Journal of Korea Game Society 2013 Dec; 13(6): 43-52
http://dx.doi.org/10.7583/JKGS.2013.13.6.43

게임 프로그래밍

효율적인 휴리스틱 계산 처리를 위한 가중치 기반의 선수행 A* 알고리즘

오민석°, 박성준
호서대학교 게임학과
ominseok@naver.com, sjpark@hoseo.edu

A Weighted based Pre-Perform A* Algorithm for Efficient Heuristics Computation Processing

Min-Seok Oh°, Sung-Jun Park
Department of Game, Hoseo University

요 약

경로 탐색은 인공지능의 매우 중요한 요소 중의 하나이며, 여러 분야에서 두루 쓰이는 과정이다. 경로 탐색은 매우 많은 연산이 필요하기 때문에 성능에 매우 중대한 영향을 미친다. 이를 해결하기 위해서 연산량을 줄이는 방식의 연구가 많이 진행되었고, 대표적으로 A* 알고리즘이 있으나 불필요한 연산이 있어 효율성이 떨어진다. 본 논문에서는 A* 알고리즘 중 연산 비용이 높은 노드 탐색 수 등 연산량을 줄이기 위해서 가중치 기반의 선수행 A* 알고리즘을 새롭게 제안한다. 제안한 알고리즘의 효율성을 측정하기 위해 시뮬레이션을 구현하였으며, 실험 결과 가중치를 이용하는 방법이 일반적인 방법보다 약 1~2배 높은 효율을 보였다.

ABSTRACT

Path finder is one of the very important algorithm of artificial intelligence and is a process generally used in many game fields. Path finder requires many calculation, so it exerts enormous influences on performances. To solve this, many researches on the ways to reduce the amount of calculate operations have been made, and the typical example is A* algorithm but it has unnecessary computing process, reducing efficiency. In this paper, to reduce the amount of calculate operations such as node search with costly arithmetic operations, we proposes the weight based pre-processing A* algorithm. The simulation was materialized to measure the efficiency of the weight based pre-process A* algorithm, and the results of the experiments showed that the weight based method was approximately 1~2 times more efficient than the general methods.

Keywords : AI, Path Finding, A*, Dijkstra, Game, Search Algorithm

Received: Sep. 05, 2013 Revised: Nov. 21, 2013
Accepted: Dec. 09, 2013
Corresponding Author: Sung-Jun Park(Hoseo University)
E-mail: sjpark@hoseo.edu

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

일반적으로 경로 탐색 알고리즘은 네트워크의 경로를 설정하거나, 로봇이나 게임 유닛 등의 이동 경로를 설정하는 등의 목적으로 사용하는 알고리즘으로 다양한 연구가 진행되었고, 그 중 인공지능 분야에서의 연구 결과, Dijkstra와 A*를 비롯하여 B*, D*, IDA*, SMA*, HPA* 등 많은 경로 탐색 알고리즘이 개발되었다[1,2,3].

A* 알고리즘의 성능은 Dijkstra 알고리즘보다 뛰어나지만 메모리를 많이 차지하고, 탐색 속도가 느려 실시간 환경에 적합하지 않으며, 맵 데이터가 고정되어있는 정적인 환경이 적합하다는 등의 문제점이 있기 때문에, 그 연산 비용을 줄이는 기술이 매우 중요하다[4,5].

D* 알고리즘은 검색 공간이 클수록 A* 알고리즘을 사용할 때에 비해서 이득이 더욱 커진다. 경로를 만드는 데 필요한 계산 비용은 그래프의 노드 개수에 비례하는데, D* 알고리즘은 그러한 재계산을 최대한 피하기 때문이다[6,7].

IDA* 알고리즘은 Open List와 Closed List를 만들지 않기 때문에, A* 알고리즘에 비해서 메모리 요구량이 매우 적지만, 검색에 걸리는 시간이 늘어날 수 있다[8,9].

HPA* 알고리즘은 모든 노드를 계층 구조를 이루는 그래프로 만들어, 높은 수준의 간략화 된 경로부터 찾은 뒤에, 낮은 수준의 세부 경로까지 순차적으로 검색하는 알고리즘이다. 적절히 계층 구조를 구성할 경우 대부분 A* 알고리즘보다 빠르지만, 모든 노드들을 계층 구조로 미리 만들어야 한다는 점과 적절한 크기의 계층 구조로 정확히 구성하기 쉽지 않다는 단점이 있다[10].

본 논문에서는 A* 알고리즘에서 연산 비용과 연관이 깊은 휴리스틱 값(Heuristic value : H)을 계산할 때, 최단 경로를 찾을 수 있을 정도로만 H를 과대평가하여, 최적의 가중치를 미리 계산해서 사용하는 방법인 가중치 기반의 선수행 A* 알고리즘(A Weighted based Pre-perform A* algorithm : WPA* algorithm)을 개발했으며, 해당 알고리즘의 성능 평가 및 고찰을 위한 실험을 수행하였다. 이 알고리즘은 맵 데이터가 변하지 않는 환경에서만 사용할 수 있으며, 이때 사용하는 방법은 A* 알고리즘뿐만 아니라 IDA* 알고리즘이나 HPA* 알고리즘 등 A* 알고리즘에서 변형된 여러 알고리즘에서도 비슷한 방식으로 사용할 수 있다[11].

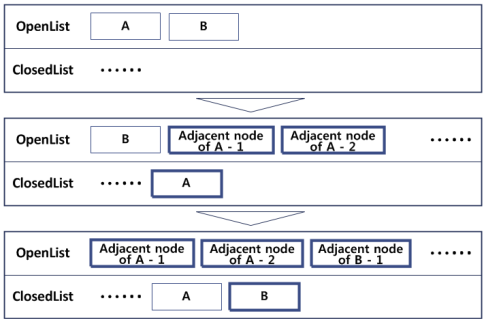
본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 설명하고 있으며, 3장에서는 WPA* 알고리즘 및 시뮬레이션 프로그램의 개요와 흐름에 대해 설명하고 있고, 4장에서는 시뮬레이션 프로그램으로 실험한 결과에 대해서, 5장에서는 결론 및 향후 연구 방향에 대해서 설명하고 있다.

2. 관련 연구

2.1 A* 알고리즘

A* 알고리즘은 기본적으로 아직 조사하지 않은 노드들 중 가장 유망한 노드를 조사하는 과정을 반복한다. 만일 그 노드가 목표이면 알고리즘은 끝나고, 그렇지 않으면 그 상태의 인접한 노드를 조사해 나간다.

A* 알고리즘은 [Fig. 1]과 같이 두 종류의 목록들을 관리한다. 하나는 아직 조사하지 않은 노드들을 담은 열린 목록이고, 또 하나는 이미 조사한 노드들을 담은 닫힌 목록이다. 알고리즘의 시작에서 닫힌 목록은 비어 있으며, 열린 목록은 오직 시작 노드만을 가지고 있다. 각각의 반복에서 알고리즘은 열린 목록의 노드들 중 가장 유망한 것을 가져온다. 그 노드가 목표가 아니면 이웃한 노드들 중 새로운 것은 열린 목록에 넣고, 이미 목록에 있는 것은 비용 비교 후 갱신한다. 목표에 도달하기 전에 열린 목록이 비게 되면 시작 노드로부터 목표 노드에 도달하는 경로가 존재하지 않는 것이다[12].



[Fig. 1] Status list of A* algorithm

A* 알고리즘에서 비용을 계산할 때는 총 2가지 비용의 합을 이용한다. 하나는 시작 노드부터 해당 노드까지의 비용인 G이고, 다른 하나는 해당 노드부터 목표 노드까지의 비용인 H이다. 이 G와 H의 합을 F라고 하는데, 여기서 H는 실제 값을 정확히 알 수 없기 때문에 추측을 통해 계산한다. 이 H가 0인 경우가 바로 Dijkstra 알고리즘이다. 또한 H를 과대평가할 경우, 더 빠르게 목표 노드로 향할 수 있지만 최단 경로를 찾지 못하는 경우도 있다.

2.2 HPA* 알고리즘

HPA* 알고리즘은 시작 노드와 목표 노드가 속한 집합을 찾은 뒤, 해당 집합의 모든 연결 노드부터 시작/목표 노드까지의 비용을 계산한다. 이어서 이 연결 노드들을 각각 시작 노드들과 목표 노드들로 가지고 높은 레벨에서 경로 탐색을 시작한다. 찾아낸 높은 레벨 경로의 시작과 끝에 처음 주어진 시작 노드와 목표 노드를 추가하고, 이 경로에 있는 노드들 간에 이동할 수 있는 세부 경로를 낮은 레벨에서 탐색해서 연결하면, 최종 경로를 얻을 수 있다. 이 때 높은 레벨에서 경로를 얻는 순간 유닛은 미리 출발하고 이동하는 동안 세부 경로를 계산하는 등의 기법을 사용할 수 있는 특징이 있다.

본 논문에서는 기존의 A* 알고리즘에서 연산 비용과 연관이 깊은 휴리스틱 값을 적절히 과대평가하여 효율적으로 처리한 기법인 WPA* 알고리즘에 대해 소개한다.

3. 시스템 구성

3.1 WPA* 알고리즘

WPA* 알고리즘은 기존 A* 알고리즘에서 H의 가중치 W만 미리 계산해서 저장한 뒤 가져와서 사용하며, 나머지는 동일하다. 즉 A* 알고리즘의 비용 계산 공식이 (eq. 1)이라고 한다면, WPA* 알고리즘은 (eq. 2)라고 할 수 있다.

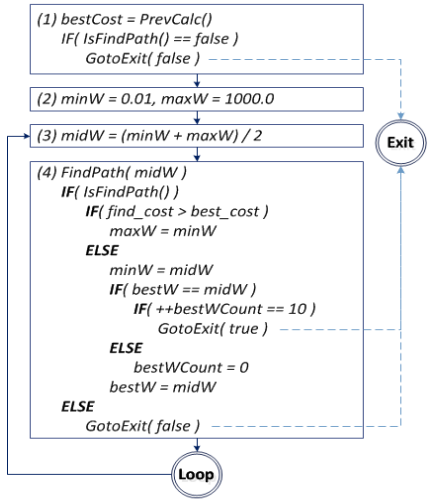
$$F = G + H$$

(eq. 1) Formula for calculating cost of A* algorithm

$$F = G + (H*W)$$

(eq. 2) Formula for calculating cost of WPA* algorithm

가중치 W는 선형 탐색 알고리즘과 이진 탐색 알고리즘 등을 통해서 계산할 수 있으며, [Fig. 2]는 이진 탐색 알고리즘을 이용한 W 계산 방법이다. 우선 Dijkstra 알고리즘을 이용해서 최단 경로의 비용을 구한 뒤, WPA* 알고리즘의 W를 조절해가면서 이진 탐색하는데, W가 일정 횟수 이상 같은 값이 나오면 종료하도록 했다.



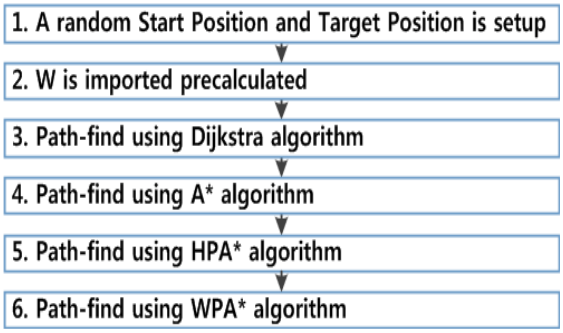
[Fig. 2] W calculation method using binary search algorithm

이때 W는 Dijkstra 알고리즘을 통해서 계산한 최단 경로의 비용과 동일한 비용이 나오는 경우에만 유효하다고 판단하기 때문에, Dijkstra 알고리즘과 같이 WPA* 알고리즘 또한 항상 최단 경로를 찾을 수 있다.

한편 W를 구하는 과정은 동일한 시작 노드와 목표 노드를 가지고 WPA* 알고리즘을 반복 실행해야하지만, 이 과정은 갈 수 있는 모든 노드들을 시작/목표 노드의 쌍으로 구성한 뒤에, W를 계산해서 파일이나 데이터베이스 등을 통해 저장해두는 전처리 과정에서 처리할 수 있다. 그 후 실제 WPA* 알고리즘을 사용할 때는 전처리 과정에서 얻어진 결과들 중 해당하는 W를 읽어서 사용하기 때문에, W를 읽는 비용만 추가로 필요하다. 단 노드의 수가 많을수록, 전처리 과정의 결과로 저장하는 데이터의 양도 많아질 수 있다.

3.2 시뮬레이션 구성

시뮬레이션은 Dijkstra 알고리즘과 A* 알고리즘, HPA* 알고리즘, 그리고 WPA* 알고리즘을 사용했을 때의 연산량과 불필요한 연산량 및 실행 시간의 차이를 알아보기 위해서 진행했으며, [Fig. 3]의 시뮬레이션 진행 순서에 따라 시뮬레이션 횟수만큼 반복했다. 이때 3~6을 진행할 경우의 연산량과 불필요한 연산량 및 실행 시간을 측정해서 비교했다.



[Fig. 3] Sequence of simulation

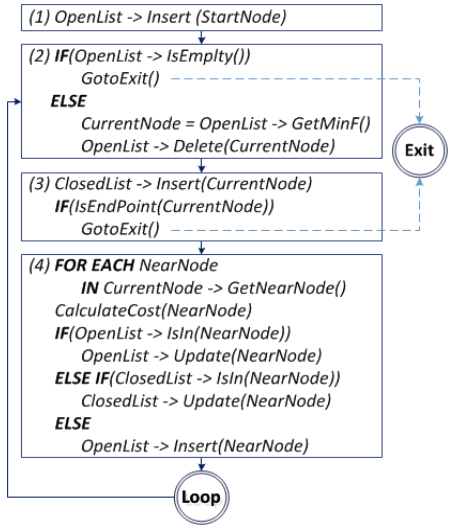
[Fig. 4]는 A* 알고리즘과 WPA* 알고리즘의 진행순서를 다이어그램으로 표한한 것이다.

(1) OpenList에 시작 노드를 삽입한다.

(2) 만약 OpenList가 비어 있으면 알고리즘을 종료하고, 그렇지 않으면 OpenList에서 F가 가장 작은 노드를 선택한 후 삭제한다.

(3) ClosedList에 선택한 노드를 삽입하고, 만약 선택한 노드가 목표이면 알고리즘을 종료한다.

(4) 현재 노드에 연결된 모든 노드에 대해 비용을 검사한다. 그 후 OpenList, ClosedList 각각에 대해 비용 비교 후 갱신한다. 만약 두 리스트 모두에 없으면 OpenList에 추가한다.

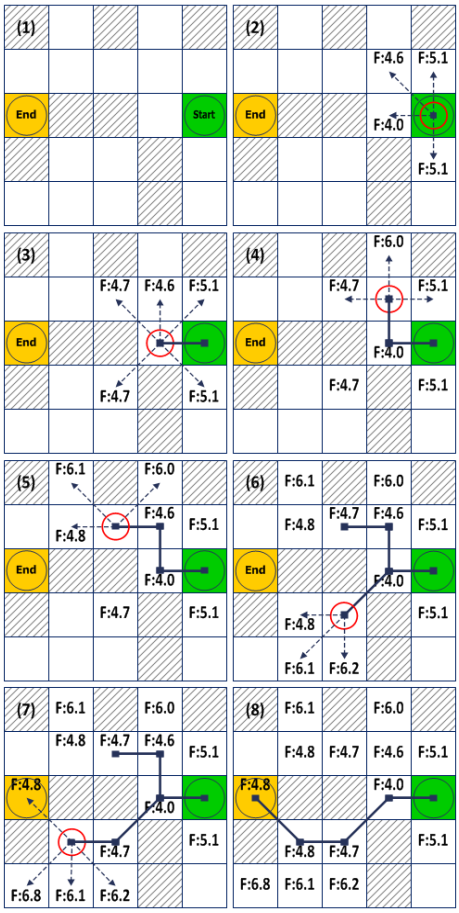


[Fig. 4] Sequence of A*/WPA* algorithms

[Fig. 4]에서 보는 바와 같이 A* 알고리즘과 WPA* 알고리즘의 진행 순서는 동일하나, (4)에서 각 알고리즘의 비용 계산 방식을 다르게 하였다. 비용 계산 시 A* 알고리즘일 경우 (eq. 1)을, WPA* 알고리즘일 경우 (eq. 2)를 이용했다. 즉 비용 계산 시 휴리스틱 비용을 계산할 때, 가중치를 적용하지 않으면 A* 알고리즘이고, 가중치를 적용하면 WPA* 알고리즘이라고 할 수 있다.

3.3 시뮬레이션 구현

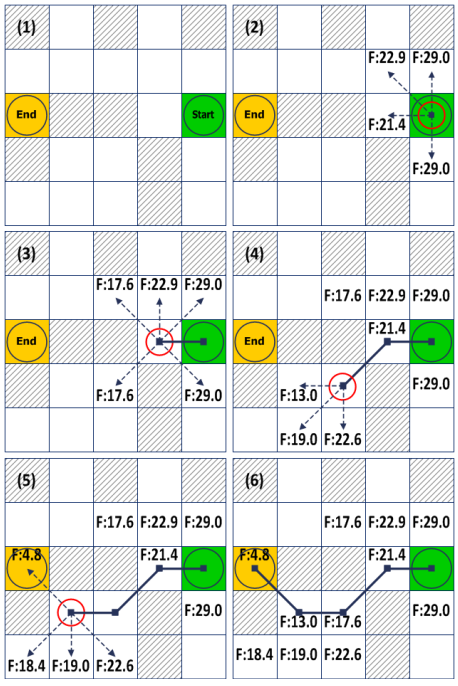
시뮬레이션 프로그램은 좌우 50개, 상하 50개의 타일로 구성된 타일맵 기반 환경으로, 갈 수 있는 곳과 갈 수 없는 곳 두 가지이다. 또한 각 노드는 자신 주변 8방향의 노드를 연결했으며, 한 노드에서 다른 노드로 이동하는 비용은 모두 동일하지만, G를 계산할 때 유클리드 거리(Euclidean Distance)에 의해 사선으로 이동할 경우 1.41421로, 그렇지 않을 경우 1.0의 비용으로 계산했다.



[Fig. 5] Flow of A* algorithm

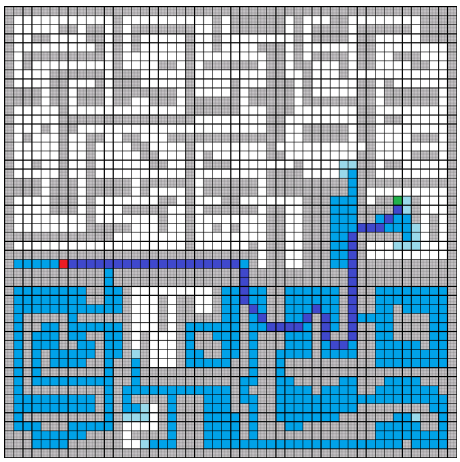
[Fig. 5]는 A* 알고리즘의 흐름을 도식화한 것으로, 왼쪽 위부터 오른쪽 아래의 순서이다. 가중치가 1.0인 전형적인 A* 알고리즘으로, 비용 계산 공식으로 [eq. 1]을 이용했으며, 총 7단계의 탐색이 필요했다. 각 타일의 위치를 좌상단부터 [x,y]로 표시했을 때, 시작 위치가 [5,3]이고 목표 위치가 [1,3]인데, 각 노드의 비용인 F가 낮은 순서에 따라 [5,3], [4,3], [4,2], [3,2], [3,4], [2,4], [1,3]의 순서로 탐색이 진행되어 경로를 찾는 것을 볼 수 있다.

[Fig. 6]은 WPA* 알고리즘의 흐름을 도식화한 것으로, 마찬가지로 왼쪽 위부터 오른쪽 아래의 순서이다. 이때, 이진 탐색 알고리즘을 통해 찾아낸 가중치 1.21와 (eq. 2)를 비용 계산 공식으로 이용했는데, 각 노드의 비용을 보면 A* 알고리즘에 비해 휴리스틱 비용을 더 높게 계산했다는 것을 알 수 있고, 탐색이 2단계 줄었으며, 탐색 노드 수 등 연산량이 약 11% 감소했다. 앞서 보았던 [Fig. 5]과 동일한 시작 위치 [5,3]과 목표 위치 [1,3]으로 탐색했는데, 각 노드의 비용 F가 낮은 순서에 따라 [5,3], [4,3], [3,4], [2,4], [1,3]의 순서로 탐색이 진행되어 경로를 찾는 것을 볼 수 있다.

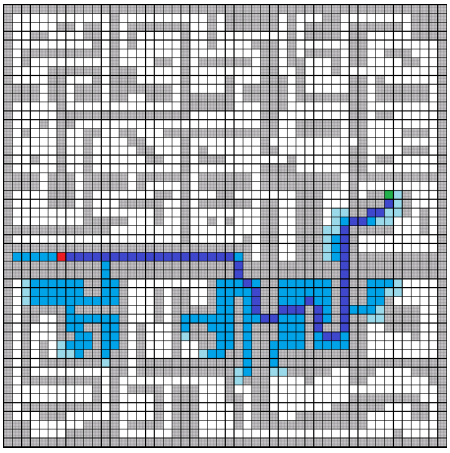


[Fig. 6] Flow of WPA* algorithm

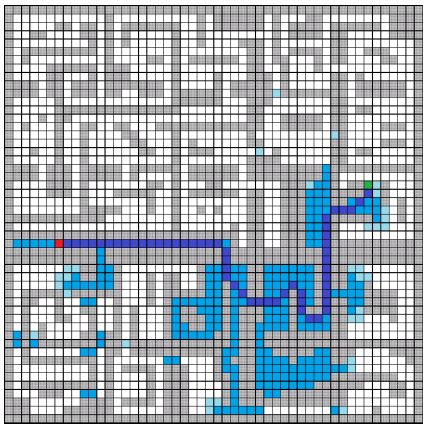
[Fig. 7,8,9,10]은 Dijkstra 알고리즘과 A* 알고리즘, HPA* 알고리즘, 그리고 WPA* 알고리즘을 실행한 프로그램의 화면이다. Dijkstra 알고리즘과 A* 알고리즘, HPA* 알고리즘은 가중치가 0.0과 1.0, 1.0인 전형적인 방식이었고, WPA* 알고리즘의 가중치는 이진 탐색 알고리즘을 통해 계산한 1.96이었다. 이 가중치에 따라 진행한 탐색 결과 화면을 보면, Dijkstra 알고리즘, HPA* 알고리즘, A* 알고리즘, WPA* 알고리즘 순으로 짙은 색의 탐색한 노드 수가 적어지며 더 적은 횟수와 연산량으로 경로를 찾는 것을 확인할 수 있다.



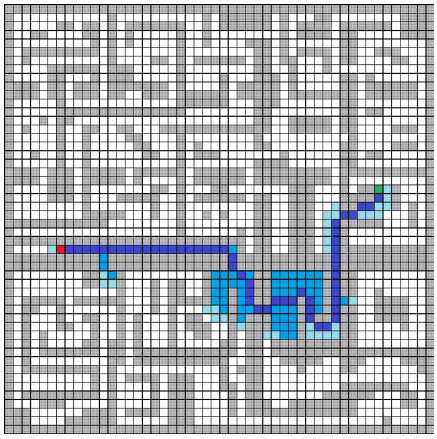
[Fig. 7] Screenshot of Dijkstra algorithm



[Fig. 8] Screenshot of A* algorithm



[Fig. 9] Screenshot of HPA* algorithm



[Fig. 10] Screenshot of WPA* algorithm

4. 실험 및 결과

4.1 실험 환경의 구성

실험은 Intel I7-3770 4.3GHz, 8GB Ram, Radeon HD 6850 1GB, Windows7 64bits 컴퓨터에서 진행했다.

4.2 실험 결과

시뮬레이션 프로그램을 이용해서 약 2천회의 시뮬레이션을 통해 얻어낸 결과를 '연산량', '불필요한

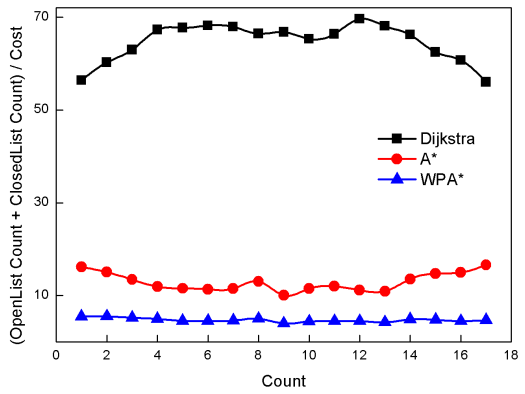
연산량', '실행 시간' 등의 세 가지 항목으로 구분해서 정리했다.

[Table 1]과 [Fig. 11]은 열린 목록의 노드 개수와 닫힌 목록의 노드 개수의 합을 찾은 경로의 비용으로 나눠서 정리한 표인데, 이 표의 Average 값이 곧 연산량과 정비례한다고 할 수 있다. 즉 이 값이 클수록 연산량이 많다고 할 수 있다.

이 때 A* 알고리즘을 이용했을 경우를 100.0%라고 하면, Dijkstra 알고리즘을 이용했을 경우 148.54%, HPA* 알고리즘을 이용했을 경우 127.31%, WPA* 알고리즘을 이용했을 경우 69.38%임을 알 수 있었다. 즉 WPA* 알고리즘의 연산량이 A* 알고리즘에 비해 약 1.44배 적다고 판단할 수 있었다.

[Table 1] Computation

	Average	Rate
Dijkstra	23.39	148.54%
A*	15.75	100.00%
HPA*	20.05	127.31%
WPA*	10.93	69.38%



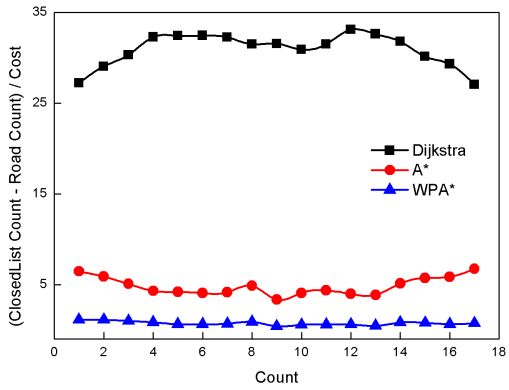
[Fig. 11] Computation

[Table 2]와 [Fig. 12]는 닫힌 목록의 노드 개수와 탐색한 경로의 노드 개수의 차를 찾은 경로의 비용으로 나눠서 정리한 표인데, 이 표의 Average 값이 작을수록 경로가 아닌 노드를 탐색하는 불필요한 연산이 적다고 판단할 수 있다.

이때 A* 알고리즘을 이용했을 경우를 100.0%라고 하면, Dijkstra 알고리즘을 이용했을 경우 158.14%, HPA* 알고리즘을 이용했을 경우 123.07%, WPA* 알고리즘을 이용했을 경우 61.59%임을 알 수 있었다. 즉 WPA* 알고리즘의 불필요한 연산량이 A* 알고리즘에 비해 약 1.62배 적다고 판단할 수 있었다.

[Table 2] Unnecessary computation

	Average	Rate
Dijkstra	10.61	158.14%
A*	6.71	100.00%
HPA*	8.26	123.07%
WPA*	4.13	61.59%



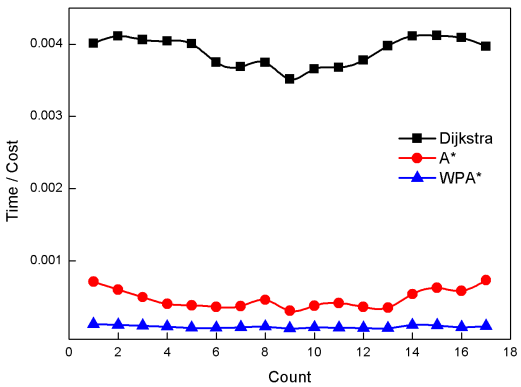
[Fig. 12] Unnecessary computation

[Table 3]과 [Fig. 13]은 경로 탐색에 필요한 시간을 찾은 경로의 비용으로 나눠서 정리한 표인데, 이 값이 곧 실행 시간의 비율이라고 할 수 있다.

이때 A* 알고리즘을 이용했을 경우를 100.0%라고 하면, Dijkstra 알고리즘을 이용했을 경우 132.80%, HPA* 알고리즘을 이용했을 경우 109.96%, WPA* 알고리즘을 이용했을 경우 60.55%임을 알 수 있었다. 즉 WPA* 알고리즘의 실행 시간이 A* 알고리즘에 비해 약 1.65배 짧다고 판단할 수 있었다.

[Table 3] Execution time

	Average	Rate
Dijkstra	16273.11ms	132.80%
A*	12253.79ms	100.00%
HPA*	13473.68ms	109.96%
WPA*	7420.27ms	60.55%



[Fig. 13] Execution time

5. 결론 및 향후 연구

본 논문은 경로 탐색에서 사용할 수 있는 새로운 알고리즘인 가중치 기반의 선수행 A* 알고리즘을 제안하였다. A* 알고리즘에서 휴리스틱 비용을 계산할 때 최적의 가중치를 미리 계산해서 사용하여, 최단 경로가 아닌 노드를 탐색할 확률을 줄여서 더 빠르게 탐색할 수 있게 하였다.

제안한 알고리즘을 이용해 시뮬레이션 프로그램을 만들어서 Dijkstra 알고리즘과 A* 알고리즘, HPA* 알고리즘, WPA* 알고리즘의 성능을 비교했다. 그 결과 WPA* 알고리즘을 이용하는 것이 A* 알고리즘에 비해 불필요한 연산이 줄었다고 판단할 수 있었고, 더 빠르다는 것을 알 수 있었으며, 약 1~2배 효율적이라는 것을 알 수 있었다.

향후 연구로는 WPA* 알고리즘에서 가중치를 미리 계산하여 사용하는 부분을 차용하여, 다른 경로 탐색 알고리즘에 적용할 예정이며, 기존 알고리즘과 성능 비교를 할 예정이다.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funed by the Ministry of Education, Science and Technology(2010-0023184).

REFERENCES

[1] Gyu-Chul Oh, Jong-Hun Park, Uk-Youl Huh, “Global Path Planning using improved A* Algorithm”, CICS ‘11, 2011.

[2] Masoud Nostrati, Ronak Karimi, Hojat Allah Hasanvand, “Investigation of the *(Star) Search Algorithms Characteristics, Methods and Approaches”, World Applied Programming, Vol.2, No.4, 251-256, 2012. 4.

[3] Hemant M. Joshi, Joshua A. McAdams, “Search Algorithms in Intelligent Agents”, Scientific Paper on various search algorithms, 2006. 3.

[4] Se-Il Lee, “Dynamic Programming Algorithm Path-finding for Applying Game”, The Korean Society Of Computer And Information, Vol.10, No.4, 2005. 9.

[5] Jin-Ho Ahn, Min-Ji Park, Sungho Kang, Byungin Moon, “Shortest Path Search Method using A* Algorithm with Priority Queue”, Korean Institute Of Information Technology, Vol.8, No.8, 1-7, 2010. 8.

[6] Kim Pallister, “Game Programming Gems 5”, Infomation Publishing Group, 469-476, 2006.

[7] Anthony Stentz, “Optimal and Efficient Path Planning for Partially-Known Environments”, In Proceedings IEEE International Conference on Robotics and Automation, 1994, 5.

[8] Scott Jacobs, “Game Programming Gems 7”, Infomation Publishing Group, 351-358, 2010.

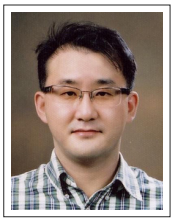
[9] Bjornsson, Yngvi, Enzenberger, Markus, Holte, Robert, Schaeffer, Jonathan. “Fringe Search: Beating A* at Pathfinding on Game Maps”, IEEE Symposium on Computational Itelligence and Games, pp. 125-132, 2005.

[10] Adi Botea, Martin M¨uller, Jonathan Schaeffer, “Near Optimal Hierarchical Path-Finding”, Journal of game development, 2004.
[11] Cazenave Tristan, “Overestimating the Admissible Heuristic of a for Multiple Sequence Alignment”, Computational Intelligence and Bioinformatics and Computational Biology, 2007. CIBCB ′07. IEEE Symposium on, 159-164, 2007. 4.
[12] Mark Deloura, “Game Programming Gems”, Infomation Publishing Group, 340-351, 2001.
[13] Taegkeun Whangbo, “Efficient Bidirectional Search Algorithm for Optimal Route”, Journal of Korea Multimedia Society v.5, n.6, 2002. 12.
[14] Taewon Kim, Kyungeun Cho, Kyhyun Um, “A Hierarchical Graph Structure and Operations for Real-time A* Path finding and Dynamic Graph Problem”, Journal of Korea Game Society, Vol.4, No.3, 2004. 9.
[15] Sung Hyun Cho, “A Pathfinding Algorithm Using Path Infomation”, Journal of Korea Game Society, Vol.13, No.1, 31-40, 2013. 3.
[16] Min-Ji Park, “A Study on the High-Speed Path Search Method using A* Algorithm”, Master Thesis of Hoseo University, 2011.
[17] Seung-Ho Ok, Jin-Ho Ahn, Sungho Kang, Byungin Moon, “A Combined Heuristic Algorithm for Preference-based Shortest Path Search”, IEEK, Vol.47, No.8, 74-84, 2010. 8.

오 민 석 (Min-Seok, Oh)

2012년 호서대학교 게임공학과 (학사)
2012년-(현) 호서대학교 게임학과 (석사)

관심분야 : 게임공학, 프로그래밍



박 성 준(Sung-Jun, Park)

1997년 호서대학교 컴퓨터공학과 (학사)
1999년 건국대학교 컴퓨터공학과 (석사)
2005년 건국대학교 컴퓨터공학과 (박사)
2006년-(현) 호서대학교 게임학과 부교수

관심분야 : 게임공학, 가상현실, HCI, Bioinformatics