10th International Research/Expert Conference
"Trends in the Development of Machinery and Associated Technology"
TMT 2006, Barcelona – Lloret de Mar, Spain, 11-15 September, 2006

ARTIFICIAL INTELIGANCE AND THE PROBLEM OF PATH FINDING IN ROBOT SIMULATIONS

Senad A. Burak
University of Sarajevo, Faculty of Mechanical Engineering
Sarajevo, Bosnia and Herzegovina

ABSTRACT
A basic problem in robotics, especially in advanced graphical simulation systems, consists of solving a problem of finding an optimal path for moving robotic objects from a known starting position to a goal point. This problem is generally complex and it is usually associated with artificial intelligence and advanced numerical computations. Standard algorithms such are A, Dijkstra's algorithm, genetic algorithms or breadth-first search are among the most popular for such problems. In this paper we present a graphical simulation system in 2D that effectively solves a path finding problem based on a well known A* algorithm. A* algorithm is used because of its popularity, flexibility and the fact that can be applied in a wide range of contexts. The programming is done in the C# language under the latest .NET 2.0 framework. The resulting software is tested on number of different configurations and optimal solutions are found quickly and effectively. It is shown however, that if the heuristic is not admissible, the applied algorithm cannot guarantee the minimal distance.*
Keywords: robotics, computer simulation, virtual prototyping, artificial intelligence

1. INTRODUCTION
An important part of any advanced robot simulation system is its ability to solve a problem of finding an optimal path for moving robotic objects from a given starting to a goal position in a known environment. This problem is generally difficult and must include some kind of heuristic to achieve a reasonable solution. Due to advances in the fields of artificial intelligence and graph theory, we have today several effective algorithms available for such purposes. The aim of this paper is to demonstrate how those techniques can be employed in a typical path finding problem in virtual robotics. A graphical 2D system with full user interface that simulates robotic movements is developed and optimal path for achieving the target position is automated. In order to represent the robot's environment we used a rectangular grid of cells, but this can be easily changed to any irregularly shaped areas. The system is flexible and easy to use for the users and allow them to quickly produce various simulations and what-if scenarios.
The most used applications where such solutions are applied are optimal route finding in city road systems, robot navigations and simulations, military applications and, of course, the modern computer games.

2. THE A* SEARCH ALGORITHM
In order to solve usally difficult and computationaly time consuming problem of optimal path finding, advanced artificial inteligence and conventional graph theory are generally used. Several algorithms that exist today in this field are well estatblished and documented. Among them, the most known and widelly used are the *breadth-first search*, *Dijkstra's* and the *A** search algorithms. All of them are based on the searching techniques through the entire configuration maps or possible routes and the optimal path is found based on heuristic function that can estimates the cost to reach the goal position according to an appropriate search criteria.

A* algorithm is probably the most used of all available algorithms, because of its flexibility to cover different configurations and type of problems, particularly if they can be represented in a state space configuration. Unlike the breadth-first search strategy, the A* algorithm uses a one-time computation of possible paths, simply by taking into account the cells that surround the current and the goal cells. On the other hand, unlike the Dijkstra's approach, A* employ the combined heuristic and cost functions. In order to do this, we represent the initial configuration as a start state and the goal configuration as a goal state. Then, for every possible movement of the robot we generate the successor states by using the heuristic function that gives us the best state that we are going to process next. Providing that the heuristic is not overestimated, we will quickly end up with a solution to the problem and possibly with the best or optimal path. The definition of the admisable heuristic function depends on the actual problem. In this paper we use a rectangular map devided into a number of discrete cells in which the robot has to move through the cells in all possible directions (diagonal included), until it reach the goal position. The A* algorithm that we are going to employ should take into account both, the distance from the initial state to the current position (the cost), and the estimated distance from the current cell to the desired state (the goal estimate). Clearly, unlike traditional algorithms, the heuristic approaches cannot guarantee the solution, so the importance of properly chosen heuristic function is mandatory.

The first operation we have to do is to set the nodes that represent the start and goal states of the problem. Next we create two lists, one called the open list and the other closed list, and initially we add the start state into the open list. The open list should hold all cells that are on the possible routes to the goal state, while the closed list is used for those cells that we have already expanded so far. Then, we loop through the open list as long as we have any element into it and calculate the total cost for every successor cells. The total cost is determined by

$$f(n) = g(n) + h(n),$$

where $g(n)$ is the cost of moving the robot from the starting position to the current state and $h(n)$ is the estimated cost to get from the current position to the goal state. In order to determine the function $h(n)$, a heuristic approach has to be employed, simply because there is no way that we can know the actual distance until the optimal path is found. The cell with the smallest total cost is then chosen as an optimal successor of the current cell n. An extended mechanism is added and called for every iteration to explore all possible routes in case the current route leads the robot to a dead end. In such cases, we exclude those routes from the open list.

At the end of the loop, the goal position is reached and the optimal path can be determined by examining the route backward to the starting position.

This process is quite straithforward and safe for typical configuration. However, the calculation of the estimated costs for large grids can be a time consuming process and it is not suitable for situations where the state configuration is changing during the process. In this paper we are not taking into an account such cases.

2.1. A* Search algorithm pseudocode

- Set the number of rows and columns of the grid
- Set the environment, the wall cells etc
- Set the start and goal nodes
- Initialise the Open and the Closed lists
- Populate the Open list with the start node
- Loop through the Open list while it is not empty
- Find the node off the Open list with the lowest *f* function
- Check if the current node in the loop is the goal node. If true, exit the loop and display the path backward as the optimal solution
- For each node that is not blocked calculate the estimated total cost to the goal position
- Add the successor nodes that are not blocked in the Open list
- Remove from the Open list the node with the smallest cost function
- Add current node to the Closed list
- Repeat the loop

We note that if $h(n)$ is zero at some configuration, then the A* becomes *Dijkstra's* algorithm and a shortest path will be guaranteed found. On the other hand, if $h(n) > g(n)$ then this algorithm cannot guarantee to generate the shortest (optimal) path, so a careful examination should be taken in such cases.

2. 2. D simulations

In Figure 1 we show the main screen of the solver that we develop using the C# programing language. The grid details (the number of rows and columns) can be easily adjusted to any desired combination and the walls (represented by dark squeres) can be iteractivelly positioned by using the mouse right button. The starting and ending points are also configurable by the end user using the mouse left button and by selecting an appropriate radio button from the Initial configuration box. Once we have a satisfactory configuration, we start the process of finding the optimal path, simply by selecting an menu item available on the perent shell window. The A* algorithm is then employed and the optimal path from the starting to the desired ending points is graphycally displayed.

The A* algorithm is computationaly time consuming process. However, the main problem with this approach can arrise in the case where the grid dynamically changes its configuration. The time required to loop through all steps can be unacceptable in such cases and an alternative method should be used, such is the DynamicSWSF-FP or Lifelong Planning A* algorithms.

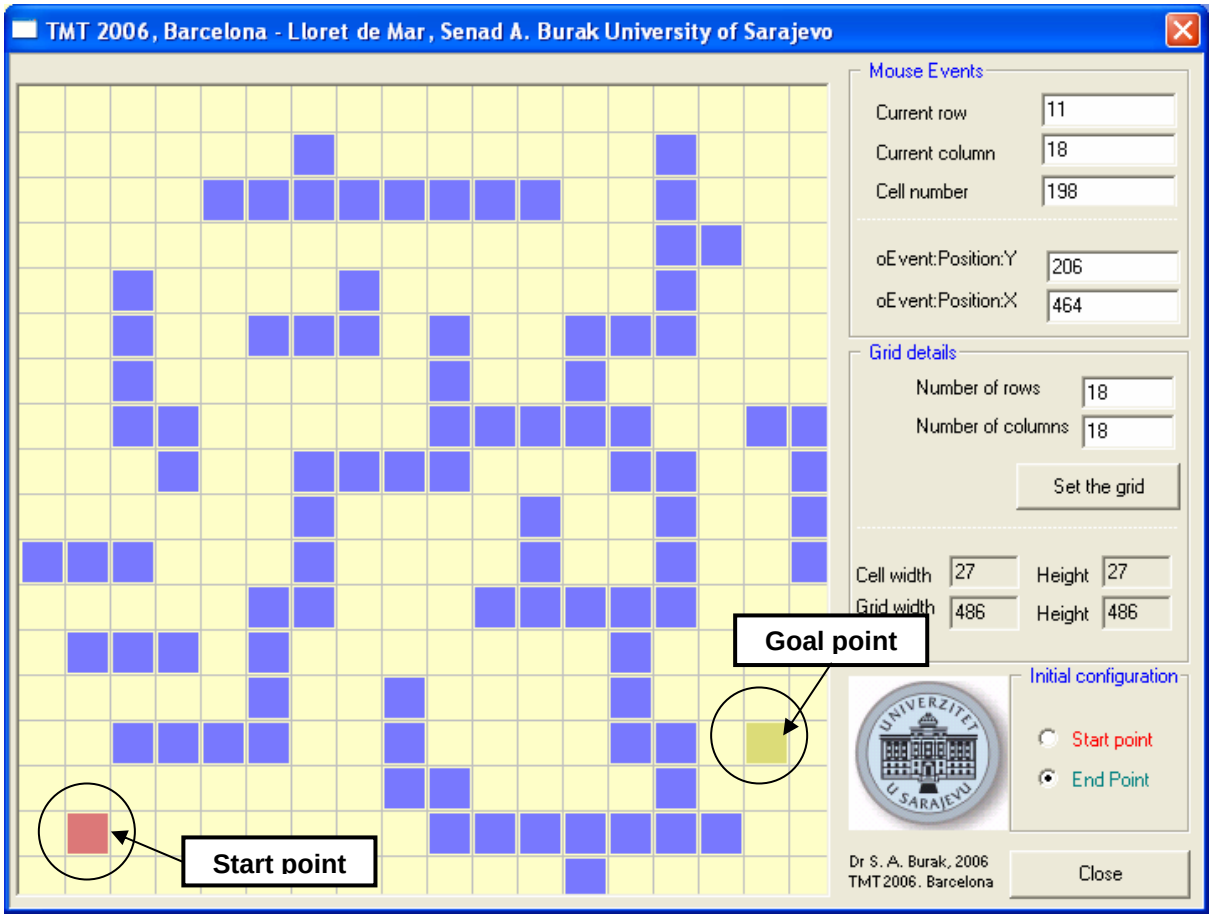

Figure 1. Path finding problem for a given grid configuration in 2D

3. REFERENCES

[1] Cao Z. L., Huang Y., and Hall E. L.: Region Filling Operations with Random Obstacle Avoidance for Mobile Robots, Journal of Robotic Systems, vol. 5, no. 2, 1988.

[2] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.

[3] Axel Pinz. Consistent Visual Information Processing Applied to Object Recognition, Landmark Definition, and Real-Time Tracking. *VMV'01*, Stuttgart, Germany, 2001.
[4] Edsger W. Dijkstra, A note on two problems in connection with graphs. *Numerische Mathematik*. 1:269—271, 1959.
[5] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs, NJ, 1995.