist[j] ← Calculate distance between μ and pj
                 else
                        targetDist[j] ← Maximum value of real type
                end if
        end for
        Sort agentIndex array by ascending order based on targetDist array

        for (j ← 1 to mi)
                k ← agentIndex[j]
                Add pk to Pi
        end for
        Apply hungarian method to pair of Pi and Ti
         for (j ← 1 to mi)
                k ← agentIndex[j]
                assignedAgentPos[k] ← Get target index of k agent is applied
                                      the Hungarian method
                isAssigned[k] ← true
        end for
end for
```

The calculation cost is a $O(M \cdot (n + \{\max_i(m_i)\}^3))$. The calculation cost of the Hungarian algorithm with the priority becomes to be same as the Nearest neighbor algorithm if $M = n$. On the other hand, it does equal to the Hungarian algorithm if M = 1. Therefore, the Hungarian algorithm with the priority is regarded as an algorithm in the middle of the Nearest

neighbor algorithm and the Hungarian algorithm from the aspect of the calculation cost.

3.4 Simulation results

We simulate three algorithms as mentioned above under 10 target positions and 10 agents. Time until the reaching all agents to all targets is measured in arranged targets and agents at random. The result with 100 times as the number of trials is shown in Table 1. From Table 1, we find

$$\text{Hungarian} < \text{Hungarian + priority} < \text{Nearest neighbor},$$

for the assignment of 10 positions, and

$$\text{Hungarian + priority} < \text{Nearest neighbor} < \text{Hungarian},$$

for the assignment of priority 4 positions.

Table 1 Simulation results on three algorithms for the assignment. The average over 100 times is presented.

Algorithm	Average	
	All(10 positions)	Priority(4 positions)
Nearest neighbor	18.25 s	9.14 s
Hungarian	11.75 s	9.72 s
Hungarian + priority	13.30 s	8.43 s

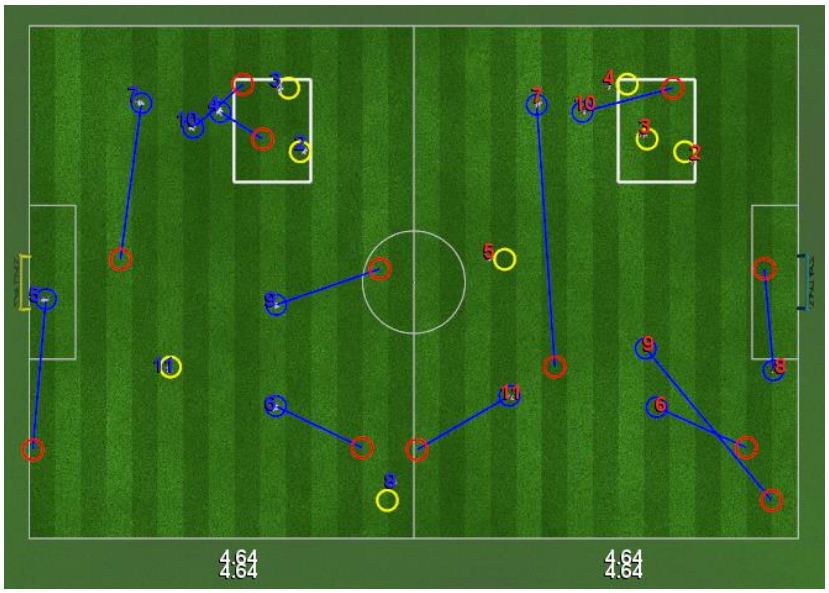

Fig. 2 Image of our simulation on a comparison among three algorithms for the assignment. Time until the reaching all agents to all targets is measured in arranged targets and agents at random by using the Nearest neighbor method and the Hungarian method on the half field on left and right sides, respectively. The small rectangle of white colored line is an area of priority 4 positions.

Also, we show the image of our simulation on three algorithms for the assignment in Fig. 2. This figure implies the simulation of the Nearest neighbor method and the Hungarian method. The blue and red colored circles are the current agent positions and the target positions, respectively. The blue lines represent the direction of movement for the agent.

4 Conclusions and Future Works

The algorithm of the Hungarian with the priority implied an advantage compared with other methods on the assignment. As the situation sometimes changes to soccer game every moment, the agents' arrangement using this way is expected to be effective.

However, we still have some problems such as reduction of the charging between agents, and the movement of the goalkeeper saving the long kick ball. At last year, CMA-ES has been used to generate the long kick motion of the agent[3]. We will attempt to generate their motions with CMA-ES or other optimization algorithms.

Acknowledgements

This work is supported in part by research grants from Fukui University of Technology.

References

1. Ishitaka, M., Kitajima, Y., Onda, K., Sugihara, K., and Yamanishi, T.: FUT-K Team Description Paper 2015, RoboCup2015, Hefei (2015).
2. Kuhn, H. W.: The Hungarian Method for the assignment problem, Naval Research Logistics Quarterly, 2, pp.83‑97 (1955); Variants of the Hungarian method for assignment problems, Naval Research Logistics Quarterly, 3, pp.253‑258 (1956).
3. Onda, K. and Yamanishi, T.: FUT-K Team Description Paper 2016, RoboCup2016, Leipzig (2016).