Research and Development on Searching a Routing Path of a Dynamic Terrain

Jui-Fa Chen+, Wei-Chuan Lin*, Chia-Che Yang+, Chih-Yu Jian+
+Department of Information Engineering, TamKang University
+E-mail: alpha@mail.tku.edu.tw
* Department of Information Technology, Tak-Ming College
*E-mail: wayne@mail.takming.edu.tw

ABSTRACT

The research topic of this paper is that how an avatar explores the environment, and find the way out to reach the pre-assigned goal. This proposed system is applied to a dynamicallly changable, continuous and large-scaled environment. To deal with the learning in such environment, the proposed system partitions the state-space into regions of states, called cells. By using cells, the states of the system can be reduced in which the number of states grows dramatically increasing in proportion to the number and quality of inputs. Through a series of maniputations of cells, the avatar can adapt itself to the changable environment. By adjusting the peferences of the proposed global-cell, the tendancy of exploration behavior of the avatar can be controlled to explore the potential path caused by the changing environment.

KEYWORDS

virtual world, dynamic environment, reinforcement learning, machine learning, curse of dimentionality.

I. INTRODUCTION

The goal of this paper is to implement an avatar by reinforcement learning(RL) [1] to find out a path to reach the assigned goal in a virtual world. The environment of this paper is dynamically changeable, large scaled and could be multi-dimensional.
There are some problems on the proposed environment. The first is how an avatar adapts itself to the dynamically changeable environment,is considered. The second is how an avatar finds out a potentially better path which the avatar is not known by the changing happened in the changing environment. In a large scaled and multi-dimensional environment, there exists an problem called “the curse of dimensionality” [2]: The number of states grows exponentially in proportion to quality, scale and number of input devices.
In this paper, section 2 is the related works about path finding and reinforcement learning. Section 3 defines the terminologies that is used in this paper. Section 4 implements a series of algorithms of cells establishment and manipulations using min-max hyperbox. Section 5 is a brief system overview. Section 6 is the implementation. Finally, the conclusion and future works are proposed.

II. RELATED WORKS

Moore et. al., [3] proposed a RL algorithm called parti-game algorithm, which used a game-theorem based algorithm for partitioning the state-space into cells, and finding paths in these cells. The parti-game algorithm partitions the state-space into cells such that the avatar doesn't have to learn all of the states. In some critical area this algorithm increases the resolution of partition to do a precise learning on such area, and this makes the path finding in a high quality. Although the parti-game algorithm learns well in large scaled multi-dimensional environment, it didn't offer a mechanism to recover the increased resolution of cells; this means parti-game algorithm may not be flexible to handle a dynamic environment.
One challenge that frequently arises in RL is the trade-off between exploration and exploitation [4]. To maximize value (expected total reward), a RL avatar should choose the preferred actions that the avatar had tried in the past and found out the effective one in producing reward. To discover such actions, the avatar needs to take actions that have not been selected before. The avatar has to *exploit* what it already knows to obtain reward. However, the avatar also has to *explore* to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively if a successful strategy is to be developed. This dilemma becomes even more challenging in a dynamic environment, where the current exploited solution may no longer be valid, and the solutions previously explored may have changed

III. DEFINITIONS OF TERMINOLOGY

Definition 1*: A* ***cell*** *is a region covers a group of states in the state-space which are essentially similar. Two states are essentially similar if they have similar reward on the same action.*

The relationship between a state and a cell is defined as equation (1):

$$\forall_a \in A, r_s(a) \cong V_c(a) \quad (1)$$

where A denotes the set of all available actions, $r_s(a)$ denotes the received reward of doing action a at state s, and $V_c(a)$ denotes the expected value that the avatar learned of doing action a in cell c.
This equation is used as a precondition of identifing which cells cover the current state s.

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

Definition 2: *The* ***transition function*** $T_c(a)$ *returns a set of cells which are the destination cells of all possible transitions triggered by doing action a in cell c.*

Definition 3: *The* ***transition probability*** $P_c(a,d)$ *is the probability of the transition triggered by doing action a in cell c; and then transfering to cell d.* $\sum_{i=1}^{j} P_c(a, d_i) = 1$, *where* $T_c(a) = \{d_1, d_2, ..., d_j\}$ *and* $d_i \in T_c(a)$.

Definition 4: *The* ***global cell*** *g, is the cell which is not really existed but standed for the whole unknown or uncertain space.*

IV. CELLS ESTABLISHMENT AND MANIPULATION IN MIN-MAX HYPERBOX

A series of algorithms are proposed by using a simple geometrical graphic, the "min-max hyperbox" [5] [6] to implement the necessary learning operations on cells.

The advantage of using the min-max hyperbox is that it uses only two points: min and max-point to represent the shape in any multi-dimensional space. In this way, the algorithms and programs written in min-max hyperbox can be portable and flexible.

In algorithm 1, cells which satisfied the equation (1) and state *s* is included in the shape, are collected in a set *Q* and then returns *Q* when all cells are checked.

Algorithm 2 collects cells that are not in the set of cells returned by function **IDENTIFY**, and the sizes of these cells would not be larger than *MaxSize* if these cells expanded to include state *s*. Finally, the collected cells are put in set *Q*, and returns *Q* when all cells are checked.

Figure 1 shows how the function **EXPANDABLE** works. In the case of (a), the cell size is still smaller than *MaxSize* if expanded to include state *s*, thus this cell should be collected. In the case of (b) if the cell is expanded, the size would be larger than *MaxSize*; therefore, this cell should not be put in the set *Q*.

Q: a set of cells, *C*: the set of all cells,
s: the current state, *c*: a cell,
c_max: the max point of cell *c*, and
c_min: the min point of cell *c*.

1. $Q \leftarrow \varnothing$
2. **for all** $c \in C$ **do**
3. **if** $\forall i \in$ all dimensions $S_i \le c\_max_i$ and $\forall i \in$ all dimensions $S_i \ge c\_min_i$ and satisfied equation (1) **then**
4. $Q \leftarrow Q \cup \{c\}$
5. **end if**
6. **end for**
7. return *Q*

Algorithm 1: **IDENTIFY**(*s*)

Q: a set of cells, *C*: the set of all cells,
s: the current state, *c*: a cell,
c_max: the max point of cell *c*,
c_min: the min point of cell *c*, and
n: the number of inputs, *MaxSize*: the maximun size of a cell.

1. $Q \leftarrow \varnothing$
2. **for all** $c \in (C$ - **IDENTIFY**$(s))$ **do**
3. **if** MaxSize $\ge \sum_{i=1}^{n}$ **(max(c_max$_i$, s$_i$) – min(c_min$_i$, s$_i$))** and satistied equation (1) **then**
4. $Q \leftarrow Q \cup \{c\}$
5. **end if**
6. **end for**
7. return *Q*

Algorithm 2: **EXPANDABLE**(*s*)

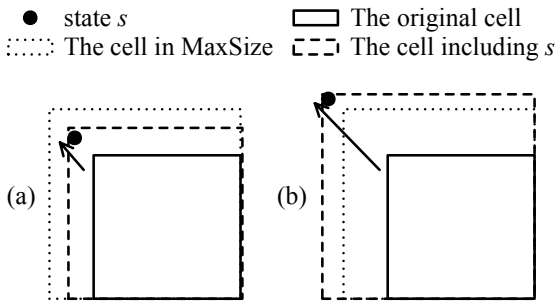

Figure 1. Expandable of Min-Max Hyperbox

The function **EXPANSION** of algorithm 3 is used to expand a cell *c* to include a state *s*.

---

s: the current state, *c*: a cell,
c_max: the max point of cell *c*,
c_min: the min point of cell *c*, and

1. **if** $c \in$ **EXPANDABLE**(s) **then**
2. **for** *i* in all dimentions **do**
3. $c\_max'_i \leftarrow$ **max**(c_max_i, s_i)
4. $c\_min'_i \leftarrow$ **min**(c_min_i, s_i)
5. **end for**
6. **end if**

Algorithm 3: **EXPANSION**(*c*, *s*)

Figure 2 shows the flow of **EXPANSION** function. In case (a), both *x* and *y* value of state *s* are larger than the max-point of cell; To expand this cell, it needs to assign both *x* and *y* value of state *s* to the max-point and let this cell includes state *s*. In case (b), only *y* is larger than the value of max-point, therefore, assign *y* as the max-point of this cell. In case (c), *x* value is smaller than min-point, therefore, assign *x* as the min-point of this cell.

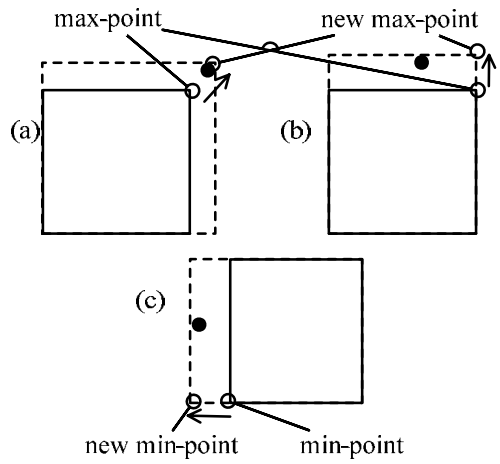

Figure 2. Expansion of Min-Max Hyperbox.

While avatar found a cell contains state *s*, but not essentially similar (e.g. not satisfied equation (1)). The **CONTRACTION** function as shown in Figure 3 is used to make such cell exclude state *s*.

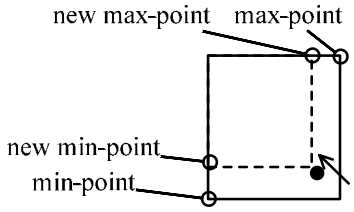

Figure 3. Contraction of Min-Max Hyperbox.

s: the current state, *c*: a cell,
c_max: the max point of cell *c*,
c_min: the min point of cell *c*, and

1. **if** $c \in$ **IDENTIFY**(*s*) **then**
2. **for** *i* in all dimentions **do**
3. **if** ($c\_max_i - s_i$) ≤ ($s_i - c\_min_i$) **then**
4. $c\_max'_i \leftarrow s_i$
5. **else**
6. $c\_min'_i \leftarrow s_i$
7. **endif**
8. **end for**
9. **end if**

Algoright 4: **CONTRACTION**(*c*, *s*)

While an avatar is at current state *s*, and state *s* is in current cell *c*. Doing action *a* may cause a transition and then receive the next state *s*' and a reinforcement-signal $r_s(a)$. The handling procedure is that at first, the avatar uses equation 2 to update the value function of cell *c*,

$$V_c(a) \leftarrow \gamma \cdot V_c(a) + (1-\gamma) \cdot V_s(a) \qquad (2)$$

where γ is a constant used to control the learning rate.

After the value function is updated, the avatar uses **IDENTIFY**(*s*') function to identify in which cell the avatar is. According to the results of **IDENTIFY**(*s*') and **EXPANDABLE**(*s*'), there are several cases of learning procedure.

1) The current cell $c \in$ **IDENTIFY**(*s*') The avatar is still in current cell *c*, there is no transition happened.
2) The current cell $c \notin$ **IDENTIFY**(*s*'), and **IDENTIFY**(*s*') ≠ ∅ The system should find the next cell *c*' from **IDENIFY**(*s*') that is most similar to state *s*'. After finding the cell *c*', the system should update the transitions of cell *c* by equation (3) and the corresponding transition probability is also updated.

$$T_c(a) \leftarrow T_c(a) \cup \{c'\} \qquad (3)$$

3) **IDENTIFY**(*s*') = ∅, but **EXPANDABLE**(*s*') ≠ ∅. The system should find the next cell *c*' from **EXPANDABLE***s*' that is most similar to state *s*'. After finding the cell *c*', the system should update the transitions of cell *c* by equation (3) and the corresponding transition probability is also updated.
4) **IDENTIFY**(*s*') = ∅, and **EXPANDABLE**(*s*') ∅, too. In this case, create a new cell *c*' centering around state *s*'. After finding the cell *c*', the system should update the transitions of cell *c* by equation (3) and the corresponding transition probability is also updated.

V. SYSTEM OVERVIEW

Figure 4 is an system overview of how it can work. The explanation is shown as followed:

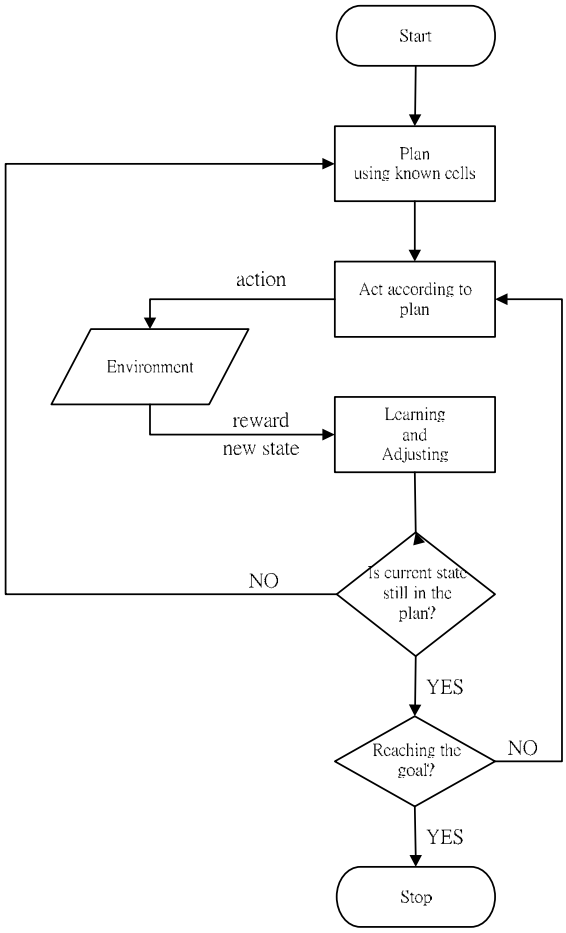

Figure 4. The system flowchart.

1) Find a plan to be the behavior policy of the avatar. The plan maybe an exploitation plan that can reach the goal from current cell *c*, or an exploration plan throw the global cell, that has the best total weight.
2) Choose an action *a* from the plan and excute the action *a*.
3) After excution of action *a*, the system receives a reinforcement-signal **r**(*s*, *a*) and next state *s*' from the environment. The system should find out, or create the destination cell *c*' which including *s*'. After finding out the cell, the system should also adjust the position and range of cells, and update the value function, transition function and transition probability if necessary.
4) Set the new current cell to *c*'. If current cell includes the goal, the job is finished. If current cell is not in the plan anymore, the system should go back to step 1 and find another plan. If current cell is still in the plan, the system should go back to step 2, and continue acting according to the plan.

VI. THE IMPLEMENTATIONS

The implementation environment is shown as figure 5.

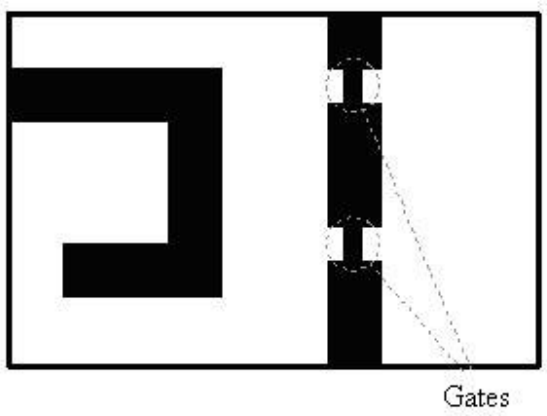

Figure 5. A maze with 2 passages and 2 corresponded gates.

As shown in figure 5, there are two gates that can be opened or closed to represent a dynamic environment. The upper side gate is marked as gate *A*, and the other one is marked as gate *B*.

First, let gate *A* be opened, gate *B* be closed, the start point is surrounded by obstacle in the left hand side as the red spot in figure 6 and the goal point is in the right hand side to the start point behind the passages. Figure 6 shows the cells contribute to the avatar for reaching the goal at the first time. The gray rectangles in figure 6 are min-max hyperbox cells that the avatar learned. The dotted lines are the track of avatar's moving from the begining.

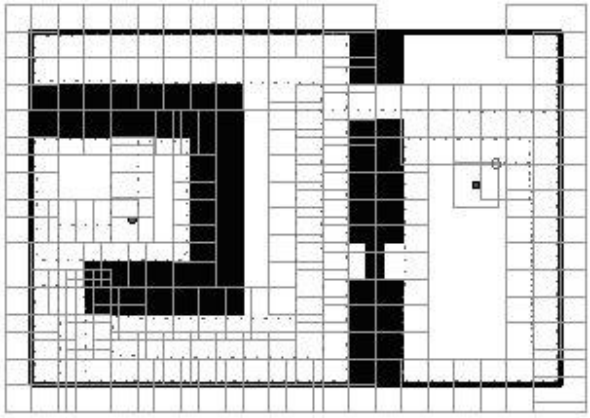

Figure 6. Cessl and track when avatar reach the goal cell in the first time.

Figure 7 shows the plan that the avatar choosed after several epochs of learning (reached the goal for several times). The environment is almost fully explored.

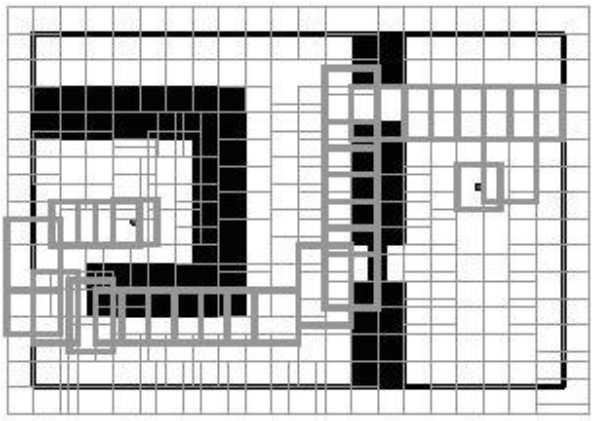

Figure 7. The plan from start to goal after several epochs of learning.

After the previous experiment, now let gate *A* be closed and gate *B* be opened to cause a changing of environment. The result is shown as figure 8. At the begining, the avatar didn't know about the changing, therefore, it still chooses, and acts according to the previous plan that passing through gate *A* until encountered the closed gate *A*. The avatar began to do exploration again and finally found a new path through the opened gate *B*. Figure 9 shows the new plan from the start point again, after the opened gate *B* is explored.

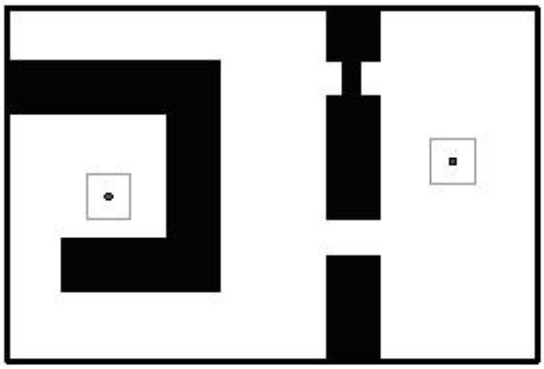

Figure 8. Switch the gates as a changing of environment.

Finally, the avatar created 165 cells for learning this 300 * 400, nearly 12,000 pixels(states) environment.

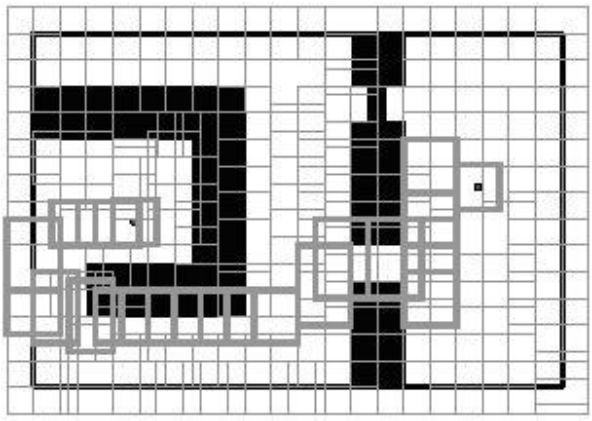

Figure 9. The plan of new environment.

VII. CONCLUSION AND FUTURE WORK

A reinforcement learning model with cells management algorithms has been proposed. From the experiment result, the avatar uses less storage and finds paths efficiently (165 cells versus 12,000 states). By modifying position and range of cells, the avatar can adapt itself to the changing of environment and keep a well balance between exploration and exploitation behavior. In this way, the avatar will tend to exploitation when the environment is unstable and exploration when the environment is steady.

The previous discussed cases are all assumed that there is always at least one solution. However, in fact, there maybe no solution exists. How the avatar deals with such kind of situation should be considered in the futual work.

REFERENCES

[1] A. W. Moore, L. P. Kaelbling, and M. L. Littman, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[2] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[3] A. W. Moore, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," in Cowan, J. D., Tesauro, G., & Alspector, L.(Eds.), *Advances in Neural Information Processing System 6*. San Mateo, CA. Morgan Kaufmann., 1994, pp. 711–718.

[4] W. Macready and D. H. Wolpert, "Bandit problems and the exploration/exploitation tradeoff," *IEEE transactions on Evolutionary Computation*, vol. 2, pp. 2–22, 1998.

[5] P. K. Simpson, "Fuzzy min-max neural networks. I. Classification," *Neural Networks, IEEE Transactions*, vol. 3, pp. 776–786, 1992.

[6] P. K. Simpson, "Fuzzy min-max neural networks – part 2: Clustering," *Fuzzy Systems, IEEE Transactions*, vol. 1, p. 32, 1993.