다익스트라* 알고리즘 : 최소검색비용 최적경로 탐색

Dijkstra* Algorithm :

Minimum Search Cost Optimal Path Exploration

요 약

최단거리 알고리즘에는 대표적으로 다익스트라 알고리즘과 A* 알고리즘이 있다. 다익스트라 알고리즘은 최단 경로를 찾을 수 있는 장점이 있으나 탐색 공간의 모든 경로를 검색하기 때문에 연산 시간이 많이 소요되는 단점이 있다. A* 알고리즘은 탐색 범위를 줄일 수 있으나, 경로상의 탐색 범위가 넓은 경우, 최단 경로를 보장하지 않는다. 본 논문에서는 다익스트라 알고리즘과 A* 알고리즘의 장점을 결합한 다익스트라* 알고리즘을 제안한다. 다익스트라* 알고리즘은 전체 경로를 여러 부분 경로로 세분화한다. 세분화된 경로는 A* 알고리즘으로 탐색하고 부분 경로별 탐색 순서는 다익스트라 알고리즘을 사용한다. 즉, 기존의 최단거리 알고리즘을 보완한 새로운 최소 검색 비용을 갖는 최적경로 탐색 알고리즘을 제안한다.

1. 서 론

대중교통 접근성을 분석할 때 실제 이동경로를 반영하기 위해 최단거리 알고리즘을 활용할 수 있다. 최단 거리 알고리즘을 활용하여 한 노드를 기준으로 그래프 상의 다른 모든 노드에 대한 최단 거리 및 이동 시간을 얻을 수 있기 때문에 노드 단위의 세밀한 접근성을 도출 할 수 있다.[1] 최단거리 알고리즘에는 대표적으로 다익스트라 알고리즘(Dijkstra's Algorithm)과 A* 알고리즘(A star Algorithm)이 있다. 다익스트라 알고리즘은 시작점에서 도착점까지의 가능한 모든 경로(path)를 탐색하여 경로의 가중치 값(weight)이 가장 작은 경로를 찾을 수 있는 장점이 있으나 탐색 공간의 모든 경로를 검색하기 때문에 계산 시간이 많이 소요되는 단점이 있다.[2] A* 알고리즘은 탐색 범위를 줄일 수 있으나, 넓은 영역을 탐색하는 경우, 가장 짧은 경로를 보장하지 않는다.[3]

본 논문에서는 앞서 말한 최단 경로 알고리즘의 단점들을 보완하기 위해 다익스트라 알고리즘의 장점과 A*의 장점을 결합하였다. 먼저, 탐색 시간을 감소시키기 위하여 전체 경로를 여러 부분 경로로 세분화하였다. 세분화된 경로는 A* 알고리즘을 이용하여 부분 경로 내 최단경로를 구한다. A* 알고리즘이 다익스트라 알고리즘 보다 짧은 연산 시간을 갖기 때문에, 다익스트라 알고리즘의 느린 탐색 속도문제를 해결할 수 있다. 또한, 세분화된 경로 탐색을 통해 넓은 영역 탐색 시 발생하는 A* 알고리즘의 단점을 해결하였다. 다음으로 최적 경로를 구하기 위해서 나눠진 경로의 탐색 순서는 다익스트라 알고리즘을 이용하여 정한다. 이때, 전체 경로 탐색에 비해 노드수가 크게 감소하기 때문에 탐색 시간 또한 현저히 줄어든다. 즉, 기존의 최단거리 알고리즘의 단점을 보완한 새로운 최적경로 탐색 알고리즘을 제안한다.

2장에서는 Dijkstra 알고리즘과 A* 알고리즘의 최단경로를 찾는 과정을 고찰해 보고 문제점을 살펴본다. 3장에서는 2장에서 고찰한 알고리즘의 단점을 보완한 새로운 최적경로 탐색 알고리즘을 제안한다. 4장에서는 제안된 알고리즘을 컴퓨터 프로그램으로 실험 결과를 고찰한 후 결론을 맺는다.

2. 관련 연구

2.1 다익스트라 알고리즘(Dijkstra Algorithm)

다익스트라 알고리즘은 어떤 간선도 음수 값을 갖지 않는 방향 그래프에서 주어진 출발과 도착 사이의 최단 경로를 계산하는 알고리즘이다. 따라서 다익스트라 알고리즘은 한 노드에서 다른 모든 노드로의 최단경로를 구하는 방법이다.[4] 다익스트라 알고리즘은 다음과 같다.[5]

```
1   function Dijkstra(G, w, s)
2     for each vertex v in V[G]
3           d[v] := infinity
4           previous[v] := undefined
5     d[s] := 0
6     S := empty set
7     Q := set of all vertices
8     while Q is not an empty set
9           u := Extract_Min(Q)
10          S := S union {u}
11          for each v with edge (u,v) defined
12                if d[v] > d[u] + w(u,v)
13                      d[v] := d[u] + w(u,v)
14                      previous[v] := u
```

알고리즘 1. 다익스트라 알고리즘

다익스트라 알고리즘은 알고리즘 1을 통해 정확한 최단경로를 제시한다. 이러한 다익스트라 알고리즘의 장점을 활용하여 다익스트라* 알고리즘은 탐색할 부분 경로를 정한다.

2.2 A* 알고리즘(A Star Algorithm)

A* 알고리즘은 실제 거리에 대한 정보와 휴리스틱 함수가 이용된다. 실제거리 $g(n)$은 출발지에서 현재 노드까지의 실제거리로 계산되며, 출발지에서 현재 노드까지의 실제거리가 누적된 값이다. 휴리스틱 함수 $h(n)$은 추측된 값으로서 현재 노드에서 목적지까지의 거리 예측 값이다. 이 두 함수의 합으로 평가함수 $f(n)$을 구할 수 있다.

A* 알고리즘에는 노드들을 관리하는 Open List와 Close List가 있는데, 각 노드들의 상태에 따라서 해당 List에 나눠 저장된다. 아직 탐색되지 않은 노드들은 Open List에 저장되고, 탐색이 완료된 노드들은 Close List에 저장된다. 즉, Open List에 저장되어 있던 노드들을 탐색을 하면 Close List로 이동되고, Open List의 상태가 비어있는 상태가 되면 알고리즘은 탐색을 종료한다.

A* 알고리즘은 다음과 같다.

(a) 출발점을 생성하고 공백으로 초기화 된 Open List에 저장한다.

(b) 평가 함수 f는 Open List 노드들의 정보로 계산되며, 계산된 f값이 가장 작은 노드를 Open List에서 선택한다. 선택된 노드는 Open List에서 제거되고, Close List로 이동한다.

(c) Open List에 저장된 노드가 없거나 현재 노드가 목적지에 도달했다면 알고리즘을 종료한다.

짧은 탐색 시간을 가지는 A* 알고리즘을 활용하여 부분 경로 탐색에 활용하였다.

3. 다익스트라* 경로 탐색 알고리즘

앞 장에서 고찰한 기존의 탐색 법이 동기가 되어, 본 논문에서는 Dijkstra 알고리즘과 A* 알고리즘을 개선한 다익스트라* 탐색 알고리즘을 제안한다. 다익스트라* 알고리즘은 시작 노드에서 도착 노드까지의 전체 경로를 R-node 기준으로 부분 경로로 세분화한다. 이때, R-node란 간선의 개수가 평균이상인 노드를 의미한다. 이때, 간선의 개수가 많을수록 가중치 선택을 유연하게 할 수 있다.

R-node 지정은 다음과 같은 기준을 따른다. 우선, 각각의 노드들이 소유하고 있는 간선 수의 평균을 구하고, 평균치보다 많은 수의 간선을 소유하고 있는 노드들을 R-node로 지정한다. 이렇게 구한 R-node들은 R-list에 저장되며, 향후 구간화 작업에 사용된다. 각각의 R-node는 R-node들간의 다익스트라 알고리즘 결과를 테이블 형태(R-table)로 가지고 있다. 이때, R-table은 경로 탐색 전에 초기화 되며 R-node의 변화가 발생하지 않을 경우 변경되지 않는다.

다익스트라* 알고리즘으로 최단 경로 거리 찾는 과정은 다음과 같다.

```
[ Initialize (R-list, node) ]
  openlist = {node}
  closelist = {}
  while
        cur = node having the lowest F value
        if cur in R-list
              return cur
        openlist.remove(cur)
        closelist.add(cur)
        for each neighbor of cur
              if neighbor in closelist
                    continue
              if neighbor not in openlist
                    openlist.add(neighbor)
```

```
[ Algorithm (S, D) ]
   R-list = {R-node}
   if S != R-node
        S = initialize (S)
   if D != R-node
        D = initialize (D)
   while
        D' = next node in start's dijkstra table
        openlist = {S}
        closelist = {}
        while open not empty
              cur = node having lowest F value
              if cur = D'
                    break
              openlist.remove(cur)
              closelist.add(cur)
              for each neighbor of cur
                    if neighbor in closelist
                          continue
                    if neighbor not in openlist
                          openlist.add(neighbor)
              if cur = D
                    return
              else
                    S = cur
                    continue
```

알고리즘 2. 다익스트라* 알고리즘

알고리즘 2에 따르면 R-table은 실행 중이던 부분 경로 탐색 완료 후, 다음 탐색 경로 선택에 사용된다. 선택된 부분 경로는 A* 알고리즘으로 진행된다. 즉, 다익스트라 알고리즘을 사용하여 전체 경로내 부분 경로 별 최단 경로를 제시하고, 부분 경로 내 최단경로 탐색은 A* 알고리즘을 이용한다.

시작 노드(S)와 도착 노드(D)가 R-node가 아닌 경우에는, A*를 이용하여 가장 가까운 R-node를 서브시작노드 또는 서브 도착노드로 지정한다. 선택된 부분 경로 내에서 A*를 이용하여 최단경로를 찾는다. 이때, 부분 경로의 도착 노드(D')가 D가 될 경우 탐색을 종료한다.

4. 실험 결과 분석

이 장에서는 다익스트라* 알고리즘의 최적 경로 탐색에 대한 컴퓨터 시뮬레이션 내용을 설명한다. 시뮬레이션 환경은 가상의 도로정보로 구축하여 실험하였다. 시작 노드(S)를 출발하여 도착 노드(D)에 도착하는 최적 경로를 탐색하는 실험환경을 설정하였다. 여기서 실험의 간편성을 위해 노드와 노드 사이의 거리는 모두 '1'로 가정하였다.

그림 1, 2, 3은 각각 알고리즘의 시뮬레이션 결과이다. 여기서 각각 알고리즘에 따른 결과는 그림 상에 검은색 선으로 표시하였고, 경로탐색을 위해 방문하였던 노드는 옅은 회색으로 표시하였다. 이때, 가상의 도로 상황을 표현하기 위해서 접근할 수 없는 경로는 짙은 회색의 노드로 장애물로 처리했다. 그림 3의 경우 R-node는 검은색 노드로 표시하였다.

세 가지 알고리즘으로 최단경로를 탐색한 결과, 연산 시간, 경로 길이, 연산 횟수가 알고리즘에 따라 차이가 있음을 알 수 있다. 그림 1은 다익스트라 알고리즘으로 전체 노드를 탐색하기 때문에 연산 횟수가 가장 많아 가장 긴 탐색 시간을 갖지만, 최단 경로를 제시한다. 그림 2는 A* 알고리즘으로 다익스트라 알고리즘에 비해 탐색 범위가 좁아 연산 횟수와 연산시간이 감소하지만 최단 경로를 제시하지 못한다. 게다가, 그림 2의 사각형 내에 보이는 것과 같이 장애물이 있을 경우 불필요한 곳을 탐색하는 상황이 발생할 수 있다. 그림 3은 다익스트라* 알고리즘으로 R-node의 다익스트라 결과를 바탕으로 최적 경로를 탐색하기 때문에 그림 2보다 최적화된 경로를 제시한다. 또한, 부분 경로 별로 A* 알고리즘을 이용하기 때문에 탐색 범위를 크게 줄일 수 있다. 따라서 표 1에 보이는 것과 같이, 가장 짧은 연산 시간과 연산 횟수를 가지며 최단 경로에 가까운 최적 경로를 제시한다.

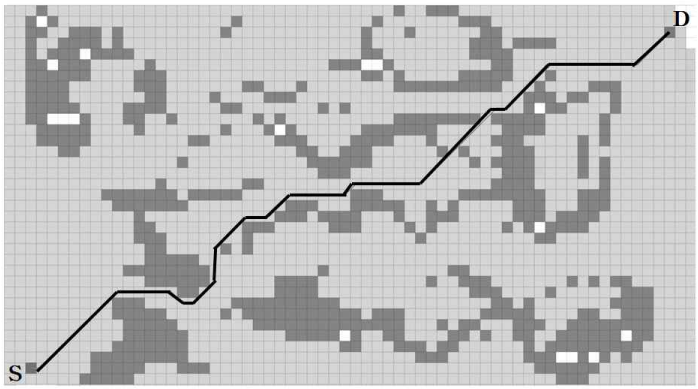

그림 1. 다익스트라 알고리즘

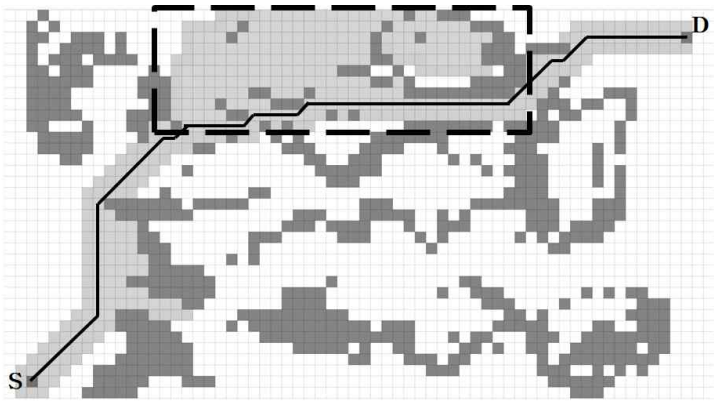

그림 2. A* 알고리즘

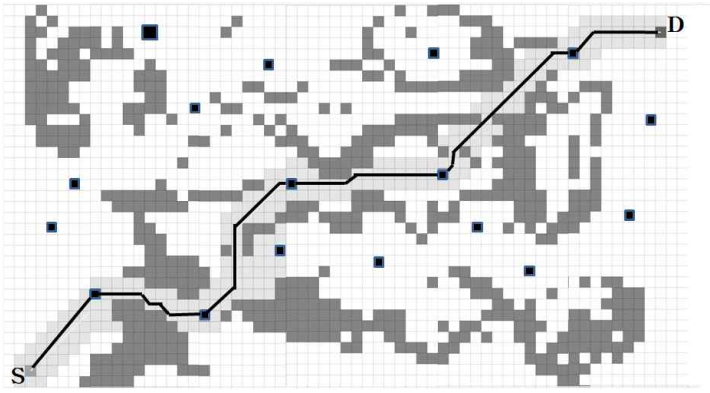

그림 3. 다익스트라* 알고리즘

	다익스트라	A*	다익스트라*
연산 시간	4.92	3.75	3.4
경로 길이	74.43	77.7	75.4
연산 횟수	3492	692	554

표 1. 알고리즘별 시뮬레이션 결과

5. 결론

다익스트라 알고리즘은 최단 경로를 찾을 수 있는 장점이 있으나 탐색 공간의 모든 경로를 검색하기 때문에 연산 시간이 많이 소요되는 단점이 있다. A* 알고리즘은 탐색 범위를 줄여 짧은 연산 시간을 갖지만, 경로상의 탐색 범위가 넓은 경우, 최단 경로를 보장하지 않는다.

본 논문에서는 다익스트라 알고리즘과 A* 알고리즘의 단점을 보완하기 위해 각 알고리즘의 장점을 결합한 다익스트라* 알고리즘을 제안한다. 다익스트라* 알고리즘은 전체 경로를 여러 부분 경로로 세분화한다. 세분화된 경로는 A* 알고리즘으로 탐색하고 부분 경로별 탐색 순서는 다익스트라 알고리즘을 사용한다. 따라서, 최적 경로를 빠르게 탐색할 수 있다. 향후 실제 도로 상황을 반영한 추가적인 연구가 필요하다.

[참고문헌]

[1]최승우, 전철민, 조성길, "정류장 단위의 미시적 대중교통 접근성 분석 - KTX 서울역 사례연구", 한국지형공간정보학회지 24(1), 2016.3, 9-16.

[2] James B. Orlin, Kamesh Madduri, K. Subramani, and M. Williamson, "A faster algorithm for the single source shortest path problem with few distinct positive lengths", J. of Discrete Algorithms 8,June 2010,189-198.

[3] 이용후, 김상운, "내비게이션 경로설정에서 최단거리경로 탐색을 위한 A*와 Dijkstra 알고리즘의 하이브리드 검색법", 전자공학회논문지 51(10), 2014.10, 109-117.

[4] 조태환, 김지원, 김병조, 윤완오, 최상방, "동적 라우팅 알고리즘의 신뢰성 향상을 위한 최단 경로 설정 알고리즘", 정보과학회논문지 : 정보통신 38(6), 2011.12, 450-459.

[5] E.WDijkstra, "A note on two problems in connection with graphs",Numerische Mathmatik,1959,1:269-271.