2014년 10월 전자공학회 논문지 제 51권 제 10 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 10, October 2014 http://dx.doi.org/10.5573/ieie.2014.51.10.109

논문 2014-51-10-15

내비게이션 경로설정에서 최단거리경로 탐색을 위한 A*와 Dijkstra 알고리즘의 하이브리드 검색법

(A Hybrid Search Method of A* and Dijkstra Algorithms to Find Minimal Path Lengths for Navigation Route Planning)

이 용 후*, 김 상 운*

(Yong-Hu Lee and Sang-Woon Kim©)

요 약

내비게이션 경로탐색 시스템에서 A* 알고리즘을 사용할 경우 경로거리가 멀수록 Open 리스트(최적의 경로를 선택하기 위해 탐색된 예비경로들의 집합)의 크기가 증가하며, 이로 인해 비교연산이 증가하게 된다. 본 논문에서는 Dijkstra의 알고리즘과 A* 알고리즘을 주기적으로 교체 적용하여 Open 리스트의 크기를 줄일 수 있는 검색 방법을 제안한다. 여기서 두 알고리즘을 교체 적용하기 위하여 $Level$이라는 이름의 파라미터를 사용한다. 미리 정해진 레벨(깊이)만큼 Dijkstra의 알고리즘으로 탐색한 다음 A* 알고리즘으로 교체되도록 한다. 이 때 Dijkstra 알고리즘의 Open 리스트에 있는 노드들을 A* 알고리즘의 평가함수로 적합도를 평가하여 가능성이 있는 노드만을 A* 알고리즘의 Open 리스트로 전달한다. 따라서 계속되는 검색과정에서 Open 리스트의 크기가 불필요하게 증가되는 것을 억제할 수 있다. 또한 Dijkstra와 A* 알고리즘을 번갈아 적용하기 때문에 A* 알고리즘으로는 찾지 못할 최적 또는 준 최적 경로를 Dijkstra의 알고리즘으로 탐색한 결과와 비슷한 수준으로 찾을 수 있게 된다. 제안한 하이브리드 검색 알고리즘을 인공 및 실제의 지도 데이터를 이용하며 실험한 결과, 기존의 탐색 알고리즘과 비슷한 수준의 최단경로거리를 유지하면서 비교연산의 수를 더 줄일 수 있었다. 이 실험에서는 Level 값은 임의로 선정하였다. 따라서 실제의 도로 상황에서 최적 Level 값을 자동 선정하는 연구는 앞으로의 과제이다.

Abstract

In navigation route planning systems using A* algorithms, the cardinality of an Open list, which is a list of candidate nodes through which a terminal node can be accessed, increases as the path length increases. In this paper, a method of alternately utilizing the Dijkstra's algorithm and the A* algorithm to reduce the cardinality of the Open list is investigated. In particular, by employing a depth parameter, named Level, the two algorithms are alternately performed depending on the Level's value. Using the hybrid searching approach, the Open list constructed in the Dijkstra's algorithm is transferred into the Open list of the A* algorithm, and consequently, the unconstricted increase of the cardinality of the Open list of the former algorithm can be avoided and controlled appropriately. In addition, an optimal or nearly optimal path similar to the Dijkstra's route, but not available in the A* algorithm, can be found. The experimental results, obtained with synthetic and real-life benchmark data, demonstrate that the computational cost, measured with the number of nodes to be compared, was remarkably reduced compared to the traditional searching algorithms, while maintaining the similar distance to those of the latter algorithms. Here, the values of Level were empirically selected. Thus, a study on finding the optimal Level values, while taking into consideration the actual road conditions remains open.

Keywords : 최단거리경로검색, Dijkstra 알고리즘, A* 알고리즘, 하이브리드 검색 알고리즘

* 정회원, 명지대학교 컴퓨터공학과(Department of Computer Engineering, Myongji University)
© Corresponding Author(E-mail: kimsw@mju.ac.kr)
※ 이 논문(제2저자)은 2012년 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업의 일부임(NRF-2012R1A1A2041661)
접수일자: 2014년08월10일, 수정일자: 2014년09월16일, 게재확정: 2014년09월30일

Ⅰ. 서 론

차량용 내비게이션은 차량의 위치를 파악하기 위해 GPS(Global Positioning System)[1]를 활용한다. 국내 내비게이션에서 사용하는 운전자용 DB에는 180만개의 링크를 이용한다. 따라서 경로를 탐색할 때, 시작지점에서 목적지점까지 거리가 멀어질수록 경로탐색에 필요한 연산비용(비교횟수)은 거리에 따라 가파르게 증가한다.

한편, 경로탐색 알고리즘에서 가장 기본이 되는 알고리즘에 $Dijkstra$의 알고리즘[2]과 A^*[3] 알고리즘이 있다. $Dijkstra$의 알고리즘은 항상 가장 짧은 경로를 찾을 수 있지만 탐색공간의 모든 경로를 모두 검색하여야 한다. 반면에 A^* 알고리즘의 경우 경험에 근거하여 탐색공간의 크게 줄일 수 있으나, 가장 짧은 경로의 탐색을 보장할 수 없다는 단점이 있다. 따라서 A^* 알고리즘에 기반 한 다양한 연구[4]가 진행되었으며, 크게 검색속도의 개선, 검색 장비에 따른 메모리용량 최소화, 실시간 검색 등의 응용으로 대별할 수 있다.

특히, 내비게이션 경로탐색을 위해 $Dijkstra$와 A^* 알고리즘을 기반으로 한 알고리즘에는 탐색시간을 단축하기 위해 양방향에서 검색하는 Bi-directional 검색 방식[5], 한 검색 영역을 소규모의 여러 영역으로 분할하여 검색하는 Stochastic time-dependent planning 방식[6]을 비롯하여 다양한 차량운행 환경에서 최단 경로탐색 알고리즘 연구[7-8]가 진행되었다.

A^* 알고리즘에 기반 한 탐색 알고리즘들은 다음과 같은 두 가지의 근본적인 문제를 가지고 있다. 먼저, 앞에서도 설명한 $Dijkstra$의 알고리즘에 비해 최적 경로탐색이 어렵다는 (보장하지 못한다는) 문제가 있으며, 둘째로 경로탐색 거리가 멀어질수록 연산비용이 급격하게 증가하는 문제를 가지고 있다. 첫 번째 문제는 알고리즘의 구조적인 문제로, 여기서는 두 번째 문제의 원인을 고찰하고 해결방안을 검토한다.

A^* 알고리즘에서는 탐색시간(연산비용)은 탐색할 노드들을 저장하는 Open 리스트 크기에 따라 비례하여 증가한다. 특히, 시작지점에서 목적지점까지의 거리가 멀수록 Open 리스크의 크기가 급격히 증가하며, 이와 비례해서 (비교)연산비용이 증가하게 된다. 따라서 A^*에 기반 한 알고리즘의 검색시간을 줄이기 위해선 A^* 알고리즘의 Open 리스트 크기를 작게 하여야 하나 이는 탐색효율을 저하시키는(잘못된 경험규칙을 적용하여 탐색에 실패하는) 원인이 되기도 한다. 이 문제를 해결하기 위해서 본 논문에서는 $Dijkstra$와 A^* 알고리즘을 하이브리드시키는 방법[12~13]을 확장 검토한다.

본 논문의 하이브리드 검색에서는 먼저 $Dijkstra$의 알고리즘으로 최단거리가 될 수 있는 경로들을 찾아 Open 리스트를 구성한 후, 탐색 알고리즘을 A^* 알고리즘을 교체한다. 즉, $Dijkstra$의 알고리즘 Open 리스트에 있는 노드들의 적합도를 (A^* 알고리즘의 휴리스틱 경험치(h)를 이용하여) 새롭게 측정하여 가능성이 높은 노드들만을 남긴 후(A^* 알고리즘의 Open 리스트에 저장한 후) Open 리스트의 내용을 모두 삭제하여 ($Dijkstra$의 알고리즘을 새롭게 시작하여) Open 리스트의 크기가 급격히 증가하는 것을 방지한다.

기존의 A^*에 기반 한 알고리즘에서 경험치(h)를 달리 사용하여 $Dijkstra$의 알고리즘 보다 연산비용(방문한 탐색공간의 크기)을 줄일 수 있다 (즉 Open 리스트의 길이를 줄일 수 있다). 그러나 h에 따라선 일부 최단거리경로를 찾지 못하는 경우가 발생한다. 그러나 하이브리드 탐색에서는 $Dijkstra$의 알고리즘과 A^* 알고리즘을 교대로 적용하여 Open 리스트의 길이를 줄이기 때문에 A^* 알고리즘만으론 찾을 수 없는 최단거리경로를 탐색할 수 있다. 즉, 탐색공간에서 주기적으로 $Dijkstra$의 알고리즘을 사용하기 때문에 $Dijkstra$의 알고리즘과 비슷한 수준의 경로탐색이 가능하다.

실험 결과, 제안한 방법으로 기존의 $Dijkstra$, A^*, $LRTA^*$, $Bi-directional$ ($Bi-dir$) 알고리즘 보다 연산이 감소할 수 있음을 보이고, A^* 알고리즘만으론 찾을 수 없는 경로를 찾을 수 있음을 보인다. 즉, 기존 $Dijkstra$ 수준의 최단거리 경로를 A^*보다 적은 탐색비용으로 탐색을 할 수 있음을 보인다. 이하, 제 Ⅱ장에서는 기존의 탐색 알고리즘을 이용한 경로탐색을 설명한다. 제 Ⅲ장에서는 제안된 경로탐색 시스템에 대한 알고리즘을 설명하며, 제 Ⅳ장에서는 컴퓨터 시뮬레이션 결과를 고찰한 후, 제 Ⅴ장에서 결론을 맺는다.

Ⅱ. 관련 연구

1. 경로탐색 알고리즘

최단경로의 탐색은 어떤 네트워크에서 시작지점에서 목적지점까지의 최단경로를 탐색하는 문제로서, 소요시간이나 메모리용량 등에서 큰 차이를 보이는 다양한 방

법이 제안되어 있다. 이하 대표적인 경로탐색 기법이면서 본 연구의 하이브리드 대상인 $Dijkstra$의 알고리즘과 A^* 알고리즘을 간단히 설명한다.

먼저, $Dijkstra$의 알고리즘은 시작지점에서 모든 이웃 경로의 탐색을 시작하며, 시작지점을 기준으로 영역을 확장하여 검색하고 목적지점인지 조사한다. 이 과정을 목적지점을 찾을 때까지 반복하는 알고리즘이다.

$Dijkstra$의 알고리즘은 최단거리 탐색을 보장하는 완전 탐색 알고리즘의 하나로 알려져 있다.

한편, A^* 알고리즘은 시작지점에서 목적지점까지 최적우선탐색 규칙을 적용하여 최단경로를 산출하는 선택적인 탐색 알고리즘의 하나이다. 즉 가장 적절한 탐색방향을 평가한 후 탐색을 진행하며, 탐색이 잘못되었을 경우 다시 뒤로 돌아와서 다른 방향을 탐색하는 목표지향적인 알고리즘이다. 이 때 n번 노드의 적합도(fitness)는 $f(n)$으로, 다음과 같이 평가한다.

$$f(n) = g(n) + h(n) \tag{1}$$

여기서, $g(n)$은 시작지점에서 n번 노드까지의 최단비용이고 $h(n)$는 n번 노드에서 목적지점까지 최단 추정비용이다. 따라서 $h(n)$ 추정치가 정확 할수록 $f(n)$의 값은 실제 최단경로의 값 $f^*(n)$에 가까워지며 A^* 알고리즘은 최단거리로 목적지점에 도달할 수 있게 된다. A^* 알고리즘의 상세에 대한 설명은 생략하며, 자세한 내용은 서베이 논문[4]을 참조할 수 있다.

2. A* 알고리즘 성능 향상을 위한 우선순위 큐

$Dijkstra$와 A^* 알고리즘의 성능 개선을 위한 초창기 기법은 탐색시간을 단축하기 위해 양방향 검색을 수행하였다. 즉, 시작지점에서 목적지점으로 $Dijkstra$이나 A^* 알고리즘을 수행하면서 동시에 또 다른 한편으로 목적지점에서 시작지점의 역방향으로 동일한 탐색 알고리즘을 수행하는 양방향 탐색 알고리즘이 있다. 예를 들면 양방향 $Dijkstra$의 알고리즘이나 양방향 A^* 알고리즘이다. 그러나 양방향 A^* 알고리즘의 경우, 순방향 알고리즘과 역방향 알고리즘이 서로 다른 경험규칙 h를 사용할 경우 두 탐색방향이 일치하지 않게 되고, 따라서 탐색공간은 기존의 단방향 A^* 알고리즘보다 더 커지게 된다.

또 다른 탐색법으로, 우선순위 큐(priority queues)를 가진 A^* 알고리즘[9]이 제안 되었으며, Open 리스트의 크기를 줄이는 것 보다 Open 리스트 내에 데이터를 빠르게 검색하기 위한 방안이다. 여기서 우선순위 큐는 완전이진트리로서, 큐에서 가장 상단에 위치하는 노드가 가장 큰 값 (혹은 가장 작은 값)을 가진다. 기존 A^* 알고리즘에서 사용하는 Open 리스트는 데이터 추가 및 삭제 시에 데이터를 재 정렬할 필요는 없지만, 우선순위 큐를 가진 A^* 알고리즘에서는 최상위 노드에 가장 작은 값을 가지며, 노드를 추가하거나 삭제할 때 마다 Open 리스트를 재 정렬해야 한다. 따라서 우선순위 큐를 가진 A^* 알고리즘에서는 재 정렬로 인한 추가적인 연산비용이 추가된다.

3. LRTA* 알고리즘

$LRTA^*$(Learning Real-Time A^*)[10-11] 알고리즘은 대표적인 실시간 경로 탐색 알고리즘이다. A^* 알고리즘은 전체 경로를 검색을 하는 반면, $LRTA^*$는 시작지점에서 한정된 탐색공간만을 검색하며, 시작지점을 이동해서 다시 한정된 공간을 검색한다. 새로운 시작지점을 이동하기 이전에 무한반복 과정을 막기 위해서 h를 갱신하여 h'라 하고 계산은 식(2)와 같다.

$$h'(s) = \min c(s,n) + h(n) \tag{2}$$

즉, 시작 노드의 추정치 $h'(s)$는 시작지점(s)부터 한정된 지역 내에 있는 노드 n까지의 거리 $c(s,n)$와 추정 거리 $h(n)$를 더한 값 중에서 가장 작은 값을 가지는 경로를 선택한다. $LRTA^*$는 실시간으로 경로 변화가 많은 환경인 경우 전체 경로를 재탐색하지 않고, 한정된 범위 내에서 경로 탐색을 수행한다. 이는 시작지점에서 목적지점까지의 최단 경로를 보장 하지는 못하지만, 빠른 시간 내에 최선 경로를 찾아낼 수 있는 효율적인 탐색 알고리즘이다. 즉, $LRTA^*$는 경로 변화가 많은 경우에 적합하며, 최단경로 탐색이 아닌 최단시간 내에 경로 탐색을 목적으로 하는 알고리즘이다.

Ⅲ. 하이브리드 경로탐색 알고리즘

앞 장에서 고찰한 기존의 개선된 탐색법이 동기가 되어, 본 논문에서는 $Dijkstra$와 A^* 알고리즘을 하이브리드시킨 탐색알고리즘을 제안한다. 두 알고리즘을 교체하기 위하여 새 파라미터 $Level$을 정의한다. 즉 하이브리드 탐색에서는 $Level$에 따라 $Dijkstra$와 A^*

알고리즘의 적합도를 식(3)과 같이 교대로 계산한다.

$$\begin{cases} f(n)=g(n), \text{ if } Level < 20 \\ f(n)=g(n)+h(n), \ otherwise \end{cases} \quad (3)$$

여기서 $Level$은 검색 시작지점에서 가장 이웃한 노드를 $Level-1$ (줄여서 $L1$으로 표기)로 정의하고, 이 노드에 다시 이웃한 노드를 $Level-2$(즉, $L2$)로 정의한다. 최대 $Level$을 10이라고 가정하는 경우, $L1$에서 $L10$까지는 $Dijkstra$의 알고리즘을 이용하여 최단거리 경로를 탐색하고, $L10$이 되는 경우에만 A^* 알고리즘으로 전환한다. 즉, $Level$ 값이 증가할수록 $Dijkstra$의 알고리즘 $OPEN$ 리스트 크기는 큰 폭으로 증가하게 되며, 이를 A^* 알고리즘의 평가기법을 도입하여 억제시키는 아이디어이다.

위 식(3)을 이용하여 적합도를 계산할 때 $Dijkstra$에서 사용하는 Open 리스트와 A^*에서 사용하는 Open 리스트는 서로 다른 독립적으로 운용되는 리스트로서 각각의 알고리즘에 맞는 Open 리스트를 정의하여 이용한다. 하이브리드 탐색 알고리즘으로 최단거리경로를 찾는 과정은 다음과 같이 세 단계로 이루어진다.

단계 1 : (초기화 단계)
(1) $Dijkstra$와 A^* 알고리즘을 위한 Open 리스트 DO와 AO를 생성한다. 또한 두 알고리즘이 공용하는 Close 리스트 $CLOSE$를 생성한다. DO는 AO 보다 $Level$ 값(l)을 더 저장할 수 있다.
(2) 시작지점 노드를 P라고 하며, P를 DO에 추가한다. 이 때 DO에는 노드 P뿐으로, P에는 $l \leftarrow 0$값을 부여한다.

단계 2 : ($Dijkstra$ 탐색과정)
(1) P 노드에 이웃한 노드 n을 찾은 다음, n이 기존 DO 혹은 $CLOSE$에 존재하지 않은 경우 n을 DO에 추가하고, n의 레벨 값을 $l \leftarrow l+1$로 갱신한다.
(2) DO에서 가장 작은 평가도 값을 선택한 노드를 m이라 하며, m을 $CLOSE$에 추가한다.
(3) 선택 노드 m이 목적노드인 경우 성공종료하고, DO가 공백 리스트인 경우 실패종료한다.
(4) 노드 m의 l이 $Level$ (예: 15) 보다 작으면 단계 2로 가고, 아니면 단계 3으로 간다.

단계 3 : (A^* 탐색과정)
(1) $AO \leftarrow AO \cup DO$로 통합한 후, AO의 모든 노드 (n_i)의 평가도를 다음과 같이 갱신한다: $f(n_i) \leftarrow g(n_i)+h(n_i)$.
(2) AO에서 가장 작은 평가도 값의 노드를 P라고 하며, P ($l \leftarrow 0$)를 $CLOSE$에 추가한다.
(3) DO를 공백 리스트로 리셋 한다.
(4) 단계 2로 간다.

Ⅳ. 컴퓨터 시뮬레이션

이 장에서는 하이브리드 알고리즘의 최단거리 경로 탐색에 대한 컴퓨터 시뮬레이션 내용을 설명한다. 먼저, 시뮬레이션 환경과 두 가지 지도 데이터(인공지도 및 실세계 지도 데이터)를 기반으로 실험 결과를 제시한다. 먼저, 인공지도 데이터를 이용하여 레벨 값(l)에 따른 연산비용을 측정한 결과를 고찰한 다음, 실제 지도 데이터를 이용한 탐색결과와 연산비용을 고찰한다.

1. 인공 지도를 이용한 시뮬레이션

가. 시뮬레이션 환경 및 사용한 지도

먼저 실제의 도로 정보와 비슷한 환경을 구축하여 실험하였다. 시뮬레이션 화면은 크게 탐색경로를 나타내는 화면영역은 최대 1600(40x40)개의 노드(경로와 경로가 만나는 교차로)를 표기할 수 있도록 하였고, 시작지점(S)를 출발하여 목적지점(E)에 도착하는 최단거리 경로를 탐색하는 실험환경을 설정하였다. 여기서 실험을 간단하게 하기 위하여 노드와 노드 사이의 거리는 모두 '1'로 가정하였다.

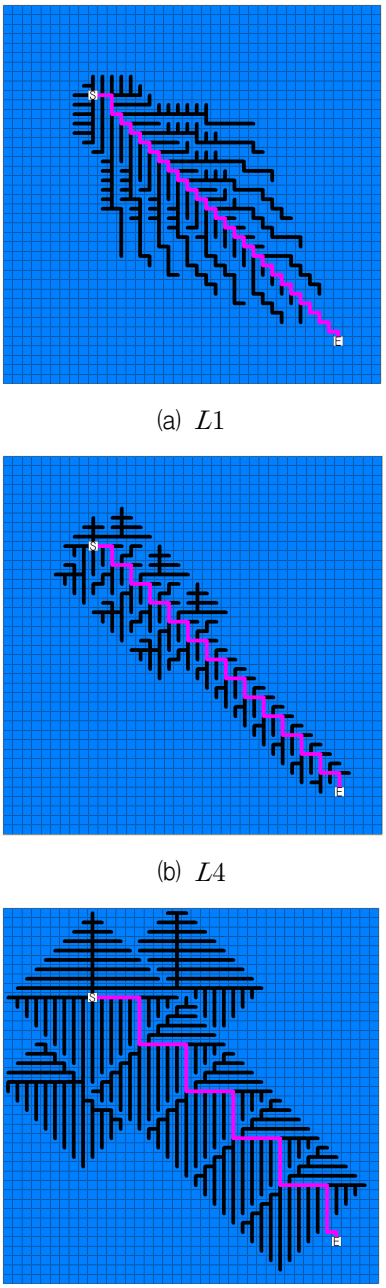
나. Level에 따른 접합도 측정 결과

그림 1은 하이브리드 알고리즘에서 $Level$을 $L1$, $L4$, 그리고 $L10$으로 설정하였을 경우 S에서 E에 도달하는 최단거리 경로를 탐색하기 위한 탐색공간을 나타낸다. 여기서 최단거리경로는 그림 중앙에 분홍색으로 표시하였고, 경로탐색을 위해 방문하였던 경로는 검은색으로 표시하였다.

그림 1(a)는 $L1$의 경로탐색 결과와 탐색공간을 나타낸다. 즉, $L1$에서는 $Dijkstra$ 알고리즘을 적용하지 않고 A^* 알고리즘만으로 경로탐색을 수행한다. 탐색공간은 기존 A^* 알고리즘과 동일한 결과를 보이고 있다.

그림 1(b)는 $L4$의 경로탐색 결과와 탐색공간을 나타낸다. 즉, S에서 세 번째의 이웃 노드까지만 $Dijkstra$ 알고리즘으로 탐색한 후, A^* 알고리즘으로 전환하게 된다. A^* 알고리즘에 의해서 선정된 노드의 적합도를 평가한 다음 다시 $Dijkstra$ 알고리즘을 적용하여 탐색한다.

마지막으로 그림 1(c)는 $L10$의 경로탐색 결과와 탐색공간을 나타낸다. 즉, S에서 아홉 번째의 이웃 노드가 찾아질 때까지 $Dijkstra$ 알고리즘을 사용한 이후에 A^* 알고리즘으로 전환한다.



(a) $L1$

(b) $L4$

(c) $L10$

그림 1. $L1$, $L4$, $L10$을 이용한 경로탐색 결과
Fig. 1. Path search result using $L1$, $L4$, and $L10$.

세 종류 하이브리드 알고리즘으로 최단거리경로를 탐색한 결과, 탐색공간이 $Level$ 값에 따라 차이가 있음을 알 수 있다. 그림 1(a) 보다 그림 1(c)의 경우 더 많은 경로를 탐색하였음을 알 수 있고, 따라서 최단경로를 찾을 수 있는 가능성은 높아진다.

그림 2는 $L1$, $L4$, $L10$의 탐색에서 Open 리스트에서 최소 (또는 최대) 적합도에 해당하는 노드를 선정하기 위한 비교 횟수를 나타낸다. 여기서 x-축은 탐색한 경로 길이(노드 수), y-축은 노드를 비교한 횟수(이를 연산비용이라 정의한다)를 나타낸다.

그림 2에서 $L1$ 탐색의 연산비용이 $L10$탐색의 비용보다 더 크다는 것을 알 수 있다. 또한 시작지점과 목적지점간 사이의 거리가 멀수록 비용의 차이가 더 커진다는 것을 알 수 있다. 즉, 그림 2에서 경로길이가 20미만의 짧은 경우엔 $L1$의 연산비용(붉은 점선과 x-축이 만드는 면적)이 $L10$의 연산비용(푸른 점선과 x-축이 만드는 면적)보다 작다. 그러나 경로길이가 40 또는 그 이상으로 길어질수록 $L1$이 $L10$보다 연산비용이 큰 폭으로 커지게 된다.

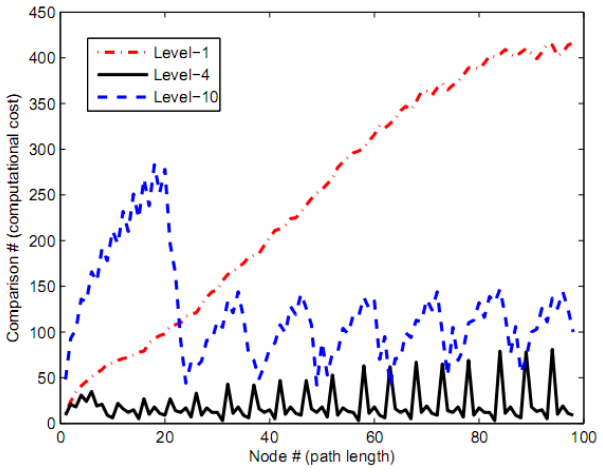


그림 2. $L1$, $L4$, $L10$ 탐색에 필요한 연산비용
Fig. 2. Calculation costs for the path search results using $L1$, $L4$, and $L10$.

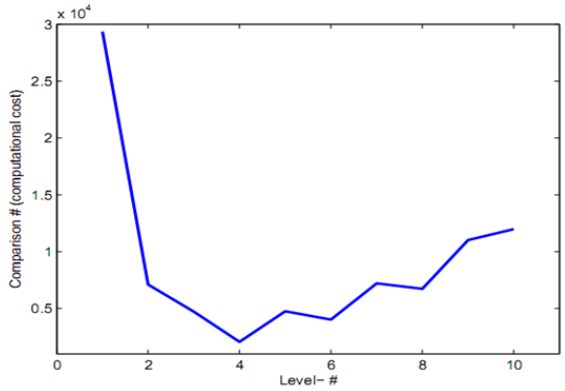


그림 3. $Level$ 값에 따른 누적 연산비용 비교
Fig. 3. Comparison of cumulative calculation costs for the values of $Level$.

또한, 그림 3은 $L1$에서 $L10$까지 $Level$ 값을 증가시키면서 탐색하였을 경우 소요된 연산비용의 누적 값을 표현한 것이다. 연산비용이 $L1$에서 가장 크고, $L4$에서 가장 적음을 알 수 있다. 그리고 $L4$이후 연산비용이 소폭 증가함을 알 수 있다. 이상의 고찰로부터 $L4$의 하이브리드 탐색을 수행하면 연산비용(비교횟수)을 가장 효과적으로 줄일 수 있음을 알 수 있다.

2. 실제 지도를 이용한 시뮬레이션

가. 실제 지도의 환경

지능형교통시스템(ITS: Intelligent Transportation Systems)용 전자도로망 지도 데이터들을 이용하며, 서울, 부산, 대전, 대구, 경기도, 그리고 강원도 지역 데이터를 통한 경로 탐색을 실험한다. 표 1은 ITS에서 제공한 지도 데이터 중에서 서울, 부산, 대전, 대구, 경기도, 그리고 강원도 지역에 대한 노드와 링크의 수를 파악한 것이다. 이 통계는 실제 지도에서 표시되는 모든 도로가 포함된 것이 아니며, 골목길과 같은 세부 도로에 대한 정보를 제외한 내용이다.

표 1. 지도 데이터의 노드와 링크 수
Table 1. The number of Node and Link by real maps.

도시	Node 수	Link 수
Seoul	8565	22182
Busan	2565	7671
Daejeon	1889	5618
Daegu	1438	4471
Kyeonggido	23747	61444
Gangwondo	6535	16184

나. 알고리즘별 경로탐색

경로탐색에서 제안 알고리즘은 $Level$ 값을 $L15$로 하였고, $LRTA^*$의 k는 16, 32, 64로 한정하였다.

표 2는 여섯 지역에 대한 경로 탐색한 결과를 정리한 것이다. 여기서 '경로거리'는 시작지점에서 목적지점까지의 거리이고, '연산비용'은 최적의 경로를 찾기 위한 연산에 대한 전체 비용이다.

또한, 표 2에서 사용한 탐색 알고리즘은 $Dijkstra$, A^*, $LRTA^*$, $Bi-dir$ 및 하이브리드 알고리즘이다. 특히 양방향 검색이 가능한 $Bi-dir$ 알고리즘은 DD, AA, DA, AD 네 가지로 분류하여 실험하였다. 여기서 DD는 양방향 모두 $Dijkstra$를, AA는 양방향 모두 A* 알고리즘을 적용한 탐색이다. 그리고 DA는 시작지점에서는 $Dijkstra$를 사용하며 목적지점에서는 A*를, 반대로 AD는 시작지점에는 A*를 사용하고, 목적지점에서는 $Dijkstra$를 사용한 탐색이다.

표 2의 경로거리에서 최단 경로 (진하게 표시한 수)를 찾아낸 알고리즘은 $Dijkstra$와 $Bi-dir$(DD)이며, 그 다음 짧은 경로를 찾은 알고리즘은 하이브리드 알고리즘이다. 이 결과로부터 하이브리드 알고리즘이 기존 A^*에 비해 더 짧은 최단 경로를 찾을 수 있음을 알 수 있다. 반면에 가장 적은 연산비용의 알고리즘은 일반적으로 $Bi-dir$(AA)이며, 그 다음이 하이브리드 알고리즘이다. 즉, 최단 경로거리 탐색은 연산비용이 증가하고, 연산비용의 최소화는 경로거리의 증가로 연결됨을 알 수 있다. 따라서 $Dijkstra$, $Bi-dir$(DD), $Bi-dir$(AA) 등은 경로거리와 연산비용의 두 평가기준을 모두 만족시키기 어렵다. 특히, AA와 DA의 경우 연산비용이 감소되지만 경로거리는 증가한다. 반면 하이브리드 알고리즘은 그 대안이 될 수 있음을 보인다. 즉, $Dijkstra$나 $Bi-dir$(DD)와 비슷한 경로거리를 $Bi-dir$(AA)에 근접하는 연산비용으로 탐색이 가능함을 보인다.

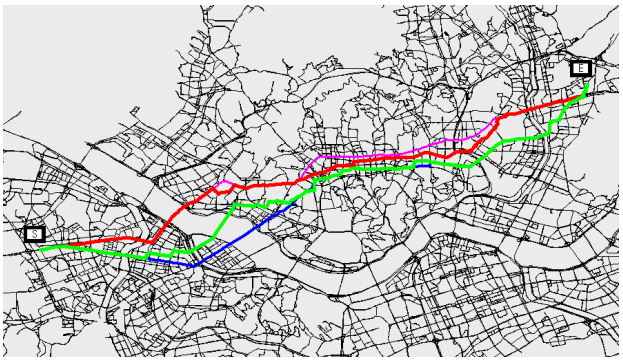
한편, $LRTA^*$는 k 값이 증가할수록 연산비용은 증가하며, 경로거리는 소폭 감소함을 알 수 있다. 예를 들어 표 2에서 Seoul 지역 k=16, 32, 64의 경우 연선비용은 큰 폭으로 증가하나 경로거리는 매우 유사하다. 이는 $LRTA^*$가 제한된 영역만을 탐색하기 때문에 잘못

된 경로가 탐색될 경우, 이를 재조정하기 위하여 큰 비용을 들여야 한다는 특성에 기인한 것으로 보인다. 한편 하이브리드 알고리즘에서는 잘못된 경로가 탐색될 경우, A^* 알고리즘으로 새 경로를 탐색할 수 있다.

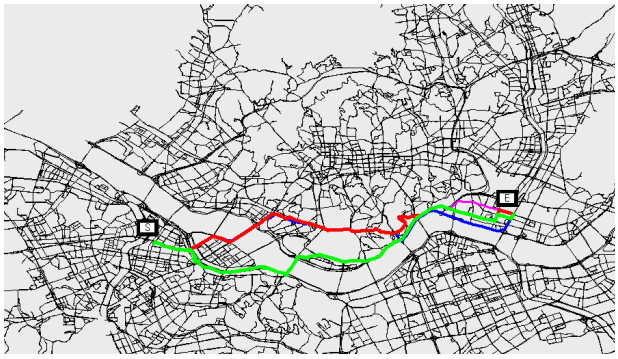
표 2의 탐색은 노드 수 및 링크 수가 서로 다른 지역에서 최단거리 경로를 탐색한 결과이었다. 이와는 달리 그림 4는 서울에서 장거리, 중거리, 단거리에 네 종류 탐색 알고리즘($Dijkstra$, A^*, $LRTA^*$(k=32), 하이브

표 2. 실제도시 맵에 대한 경로거리와 연산비용 (여기서 최소 경로거리와 최소 연산비용은 진하게 표시)
Table 2. Path distance and operation cost for real city maps. Here, the minimal distances and the smallest costs are highlighted in bold-faced.

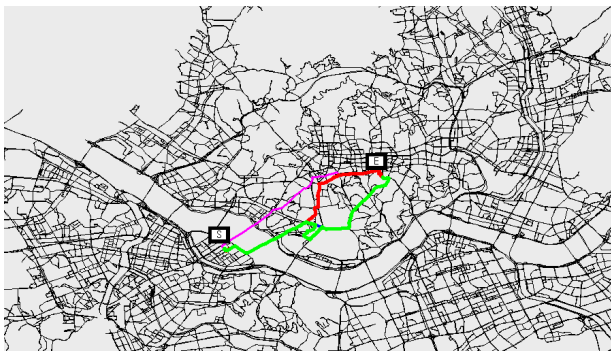
	Area	Dijkstra	A*	LRTA*			Bi-directional				하이브리드 탐색법
				k=16	k=32	k=64	DD	AA	DA	AD	
경로거리	Seoul	**272123**	284222	295718	295623	295306	**272123**	286924	291943	283790	283912
	Busan	**167604**	169158	265240	261554	267604	**168966**	170794	170794	168841	167803
	Daejeon	**281592**	294378	337002	293399	293399	**281592**	294378	293229	293399	293229
	Daegu	**465465**	472703	477256	475566	474084	**465465**	472703	472292	475566	472292
	Kyeonggido	**633475**	646566	2609902	6226063	1447342	**633475**	647037	647045	646566	638990
	Gangwondo	**1829053**	1863230	6140232	6083315	7689602	1835400	1835400	1889106	1862478	1835982
연산비용	Seoul	1273491	48426	46713	138282	404783	1154780	21676	25941	50615	**12012**
	Busan	130624	6281	15359	43747	86963	59493	**2690**	3700	6396	3772
	Daejeon	77330	47036	18715	28214	31179	49585	4823	**3936**	23463	11169
	Daegu	45542	6790	14157	38976	85435	45542	5700	**5073**	38976	**5073**
	Kyeonggido	2910062	319247	176535	1152598	564053	661488	**25865**	86222	43840	68955
	Gangwondo	388955	118676	109894	329563	778123	274514	118246	119544	118252	**40079**



(a) 장거리 (path1)



(b) 중거리 (path2)



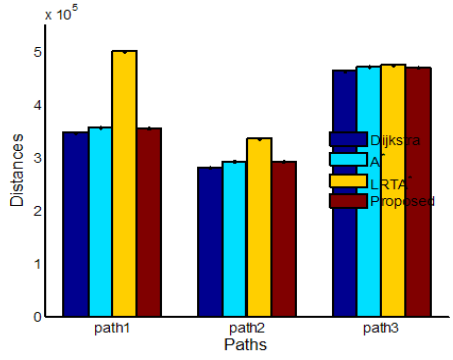
(c) 단거리 (path3)

그림 4. 서울 지도를 이용한 세 종류 경로 탐색 결과
Fig. 4. Search results for the three paths in Seoul maps.

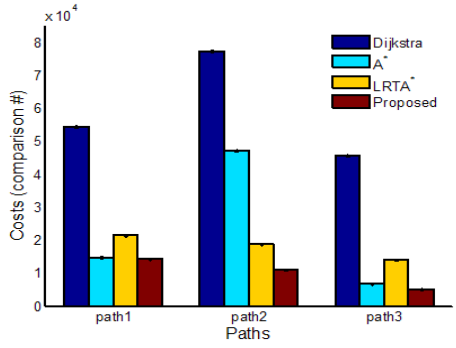
리드 알고리즘)을 적용한 결과를 보인다. 여기서 분홍색은 $Dijkstra$, 파란색은 A^*, 초록색은 $LRTA^*$, 그리고 빨간색은 제안 알고리즘의 경로이다.

또한, 그림 5는 그림 4에서 수행한 서울(노드 수 및 링크 수를 동일하게 한 지역)에서 탐색 경로거리를 장-중-단 거리로 달리하였을 경우 경로거리와 연산비용을 비교한 것이다.

먼저, 그림 5 (a)의 결과로부터 최단거리 경로탐색을 수행한 경로거리에 대한 고찰은 다음과 같다. path1과 path2의 경우 $LRTA^*$ 알고리즘으로 탐색된 경로거리는 그 외 다른 알고리즘($Dijkstra$, A^*, 하이브리드 알고리즘)의 경로거리보다 다소 길지만(높지만) path3의 경우엔 모든 탐색 알고리즘의 경로거리가 비슷함을 보인다. 이와 같은 결과는 그림 4 (a), (b), (c)의 결과로부터도 관찰할 수 있다. 그림 5 (b)의 결과로부터 최단거리 경로탐색을 수행한 연산비용은 path1, path2, path3의 모든 경로에 대하여 제안 알고리즘이 다른 알고리즘



(a) 경로거리



(b) 연산비용

그림 5. 서울에서 장-중-단거리에 대한 경로거리와 연산비용

Fig. 5. Path distance and operation cost for Long-middle-short distance paths in Seoul.

표 3. 표 2의 데이터에 대한 최적의 $Level$ 값

Table 3. Optimal $Level$ values for the data of Table 2.

도시	경로거리		연산비용	
	최소거리	level 값	최소비용	level 값
Seoul	291346	15	27439	12
Busan	351378	20	13081	17
Daejeon	289496	16	14736	20
Daegu	466876	16	5412	16
Kyeonggido	635809	11	12016	11
Gangwondo	1843066	16	38766	10

보다 우수함(높이가 낮음)을 알 수 있다. 이와 같은 고찰로부터 하이브리드 알고리즘이 일반적으로 다른 알고리즘에 비해 연산비용 면에서 더 우수함을 알 수 있다.

끝으로, 표 3은 표 2의 '경로거리' 및 '연산비용'을 고려하여 선정한 최적의 $Level$ 값이다. 즉, 표 2의 실험은 간단한 실험을 위해 하이브리드 알고리즘의 경우 $L15$를 공통으로 사용하였다. 그런데 실험 결과, 제안된 하이브리드 알고리즘의 성능(연산비용)은 $Level$ 값에 민감하였음을 보였다. 따라서 주어진 도로환경에 하이브리드 알고리즘을 효율적으로 적용하기 위해선 최적의 $Level$ 값을 선정하여야 한다. 이 문제를 검토하기 위하여 여러 가지 $Level$ 값으로 경로탐색을 수행한 다음 탐색된 경로거리와 소요된 연산비용이 '최소'가 되는 $Level$ 값을 최적 값으로 선정한 결과는 표 3과 같다.

표 3의 결과는 경로탐색을 위한 지역특성(탐색 경로의 노드 수 및 링크 수) 또는 선정기준 (경로거리 및 연산비용)에 따라 서로 다른 결과를 보인다. 예를 들어, 대구의 경우 경로거리와 연산비용 모두 $L16$이었지만, 다른 지역의 경우 두 기준으로 선정한 최적 값이 서로 다르게 나타났다. 또한, 대전과 대구의 경우 경로거리를 기준으로 선정한 최적 값은 $L16$으로 같았으나 연산비용을 기준으로 선정한 최적 값은 각각 $L20$과 $L16$이었다. 따라서 최적의 $Level$ 값을 선정하기 위한 별도의 학습이 필요하며, 앞으로의 과제이다.

Ⅳ. 결 론

내비게이션 최단경로설정을 위해 다양한 환경에서 작동하는 많은 알고리즘들이 제안되었다. 그러나 지도가 클 경우, 경로탐색을 위한 비교연산의 횟수가 크게 증가하는 문제가 있다. 이 문제를 해결하기 위하여 본

논문에서는 완전탐색 알고리즘의 하나인 $Dijkstra$의 알고리즘을 기본 탐색방식으로 사용하면서 선택적 알고리즘인 A^* 알고리즘의 기법을 활용하는 하이브리드 탐색을 검토하였다. 특히 $Dijkstra$의 알고리즘과 A^* 알고리즘이 별도의 Open 리스트를 사용하는 방법으로 Open 리스트의 무분별한 증가를 방지하여 비교횟수(연산비용)를 줄였다. 또한 $Dijkstra$의 알고리즘으로 최단거리 경로를 탐색하기 때문에 A^* 알고리즘만으론 찾을 수 없는 경로를 찾을 수 있도록 하였다.

제안 알고리즘을 평가하기 위해 $1600(40\times40)$개 노드의 인공 지도 데이터와 정부에서 제공한 지능형교통시스템용 전자도로망 지도 데이터를 사용하였다. 이 지도 데이터를 대상으로 $Dijkstra$, A^*, $LRTA^*$, $Bi-dir$ 알고리즘 및 하이브리드 알고리즘을 적용하여 최단거리 경로탐색을 수행하였고, 탐색결과를 기반으로 경로거리와 연산비용을 비교 분석하였다. 본 시뮬레이션 결과로부터 제안된 하이브리드 알고리즘의 탐색경로가 $Dijkstra$ 알고리즘과 비슷한 결과를 보였으며, 연산비용은 A^*와 비슷한 결과를 나타내었다. 그러나 제안 알고리즘의 성능(경로거리 및 연산비용)은 $Level$ 값에 민감하였다. 즉, 성능은 $Level$ 값이 증가할수록 $Dijkstra$ 알고리즘에 근접하고, 작아질수록 A^* 알고리즘에 접근하였다. 따라서 시작지점과 목적지점간 사이의 거리 및 지역특성에 따른 최적의 $Level$ 값을 자동으로 결정하기 위한 추가적인 연구가 필요하다.

REFERENCES

[1] Hofmann-Wellenhof, B., Lichtenegger, H., Collins, J., *Global Positioning System: Theory and Practice*, Springer-Verlag, New York, 2001.
[2] Dijkstra, E. W., "A note on two problems in connextion with graphs", *Numerische Mathematik*, vol. 1, no. 1, pp. 269-271, 1959.
[3] Hart, P. E., Nilsson, N. J., Raphael, B., "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on System Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, 1968.
[4] Rios, L. H. O., Chaimowicz, L., "A survey and classification of A* based best-first heuristic search algorithms", *Proc. of the 20th Brazilian Symposium on Artificial Intelligence (SBIA 2010)*, vol. 6404, pp. 253-262, 2010.
[5] Lee, J., Kim, J., and Jeon, H. S., "Performance evaluation of different route planning algorithms in the vehicle navigation system," *Journal of Korean Association of Information Education*, vol. 2, no. 2, pp. 252-259, 1998.
[6] Flinsenberg, I. C. M., *Route Planning Algorithms for Car Navigation*, PhD Thesis, Technische Universiteit Eindhoven, The Netherlands, 2004.
[7] Ok, S. -H., Ahn, J. -H., Kang, S., and Moon, B., "A combined heuristic algorithm for preference-based shortest path search," *Journal of the Institute of Electronics Engineers of Korea*, vol. 47(TC), no. 8, pp. 716-726, 2010.
[8] Lee, B. -W., Choi, W. -K., and Jeon, H. -T., "Intelligent navigation system using fuzzy logic," *Journal of the Institute of Electronics Engineers of Korea*, vol. 43(CI), no. 4, pp. 67-72, 2006.
[9] Park, M. -J., A *Study on the High-Speed Search Method Using A* Algorithm*, Master Thesis of Electronic Engineering, Hoseo University, Cheonan, Korea, 2011.
[10] Korf, R. E., "Real-Time Heuristic Search", *Artificial Intelli.*. vol. 42, no. 2-3, pp. 189-211. 1990.
[11] Bulitko, V., Lee, G., "Learning in Real Time Search: A Unifying Framework", *Journal of Artificial Intelligence Research*. vol. 25, pp. 119-157, 2006.
[12] Lee, Y. -H., Kim, S. -W., "A method of finding the optimal paths on image maps for navigation system," *Proceedings of IPIU 2014*, Jeju, Korea, Feburary 2014.
[13] Lee, Y. -H., Kim, S. -W., "A Hybrid Search Method to Find Minimal Length Paths for Navigation Route Planning," *Proc. of ICAI 2014*, Las Vegas, NV, pp. 378-384, July 2014.

저 자 소 개

이 용 후(정회원)
대한전자공학회 논문지
제 50권 제 8호 참조

김 상 운(정회원)-교신저자
대한전자공학회 논문지
제 50권 제 8호 참조