Path Planning with Cellular Automata

Abrie Greeff
B.Sc Hons (Computer Science)
Department of Computer Science
University of Stellenbosch

March 24, 2006

1 Introduction

A cellular automata (CA) is a n-dimensional lattice, where each cell of this lattice has a discrete state. Furthermore a CA is a dynamic system that has a set of rules that describe every state. These set of states are very simple but together they create a continuous dynamic system.

This report examines robot path planning as an application of a CA system. The implemented algorithm is based upon an algorithm proposed by C. Behring [1]. The purpose of this algorithm is to find a collision free optimised route for a robot given that the terrain from his initial position to his goal position is known. In addition to the path planning system another separate system was developed to model a simple dependant CA. For the rest of this report we only consider two-dimensional CA when referring to a CA.

2 Formal Description

2.1 Cell Space

We define the cell-space of a two-dimensional cellular automata as $L = \{(i,j) \mid i,j \in \mathbf{N}, 0 \leq i \leq n, 0 \leq j \leq m\}$, where i,j are the corresponding column and row numbers of the lattice and m,n is the size of the lattice.

2.2 Moore Neighbourhood

We define the Moore neighbourhood of a cell i,j as $N_{i,j} = \{(k,l) \in L \mid |k-i| \leq 1 \wedge |l-j| \leq 1\}$, where L is the cell space. The Moore neighbourhood corresponds to the nine surrounding cells of the cell at i,j.

3 Algorithms

3.1 Dependant CA

This CA has a list of rules that is defined for example in a text file. These rules define what other cells a certain cell is dependant on to compute its state. In this dependant CA every cell has two possible states namely on and off. The function used to calculate the state of the cell is the bitwise exclusive-or (XOR) of the cells it's dependant on. The system runs continuously and the state of each cell is calculated with every iteration. Certain cells can also be set on before the first iteration starts. A typical text file containing the rules may contain the following.

```
7,0 DEPENDS 0,1 12,12
6,0 DEPENDS 1,8 1,12
7,2 DEPENDS 6,0
5,1 DEPENDS 0,2 1,0
INITIAL 0,0
```

3.2 Robot Path Planning

The robot path planning CA is created in two phases. The first phase creates the initial configuration space for the CA. The second phase floods the CA and hopefully a route can be determined for the robot.

3.2.1 Phase One

In the first phase all the information for the system needs to be gathered. The system needs to know where the robot are, where it wants to go, the size of the CA and where the walls are. To obtain this information it is loaded for example from a text file. This file may contain the following information.

```
SIZE 9,14
WALL 5,1 6,1 7,1 1,2 3,2 4,2 5,2 0,5 3,5
INITIAL 3,0
GOAL 4,13
```

Once this information has been loaded the CA is built. Every cell can have five possible states: open, closed, initial, goal and Manhattan distance to the goal. The Manhattan distance is defined as $g((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 = y_2|$. This is equal to the length of all paths connecting (x_1, y_1) and (x_2, y_2).

The CA is first evolved using the Moore neighbourhood with the following rule.

$$s_{i,j}(t+1) = \begin{cases} 1, & s_{i,j}(t) = 0 \wedge \exists (k,l) \in N_{i,j}(t) | s_{k,l}(t) = 1 \\ s_{i,j}(t), & otherwise \end{cases}$$

After this rule has been applied on the CA the first phase is complete.

3.2.2 Phase Two

The initial configuration has now been done in the previous phase. This phase now floods the rest of CA with a different rule. This rule also uses the Moore neighbourhood in its calculation.

$$s_{i,j}(t+1) = \begin{cases} s_{k,l}(t)+1, & s_{i,j}(t) = 0 \wedge \exists (k,l) \in N_{i,j}(t) | s_{k,l}(t) \geq 3 \\ s_{i,j}(t), & otherwise \end{cases}$$

This flooding starts from the goal cell. The flooding continues until either the initial cell is reached or all cells have non-zero entries. If the initial cell was reached the robot can now traverse the system. The robot decides which route to follow by watching his neighbours and their respective Manhattan distances and choosing the direction with the smallest distance.

4 Implementation

These algorithms were implemented in a Java enviroment. For both algorithms the CA was represented with a two-dimensional array containing objects at each cell. These objects contained basic functions to access their data. Both used a set bits to hold the relevant information. The object for the path planning algorithm used an extra integer to keep track of the Manhattan distance for each cell. To make it easier to visualize the running of both systems the Java graphics library was used to draw the CA on the screen.

5 Conclusion

The dependant CA was implemented first and therefore took a bit longer than the path planning CA. The path planning CA was adapted from the dependant CA and no conclusion can be made which system was more difficult to code. The path planning CA did however give a much better visual representation when it was running. This made it possible to easier identify possible problems in the implementation of the algorithm.

6 References

1. C. Behring, M. Bracho, M.Castro and J.A. Moreno. *An Algorithm for Robot Path Planning with Cellular Automata*

2. Eric W. Weisstein et al. *Taxicab Metric* From *MathWorld*–A Wolfram Web Resource. http://mathworld.wolfram.com/TaxicabMetric.html