# A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with A Common Goal

**Yashar Tavakoli** **H.Haj Seyyed Javadi** **Sepideh Adabi**

Computer Engineering Department, University of Islamic Azad –Science and Research, Tehran, Iran
Department of Mathematics and Computer Science, Shahed University, Tehran, Iran
Computer Engineering Department, University of Islamic Azad -East Tehran (Ghiam dasht), Tehran, Iran

**Summary**
The interest in MAS (Multi-Agent Systems) is increasing and an important task is providing these systems with sophisticated planning algorithms. One of the major and complex planning problems in MAS is indeed path planning problem; in this way, the case in which every agent has its own goal has been more discussed than the one in which there is a common goal among the agents, while we know in the recent problem, there are potentially more collisions and bottlenecks making it a hard problem to approach. Through this paper, we will address certain boundaries of this problem and introduce an algorithm to achieve a proper solution to this optimization problem.

***Key words:***
*Cellular Automata; Multi-Agent System; Path planning.*

## 1. Introduction

A lot of approaches in MAS path planning problem have been introduced so far: [1] deals with RTS (Real-Time Strategy) games in which agents must find non-colliding routes in a grid and several versions of A* have been introduced to solve the problem. In a high level of abstraction, [2] assumes that the “road map” is in the form of a graph and solves the problem by partitioning the graph into some certain sub-graphs and applying breadth-first search or traditional A* to sub-graphs. [3], instead of path planning alone, applies motion planning which consists of path planning and trajectory planning; a centralized planner benefits a collision map to achieve a collision free solution as trajectory planning while every agent is associated with a certain priority. [4] again discusses RTS games and path planning problem in large environments and introduces a certain strategy using a graph model and Dijkstra's algorithm. All of these papers consider separate destinations for different agents while [5] benefits a co-evolutionary genetic algorithm to solve the problem of only two agents (while we focus on the case in which there are several agents) , a common goal and several obstacles in Cartesian coordination. Since we benefit cellular automata, we address [6] , a paper on single-agent path planning which tries to find minimum distances using a cellular automaton and applying certain heuristics.

Now suppose that there are several geographically distributed agents with the same priorities. Agents must move for example to a station in a real environment or a fort in an RTS game; we are interested in minimizing the total time in which all the agents have reached the goal by applying a collision free ( agent-agent and agent-obstacle) strategy.

## 2. Modeling the problem

The traditional approach to multi-agent control, may be classified into two approaches, centralized and distributed (or centralized and decoupled, [7] and [8]). [3] compares these approaches. Here we benefit a central planner which is in charge of movements; hence we have applied centralized approach.

### 2.1 Building a framework

Consider an $m\times m$ grid of cells; each cell is associated with Cartesian coordination. A cell may be a home to three kinds of objects:

1.Agents (represented by numbers).
2.Obstacles (represented by black colors).
3.Goal (represented by letter “G”).

We will denote the number of agents by $n$ ($n < m^2$). If a certain cell is occupied by an agent, we call it an occupied cell. If a certain cell is not occupied and it’s not filled by an obstacle then it’s a free cell.

A grid with a number of objects forms an environment. The term configuration stands for the arrangement of the objects in a certain environment.

This framework is very appropriate for RTS games [1], but it may be also applied to real environments by discretizing an environment using a procedure like the one described in [6].

Fig. 1 illustrates an environment which exhibits a certain configuration.

Manuscript received July 5, 2008.
Manuscript revised July 20, 2008.

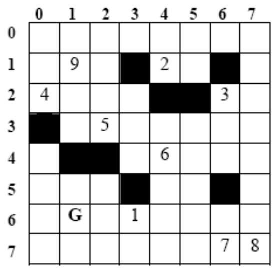


Fig. 1 An environment with a certain configuration

We distinguish vertical/horizontal movements and diagonal movements (to approach the Euclidean norm); for the sake of easy computation, each vertical or horizontal movement costs 10, while each diagonal movement costs 14 units of distance. As one can observe in Fig.2, an agent is able to move in 8 geographical directions in case they are free.

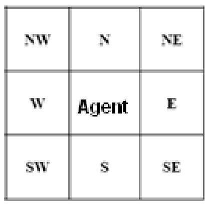


Fig. 2 Direction symbols

Let's consider the time factor in our framework. Instead of time we introduce time steps; if in each single time unit, a single distance unit can be taken by an agent, with regard to costs of distances (10 and 14) time steps are, 10,14,20,24,28,44,48,50,… . Time steps carry an additional concept for us, only in these time steps an agent may be finished with its currently decided movement.

We define a set named *unfinished-agents*, in the beginning all the agents are in *unfinished-agents*, if an agent reaches the goal we remove it from *unfinished-agents* set and then we will add it to *finished-agents* set. If an agent occupies the goal cell in current time step, the goal will be free of that agent in the next time step. A decision indicates a certain direction that must be taken by an agent; a decision can be made for an agent only if it's finished with its currently decided movement. Suppose a case in which an agent is still moving (i.e. it hasn't finish with its currently decided movement), then we rationally assume that both start and destination cells are occupied (since our model is discrete). We also define two other sets; set of *moving-agents* (in which every agent exactly fills two neighbor cells) and set of *arrived-agents* (in which every agent fills only a single cell).

These recent sets partition *unfinished-agents* set in each time step. Those agents who could not make their movements to decided cells in previous time step, exactly because the cells were occupied (agents have to wait at least one time step if the decided cells are occupied), in current time step, are again in *arrived-agents* set.

## 2.2 Formulating the problem

Now we can exactly define the problem and determine its particular boundaries. Since central planner's first objective is leading the agents toward the goal, there should be time step $T$, in which, |*finished-agents*|=$n$ ($T$ is exactly the certain time step in which the last agent reaches the goal).

The second and more challenging objective of central planner, is minimizing $T$. Consider agent $i$ in an environment, and time step $t_i$ in which, agent $i$ reaches the goal. Every path which an agent takes toward the goal is associated with a certain length (which is also measurable by time), let's name it $d_i$ for agent $i$. Furthermore, an agent on its way toward the goal, may be forced to wait for several time steps, for agent $i$ we denote it by $w_i$. Clearly, $t_i = d_i + w_i$ , and central planner tries to minimize $\sum_{i=1}^{n} t_i$ (a variant of minimizing $T$) .

To decrease $t_i$ we can not simply decrease both $d_i$ and $w_i$ , as if we try to decrease $w_i$ , $d_i$ will relatively increases and vice versa. The reason is quite obvious; in most of our favorite environments with a lot of agents and obstacles if all the agents take their shortest paths toward the goal, there will be a lot of collisions, therefore if we are interested in decreasing the number of collisions (i.e. decreasing $w_i$s), central planner should force the agents to take longer paths (i.e. increasing $d_i$) to avoid agent-agent collision, and vice versa.

This inverse relation between $d$ and $w$ may become stronger or weaker as configuration changes. Even $t_i$s are in relatively inverse relation with each other, for example if we try to decrease $t_i$ for agent $i$, there may be other agents who reach the goal at least a single time step later, as agent $i$ will occupy goal cell at least a single time step sooner, thus a sophisticated algorithm tries to balance $d_i$s , $w_i$s and $t_i$s in order to minimize $\sum_{i=1}^{n} t_i$ .

## 3. Cellular automata

As described in [11], Cellular Automata (CA) is decentralized, discrete space-time systems that can be used to model physical systems. Cellular Automata are formally defined as quadruples (d, q, N, $f$).The integer d is the *dimension* of the working space, obviously it is possible to create one, two or more dimensions automata. Q= {0, 1,…, q-1} is called the set of *states.* The neighborhood N= ($n_1$,…, $n_v$) is a v-tuple of distinct vectors of $Z^d$. The $n_i$'s are the relative positions of the neighbor cells with respect to the cell, the new state of which is being computed. The states of these neighbors are used to compute the new state of the center cell. The *local function* of the cellular automata $f$: $Q^v \leftrightarrow Q$ gives the local transition rule. A *configuration* is a function from $Z^d$ to Q. The set of all configurations is $C=Q^{Z^d}$. The *global function* A of the cellular automaton is defined via $f$ as follows:

$$\forall c \in C, \forall i \in Z^d, A(c)(i) = f(c(i+n_1),...,c(i+n_v))$$

Fig.3 illustrate a one-dimensional, binary-sate, nearest-neighbor (r =1) cellular automata with N=11.

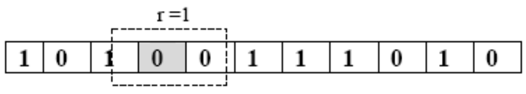


Fig. 3 Illustration of a one-dimensional, binary-sate, nearest-neighbor (r =1) cellular automaton with N=11.

## 4.Solving the problem

Suppose in Fig.4 we are interested to find the minimum distance between the agent and the goal.

The straight forward algorithm for this problem is traditional A*, but let's investigate a new strategy; we define a cellular automaton which dynamically calculates the minimum distances between each cell and the goal, the transition rules are:

$$q_{i,j}^{t+1} = \begin{cases} 1 \quad ; \; q_{i,j}^t = 1 \quad \left(\begin{array}{l}\text{1 stands for obstacle}\\ \text{in cellular automaton}\end{array}\right) \\ \min\left\{\begin{array}{l} q_{i,j}^t \,,\; q_{i-1,j-1}^t + 14 \,,\; q_{i-1,j} + 10, \\ q_{i-1,j+1}^t + 14 \,,\; q_{i,j-1}^t + 10, \\ q_{i,j+1} + 10 \,,\; q_{i+1,j-1}^t + 14, \\ q_{i+1,j}^t + 10 \,,\; q_{i+1,j+1} + 14 \end{array}\right\} \quad \underset{\substack{r=-1,0,1\\ s=-1,0,1}}{\forall q'_{i+r,j+s} \neq 1} \;; O.W. \end{cases}$$

Introduced cellular automaton terminates when no further updates occur. In Fig.5, the entire procedure for an instance has been illustrated.

Now we are provided with two global schemes which are both associated with Cartesian coordination, denote them by V-*scheme* which contains the minimum distances between each cell and the goal and D-*scheme* which is a map of the optimal direction(s) for each cell (following the optimal directions an agent can take its shortest diagonal path(s) toward the goal from an arbitrary location).

Using V-*scheme*, optimal direction(s) for a cell can be calculated by 8 comparisons, therefore having V-*scheme*, we can calculate D-*scheme* in $O(m^2)$ (although D-*scheme* can be

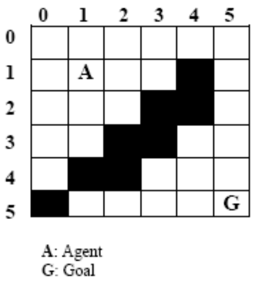


Fig. 4 A sample environment with a single agent and the goal.

dynamically computed along with *V-scheme*.

Since obstacles are fixed, it's proper to consider the cellular automaton as preprocessing [3], our implementation on a Pentium 4 machine, takes about *1 sec* to compute V-*scheme* for a 1024×1024 grid.

Obviously in case there's only one agent, A* is a faster algorithm to find the shortest path but as the number of the agents grows, A* will loose it's efficiency gradually [9] , while in our strategy, having D-*scheme*, every agent located in every arbitrary location can find its shortest path in, in $\theta(\ell)$ which $\ell = O(m^2)$ , therefore, one can find the shortest paths for all the agents in $O(nm^2)$.

Yet the main benefit of such approach in path finding for multi-agent systems with a common goal hasn't been discussed; provided schemes will give us an idea about the future movements of the agents as they tend to take their shortest paths.
Let's compare two different algorithms; in both of the algorithms, in each time step, central planner with respect to a kind of order for agents in *arrived-agents* set decides a proper direction to take.

The first algorithm only tries to reduce $d_i$*s* greedily, hence, central planner always chooses the optimal directions for each agent, therefore A* can be applied for each agent, but because of growing number of agents we prefer to use D-*scheme*, thus our first algorithm computationally takes $O(nm^2)$ .

The other algorithm tries to behave more intelligently; in order to estimate how much it takes toward the goal ($t_i$ ) if central planner force agent $i$ to step into a cell as the decided direction (not necessarily the optimum direction) we can add the time which is needed to get to decided cell from the current cell to the time which is needed to reach the goal from decided cell and then to an

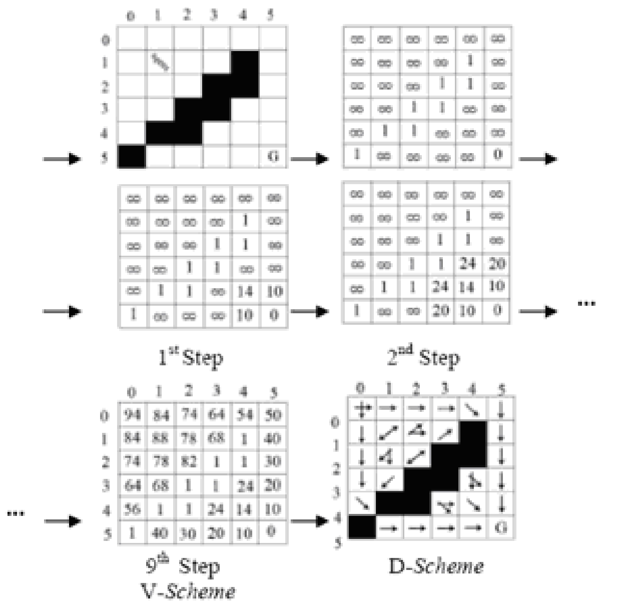


Fig. 5 Cellular automaton which dynamically calculates the minimum diagonal distances between each cell and the goal.

estimated time in which agent *i* should wait if it takes the decided direction. Using D-*scheme*, we can easily calculate the number of collisions between an agent if it takes the shortest path beginning by a certain direction, let's call it the probable path, and the other agents if they all take their shortest paths, one by one. This fact serves as our heuristic to solve the problem more intelligently.

Assume that the central planner is going to decide for an agent which direction to take, according to our heuristic, we define function *cost*:

$$Cost(i,j,direction)=v(direction)+V(i,j,direction)+h\times collisions(i,j,direction)$$

*cost* takes the current agent location, and a possible direction (i.e. a direction which is not obstacle), as arguments and returns a positive real number. Central planner always chooses the possible directions with lower costs for the agents in *arrived-agents* set in each time step. Suppose we have direction ***NW*** and location (r,s); *v* determines the required time units for the agent to get to ***NW*** cell, thus v( ***NW*** ) = 14, *V* returns the neighbor cell's value in V-*scheme*, with respect to the agent current location and a specified direction so *V* ( r , s , ***NW*** ) = V-*scheme* [ r - 1 , s - 1 ] which is the minimum distance from the decided cell toward the goal and finally *collision* returns a relatively good estimate of the number of collisions if the direction ***NW*** is taken by the agent located in (r,s) using the discussed procedure.

Let's discuss the parameter *h*. *cost* gives central planner an estimate of the total time in which an agent reaches the goal if it takes a specified direction; in order to build this function what is required is not actually the estimated number of collisions, it is the estimated waiting time, in this way, estimated number of collisions will only help us to calculate the waiting time if we have an idea about the duration of a single stop to avoid collision (i.e. how much an agent must wait for an occupied cell to become free). *h* is a free parameter which represents this value.

There is a tight relation between a certain configuration and the value of *h*. As the number of the agents and specially their density grows, *h* will also grow. In an environment with limited number of paths toward the goal, there will be usually longer queues and it's wise to pick greater values of *h* too. For $h = 0$ the algorithm's behavior will be equivalent to A*'s behavior which gives us a proper flexibility. The reader may also easily verify that $v + V$ forms $d_i$s while $h\times collisions$ stands for $w_i$s. In fact in each time step, for all the agents in *arrived-agents* set, central planner tries to minimize $t_i$s , by balancing $d_i$s and $w_i$s through making certain decisions.

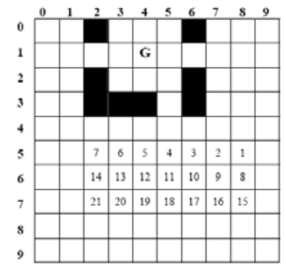
Now, we are ready to calculate an upper bound for time complexity of the current algorithm; in the worst case, in every time step for each agent, eight probable path and $n - 1$ shortest paths must be calculated, which totally takes $O(nm^2)$ Let's bound the possible number of time steps by M, hence the algorithm will take, $O(n(M(nm^2))$ or simply $O(Mn^2m^2)$. It should be stated that the value of M depends on the value of *h*, since by changing the value of *h* in a particular problem *T* may change, but to have an idea about how big M is, our simulations show that for all values of *h* (even very great ones), $M << nm^2$ (i.e. it never reaches the *T* in which there are no simultaneously movements and every agent with respect to a kind of order sequentially waits until the other agents reach the goal, and then moves). Thus, $M = O(nm^2)$ and a bound for the algorithm would be $O(n^3m^4)$.

## 5. Experimental results

We have applied introduced algorithms to a wisely populated environment. The agents and obstacles have been arranged in such a way that increase the density and restrict the number of possible paths toward the goal. This configuration will increase waiting times implying for some $h > 0$ we expect to see a better result than $h = 0$ (A*). To avoid complexity we have considered *h* as an integer.

Fig.6, shows the original problem and results for the first algorithm and the second algorithm while applying different values of *h*. Fig.7-(a), illustrates the configuration of the environment for the selected time step 132 if we pick *h = 2* (the most optimal value), while Fig.7-(b) illustrates the configuration in the same time step if we apply A*(in both figures as stated before, an agent who has occupied two neighbor cells, hasn't arrived to its destination cell yet). The D-*scheme* for the problem which is provided in Fig.7-(c), help the reader to easily verify the potential of bottleneck and long queues in case we greedily only look at $d_i$s.

A deeper look reveals that the second algorithm gain a lower $\sum t$ because it wisely distributes the agents and gives them the chance to get closer to the goal gradually instead of wasting time in long queues.



First Algorithm (A*) :
T = 512
Second Algorithm :

| h | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 50 | 100 | 500 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T | 286 | 270 | 280 | 284 | 288 | 276 | 296 | 282 | 282 | 288 | 290 | 310 | 310 | 310 | 324 |

Fig. 6 The results of the first and second algorithm while applying different values of *h*.

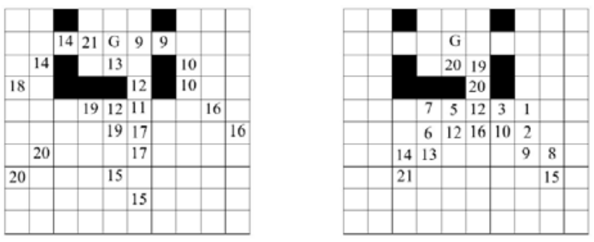
## 6. Conclusion

In this paper an algorithm was introduced to deal with the path planning problem in MAS while several agents try to reach a common goal. We also computationally bounded the algorithm. The proposed approach wisely distributes the agents to avoid long queues; this fact reveals the suitability of applying the algorithm in those MAS environments with a common goal which give rise to long queues like those with just a few possible paths toward the goal. The simulations also indicate that the proposed algorithm could achieve a far better total time than the traditional A*.

## References

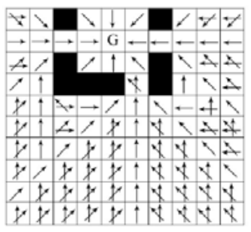
[1] D.Silver, *"Cooperative path-planning"*, AI Programming Wisdom, 2006.

[2] M.Ryan," *Graph Decomposition for Efficient Multi- robot Path Planning"*, In Proceedings of 7th International Joint Conference on Artificial Intelligence, 2003.



8 agents have already reached the goal 4 agents have already reached the goal

**(a)** **(b)**



**(c)**

Fig. 7 the original problem and results for the first algorithm and the second algorith

[3] S.H.Ji, J.S.Choi, B.H.Lee, "*A Computational Interactive Approach to Multi-agent Motion Planning*", International Journal of Control, Automation, and Systems vol.5, no. 3, pp. 295-306, June 2007.

[4] O.Arikan, S.Chenney, D.A.Forsyth," *Efficient multi-agent path planning*", In Proceedings of the Eurographic Workshop on Computer Animation and Simulation (Manchester, UK, September 02 - 03, 2001). W. Hansmann, W. Purgathofer, and F. Sillion, Eds.Springer-Verlag New York, New York, NY, 151-162.

[5] Z.Cai, Z.Peng, "*Cooperative Coevolutionary Adaptive Genetic Algorithm in Path Planning of Cooperative Multi-Mobile Robot Systems*", Journal of Intelligent and Robotic Systems 33: 61-71,2002.

[6] C.Behring, M.Bracho, M.Castro, J.A.Moreno," *An Algorithm for Robot Path Planning with Cellular Automata*",In Proceedings of the Fourth international Conference on Cellular Automata For Research and industry:theoretical and Practical Issues on Cellular Automata (October 04 - 06, 2000). S. Bandini and T. Worsch, Eds. Springer-Verlag, London, 11-19.

[7] J.C.Latombr, "*Robot Motion Planning*", Kluwer Academic Publishers, 1991.

[8] K. Fujimura, "*Motion Planning in Dynamic Enviornmen"t*, Springer-Verlag, New York, 1991.

[9] R.Leigh, S.J.Louis, C.Miles, "*Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms*", In Proceedings of the 2007 IEEE Symposium on ComputationalIntelligence and Games (CIG 2007).

[10] S.Russel, P.Norvig, "*Artificial Intelligence A Modern Approach*", SPrentice Hall Series in Artificial Intelligence, 2003.

[11] B. Durand, E. Formenti, A. Grange and Z. Róka. "Number conserving cellular automata: new results on decidability and dynamics". Discrete Mathematics and Theoretical Computer Science, AB: 129-140, 2003.