World of Computer Science and Information Technology Journal (WCSIT)
ISSN: 2221-0741
Vol. 1, No. 8, 346-350, 2011

Solving the Puzzle Problem using a Multiagent Approach

Kamel Khoualdi
Management Information Systems Department
King Abdulaziz University
Jeddah, Saudi Arabia

Marwan El-Haj Mahmoud
Computer Science Department
Philadelphia University
Amman, Jordan

Abstract— Toys problems, such as the puzzle problem, are solved using classical artificial intelligence search algorithms Such as Breadth-first search and depth-first search. These strategies requires the generation of a graph known as the state space search that consist of the different states a problem may have. Using the above search techniques, a solution of the problem consists of a systematic exploration of the different state, starting from an initial state and moving towards a final state. This approach is time and memory consuming.
In this paper, we propose a multiagent approach as an alternative to solve the puzzle problem. Each block in the puzzle is considered as a reactive agent. The paper shows how the solution is reached through the interaction of the agents.

Keywords- multiagent systems; puzzle problem; reactive agents; search methods.

I. INTRODUCTION

Search algorithms constitute the more powerful approach for problem solving in artificial intelligence (AI) [1]. For example toy problems such: blocks world, puzzle, salesman, etc can be solved using these search strategies.

Search techniques are a general mechanism for problem solving that take a place in a space called states space search. Search algorithms examine all the alternatives systematically within this space until reaching the solution of the problem. The state space describes all the possible states of the problem. A possible action on this space marks the transition from one state to another. The state space can be viewed as a graph where each node represents a state and where a link between two nodes represents a transition from a state to its successor. The problem is defined by giving an initial state from which the search begins and final state where the problem ends. Search algorithms have to explore the graph trying to find a path that leads from the initial state to the goal. The solution of the problem is therefore a sequence of nodes leading from the start node to the goal.

Many search algorithms were proposed. They are classified into two main categories: uninformed and informed search strategies. Uninformed search algorithms, called also blind search, use only the information available on the problem definition. They have no additional information to guide their search. Breadth-first search and depth-first search are the well known algorithms for this kind of search. The main drawbacks of uninformed search algorithms are the time required by the search and the amount of memory required to store the states to be examined. For a big search problem with a huge number of nodes, the search process may cause a memory overflow. This is phenomenon is known as combinatorial explosion. Informed search problem such as best-first search try to reduce the time and the memory space by using heuristics that guide the search process. But still informed search are time and space consuming.

Multiagent approach can be used such problems. The main idea is that there is no search graph to consider and thus the combinatorial explosion problem is avoided. To clarify this approach, the puzzle problem is proposed.

The next section presents an overview of multiagent systems. Section 3 presents the puzzle problem, while section 4 exposes our approach to solve this problem.

II. REACTIVE AGENTS

Multiagent systems [2] are a community of autonomous agents working together in order to achieve a goal. We distinguish between two main streams in a multiagent environment. Agents can be cognitive or reactive [3][4]. A cognitive agent [5] is distinguished by an explicit representation of the environment, possessing skills, intentions, beliefs, etc. Cognitive agent behavior is a consequence of its observations and its beliefs. To achieve their goals, cognitive agents can use sophisticated models of cooperation negotiation, etc.

In contrast, reactive agents [6], object of our study, stipulate that agents have not to be intelligent, but, on the contrary, intelligence emerges from interaction of non intelligent agents. Thus, a reactive agent, is fine-grained simple agent with non explicit representation of its environment. Its behavior is very simple and, in general, of type stimuli-response. Minsky (1986) postulates that intelligence of a society of minds is the result of the interaction of a huge number of simple agents which considered individually have no intelligence.

Many applications were developed using the reactive approach. We can mention, for example, Brooks's robots [7], Steels' robots [8], Pengi problem [9], ant society [10], and eco-problem solving [11].

Eco-problem solving is a particular approach for solving distributed problem solving. It postulates that problem solving can be viewed as defining a population of simple autonomous agents where the behavior and the interaction between these different entities lead to reach a stable state that corresponds to problem solution. The solution of the problem emerges from the interaction of these agents.

In contrast of the classical approach of problem solving based on the exploration of a state space, the eco-problem solving technique doesn't use the state notion. It is mainly based on the agents' interaction. The solution of the problem emerges by side effects due to the interaction of the agents. This interaction resulting from the behavior of the agents enables the system to evolve to a stationary or stable state representing the solution of the problem. Memorizing agents' actions, while looking for a stable state, results in a plan of actions. This plan gives the possible actions that lead from the initial state to the goal.

In the following section, we will describe how a reactive agent approach can be used to solve the puzzle problem.

III. PUZZLE PROBLEM

A. *Problem Specification*

The problem we are dealing with is to choose a number N to specify the puzzle size according to the equation (Size = 2 * N + 1).

- White Block.
- Blue Block.
- Space Block. (S)

After specifying the size of the puzzle, we choose an initial state of the puzzle (randomly , or according to user choice).For example if we choose N = 2, then Size = 5. Suppose that the initial state is (Fig. 1.):

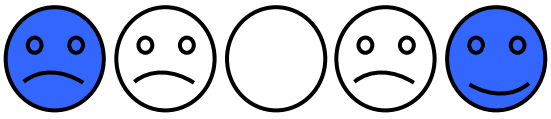


Figure 1. Initial state

The goal that we want to reach is to make the white blocks to the left and the blue blocks to the right. The location of the space block is not important. We notice that the last blue block is happy; that is because there is no white block after it which means that it is already at the right place.

The possible goal states are (Fig. 2):

1.
2.
3.
4.
5.

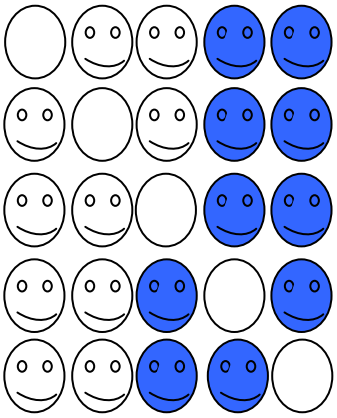


Figure 2. Possible goal states

The constraints applied for moving from the initial state to the final state are:

- You can exchange place for any block with the space block (S) only.
- A block to be exchanged should not be far than N places from the space block.

B. *Traditional AI Ttechniques as a Solution Approach*

As it is mentioned earlier, this problem can be solved using traditional AI techniques such a breadth-first search, depth-first search, etc. But, those techniques are space and time consuming due to size of the state space that represents all the possible states of the problem. The state space is modeled as a graph where each state corresponds to a node. Search algorithms are applied to this graph, and the number of nodes to be expanded becomes rapidly huge.

C. *Multiagent Systems as a Solution Approach*

The problem can be solved using a multiagent system. In this case, each agent is considered as a non "intelligent" agent that has its own knowledge and behavior.

The agent's knowledge is very simple such as its color, its location, its state, etc. The behavior of the agent is of stimulus-reaction type, i.e. it reacts according to the changes it perceives from its environment. We have three types of agents: blue agent, white agent, and space agent. Fig. 3 shows the agent's internal structure.

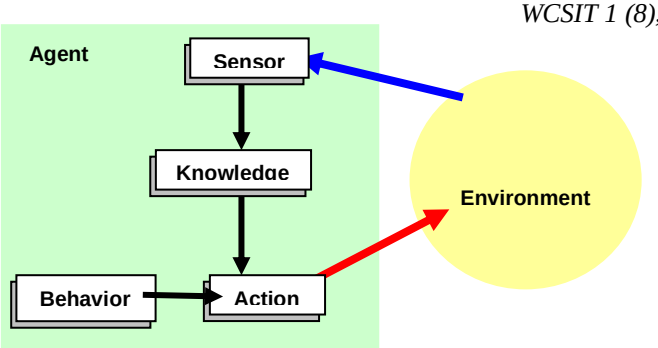


Figure 3. Internal structure of the Agent

The internal structure of each agent is described below:

1) Blue and white agents structure

a) The blue (white) agent's knowledge consists of:

- Color: blue (white).
- Location: its place in the puzzle.
- Satisfy: true or false (indicator of agent's satisfaction).
- Move: gives the agent's move number in the process, for example is it the first one to make a move.
- Move: gives the agent's move number in the process, for example is it the first one to make a move.
- Face: object (agent's interface)

b) The blue (white) agent's behavior consists of:

- Is_Satisfied: returns true if it is satisfied about its place, false elsewhere.
- Can_Move: returns true if it can move according to the exchange constraints, false elsewhere.
- Agree_OnMove: returns true if the move doesn't change its state from satisfied to unsatisfied, false elsewhere.
- Refuse_Move: returns true if it is the agent who made the last move, false elsewhere.

2) Space agent structure

a) The space agent's knowledge consists of:

- Location: its place in the puzzle.
- Position: left, right, or middle according to the puzzle.
- Face: object (agent's interface).

b) The space agent's behavior consists of:

- Can_You_Move: asks an agent if it can move or not.
- Agree_To_Exchange: asks an agent for a permission to exchange places.
- Get_Position: specifies its position (left, right, or middle).
- Apply_Right_rule: if agent's position is left, right rule is applied.
- Apply_Left_rule: if agent's position is right, left rule is applied.
- Apply_Middle_rule: if agent's position is middle, middle rule is applied.

As mentioned above blue and white agents belongs to the same class, they only differ by their color. However space agent may be seen as a coordinator since it coordinates the moves of blue and white agents through its left, right, and middle rules. These rules constitute the core of the movement process. The following algorithms clarify how each rule work.

c) Left rule algorithm

```
1: if not last agent preceding space agent then
   { if agent color is Blue then
     { if (Ask? (Can move and agree to move)) then
       { exchange (agent.location, space.location)
         update necessary fields
         exit ()  }
       else {
         go to next agent
         go to 1  }  }
     else {
       go to next agent
       go to 1  }
   else {
       ask agent if Blue and White satisfied then
                goal reached
       else if space in middle then goal reached
            else {
                 exchange (agent.location,
                 space.location)
                 update necessary fields }
       }
```

d) Right rule algorithm

Right rule is applied when the space agent's position is on the Left, or can be motivated by middle rule. In this case, a dialogue occurs between the space agent and the agent succeeding it to see if there is a possibility of to permute their places. The dialogue starts with agents succeeding space agent from extreme right to left according to the following procedure:

```
1: if not agent succeeding space agent then
   { if agent color is White then
     { if (Ask? (Can move and agree to move)) then
       { exchange (agent.location, space.location)
         update necessary fields
         exit ()  }
       else {
         go to previous agent
         go to 1  }  }
     else {
       go to previous agent
       go to 1  }
   else {
       ask agent if White and Blue satisfied then
                goal reached
       else if space in middle then goal reached
            else {
                 exchange (agent.location,
                          space.location)
                 update necessary fields }
       }
```

e) Middle rule algorithm

Middle rule is applied when space agent is in the middle. In this case, the right rule is examined to see if it applies to the agent that was the last one to make a move. If this is not the case, apply the right rule. The following procedure explains how middle rule works:

if *right rule doesn't apply to the agent who makes the last move* ***then***
apply right rule
else
apply left rule

IV. Implementation

The puzzle problem can be implemented using object oriented programming. A class may represent an agent. We have two main classes: Block class and Space class. Instances from the Block class represent blue and white agents. Instance from the Space class represents the space agent. Knowledge and behavior of each agent correspond respectively to the attributes and methods of a class.

One of the problems of this technique is that agents can move indefinitely. Therefore, this situation has to be avoided. To prevent an infinite cycle to occur, we must know its causes. A cycle may occur, if an agent iterates the same movement yielding thus to a replication of a state, which means that the same rule is applied repeatedly. The rules have to be checked to avoid cycle to occur.

In the left rule, agents with blue color always move from left to right. If the agent is not a blue one, the nearest white agent moves (its position is just before space agent). According to this rule, the blue agents will move to the right and white agent if not satisfied will move near to middle. So no blue agents will go back using this rule.

In the right rule, agents with white color always move from right to left. If the agent is not a blue one, the nearest blue agent moves (its position is just after space agent). According to this rule, the white agents will move to the left and blue agent if not satisfied will move near to middle. So no white agents will go back using this rule.

In the middle rule, a cycle may occur. It can be prevented, as mentioned earlier, by checking the last move (last move is checked only when space agent is in the middle) that guarantees avoiding an infinite cycle.

So, since right rule (respectively left rule), eventually, moves all white (respectively blue) blocks to the left (respectively to the right) and stops when the goal is reached, middle rule also guarantees that no cycle may occur.

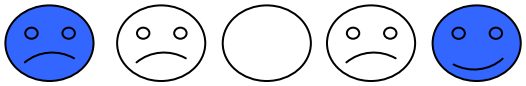
V. Examples of Implementation

The following examples show how the puzzle problem is implemented.

A. Example1

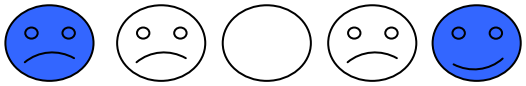
Consider the following example 5 puzzle blocks problem (Fig. 4):

0. Initial state



1. Applying middle and right rules

2. Applying left rule



3. Applying middle and left rules

4. Applying right rule

5. Applying right rule

6. Applying left rule

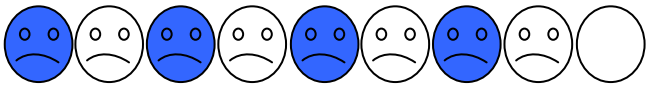
Goal

Figure 4. Example1

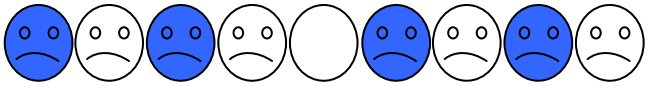
B. Example2

Consider the following example 9 puzzle blocks problem (Fig. 5):

0. Initial state

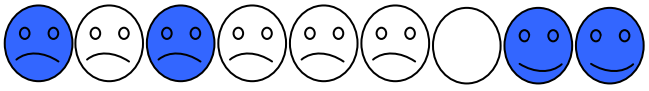


1. Applying left rule



2. Applying middle and right rules

3. Applying left rule



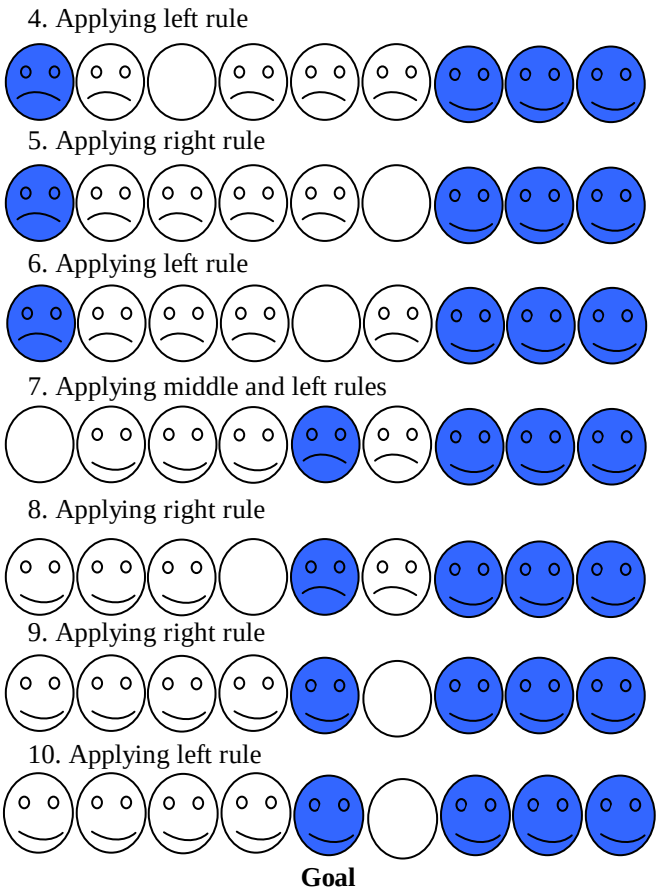


Figure 5. Example2

VI. CONCLUSION

Solving the puzzle problem using a multiagent approach shows how this approach is efficient comparing with the traditional techniques used in artificial intelligence. In fact, the generation of the state space requires not only a huge amount of memory space to store the different states to be examined, but also requires a long time to explore the state space. So, the traditional techniques are very expensive in time and memory storage. On the contrary, using multiagent approach eliminates the need to generate a state space. The solution emerges from the interaction of small non-intelligent agents. The number of agents' moves is considerably small comparing with traditional methods. Also, memorizing the different moves produces a plan of actions leading from different initial states to a goal state.

REFERENCES

[1] S. Russell, and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 3rd ed., 2009.

[2] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed., John Wiley & Sons, 2009.

[3] K. Fischer, I. J. Timm, and E. André, "Introduction to the special issue on the Fourth German Conference on Multiagent System Technologies (MATES)," Autonomous Agents and Multi-Agent Systems (AAMAS) vol. 18, no 2, pp. 219, 2009.

[4] N. Vlassis, "A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence", Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 1, no. 1 , pp. 1-71, 2007.

[5] J. L. Dessalles, J. Ferber, and D. Phan, "Emergence in agent based computational social science: conceptual, formal and diagrammatic analysis," Intelligent Complex Adaptive Systems, vol. 24, pp. 1-24, 2008.

[6] M. R. Abdessemed, K. Khoualdi, and A. Bilami, "Optimization of clustering time by a group of autonomous robots making use of an exclusive multi-marking," Journal of Computer Science, vol. 6, no 12, pp. 1465-1473, 2010.

[7] R. A. Brooks, L. Aryananda, A. Edsinger, P. Fitzpatrick, C. Kemp, U. M. O'Reilly, E. Torres-Jara, P. Varshavskaya, and J. Weber, "Sensing and manipulating built-for-human environments", International Journal of Humanoid Robotics, vol. 1, no 1, pp. 1–28, 2004.

[8] L. Steels, "When are robots intelligent autonomous agents?," Robotics and Autonomous Systems, vol. 15, issues 1-2, pp. 3-9, 1995.

[9] L. Leahu, P. Sengers, and M. Mateas "Interactionist AI and the promise of ubicomp, or, how to put your box in the world without putting the world in your box," Proceedings of the 10th international conference on Ubiquitous computing, pp. 134-143, 2008.

[10] R. O'Grady, A. L. Christensen, C. Pinciroli, M. Dorigo, "Robots autonomously self-assemble into dedicated morphologies to solve different tasks," 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), vol. 1, pp. 1517-1518, Toronto, Canada, May 10-14, 2010.

[11] A. Drogoul, D. Vanbergue, and T. Meurisse. "Multi-agent based simulation: Where are the agents?," Lecture Notes in Computer Science, pp. 1-15, 2003.

AUTHORS PROFILE

Kamel Khoualdi is currently assistant professor in management information systems department, King Abdulaziz University, Jeddah, Saudi Arabia. He received his Ph.D. degree in computer science from Pierre & Marie Curie University (Paris 6), France, in 1994. He received his M.Sc. degree in Computer Science from Paris 6 University, France, in 1990. He received his B.Sc. in computer science from Batna University, Algeria, in 1988.

His main research interests concern artificial intelligence, multiagent systems, knowledge-based systems, and e-learning.

Marwan El Haj Mahmoud is currently a lecturer in Philadelphia University, Amman, Jordan. He received his M.Sc. degree in computer science in 2003. He recived his B.Sc. in computer science in 1999.

His main research intersets concern artificial intelligence and multiagent systems.