Strategies for Multi-Agent Coordination in a Grid World

Eder Mateus Nunes Gonçalves and Guilherme Bittencourt

UFSC - Federal University of Santa Catarina
DAS - Automation and Systems Department
88040-900 Florianópolis - SC - Brazil
{ eder | gb }@das.ufsc.br

Abstract. In this work, we describe strategies for multi-agent coordination, where adequate coordination means a system performance increase. In the main strategy, when a agent can't act by any reason, it choose the agent more capable in the environment to execute these action. The results show the strategy efficacy, especially when the environment increase the necessity for a reaction.

1 Introduction

The advantages obtained with a multi-agent approach can be easily lost if an adequate coordination process between agents can not be established. To explore the real possibilities of a multi-agent strategy, the agents in the society must be able to cooperate in a coordinated way. In the Artificial Intelligence literature, several research problems where proposed where these coordination strategies can be implemented and tested, e.g., robot soccer, rescue are surveillance activities, etc. One common artificial environment consists of a world in a grid format, that agents should explore to find resources, normally associated with "food". The idea of all these problems is to measure how the performance of the agents, in this case the quantity of food that is collected, increase with coordination, i.e., what is the impact of team work.

In this context, we have developed a software[1] that simulates a grid world, with twenty lines and twenty columns, where food can appear at a randomly chosen case of the world at regular time intervals. Four agents should coordinate their actions in the order to collect the maximimal quantity of food in a given period of time. Any agent that finds a food unit, should depose it into a given storage case. To evaluate the coordinate strategies we can modify the environment conditions to determine the better moment to use the strategy. The software was implemented in C++, using object orientation. The agents are implemented by a knowledge-based system, whose rules are explicitly codified into the software main program. As a base case, the performance without any cooperation strategy is considered. In this case, the agents search for food alone,

[1] The software can be downloaded from:
http://www.das.ufsc.br/~eder/clima.src.tar.gz.

without taking into account the other agents. As a first cooperation strategy we propose some actions to be taken when an agent finds food in the way to the depot.

2 The Environment: A Grid World

The environment consists of a grid, a matrix with twenty lines and twenty columns. The intersection between a line and a column is called a *case*. At each moment, each agent is located in exactly one case. One case is chosen to be the depot case, where all collected food should be stored. The simulation occurs in cycles, i.e., the temporal unit is a cycle. In a cycle, an agent can perform one, and only one, action, that can be either a movement to an adjacent case, or the emission of a message. Initially, the agents are located in the grid corners and the case depot is located in the center of the grid, in position (10,10).

According to the environment rules, one food unit appears automatically at every n cycles, where n is a simulation parameter. The experiments showed that the strategy efficacity varies with the parameter value. Others constraints were considered to create a problem more adequate to the multi-agent approach. All simulations were performed using a time interval of twenty seconds. In this way, we treat with a response pattern. The agent can carry only one food at a time. Once an agent collected a food, it can only leave it in the depot. There is no direct communication, all communication traffic is carried out by a communication manager. When an agent needs to send a message, this message is transmitted to a mailbox, and the communication manager delivers it to the message receiver. An agent can receive several messages, but it can participate in only one cooperation process at each time, i.e., during one cooperation process, the agent can not engage into a new cooperation process.

The software was implemented in the C++ programming language [1], using an object-oriented approach. A total of seven classes were implemented. The classes are: (i) **Position**: indicates a position in the grid; (ii) **Food**: contains the list of positions in the grid where the food is located; (iii) **Message**: contains constructors for messages with different performatives; (iv)**Mailbox**: used by the environment to manage the agent's messages; (v) **Clock**: implements the environment clock; (vi) **Interface**: verifies and validates the agent's actions in the environment, besides providing the input information for the agents; (vii)**Agent**: provides the agent internal variables and the actions in the environment, including movements and messages exchanges.

The main program, clima.cc, implements the four agents and their interactions with the environment are implemented, besides providing the interface with the exterior. This interface is in command-line form. In every cycle, the system updates the environment state, i.e, the simulation time, the cases with food, the agents positions, the number of foods collected, the number of food units already stored in the depot and the number of food units collected individually by each agent and the positions where they were collected.

For simplicity, the program was implemented in a totally sequential way. A more realistic approach would consider five different processes, one for for each agent and one for the environment, however the sequential approach response was satisfactory with the constraints described in this section.

3 The Cooperative Strategy

Each agent encapsulates a knowledge-based system. Nevertheless, the multi-agent system is a homogeneous one, where all the agents have the same knowledge about the domain. In each cycle, the agent is allowed to perform only one action, that can be either a movement to a new case, or a message emission to another agent, through the environment mailbox.

In a first strategy, there is no cooperation between the agents. Each agent search for food independently. The knowledges bases are build using the methodology described in [2], where a High-Level Petri net is used to describe the agent knowledge. This Petri net can be seen in the figure 1. In this first strategy, the transition $t5$ does not exist.[2]

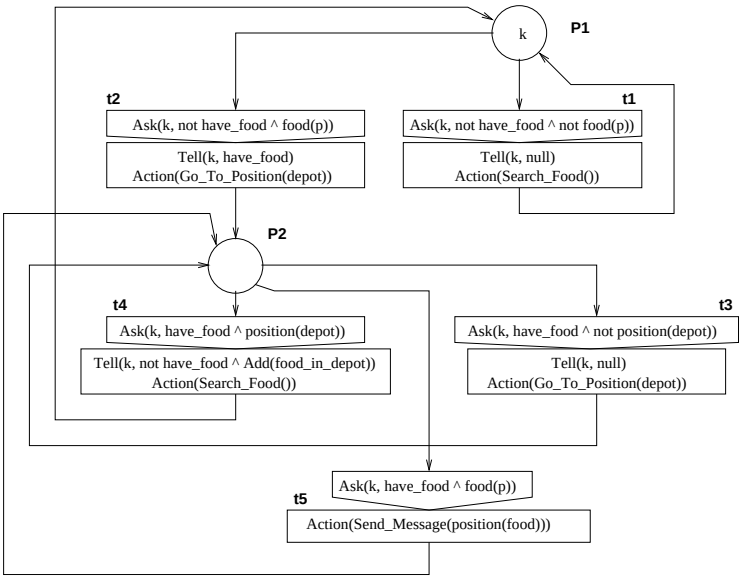

Fig. 1. The knowledge base of each agent represented by a Petri net.

The token represents the knowledge base (K). When this token enables a transition, an Ask is made to the K, that must return an answer. This answer

[2] The knowledge base represented in the Petri net is a simplified one. However, it is enough to understand the agent dynamic.

is represented by a directive $Tell$, where the inference engine returns the results to K. Besides that, an action is also inferred. This Petri net, in a next stage, is translated into a rule set that constitutes the knowledge base of each agent.

According to the knowledge base presented in figure 1, when an agent finds a food unit in its way to the depot case, it follows its way, and the found food unit is not collected. In a second strategy, when this state happens, a cooperation process is started, represented by the inclusion of the transition $t5$.

Now, when an agent finds a food unit in its way to the depot, it sends a message to the other agents with the food position. The agents answer this message telling the distance between them and the food position. The closest agent is considered the winner and starts a cooperation. When the food is stored in the depot, the cooperation is finished.

Following the FIPA-ACL [3], a message is constituted by a performative field, a sender field, a receiver field and a content field. The content field contains the agent position or the distance between the agent and the food. The sender and receiver fields contain, respectvilly, the identification of the agent that has sent the message and the identification of the agent that receives it. When the receiver field contain "all", all the agents receive the message.

The performative field describes the type of communication act intended with the message. A *request* is used when a cooperation is requested. A request should be answered with a *propose* or with a *refuse*. In the first case, the sender agent makes a proposal telling its distance from the food unit. In the second case, the agent is not ready to cooperate, because it is either carrying a food unit or involved in another cooperation process. The requesting agent receives the proposals and chooses the best one. In a last step, it sends a message with an *accept* performative to the winner and one with a *reject* performative to all the others. It is important to consider that only one message is delivered per cycle.

4 Results

The simulations were ran in a computer with Intel Celeron 2 Ghz processor, 256MB of RAM memory, running Mandrake Linux 10.0, using the gcc-2.96.

The performance of the system is measured by the number of cycles that the system needs to store a food. Considering a period of 80 cycles to appear a new food, the four agents, without any cooperation strategy needs 80.7 cycles to collect and store a food in the depot. If we consider the cooperation strategy the media is all the same: 80.6. The same results are obtained if we diminish the period down to 10 cycles. With a period of 40 cycles, the multi-agent system needs 41.1 cycles to collect and store one food unit, with or without the cooperation strategy.

With a period of 10 cycles between each new food unit, the strategy starts to make some difference. Without cooperation, the agents need 15.2 cycles to find and store a food unit. Using cooperation, this is reduced 14.8 cycles, still a minimum difference. When the period is set to 5 cycles, a greater difference

appears: 10.9 cycles without cooperation and 10.1 cycles with cooperation. In fact, the smaller the cycle period, the greater is the number of times a conflict occurs. We consider a conflict when an agent carrying a food to the depot finds another food in its way.

5 Conclusion

There are many ways to cooperate in this problem. However this work looks to focus in one approach and analyzes its impact. This strategy consists in starting a cooperation strategy when an agent finds a food in its way to the depot. In this case, the agent is still carrying a resource, and is not able to take another one. In this case it sends a message to the the other agents to discover who is the closest agent. Once this agent is determined, it should go to food and collect it.

In fact, this is a special case inside the environment dynamics. It almost never happens. However, if the period between to succesive food introductions is diminished, the probability that this situation happens is increased, and the cooperation turns into an alternative to improve the system performance. This is the main conclusion of the results presented. The smaller the period between food appearances, the greater is the effect of the cooperation strategy.

It is important to see that it is not the only way to cooperate in this problem. Others ways include designate a fix collector that must collect the foods found by the others agents. In other case, when an agent find a food, it can ask to the others what they are doing. If there is an agent that is storing the food, it is informed with the position of the new food, and then goes to it. In others words, the collector role, in this case, is dynamic.

The approach presented here can model situations in real problems, like the collecting robots and intelligent routing in networks.

References

1. Stroustrup, B.: The C++ Programming Language. Addison Wesley Longman (1999) ISBN 0-201-88954-4.
2. Gonçalves, E.M.N., Bittencourt, G.: A planning-based knowledge acquisition methodology. In: Congress of Logic Applied to Technology (LAPTEC), IOS Press (2005)
3. Foundation for Intelligent Physical Agents http://www.fipa.org: FIPA ACL Message Structure Specification. (2002)