Algorithm for Search Space Reduction based on Dynamic Heuristic Value Change

Hyung-Soo Kim* · Kyung-Seob Moon**

요 약

실시간 전략 게임은 인간 혹은 컴퓨터를 상대로 하는 게임 장르이다. 이것은 턴방식의 컴퓨터게임과는 게임 진행 방식이 상이하다. 체스와 같은 턴방식은 오직 한 명의 플레이어의 동작을 허용하지만 실시간 전략게임은 복수의 플레이어의 동시다발적인 동작을 허용한다. 따라서 실시간 전략게임에서는 게임 내 유닛들의 이동경로는 자원채취, 건물건설, 그리고 전투 프로세스들의 처리를 위한 충분한 시간을 확보하기 위해 신속히 계산되어야 한다. 경로계산에 필요한 메모리, 탐색공간, 유닛들의 반응속도를 향상시키려는 여러 접근방식들이 소개되고 있다. 현재의 경로계산 알고리즘들은 최상 경로계산에 치중한 나머지 실시간 전략게임에서의 계산 과부하 문제를 고려하고 있지 않다. 이런 점에서 본 논문은 탐색공간을 줄이고 유닛들의 반응속도를 높이는 DHA*(Dynamic Heuristic A*) 알고리듬을 제안하고, 기존 A* 알고리듬과 비교하여 본 제안 DHA* 알고리듬의 성능이 우수함을 입증한다.

Abstract

Real time strategy game is a computer game genre of playing with human or computer opponents in real time. It differs from turn-type computer games in the game process method. Turn type games, such as chess, allow only one player to move at a time. Real time strategy games allow two or more players to move simultaneously. Therefore, in real time strategy computer games, the game components' movement plans must be calculated very quickly in order to not disturb other processes such as gathering resources, building structures, and combat activities. There are many approaches, which can reduce the amount of memory required for calculating path, search space, and reactive time of components. (or units). However, existing path finding algorithms tend to concentrate on achieving optimal paths that are not as important or crucial in real time strategy game. This paper introduces Dynamic Heuristic A*(DHA*) algorithm which is capable of reducing search space and reactive time of game units and compares with A* algorithm using static heuristic weighting.

Keywords

Search Space Reduction, Heuristic Value, Dynamic A*, DHA*

Ⅰ. Introduction

As the use of internet increases, it is necessary to get a useful real time information in the large scale database. In real time strategy games, most of the simple games adopt the path searching method with the artificial intelligence techniques and also use a finite state machine to represent the action of a character[1]. The units path calculation should be performed very rapidly

* Division of Computer Information, Cheju Halla College, Korea

**School of Information Technology, Griffith University, Gold Coast Campus QLD Australia 4217

접수일자 : 2002. 10. 18

so as not to disturb other processes such as gathering resources, building structures, and combat activities. Also, it is important to show immediate reaction when the user commands the units to move. This can increase the reality of computer games. However, due to the limitation of computer processing power it can be problematic to calculate all the possible paths of units. It is not uncommon to have several hundred units in real time strategy games. Therefore, calculating those paths will have an effect on other processes in the games so a reasonable path finding algorithm is essential. Existing search algorithms can be divided into off-line search and on-line search. The off-line search algorithm calculates the complete path using omniscient view and after completing the path calculation, the units move along the path. The on-line search algorithm calculates complete or partial path using complete or coarse map information and after completing the path calculation, the units move along the path. However, on-line search algorithms may find discrepancies between provided map information and real map information. As a result, on-line search algorithms can correct those discrepancies while units are moving and fix incorrect map information. Off-line search algorithms can be divided into blind-search and heuristic search[2]. The blind-search method includes breadth-first, depth-first, iterative deepening, iterative broadening. Best-first, A* and IDA* are included in the heuristic search[3]. On-line search algorithms include D*, RTA*, LRTA* and CRTA*[4].

Generally A* algorithm or the variation of A* is used in real time strategy game. Best-first is a variation of A* which sacrificed optimality, increased heuristic weight and used A* heuristic overestimation. This is due to process management, not only path calculations but also building structures, battle management and gathering resources[5].

However, the process for initial path generation and correcting map discrepancy can causes undesirable effects on the system, as well as reducing the reality of the game and overall process speed. Therefore, a new approach is needed in order to reduce the search space and required memory as well as increase the reactive time of game units.

This paper proposes a new algorithm, Dynamic Heuristic A* algorithm (DHA*), that requires less memory and processing time for path finding. Heuristic overestimation may improve performance in terms of reducing search space. However, applying static heuristic overestimation to a crowded environment can possibly increase search space. DHA* divides map environment into many blocks and analyses the block. The adequate heuristic weights are applied for path calculation and also described the result for the effect of applying dynamic heuristic weight compared with A*. By the result, we show that DHA* can be applied to both dynamic environment and unknown or partially known environment.

II. Description of search algorithm

2.1 Off-line path search

2.1.1 A* algorithm

Off-line search algorithms find paths using initial map information and units move along the paths. The representative ones in off-line search algorithms are A* and IDA*. A* algorithm[6] and the variations of A* are in widespread use for real time strategy games because of the efficient and optimal outcomes that can be achieved. However, when using these algorithms the complete path must be calculated before the units can be moved. Also, there may be

discrepancies between calculated path and gathered map information. While movements are recognised the path must be recalculated from the location or the start location to the destination[7].

A* algorithm uses heuristic function and judges the travel cost of a node between start position and destination position as follows expressed.

f(x) = g(x) + h'(x) where;

f(x): the travel cost of going through x node from start position to destination position;

g(x): the actual travel cost from start position to x position

h'(x): the estimated travel cost from x position to destination position.

h'(x) can be calculated using several heuristic with manhattan distance, diagonal distance and straight line distance, which are those heuristics that are being used in a grid map environment

Manhattan distance can be expressed as h'(x) = | X(x) - goal(x) | + | X(y) - goal(y) | and is the sum of the x distance and y distance between X position and goal position. Diagonal distance can be expressed as h'(x) = max(| X(x) - goal(x) | , | X(y) - goal(y) |) and is the maximum value between the x distance and y distance between X position and goal position. Straight line distance can be expressed as $h'(x)=\sqrt{|\,|X(x)-goal(x)|+|X(y)-goal(y)|\,|}$ and is the geometric distance between X position and goal position.

The role of g(x) in A* algorithm is to allow the algorithm to choose a node which is approaches the closest direction to the goal node. When finding the path itself is the reason for performing A*, the role of g(x) is essential. However, if finding a solution as soon as possible is the aim of performing A*, then set g is always 0 (zero) and only the h(x) function is used. When the aim of performing A* is finding the path that contains the minimum number of nodes, the g(x)function must be set so that the travel cost from x to the previous node is equal to 1 (one). If nodes have various travel costs from one node to successor nodes, then calculate the g(x) value using an appropriate travel cost generation system to calculate optimal value. The A* algorithm can be used for finding either the optimal path or the most direct or shortest path.

h'(x) is the estimated value of h(x) which is the actual distance between the x position and the destination position. If h'(x) is the perfect estimated value, then A* will not search any other path to finds a path to destination position. The more efficient h'(x) is, the less A* searches for optimal paths. If h'(x) is always 0 (zero), then the search will be controlled by g(x) and when g(x) is equal to 1 (one), then it will be a breadth-first search. If h'(x) is admissible, h'(x) $\leq$ h(x), and if a path exists, then A* algorithm must find the optimal path from start position to destination position. However if h'(x) overestimates h(x), then A* will stop searching once a path is found, and consequently the optimal path may not found. Therefore changing heuristic weight can reduce or increase search space.

2.1.2 Iterative-deepening A*

Basically, all nodes in A* algorithm that have been explored are recorded and kept in OPEN and CLOSE lists to calculate optimal path. This means that the memory requirement for path calculation using A* is high. The number of nodes grows exponentially with solution cost in A* on a tree. Iterative-deepening-A* (IDA*) proposed by Korf[8] solves the memory constraint of A* without sacrificing solution optimality. However IDA* tends to increase computation time while reducing memory requirements. IDA* sets the cutoff value using

heuristic estimation and performs depth-first search. It uses f(x) to calculate travel cost and if f(x) exceeds the cutoff value, then the path is cut off and the search backtracks before continuing. The cutoff value is set up to the minimum f(x) and it repeats these processes until the goal node is found.

Each iteration of the algorithm is a depth-first search that keeps track of the cost, f(x) = g(x) + h'(x) of each node generated. It's memory requirement is linear with respect to the maximum search depth and if the heuristic function is admissible, IDA* finds an optimal path. However, due to the iteration of calculation, it tends to be slower than A* in terms of calculation speed. Furthermore, usually the map environment of real time strategy game is a grid map environment in which search space does not grow exponentially while search process goes on.

2.2 On-line path search

2.2.1 Real time A*

On-line search algorithms can be used when no map or only partial map information is available. Also, when map environments are changed dynamically, these algorithms can manage to correct map discrepancy while units or robots are moving. At the early stage of developing on-line search algorithms, map environment was considered to be fixed or unchangeable[9]. Lumelsky and Stepanov initially assumed the environment to be devoid of obstacles and calculated the path to a goa[10]. When map discrepancies were found, a new path was calculated from that position or the start position to the goal position. For a path of N steps, approximately N2 path steps are computed over time. Therefore, path recalculation is not an efficient method, especially in expansive environments where the goal is at a distance.

Real time A* algorithm(RTA*) is on-line search algorithm that uses initial map information to estimate the cost to the goal for each state and efficiently updates it with backtracking costs as the unit moves through the environment[11]. This approach is complete and locally optimal on trees. However, In RTA* the heuristic estimation is in general not admissible. RTA* generates initial map and corrects discrepancy between initial map and gathered map information. The processing time for generating and correcting discrepancy grows exponentially by the size of the search space. RTA* uses initial map information and calculates travel cost to goal position. While following the path, units collect surrounding map information. If the discrepancy is found, then the units back track and correct the map information. The formula of the RTA* is as follows,

f(x) = g(x) + h'(x) where:

f(x): the travel cost of going through x node from start position to destination position

s: the successor of x

g(x): the travel cost from x node to s

h'(x): the estimated travel cost from x position to destination position.

h'(s): the heuristic value of the successor (s).

Firstly, RTA* sets up h'(x) as the sum of g(x) and second best value of h'(s). h'(s) is successor s' heuristic value. The second step is to move to the node that has the best f(x) value. These procedures are repeated until goal node is found. RTA* is guaranteed to find a solution if there is a solution, and it makes locally optimal paths with the information it has been given so far.

2.2.2 Dynamic A*

The efficiency of A* in unknown or partially known environments can be improved using Dynamic algorithm A*(D*) which uses RAISED and LOWER states to propagate correct travel

cost when map discrepancies are found[5]. D* can be used when complete map information is not available and never inferior to A*. However, D* uses more space than A* because it keeps its internal information such as OPEN/CLOSE sets, path tree and g values. It can be used in situations where partial or no map information is available or/and environments are changed dynamically and also corrects travel cost of only appropriate nodes in the map only when discrepancies between initial map information and collected map information are found. RAISED states are used to propagate increased travel cost when path cost is increased. LOWER states are used to propagate decreased travel cost when path cost is decreased. D* assigns 0 (zero) to an unknown node's estimated travel cost and puts the node into OPEN list. When it finds an obstacle that is not in the map, it compares the value of nodes in the OPEN list and corrects the travel cost of nodes. Also, if an obstacle in the map is no longer there, the nodes nearest the position are recalculated to lower travel cost and also uses the optimistic optimal. In an unknown environment, it assumes there is no obstacle and lowers the travel cost of unknown nodes. When the discrepancies are found, appropriate travel cost changes are propagated through from the units' position to effective scope. D* satisfies completeness and can find optimal path given the information it has received so far. However when initial paths are calculated, it has to use the whole map. If there is only partial or no map information, then overloaded path recalculations have to be performed.

III. Modeling of DHA* Algorithm

3.1 Definition of DHA*

Now, we suggest Dynamic heuristic A* algorithm (DHA*) reduced search space and reactive time of game units. This approach uses the characteristic of A* algorithm that specifies that if h'(x) overestimates h(x), then search space will be reduced more than for admissible heuristic. DHA* divides the map into several blocks, analyses the block and decides appropriate heuristic weight. DHA* uses f(x) = g(x) + h'(x) and calculates paths to goal node. h'(x) is calculated using heuristic weight value depending on the density of obstacles in the block that is occupied by the current units.

DHA* changes heuristic weight dynamically depending on the density of obstacles on the blocks and as soon as finishing path calculation in one or several blocks the units starts moving towards the goal node. Therefore, DHA* can reduce processing time and search space for path calculation and improves the reactive speed of units

3.2 Algorithm Description

DHA* uses Manhattan distance which is the sum of the x distance and y distance between X position and goal position to calculate the distance between two positions. Target node can be decided by using the sum of g(x) and h(x) as A* algorithm uses. Table 1 illustrates the process of DHA* algorithm.

Table 1. The suggestion of DHA* algorithm

1. Divide map into blocks.
2. Analyse start block and decide heuristic weight. If the density of an obstacle is high then use low heuristic weight to increase search space.
3. Calculate path using $f(x) = g(x) + h'(x)$ until the path reaches neighbouring blocks.
4. If the block is the previous block then enclose the entrance of current block to the neighbouring block and move to the neighbouring block
5. Repeat 2 until a goal node is found.

3.3 Illustration of Operation

DHA* was tested using Pentium Ⅳ 1.6 GHz system and programmed in Lingo scripts in director 8.0. A* and DHA* use heuristic weights 1 to 5 to reduce search space but DHA* algorithm applies heuristic weighting dynamically. Fig.1 shows a 10% density typical map and a solution generated by A* with heuristic weight of 1. Fig.2 shows a solution as generated by DHA* on the same map as in Fig.1. As it seen in Fig 2, DHA* divided the map into many blocks, analysed the density of obstacles and applied appropriate heuristic weight. The searched space of Fig.1 is decreased in Fig. 2 due to the dynamic heuristic weight changes. However, generated solutions are exactly the same.

In both Fig.1 and Fig.2, the letter "S" (middle-left of the map) shows the start position and the letter "E" (middle-right of the map) shows the destination position. The search starts from the block that includes "S" and goes on until the path meets neighbouring blocks. When a neighbouring block is encountered, then units start to move to the encountered block. These processes continue until "E" mark is found.

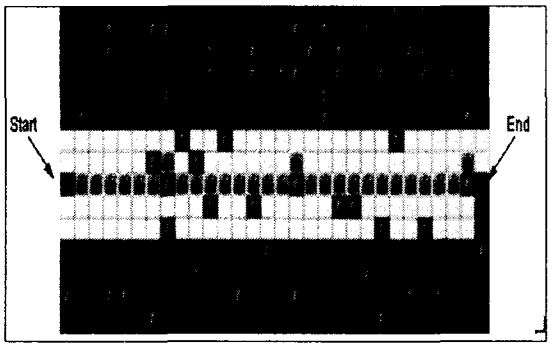

Fig. 1. Solution used A*(h1) in 10% density map

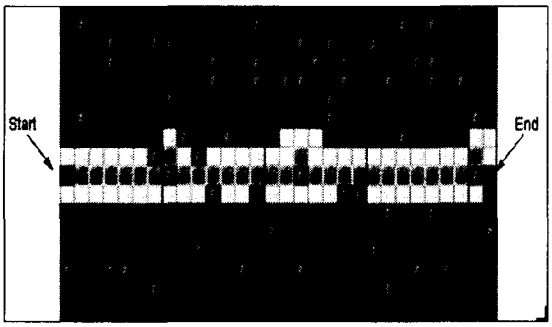

Fig. 2. Solution used DHA* in 10% density map

Ⅳ. Experimental Evaluation

4.1 Experimental Procedure

To examine the efficiency of DHA*, randomly generated maps were used. Maps of overall obstacle density 10%, 20%, 30%, 40% and 50% were generated and five different approaches: DHA*, A* with heuristic weight 1, 3, 5, 10, were used. Obstacles are not passable. Because of the obstacle density levels used, the difficulty of the searches and obstacles provided various levels of challenge. The map width and height was 30 by 28 cells. Experiments were performed 300 times for each approach on each obstacle density.

4.2 Result Evaluation

Table 2 shows the results of the comparison for environments of size 30 by 28 cells. As it can be seen from the results, DHA* decreased search space by an average of 55% than average search space of A* with heuristic weight 1 but slightly dropped down optimality by average 1%. Overestimated A* algorithms also increased their performance and the performance differences were not significant between DHA* and overestimated A* algorithms. As it can be seen from Fig.3, when the density of obstacle was high the searched space decreased. However if the density of obstacle was higher than 50%, then the searched space increased due to the increment of detour caused by obstacles. Fig. 4 shows the change of traveled distance. The traveled distance was increased when the density of obstacles was increased.

Table 2. Comparison result of the Obstacle Density

Obstache Density(%)	Factor	DHA*	A*(h1)	A*(h3)	A*(h5)	A*(h10)
10	Traveled distance	28.00	28.00	28.00	28.00	28.00
	Optimality Factor	1.00	1.00	1.00	1.00	1.00
	Searched Space	166.95	291.12	175.48	174.59	174.74
	Speed up Factor	0.57	1.00	0.60	0.60	0.60
20	Traveled distance	28.10	28.00	28.03	28.03	28.09
	Optimality Factor	1.004	1.00	1.001	1.001	1.003
	Searched Space	145.60	266.34	151.21	152.28	152.53
	Speed up Factor	0.55	1.00	0.57	0.57	0.57
30	Traveled distance	28.27	28.13	28.14	28.33	28.45
	Optimality Factor	1.005	1.00	1.00	1.007	1.011
	Searched Space	130.96	232.23	133.41	134.81	134.45
	Speed up Factor	0.56	1.00	0.57	0.58	0.58
40	Traveled distance	29.11	28.56	29.17	29.35	29.61
	Optimality Factor	1.019	1.00	1.021	1.028	1.037
	Searched Space	114.48	204.70	132.87	127.47	127.87
	Speed up Factor	0.56	1.00	0.65	0.62	0.62
50	Traveled distance	31.63	30.77	31.62	31.69	33.04
	Optimality Factor	1.028	1.00	1.028	1.030	1.074
	Searched Space	104.72	207.80	147.11	138.52	136.60
	Speed up Factor	0.50	1.00	0.71	0.67	0.66

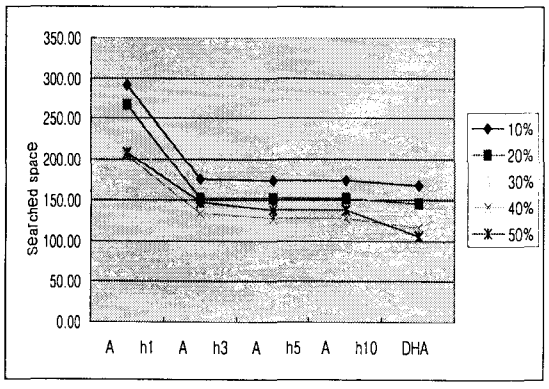

Fig. 3. Change of searched space

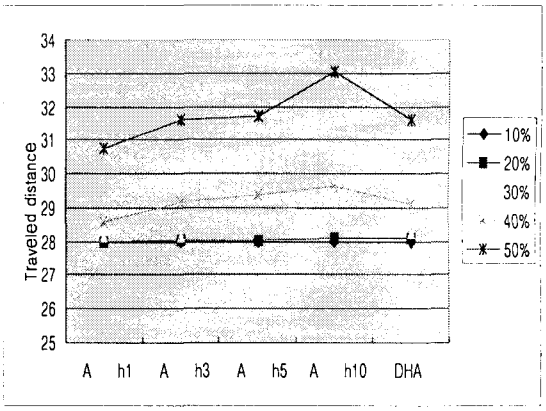

Fig. 4. Change of traveled distance

V. Conclusion

This paper examined the characteristics of various search algorithms that use heuristics. Also, it looked over the disadvantages and advantages of A* on real time strategy game. DHA* was suggested as a solution to solve the disadvantages such as memory amount, search space, and reactive time problem. The DHA* was examined by experiments through various circumstances. As it can be seen the experiments results, DHA* can reduce search space by average 55% but it deteriorates optimality by average 1%. However, the performance difference between DHA* and static overestimated heuristic A* is not remarkable. It seems like it caused by random obstacle generation system. The system generated almost uniform obstacle density. Therefore it failed to generate maps that dynamic heuristic weight change approach can show its advantages. Increasing optimality is crucial when the density of obstacles is increased due to the tendency of wandering about of DHA*. The rolls of block size in DHA* will be also examined. The completeness and soundness must be proved for usability proof of DHA*. Future work will

toward for global optimality improvement and efficiency improvement over static heuristic overestimation.

References

[1] Hyung-Soo Kim and Hong-Gi Kim, "Algorithm for Knowledge Discovery based on Data Classification and Approximate Inference", 2nd International Symposium on AISCP, VolⅡ, No.2, KAIST, Korea, pp.27-31, 2001.

[2] Rich, E., Artificial Intelligence, McGraw-Hill, 1991.

[3] Ginsberg, M., Essentials of Artificial Intelligence, Morgan Kaufmann Publishers, Inc. 1997.

[4] Edelkamp, S. and Eckerle, J., "New strategies in learning real time heuristic search", National Conference on Artificial Intelligence (AAAI), Workshop on On-line Search Providence, Rhode Island, AAAI-Press, pp.30-35, 1997.

[5] Kawick M, Real Time Strategy Game Programming Using MS Direct X, Wordware Publishing, Inc. 2000.

[6] Hart, P.E., Nilsson, N.J. and Raphael, B. "A formal basis for heuristic determination of minimum path cost", IEEE Trans. On SSC 4:100, 1968.

[7] Amit, J. P., "Amit's thought on path-finding, <http://theory.stanford.edu/ ~amitp/ Game Programming/>, 2001.

[8] Korf, R.E., "Depth-first iterative-deepening: An optimal admissible tree search", Artificial Intelligence 27(1), pp. 97-109, 1985.

[9] Latombe, J. C., Robot Motion Planning, Kluwer Academic Publishers, 1991.

[10] Lumelsky, V. J. and Stepanov, A. A., "Dynamic Path Planning for a Mobile Automation with Limited Information on the Environment, IEEE Transactions on Automatic Control, Vol. AC-31, No. 11, November, 1986.

[11] Korf, R.E., "Real-time heuristic search", Artificial Intelligence 42(2-3), pp.189-211, 1990.

[12] Stentz, A., "Optimal and efficient path planning for partially-known environments", Proceedings of the IEEE International Conference on Robotics and Automation, 1994.

저자소개

김형수(Hyung-Soo Kim)

1998년 충북대학교 전자계산학과 졸업(이학박사)

1991년 숭실대학교 정보산업과 졸업(이학석사)

1985년 성균관대학교 정보처리과 졸업(경영학석사)

1981년 제주대학교 수학교육과 졸업(이학사)

1992년~현재 제주한라대학 컴퓨터정보계열교수

※관심분야: 퍼지 및 러프이론, 게임이론, 정보검색, 멀티미디어, 웹에이전트 시스템.

문경섭(Kyung-Seob Moon)

2002년 호주 그리피스 멀티미디어과 졸업(공학석사) 및 박사과정 재학

2000년 호주 그리피스 멀티미디어과 졸업(공학사)

※관심분야: 무선인터넷, 멀티미디어, 게임이론.