한국지능정보시스템학회논문지 제11권 제1호 2005년 6월(pp.153-167)

공간 탐사를 위한 실시간 그래프 탐색

최은미
경기대학교 정보과학부 전자계산학과
(sogmi@kyonggi.ac.kr)

김인철
경기대학교 정보과학부 전자계산학과
(kic@kyonggi.ac.kr)

본 논문에서는 이동 로봇이나 자율 캐릭터 에이전트로 미지의 환경을 탐사하는 문제를 다룬다. 전통적으로 공간 탐사 문제를 해결하기 위한 연구노력들은 주로 그래프기반의 공간 표현법들과 그래프 탐색법들에 초점을 맞추어 왔다. 최근 들어, 공간탐사를 위한 가장 효율적인 그래프 탐색법들 중 최대 min(mn, d^2+m) 에지들만을 탐색하는 EXPLORE 알고리즘이 발견되었다. 이때 d 는 그래프의 부족도(deficiency)를 나타내고, m은 그래프 에지들의 수를, n은 그래프 노드들의 수를 나타낸다. 본 논문에서는 자율 에이전트에 의해 미지의 공간을 탐사하는 실시간 그래프 탐색 알고리즘 DFS-RTA*와 DFS-PHA*를 제안한다. 두 알고리즘들은 모두 EXPLORE 알고리즘과 같이 깊이-우선 탐색(DFS)을 기초로 하고 있으며, 직전 노드로의 빠른 후진을 위해 각각 실시간 최단 경로 탐색 방법인 RTA*와 PHA*를 적용하는 것이 특징이다. 본 논문에서는 대표적인 3차원 온라인 게임 환경인 Unreal Tournament 게임과 지능형 캐릭터 에이전트인 KGBot를 이용한 실험을 통해 두 탐색 알고리즘의 완전성과 효율성을 분석해본다.

논문접수일 : 2005년 4월 게재확정일 : 2005년 6월 교신저자 : 김인철

1. 서론

3차원 게임환경에서 활동하는 캐릭터 에이전트들이나 실세계 공간에서 활동하는 로봇에게 가장 중요한 문제 중의 하나는 어떻게 하면 주어진 미지의 공간을 효과적으로 돌아다니며 이동 경로들을 파악하느냐 하는 공간 탐사 문제(space exploration problem)이다. 인공지능 분야에서는 전통적으로 이 문제를 그래프 기반의 공간 표현법과 그래프 탐색 알고리즘들로 해결하려고 노력해 왔다. 따라서 많은 공간 탐사를 위한 그래프 탐색 알고리즘들이 제안되어 왔으나, 이들은 대부분 양방향으로 이동 가능한 무향 그래프(unidirectional graph)를 가정하거나 실제 특정 노드를 방문하지 않고도 그 노드에 이웃한 노드들을 사전에 모두 알 수 있다고 가정하고 있다. 공간 탐사를 위한 알고리즘에 요구되는 대표적인 성질은 탐색의 완전성(completeness)과 효율성(efficiency)이다. 본 논문에서는 미지의 유향 공간 그래프를 빠짐없이 모두 방문하는 완전성과 중복 방문 노드들의 수를 최소화할 수 있는 실시간 탐색 알고리즘 DFS-RTA* 알고리즘과 DFS-PHA* 일고리즘을 제안한다. 두 알고리즘들은 모두 깊이-우선 탐색(DFS)을 기초로 하고 있으며, 직전 노드로의 빠른

* 이 논문은 산업자원부 지원으로 수행하는 21세기 프론티어 연구개발사업(인간기능 생활지원 지능로봇 기술개발사업)의 일환으로 수행되었습니다.

후진(backtrack) 을 위해 각각 실시간 최단 경로 탐색 방법인 RTA*와 PHA*를 적용하는 것이 특징이다. 본 논문에서는 대표적인 3차원 온라인 게임 환경인 Unreal Tournament 게임과 지능형 캐릭터 에이전트인 KGBot를 이용한 실험을 통해 두 탐색 알고리즘의 완전성과 효율성을 분석해본다.

2장에서는 본 연구와 관련 있는 기존의 공간 탐사 연구들과 최단 경로 탐색 연구들을 정리해보고, 3장에서는 공간 탐사에 적용할 실시간 최단 경로 탐색 알고리즘들인 RTA*와 PHA*의 세부사항을 살펴본다. 4장에서는 깊이-우선 탐색에 기초한 공간 탐사 알고리즘들인 DFS-RTA*와 DFS-PHA*를 제안하고, 5장에서는 두 알고리즘의 구현과 실험에 대해 설명하며, 끝으로 6장에서는 결론과 함께 향후 연구 방향에 대해 기술한다.

2. 관련 연구

2.1 공간 탐사

공간 탐사 문제란 일반적으로 이동 로봇이나 지능형 에이전트가 지형이 복잡한 미지의 공간을 돌아다니며 공간에 대한 완전한 맵(map)을 효과적으로 작성하는 문제를 말한다. 이 문제는 이동 공간에 대한 표현법에 따라 접근법이 크게 달라지는데, 가장 많이 사용되는 이동 공간 표현법으로는 그래프 표현법과 다각형 표현법 등이 있다. 인공지능 상태공간(state space) 표현법에서 유래된 이동점(waypoint)들의 그래프는 현재 주로 3차원 컴퓨터 게임에서 캐릭터 에이전트들의 이동을 위해 많이 이용되고 있으며, 자유공간을 다수의 다각형(polygon)들로 표현하는 다각형 표현법은 복잡한 실외 지형에서 동작하는 이동 로봇을 위해 많이 이용되고 있다.

그래프 표현법에서는 공간을 하나의 유향 그래프(directed graph)로 나타내는데, 이때 각 노드는 하나의 이동점을, 두 노드를 잇는 각 에지(edge)는 두 노드 간에 직접 접근 가능한 경로가 존재함을 표시한다. 공간 탐사 연구에서는 하나의 강 연결 그래프(strongly connected graph)를 가정한다. 강 연결 그래프란 그래프 상의 임의의 두 노드 a와 b에 대해, a에서 b로의 경로뿐만 아니라 b에서 a로의 경로도 존재하는 그래프를 말한다. 일반적으로 공간 탐사에 가장 효율적인 그래프 형태를 오일러 그래프(Eulerian Graph)로 본다. 오일러 그래프는 임의의 한 노드에서 시작하여 모든 에지를 단 한 번씩만 방문하고 다시 그 노드로 돌아올 수 있는 그래프를 말한다. 에지의 수가 m개인 오일러 그래프를 로봇이 탐사하는데는 최대 4m개의 에지만 방문하면 되는 것으로 알려져 있다. Kutten은 임의의 한 그래프가 오일러 그래프와 얼마나 차이가 나는지를 나타내는 척도로 결핍도(deficiency)를 제안하였다. 한 그래프의 결핍도 d는 그 그래프를 오일러 그래프로 만들기 위해 필요한 에지들의 개수를 나타낸다. 따라서 결핍도가 낮은 그래프는 결핍도가 높은 그래프에 비해 보다 효율적으로 탐사할 수 있는 것으로 알려져 있다. Deng과 Papadimitriou는 결핍도가 d인 그래프를 최대 개의 에지들만 방문함으로써 탐사할 수 있는 알고리즘을 제안하였으나, 최근들어 최대 개의 - 여기서 m은 총 에지의 수를, n은 총 노드의 수를 가리킴 - 에지만을 방문하는 더욱 효율적인 알고리즘들이 Albers의 연구(Albers,S. 1997)와 Kwek의 연구(Kwek S. 1997)에 의해 독립적으로 발표되었다. 특히 이들 중에서 Kwek의 알고리즘은 이미 잘 알려져 있는 깊이- 우선 탐색(Depth-First Search, DFS)을 공간 탐사에 적용한 것으로서, 그

래프 이론에 기초한 Albers의 Balance 알고리즘에 비해 이해하기 쉽고, 구현하기 용이한 장점을 가지고 있다.

2.2 최단 경로 탐색

공간 탐사외에 가장 많이 요구되는 로봇의 공간 이동행위는 특정 목표 지점까지 가장 짧은 경로로 이동하는 것이며, 그래프 상에서 이러한 문제를 푸는 것을 최단 경로 탐색(shortest path finding)이라 부른다. 그래프상에서 최단 경로를 탐색하는 방법들은 크게 오프라인 탐색(off-line search)과 온라인 탐색(on-line)으로 나누어 볼 수 있다. 오프라인 탐색방법은 로봇이 그래프상의 각 노드를 직접 방문하지 않아도 그 노드와 에지로 연결된 이웃 노드들을 미리 알 수 있다는 가정에서, 실행 이전에 시작 노드에서 목표 노드까지 전체적인 최단 이동 경로를 계획하는 방식이다. 반면에, 온라인 탐색방법은 그래프상의 각 노드를 직접 방문하지 않고서는 연결 에지들과 이웃 노드들을 알 수 없고 목표 노드까지 전체 이동 경로를 계획할 때까지 실행을 미룰 수 없는 실시간적 제약이 있다는 가정에서, 매 단계마다 진행할 최선의 다음 노드를 계획하고 그 노드로 실제 이동하는 과정을 반복함으로써 목표 노드까지 최단 이동 경로를 찾아내는 방식이다.

대표적인 오프라인 탐색방법으로는 Dijkstra 알고리즘과 A* 알고리즘 등이 있다. 그래프의 크기가 작고 전체 그래프의 연결구조가 인접 행렬(adjacency matrix)이나 인접 리스트(adjacency list) 형태로 미리 주어지는 경우에는 Dijkstra 알고리즘이 매우 효율적이다. Dijkstra 알고리즘(Dijkstra, 1959)은 그래프의 모든 노드와 에지가 컴퓨터 메모리에 이미 저장되어 있어 모든 노드의 접근시간이 일정하게 같다고 가정하고, 그래프상의 임의의 두 노드간의 최단 이동 경로를 찾아준다. 반면에, 그래프의 크기가 매우 커서 전체 그래프를 한꺼번에 메모리에 저장하기 어려운 경우에는 주로 휴우리스틱 탐색방법(heuristic search)이 많이 이용되는데 대표적인 휴우리스틱 탐색방법의 하나가 A* 알고리즘이다. A* 알고리즘(Pearl, J. 1982)은 각 노드 n에 대해 평가치 f(n)=g(n)+h(n)를 계산하여 이 평가치가 최소인 노드를 다음 확장 노드로 선택하는 과정을 반복한다. 즉, A* 알고리즘은 다음 확장 노드를 선택할 때 시작 노드에서 그 노드까지의 최단 거리 추정치와 그 노드에서 목표 노드까지의 최단 거리 추정치를 합한 평가함수를 이용함으로써 일정한 조건하에서는 언제나 목표 노드까지 최단 이동 경로를 찾을 수 있는 효율적인 알고리즘이다.

최근에는 실시간성을 고려한 온라인 최단 경로 탐색방법들에 대한 연구도 활발한데, 대표적인 온라인 탐색 알고리즘으로는 RTA*(Real-Time A*) 알고리즘(Kort, R.E. 1990)과 PHA* 알고리즘(Felner A. 2004)등이 있다. 두 알고리즘 모두 탐색을 통한 경로 계획과 실제 이동을 번갈아 수행하는 방식을 취하고 있어, 부분적으로만 알려진 공간이나 미지의 공간에서 에이전트의 실시간 최단 경로 탐색에 매우 효율적인 것으로 알려져 있다. 한편, 또다른 실시간 최단 경로 탐색방법인 D* (Dynamic A*) 알고리즘(Stentz, A. 1994)은 백포인터(backpointer)를 이용하여 경로를 계획한 다음, 실제 이동을 통하여 새로운 정보를 습득함으로써 경로를 재 계획 한다. D* 알고리즘은 이러한 과정을 반복함으로써 탐색 도중 장애물을 만나더라도 최적의 경로를 보장해 주는 효율적인 알고리즘이다. 하지만 D* 알고리즘은 백포인터를 사용하여 탐색을 전개하기 때문에 각 에지의 양방향성이 보

장 되지 않는 유향 그래프에서는 효과를 보기 어렵다.

3. 실시간 최단 경로 탐색

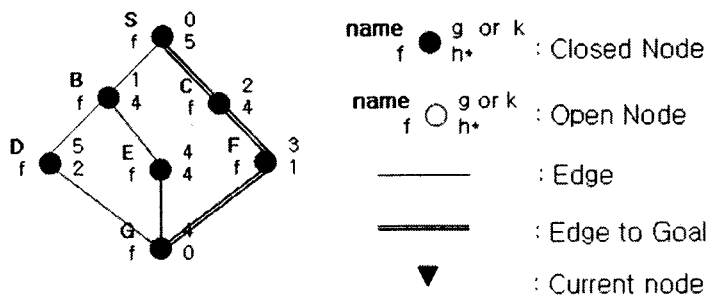
앞서 설명한 바와 같이 실시간 최단 경로 탐색 알고리즘들은 에이전트가 직접 실세계 공간에 놓여 경로탐색과 이동을 동시에 펼치는 온라인 탐색 알고리즘들이다. 이 장에서는 본 논문에서 이용할 대표적인 실시간 최단 경로 탐색 알고리즘들인 RTA*와 PHA*에 대해 자세히 살펴본다.

3.1 RTA* 알고리즘

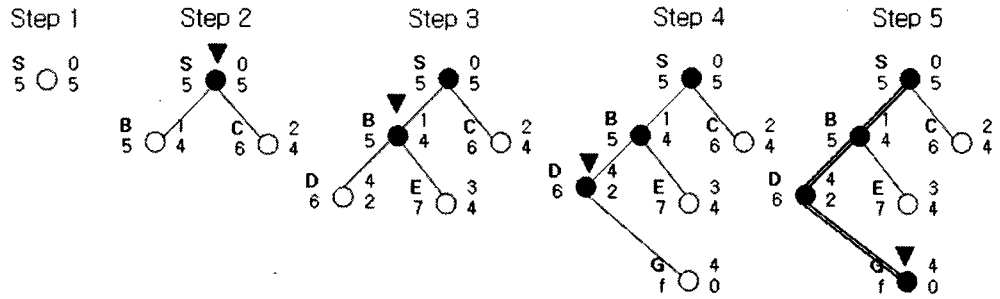
RTA*(Real-Time A*) 알고리즘(Kort.R.E, 1990)은 대표적인 실시간 경로 탐색 알고리즘으로서, 이웃 노드들 중에서 현재 노드로부터 그 노드를 거쳐 목표 노드까지 이르는 추정거리가 가장 짧은 노드를 선택하여 이동하는 과정을 반복한다. 각 이웃 노드 y에 대한 평가함수 f(y)를 다음과 같이 계산한다. f(y) = k(x, y) + h(y). 즉, 현재 노드 x에서 이웃 노드 y까지의 거리 k(x, y)에 이웃 노드 y에서 목표 노드까지의 추정 거리 h(y)를 더한 값을 평가치로 삼아 이러한 평가치가 가장 낮은 이웃 노드를 다음 노드로 선택한다. 이 알고리즘은 매 단계 현재 노드의 이웃 노드만을 대상으로 국지적인 범위내에서 다음 노드를 선택함으로써 출발 노드부터 목표 노드까지의 최단 경로를 보장 하지는 못하나, 빠른 시간 내에 최선 경로(optimal path) 혹은 차선 경로(suboptimal path)를 찾아낼 수 있는 효율적인 탐색 알고리즘 이다.

RTA* 알고리즘의 동작방식을 설명하기 위해 [그림 1]과 같은 예제 그래프를 가정한다. 그림 1에서 시작 노드는 S로, 목표 노드는 G로 표시하였으며, 각 노드에 첨자로 표현된 g(x)값은 시작 노드에서 노드 x까지 거리, k(x)는 현재 노드(역삼각형으로 표시)에서 노드 x까지 거리, h(x)는 노드 x에서 목표 노드까지 추정거리를 각각 나타낸다.

[그림 2]는 [그림 1]의 예제 그래프에 대한 RTA* 알고리즘의 탐색과정을 표현한 것이다. 예컨대, 탐색이 세 번째 단계(step 3)까지 진행되었다고 가정하면, 현재 방문노드는 B이고 이웃한 자식 노드들인 D와 E중에서 하나를 다음 노드로 선택하여야 한다. 이때 노드 D에 대해 k(D)=4, h(D)=2이며, 노드 E에 대해 k(E)=3, h(E)=4이므로, 다음 노드는 D로 선택된다.



[그림 1] 예제 그래프



[그림 2] RTA* 알고리즘을 이용한 탐색 과정

3.2 PHA* 알고리즘

또다른 실시간 최단 경로 탐색 알고리즘인 PHA* 알고리즘(Felner A. 2004)은 A* 알고리즘처럼 그래프 전체를 대상으로 평가함수 f(x)=g(x)+h(x)의 값이 최소인 노드를 다음 방문 노드로 선택하는 방식을 취한다. 이때 g(x)는 시작노드에서 노드 x에 이르는 거리를 나타낸다. 그래프 전체 노드를 대상으로 다음 방문할 노드를 결정하기 때문에 다음 방문할 노드가 현재 위치에서 멀리 떨어지는 경우도 발생하며, 현재 위치에서 방문할 노드까지 실제로 이동해가기 위해서는 이미 방문한 적이 있는 노드들을 중복 방문하기도 한다. 하지만 PHA*알고리즘은 A* 알고리즘과 같이 언제나 그래프 전역의 노드를 선택 대상으로 삼을 뿐 아니라 평가치도 시작 노드에서 해당 노드까지의 거리를 고려함으로써, 최적의 경로 탐색을 보장한다.

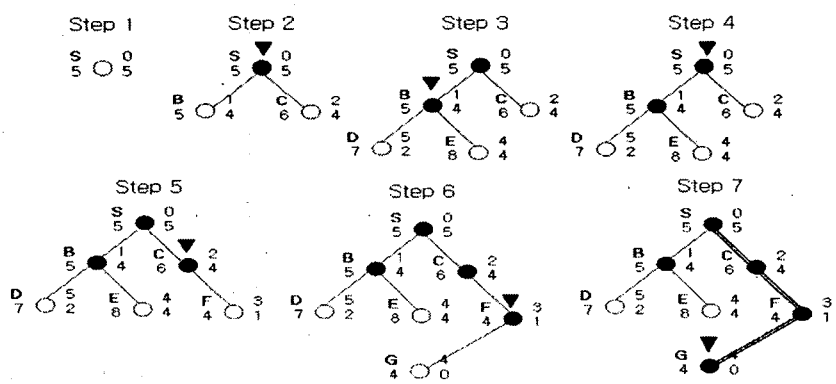
PHA* 알고리즘은 일반적으로 두 레벨로 구성되며, 상위 레벨에서는 A* 알고리즘과 동일하게 전역적 관점에서 평가치 f(x)가 가장 낮은 노드를 다음 방문 노드로 선택하는 일을 담당하고, 하위 레벨에서는 선택된 노드까지 실제로 이동해가는 일을 담당한다. 그리고 상위 레벨에서 선택된 노드까지 효과적으로 이동해가기 위해서는 다시 하위 레벨에서 다양한 탐색방법을 적용할 수 있다. 하위 레벨에 적용 가능한 탐색방법들에는 이미 상위 레벨의 탐색을 통해 알려져 있는 부분 그래프(partial graph)상의 최단 경로를 이용하는 방법과 새로운 노드들에 대한 탐색을 포함하는 깊이-우선 탐색(Depth-First Search, DFS) 방법들이 있다. 하위 레벨의 깊이-우선 탐색방법들은 에이전트가 전통적인 깊이-우선 탐색방식에 따라 막다른 종점(deadend)을 만나거나 선택된 목표 노드에 도달할 때까지 매 단계 현재 노드의 이웃 노드 중 아직 방문하지 않은 비방문 노드를 하나씩 선택하여 앞으로 앞으로 이동해가는 것을 의미한다. 이러한 하위 레벨의 깊이-우선 탐색도 매 단계 이웃한 비방문 노드들 중 어떤 노드를 우선적으로 선택하느냐 하는 여러 가지 전략이 있을 수 있으며, 그 중에서 현재 노드에서 이웃 노드까지의 거리와 이웃 노드에서 목표 노드까지의 거리의 합이 최소가 되는 노드를 우선적으로 선택하는 A*DFS가 비교적 더 우수한 것으로 알려져 있다.

[그림 3]은 앞서 소개한 [그림 1]의 예제 그래프에 대한 PHA* 알고리즘의 탐색과정을 보여주고 있다. 탐색이 세번째 단계(step 3)까지 진행되었다고 가정하면, 현재 노드는 역시 B이고 선택 가능한 노드들의 집합은 새로운 자식 노드들인 D와 E 뿐만 아니라 아직 방문 하지 않은 조상 노드인 C도 포함한다. 이때, 노드 C의 평가치는 f(C) = g(C) + h(C)=2+4=6이고, 노드 D의 평가치는 f(D)= g(D)+h(D) = 5+2=7이며, 노드 E의 평가치는 f(E)=g(E)+h(E) = 4+4 = 8이므로, 최소 평가치를 가지는 노드 C가 다음 노드로 선택 된다. 따라서 에이전트는 하위 레벨의 탐색을 통해 현재 노드 B에서 다시 시작 노드 S를 거쳐(step 4) 다음 노드인 C로 이동하여야 한다(step 5). 참고로, [그림 1]의 동일한 예제 그래프에 대한 두 알고리즘 RTA*와 PHA*의 탐색결과를 비교해보면 다음과 같다. RTA* 알고리즘은 S-B-D-G의 실제 경로 이동을 통해 시작 노드에서 목표 노드에 이르는 경로비용이 7인 차선 경로 S-B-D-G를 찾아내었고, 반면에 PHA*는 S-B-S-C-F-G의 실제 경로 이동을 통해 경로비용이 6인 최단 경로 S-C-F-G를 찾아내었다.

4. 공간 탐사를 위한 그래프 탐색

4.1 문제 정의

유향 그래프를 이용한 미지의 공간 탐사 문제는 다음과 같이 정의 할 수 있다. 한 에이전트는 현재 노드에서 출력 에지(outgoing edge)들 중 하나를 선택하고 이것을 방문하는 과정을 반복함으로써, 하나의 강 연결 유향 그래프(strongly connected directed graph)로 표현된 미지의 공간을 탐사하여야 한다. 이때, 에이전트는 자신이 한번이라도 방문한 적이 있는 모든 에지와 노드들은 기억할 수 있으며, 따라서 그들을 재방문할 경우에는 재방문 사실을 스스로 인식할 수 있다. 하지만 에이전트는 그래프상에 얼마나 많은 수의 노드들과 에지들이 존재하는지 미리 알지 못하고, 또 아직 방문하지 않은 각 에지들이 어떤 노드들과 연결되는지 알지 못한다. 에이전트가 그래프상의 모든 에지들(과 모든 노드들)을 방문하였을 때, 우리는 이 에이전트가 그래프로 표현된 미지의 공간을 모두 탐사하였다고 말한다. 에이전트의 목표는 가능한 최소의 에지와 노드들을 방문함으로써 주어진 그래프 공



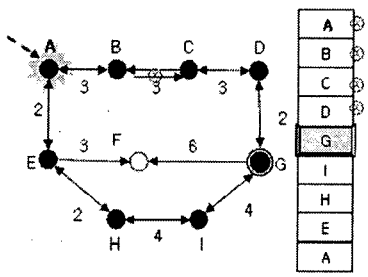
[그림 3] PHA* 알고리즘을 이용한 탐색 과정

간을 모두 탐사하는 것이다.

4.2 SimpleDFS 알고리즘

SimpleDFS 알고리즘은 무정보 그래프 탐색(uninformed graph search) 알고리즘인 깊이-우선 탐색(Depth First Search, DFS)을 미지의 공간 탐사를 위해 적용한 하나의 공간 탐사 알고리즘이다. 원래 깊이-우선 탐색은 탐색에 특별한 휴우리스틱(heuristic)을 이용할 수 없기 때문에 목표 노드를 만날 때까지는 체계적으로 모든 노드들을 다 방문해보는 일종의 전역 탐색(exhaustive search)의 성질을 가진다. 하지만 잘 알려진 바대로 깊이-우선 탐색(DFS)은 넓이-우선 탐색(Breadth First Search, BFS)과 같은 다른 전역 탐색 알고리즘들에 비해 탐색 중에 기억하고 있어야 하는 노드 수가 적고 중복 방문 노드의 수도 적어 매우 효율적이다. 따라서 이러한 특징을 지닌 DFS를 전체 맵(map)이 알려져 있지 않은 미지의 공간 탐사에 적용하면 효과를 얻을 수 있다.

SimpleDFS는 기본적으로 깊이-우선 탐색방법에 따라 탐색을 전개하되, 이웃한 비방문 노드들 중 임의의 노드를 다음 노드로 선택하지 않고, 현재 노드로부터 가장 가까운 근거리에 있는 이웃 노드를 우선 선택한다. 또 SimpleDFS는 실세계 공간의 특성을 반영하여 지형상 현재 노드에서 직전 노드로 바로 되돌아 갈 수 없는 경우, 다른 경로로 우회해서라도 가능한 최단 경로로 직전 노드까지 되돌아 간다는 점이 큰 특징이다. SimpleDFS의 탐색과정 중 만약 비방문 이웃 노드가 존재하지 않은 경우에는 현재 노드에서 더 이상 탐색을 계속할 수 없다. 우리는 이와 같은 상황을 "STUCK"이라 부르고, STUCK이 발생하면 방문 기록을 보관하고 있는 스택(stack)으로부터 되돌아갈 노드를 찾아낸다. 되돌아갈 노드는 아직 비방문 이웃 노드(unvisited neighbor node)를 포함하고 있는 가장 최근 방문 노드가 된다. 일단 되돌아 갈 노드가 선택되었으면 그 노드까지 가장 짧은 경로를 통해 이동한다. 후진이 성공하였으면 그 노드로부터 SimpleDFS 탐색은 계속된다. 만약 스택의 모든 노드를 꺼내도 비방문 이웃 노드를 가진 노드를 발견할 수 없다면, 이때는 이미 그래프 상의 모든 노드를 다 방문한 것으로 판단하고 알고리즘을 종료한다. 이와 같은 SimpleDFS알고리즘은 미지의 공간 탐사를 위해Kwek이 제안한 원래의 깊이-우선 탐색 알고리즘에 현재 노드에 가장 가까운 이웃 비방문 노드를 우선 선택하는 전략을 적용하여 보다 구체화한 세부 알고리즘으로 볼 수 있다. 따라서 SimpleDFS알고리즘 역시 총 에지의 수가 m개인 그래프를 탐사하는데 최대 개의 에지만을 방문하면 되는 높은 효율성을 지니



[그림 4] SimpleDFS 알고리즘의 탐색 예

고 있다.

[그림 4]는 SimpleDFS에 따라 그래프를 탐색하던 중에 STUCK이 발생한 상황을 나타내고 있다. A노드를 출발하여 A-E-H-I-G-D-C를 거쳐 다시 A노드에 도착 하였을 때 STUCK이 발생하게 된다. 이때 스택에 저장된 노드들을 하나씩 차례대로 꺼내어 비방문 이웃 노드를 적어도 하나 가지고 있는지 체크한다. 예컨대, A, B, C, D 노드들은 각각의 출력 에지에 연결된 모든 이웃 노드들을 이미 다 방문한 상태이기 때문에 후진 노드로 고려하지 않는다. 대신 스택의 그 다음 노드인 G 노드의 경우, 아직 방문하지 않은 이웃 노드인 F를 포함하고 있으므로, G를 후진 노드로 선택한다. 따라서 STUCK이 발생한 현재 노드B에서 출발하여 후진 노드 G까지 최단 경로 탐색 알고리즘을 적용하여 이동한다. [그림 5]는 이와 같은 SimpleDFS 알고리즘을 요약한 것이다.

```
Function : simpleDFS(S_NODE)
C_NODE = S_NODE   /* start node */
N_NODE, Y_NODE = null
do
  while(exhausted(C_NODE) != true)
    do
    N_NODE = selectNextNode(C_NODE)
    advanceTo(N_NODE)
    push( HISTORY, C_NODE ) /* history stack */
    C_NODE = N_NODE
    end
  do
    Y_NODE = pop(HISTORY)
  until((exhausted(Y_NODE) != true) or
      (empty(HISTORY) = true))
  if (exhausted(Y_NODE) != true),
    moveShortestPath(C_NODE, Y_NODE)
   C_NODE = Y_NODE
until (empty(HISTORY) = true)

Function : exhausted(C_NODE )
for each neighbor Y_NODE of C_NODE
  if unvisited(Y_NODE), return false
return true

Function : selectNextNode(C_NODE)
f = 10000   /* a large value */
for each neighbor Y_NODE of C_NODE
   if distance(C_NODE, Y_NODE) < f,
     B_NODE = Y_NODE
     f = distance(C_NODE, Y_NODE)
return B_NODE
```

[그림 5] SimpleDFS 알고리즘

4.3 DFS-RTA* 와 DFS-PHA* 알고리즘

앞서 소개한 공간 탐사를 위한 SimpleDFS 알고리즘에서 STUCK이 발생할 때마다 후진 노드까지 어떤 방법으로 되돌아가느냐 하는 결정은 SimpleDFS에 의한 공간 탐사 과정 전체의 효율성에 큰 영향을 미친다. 후진 노드까지의 이동 경로 탐색을 위해 다시 깊이-우선 탐색(DFS), 넓이-우선 탐색(BFS), 그리고 반복적 깊이-우선 탐색(Iterative Deepening Search, IDS) 등의 다양한 탐색방법을 적용할 수 있다. 하지만 실시간 제약과 탐색의 효율성을 고려한다면 A* 알고리즘에 기초한 실시간 최단 경로 탐색방법을 적용하는 것이 바람직 할 것이다.

본 논문에서 제안하는 공간 탐사 알고리즘들인 DFS-RTA*와 DFS-PHA*는 SimpleDFS알고리즘을 기초로 하되, 후진 노드(bactrack node)까지 빠른 이동을 위해 실시간 최단 경로 탐색 알고리즘인 RTA*와 PHA*를 각각 적용한 알고리즘들이다. [그림 6]과 [그림 7]은 각각 실시간 최단 경로 탐색 알고리즘인 RTA*와 PHA*를 표현한 함수들로서, [그림 5]의 SimpleDFS 알고리즘에서 후진 노드까지 최단 경로로 이동하기 위해 호출하는 추상함수인 moveShorestPath(C_NODE, Y_NODE)

```
Function : RTA(S_NODE, G_NODE)
C_NODE = S_NODE ; best = 0
do
  for each neighbor Y_NODE of C_NODE
    f = h(Y_NODE,G_NODE) + k(C_NODE, Y_NODE)
    if f < best, B_NODE = Y_NODE ; best = f
  advanceTo(B_NODE)
  C_NODE = B_NODE
until C_NODE = G_NODE
```

[그림 6] RTA* 알고리즘

```
Function : PHA(S_NODE, G_NODE)
C_NODE = S_NODE ; best = 0
do
 N_NODE = best node from OPEN_LIST
 if N_NODE = explored
   lowerLevel(C_NODE , N_NODE)
   C_NODE = N_NODE
until C_NODE = G_NODE

Function : lowerLevel(C_NODE, G_NODE)
S_NODE = C_NODE
f = 0
do
  for each neighbor Y_NODE of C_NODE
   f = g(Y_NODE) + h(Y_NODE,G_NODE)
    if f < best, B_NODE = Y_NODE ; best = f
  moveTo(B_NODE)
  C_NODE = B_NODE
until C_NODE = G_NODE
```

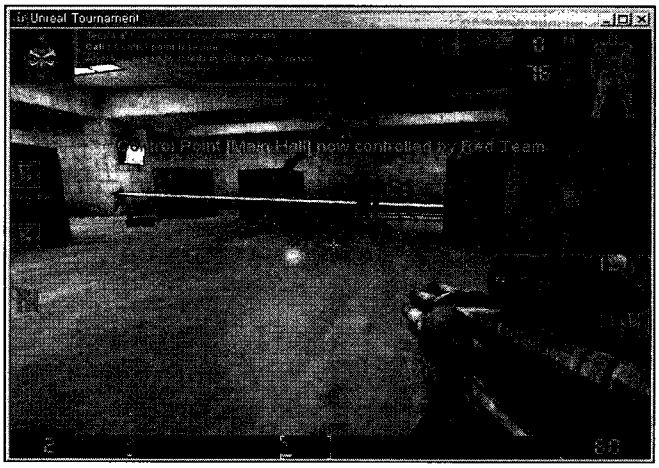
[그림 7] PHA* 알고리즘

를 대신해 사용된다. 전체적으로 탐색의 효율을 높이기 위해 DFS-RTA*와 DFS-PHA*에서는 후진 노드까지의 이동을 위한 하위 레벨의RTA* 및 PHA* 탐색동안 발견된 새로운 노드들과 방문 노드들을 상위 레벨의 후속 DFS 탐색에서도 이용할 수 있도록 설계하였다.

5. 구현 및 실험

5.1 구현

본 논문에서 제안한 공간 탐사 알고리즘들은 복잡한 3차원 공간에서 활동하는 지능형 캐릭터 에이전트 KGBot의 공간 탐사 행위 구현을 위해 적용되었다. KGBot은 Unreal Tournament 게임과 Gamebots 시스템 환경에서 사람을 대신해 자율적으로 캐릭터를 제어하도록 개발된 하나의 보트 클라이언트(bot client)이다. [그림 8]은 일인칭 슈팅(First-Person Shooting) 게임인Unreal Tournament게임의 한 화면을 보여주고, [그림 9]는 클라이언트형 보트의 연동을 지원하기 위해 UT게임을 확장한 Gamebots시스템의 구조를 보여준다(Adobbati, R. 2001). KGBot는 동적 환경에서 에이전트의 빠른 반응적 행위와 복잡한 목표-지향적 행위들을 모두 효과적으로 표현하고 실행할 수 있는 에이전트 구조 CAA를 이용하여 개발되었다(김인철, 2003). CAA에서 각 행위는 적용 전조건(precondition), 실행 몸체(body), 만족해야 하는

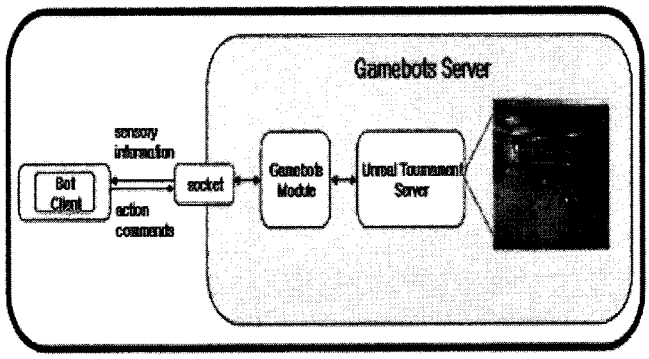


[그림 8] Unreal Tournament 게임환경

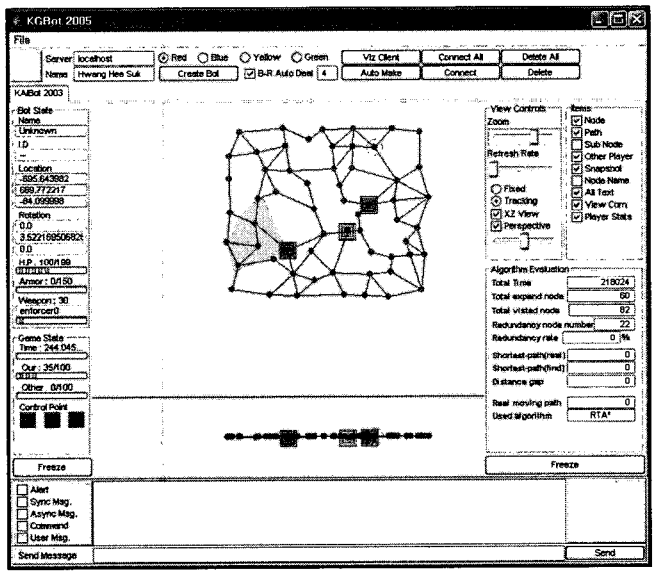
문맥조건(context condition) 등으로 표현되며, 비록 전조건이 만족되어 수행을 시작한 행위라도 수행 중 환경 변화에 의해 문맥조건을 더 이상 만족하지 못하면 언제든지 수행을 멈추고 그때 상황에 적합한 새로운 행위로 실행 전환이 일어나게 된다. KGBot의 공간 탐사 행위도 이러한 CAA의 한 행위로 구현되었으며, DFS-RTA* 및 DFS-PHA* 알고리즘은 공간 탐사 행위의 주된 몸체부분을 이룬다. [그림 10]은 이렇게 구현된 KGBot가 미지의 공간을 효과적으로 탐사하고 있는 모습을 그래픽 사용자 인터페이스를 통해 보여주는 화면이다.

5.2 실험

Unreal Tournament 게임의 3차원 맵과 Gamebots 시스템, 그리고 그 위에서는 동작하는



[그림 9] Gamebots 서버와 보트 클라이언트



[그림 10] 공간 탐사 중인 KGBot의 실행화면

지능형 캐릭터 에이전트인 KGBot를 이용하여, 본 논문에서 제안한 DFS-RTA*와 DFS-PHA*의 완전성과 효율성을 분석하기 위한 실험을 수행 하였다. DFS-RTA* 및 DFS-PHA*와의 비교를 위해 별도의 공간 탐사 알고리즘인 DFS-DFS를 구현하고 실험에 이용하였다. DFS-DFS 알고리즘은 SimpleDFS 알고리즘에서 후진 노드까지의 이동을 위해 또다시 깊이-우선 탐색(DFS)을 적용하는 공간 탐사 알고리즘이다. 또한, 실험을 위해 총 노드의 수가 다른 4 개의 3차원 맵을 준비하였다. 4개의 맵은 각각 90, 120, 150, 180개의 노드를 포함하고 있으며, Unreal Tournament 게임에서 사용되는 맵의 단위로 가로 세로의 길이가 100*100인 사각형 안에 평균 10개의 노드가 포함 되도록 구성 하였다. 이는 노드들이 너무 한 곳에 집중적으로 분포되거나, 혹은 너무 멀리 떨어져 분포하지 않도록 방지하기 위함이다. 따라서 본 실험에서는 노드 수가 많다는 것은 곧 맵의 크기가 크다는 것을 의미하기 때문에 좁은 공간를 탐사 할 때와 넓은 공간를 탐사 할 때를 비교해 볼 수도 있다. 실험에 사용되는 3차원 공간 맵은 몇 개의 복층으로 구성되거나 다양한 장애물과 함정 그리고 낭떠러지들을 포함할 수 있기 때문에 노드간의 에지가 단방향성을 가질 수도 있다는 점을 고려하면서, 그래프 상의 임의의 두 노드간에 양방향 도달 가능한 경로가 존재하도록 각 맵을 구성하였다. 또, 공간 탐사는 어느 위치에서부터 시작하느냐에 따라 실험 결과에 큰 영향을 미칠 수 있기 때문에 각 맵마다 5 개의 서로 다른 시작 위치를 잡아서 실험하였다. 전체 실험은 각기 다른 맵과 각기 다른 시작 위치에 대해 알고리즘별로 20회씩 총 60회의 단위 실험을 수행하는 방식으로 진행되었다. 그리고 각 단위실험을 통해 탐색과정 동안의 총 소요시간, 총 방문 노드수, 총 이동거리 등을 측정하여 서로 비교하였다.

총 소요시간은 시작 노드에서 출발하여 모든 에지들과 노드들을 다 방문한 시점까지 소요된 총 시간으로서, 이 시간에는 캐릭터 에이전트가 각 노드간을 실제로 이동하는데 소요된 시간과 함께 방문한 각 노드에서 이웃한 노드들을 모두 파악하기 위해 몸을 360도 회전하는데 필요한 지연시간, 그리고 다음 방문 노드를 계획하는데 걸린 계산 시간 등이 모두 포함되어 있다. 총 방문 노드수는 시작 노드에서 출발하여 탐사를 종료할 때까지 방문한 모든 노드들을 계산한 것으로서, 이것은 SimpleDFS 알고리즘을 이용한 탐사과정 동안 방문한 노드의 수와 RTA* 및 PHA* 를 이용한 실시간 경로 탐색과정 동안 방문한 노드의 수를 모두 포함한다. 마지막으로 총 이동거리는 시작 노드에서 출발하여 탐사를 완료할 때까지 캐릭터 에이전트가 실제로 이동한 거리의 합이다.

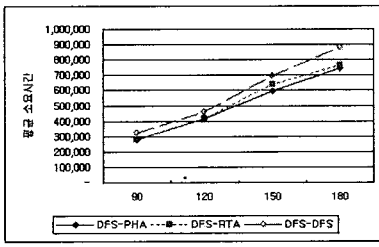
<표 1>은 총 노드수가 90, 120, 150, 180개로 서로 크기가 다른 4개의 맵을 이용하여 각 5회씩 서로 다른 시작노드를 기초로 실험한 결과와 맵 별로 평균을 계산한 결과를 보여주고 있다. 또한 DFS-PHA*와 DFS-RTA* 그리고 DFS-DFS를 시작노드 기준으로 총 소요시간, 총 방문 노드 수, 총 이동거리로 구분하여 보여주고 있다.

[그림 11]과 [그림 12], 그리고 [그림 13]은 맵 크기에 따른 평균 소요시간, 평균 방문 노드수, 평균 이동거리를 각각 알고리즘 별로 비교하여 보여주고 있다.

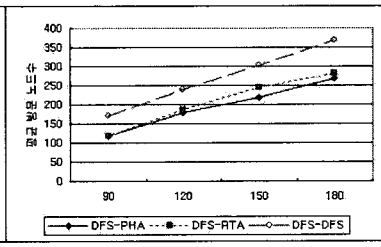
실험결과를 살펴보면, 실험에 사용된 세 가지 알고리즘 모두 주어진 맵상의 모든 노드를 빠짐없이 다 방문하는 완전성을 보였으나, 탐색 효율성 면에서는 약간의 차이를 보였다. 전체적으로 모든 비교 척도에서 본 논문에서 제안한 DFS-RTA*와 DFS-PHA*가 DFS-DFS에 비해 우수한 결과를

<표 1> 실험 결과

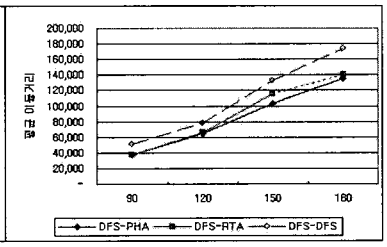
실험 횟수	총노드수	시작노드	총 소요시간			총 방문 노드수			총 이동거리		
			DFS-PHA	DFS-RTA	DFS-DFS	DFS-PHA	DFS-RTA	DFS-DFS	DFS-PHA	DFS-RTA	DFS-DFS
1	90	PathNode23	273,924	284,399	332,287	116	118	180	36,481	37,133	54,023
2	90	PathNode148	276,367	276,157	296,607	117	118	151	36,571	37,111	44,218
3	90	PathNode160	285,661	286,983	323,845	120	120	180	37,589	37,589	53,389
4	90	PathNode201	272,672	272,071	313,000	114	116	164	35,726	36,266	50,424
5	90	PathNode176	278,981	285,581	337,855	122	122	181	38,277	38,074	55,015
평 균			277,521	281,038	320,719	118	119	171	36,929	37,235	51,414
6	120	PathNode23	415,287	429,087	465,109	181	197	248	63,911	69,269	81,006
7	120	PathNode50	401,968	408,428	456,536	166	172	232	60,693	62,550	77,244
8	120	PathNode100	421,166	422,027	450,748	181	187	236	66,044	66,523	76,801
9	120	PathNode56	400,386	411,221	461,253	176	177	242	61,385	63,460	78,037
10	120	PathNode88	431,530	422,427	461,484	191	195	242	68,492	68,324	78,401
평 균			414,067	418,638	459,026	179	186	240	64,105	66,025	78,298
11	150	PathNode23	581,166	609,637	710,131	211	231	306	99,556	107,240	135,892
12	150	PathNode113	598,620	665,777	700,788	219	263	306	104,368	123,989	135,656
13	150	PathNode2	617,158	642,554	696,592	235	249	312	109,332	114,609	135,253
14	150	PathNode64	571,522	630,377	680,178	206	242	288	98,424	114,188	130,213
15	150	PathNode80	598,881	637,126	675,351	218	244	298	104,235	116,531	129,562
평 균			593,469	637,094	692,608	218	246	302	103,183	115,311	133,315
16	180	PathNode23	753,183	779,541	894,085	271	291	373	135,863	143,701	175,889
17	180	PathNode64	776,537	787,472	860,788	274	282	355	141,307	143,288	169,324
18	180	PathNode80	715,067	726,625	885,893	256	260	372	129,352	130,131	171,471
19	180	PathNode113	779,170	815,433	890,971	280	311	373	143,983	154,600	177,033
20	180	PathNode120	711,372	715,929	882,829	255	258	373	128,594	128,969	174,389
평 균			747,066	765,000	882,913	267	280	369	135,820	140,138	173,621



[그림 11] 평균 소요시간 비교



[그림 12] 평균 방문 노드수 비교



[그림 13] 평균 이동거리 비교

보인다는 것을 확인 할 수 있다. 또한, 평균적으로 DFS-PHA* 알고리즘이 DFS-RTA* 의 결과보다 우수함을 알 수 있다. 이는 경우에 따라 후진 탐색 시 DFS-RTA* 알고리즘이 최적의 경로를 찾는데 실패한 것이 원인이 된 차이점이라는 것을 알 수 있었다. 반면에 PHA* 알고리즘은 비록 후진 탐색 시 초기에는 DFS-RTA* 보다 좀 더 많은 노드를 방문하지만 언제나 후진 노드까지 최단 경로를 찾을 수 있어 전체적으로는 더 효율적인 후진 탐색이 가능했다. 이것은 DFS-RTA*에 비해 좀더 많은 계산량을 요구하는 DFS-PHA* 알고리즘의 단점을 상쇄하는 장점으로 파악된다.

6. 결론

본 논문에서는 자율 에이전트에 의해 미지의 공간을 탐사하는 실시간 그래프 탐색 알고리즘인 DFS-RTA*와 DFS-PHA*를 제안하였다. 그리고 복잡한 3차원 공간 맵을 가진 온라인 게임 환경에서 다양한 실험을 전개하여 두 알고리즘의 완전성과 효율성을 분석 하였다. 실험에서 두 알고리즘은 모두 주어진 맵의 모든 노드들과 에지들을 빠짐없이 탐색하는 완전성을 보였고, 후진 탐색을 위해 깊이-우선 탐색(DFS)를 적용하는 DFS-DFS 알고리즘과의 비교에서도 보다 높은 성능을 확인할 수 있었다. 또한 제안한 두 알고리즘 중에서도 DFS-PHA* 알고리즘이 DFS-RTA* 알고리즘보다 소요시간, 방문 노드수, 이동거리 면에서 평균적으로 우수한 결과를 보여 주었다.

본 논문에서 제안한 공간 탐사 알고리즘들은 향후 다중 에이전트에 의한 분산 공간 탐사 알고리즘으로 확장할 경우 매우 효과가 있으리라 판단된다. 다중 로봇에 의한 분산 공간 탐사에 관한 기존의 연구들은 대부분 각 개별 로봇에는 나름의 자율 공간 탐사 알고리즘을 두지 않는 대신 각 로봇의 개별 이동 동작을 중앙에서 하나 하나 제어하는 방식으로 로봇들간의 행위를 조정하여 왔다. 본 본문에서 제안한 개별 에이전트의 자율 공간 탐사 행위를 기초로 보다 유연한 분산 공간 탐사 메커니즘을 고안하는 것이 가능하리라 판단한다.

참고문헌

[1] Adobbati, R., et al. "Gamebots : A 3D Virtual World Test-bed for Multi-Agent Research", *Proceedings of Agents-01 Workshop on Infrastructure for Agents*, MAS, and Scalable MAS, Montreal, Canada. (2001).

[2] Albers, S. and Henzinger M. "Exploring Unknown Environments", *Proceedings the 29th Annual ACM Symposium on Theory Computing*, (1997), 416-425.

[3] Bender, M., Fernandez, A., Ron, D., Sahai, A., and Vadhan, S. "The Power of a Pebble: Exploring and Mapping Directed Graphs", *Proceedings of STOC-98*, (1998), 269-278

[4] Deng, X., Kameda, T., and Papadimitriou, C. "How to Learn in an Unknown Environment", *Proceedings of the 32nd Symposium on the Foundations of Computer Science*, (1991), 298-303.

[5] Dijkstra, E. "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, Vol.1, (1959), 269-271.

[6] Felner A., Stern R., Kraus S., Ben-Yair A., Netanyahu N.S. "PHA*: Finding the Shortest Path with A* in An Unknown Physical Environment", *Journal of Artificial Intelligence Research (JAIR)*, Vol.21 (2004), 631-670

[7] Kitamura, Y., Teranish, K., and Tatsumi, S. "Organizational Strategies for Multiagent Real-time Search", *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-96)*, (1996), 150-156

[8] Knight, K. "Are Many Reactive Agents Better Than a Few Deliberative Ones", *Proceedings of IJCAI-93*, (1993), 432-437

[9] Korf, R.E. "Real-time Heuristic Search", *Artificial Intelligence*, Vol.42, No.3,(1990), 189-211

[10] Kwek, S. "On a simple Depth-First Search Strategy for exploring Unknown Graphs",

Proceedings of the 5th International Workshop in Algorithms and Data Structures, LNCS. 1272, (1997), 345-353

[11] Pearl, J. and Kim, J.E. "Studies in Semi-Admissible Heuristics", *IEEE Transaction on PAMI*, Vol.4, No.201, (1982), 392-400

[12] Stentz, A. "Optimal and Efficient Path Planning for Partially-known Environments", *Proceedings of IEEE International Conference on Robotics and Automation*, (1994), 3310-3317

[13] Stern R. "*Optimal Path Search in Unknown Physical Environments*", M.Sc Thesis, CS Dept., Bar-Ilan University, Israel, (2001).

[14] Yokoo, M. and Kitamura, Y. "Multiagent Real-time A* with Selection: Introducing Competition in Cooperative Search", *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-96)*, (1996), 409-416

[15] 김인철."3차원 가상환경에서 동작하는 지능형 에이전트의 구조와 경로 찾기 행위", *정보처리학회논문지 B.* 제10-B권 제1호, (2003), 1-12

[16] 이만재. "게임에서의 인공지능 기술". *정보처리학회지*, 제9권 제3호, (2002), 69-76

Abstract

Real-time Graph Search for Space Exploration

Eunmi Choi* · Incheol Kim*

In this paper, we consider the problem of exploring unknown environments with a mobile robot or an autonomous character agent. Traditionally, research efforts to address the space exploration problem have focused on the graph-based space representations and the graph search algorithms. Recently EXPLORE, one of the most efficient search algorithms, has been discovered. It traverses at most min(mn, d^2+m) edges where d is the deficiency of a graph, m is the number of edges and n is the number of vertices. In this paper, we propose DFS-RTA* and DFS-PHA*, two real-time graph search algorithms for directing an autonomous agent to explore in an unknown space. These algorithms are all built upon the simple depth-first search (DFS) like EXPLORE. However, they adopt different real-time shortest path-finding methods for fast backtracking to the latest node, RTA* and PHA*, respectively. Through some experiments using *Unreal Tournament*, a 3D online game environment, and *KGBot*, an intelligent character agent, we analyze completeness and efficiency of two algorithms.

Key words : Space Exploration; Real-time Graph Search; RTA* Algorithm; PHA* Algorithm; Depth-First Search

* Department of Computer Science, Kyonggi University