휴리스틱 탐색 알고리즘을 사용한 국도 이용 여행 최적 경로의 웹 사이트 구현에 관한 연구

서 태 양* · 임 재 걸**

< 목 차 >

Ⅰ. 序 論

현대적 의미의 관광개발은 관광자원의 특징을 살려 가치를 증대시키는 모든 일을 포함하는 것으로 자원의 가치를 쉽게 느낄 수 있도록 하는 교통 및 관광안내 시스템의 정비는 물론 보존·보호사업, 관광정보 제공 등도 개발의 범주에 포함된다(서태양, 1996). 특히 관광량의 증대와 지역적 범위가 확대되고 있는 오늘날 관광 목적지 결정에 관광루트가 차지하는 비중은 점차 높아지고 있으며(김병문, 1986), 관광산업에 있어서 교통은 접근성이 중요한데 관광시장과 관광지와의 접근성은 거리와 교통시설, 교통기관이 등이 중요한 영향을 미치는 것으로 제기되고 있다(김종은·길용현, 1988). 우리나라의 경우 고속도로가 많이 개설되기는 했으나 지방관광에 있어서는 이용이 어려운 실정이며, 현재 전국의 관광지들을 최종적으로 이어주고 있는 관광루트는 국도이다. 따라서 경제적으로 어려운 이 시기에 새로운 루트를 형성한다는 것은 어려운 일일 수 있으며, 이미 개설되어 있는 기존국도의 활용도를 높인다면 관광객의 편의 제공은 물론 지역관광의 활성화에도 크게 기여할 수 있을 것

* 동국대학교 관광대학장, 대한관광경영학회장
** 동국대학교 정보관리처장, 동국대학교 전산학과 교수

으로 판단된다. 우리나라의 경우 관광목적지 선택관련 논문(이태희, 1998; 김향자, 1997; 김영문외, 1996;김원인, 1994)은다수 발표되었으나, 최근까지 국도 이용 여행 최적경로에 관한 연구는 극히 드물며, 전산분야에 있어서의 기술적인 연구로는 Skvarcius (1980), Gelperin (1977), Nilsson (1968), Horowitz (1984) 등의 연구를 들 수 있다. 미국에서는 1970년대 이후 관광학 분야의 주요 연구로서 여가 및 레크레이션 활동, 관광계획 및 개발, 위락여가 동기,호텔경영, 여행목적지선택결정요인 등에 관한 연구들이 활발하게 진행되어 왔으며, 그 밖에도 Frank(1984), Kaspar(1984), Haahti(1986), Ahmed(1991) 등의 관광지 포지셔닝에 관한 연구가 있으나, 관광지 접근을 위한 최적경로 연구는 극히 드문 것으로 나타났다. 따라서 관광 대중화 시대의 특징인 자동차 여행자들을 위한 목적지 최적경로에 대한 연구의 필요성이 절실한 입장이다.

관광은 주체인 관광객이 객체인 관광대상, 즉 관광지를 찾아감으로서 성립이 되는데 아무리 매력적인 관광대상이 있어도 접근할 수 없으면 의미가 없다. 즉 관광지의 발전조건은 교통이 좌우하며 접근성이 좋아야 관광지가 활성화 될 수 있다. 특히 자동차 여행이 일반화되어 있는 현대는 도로교통이 관광지 발전을 규정하는 주 요소가 되고 있다.

경제가 어려운 오늘날 관광발전을 위한 자본의 투자는 극히 어려운 처지로 기존 관광지의 활용극대화가 어느때 보다도 중요시 된다. 따라서 고비용을 투입하여 건설된 국도가 정보 부족으로 효율적으로 이용되지 못함은 국력의 낭비이며 관광개발의 경직성을 보여주는 안타까운 일이다. 본 연구는 국도이용 여행 최적 경로에 관한 제공 웹사이트 개발을 목적으로하며 구체적인 내용은 다음과 같다.

첫째, 관광객이 원하는 관광목적지를 가장 저렴한 경비로 최단시간에 도착할 있는 정보를 제공한다.

둘째, 관광객에게 관광Route에 대한 정보제공으로 관광계획 수립에 도움을 주며 교통혼잡 해소에 기여한다.

셋째, 국도를 이용한 효율적인 접근방법을 통해 관광객의 만족도를 높이고, 관광지 활성화에 기여한다.

Ⅱ. 이론적 배경

1. 도시간의 최단 경로 구하는 알고리즘

여행 최단 경로를 구하려면 전국 각 시, 군간 국도의 거리가 필요하다. 전국 시, 군의 수가 150여개에 이르므로, 22,500여 경우의 수에 대하여 시,군간의 거리와 경로를 작성하여야 한다. 이것을 지도에서 직접 찾아 기입한다는 것은 불가능한 일이다. 이 문제를 풀기 위하여 그래프 이론에서 개발된 "주어진 정점에서 다른 모든 정점까지의 최단 경로 찾기" 알고리즘을 사용하였다.

그래프는 정점의 집합, V와 V상의 관계인 E로 구성된다. E의 원소를 간선이라 한다. 각각의 간선에 실수가 연합되어 있는 그래프를 weighted graph라 한다. 정점과 간선의 나열 v_0 e_1 v_1 e_2 v_2 ... e_k v_k는 e_i가 (v_{i-1}, v_i)인 간선일 때 v_0에서 v_k로 가는 경로라하고 모든 e_i의 weight의 합을 이 경로의 비용이라 한다. Weighted graph에서 이웃하는 정점간의 거리를 바탕으로 주어진 정점으로부터 다른 모든 정점 각각까지의 최단 거리 및 경로를 구하는 알고리즘, Dijkstra's algorithm (Skvarcius, 1980)은 <표1>과 같다. 전국 각 시군을 그래프의 정점으로 하고, 각 시군간의 국도를 간선으로 하며, 국도의 거리를 간선의 weight로 하는 그래프에 <표1>의 방법을 적용하면 각 시군간 국도의 거리를 구 할 수 있다.

<표 1> Dijkstra's Algorithm

```
/* Let (V, E) be a weighted digraph. Let A be a vertex. The object is to find, for
each vertex x, the distance, d[x], from A to x and the shortest path from A to x. We
assume that for y and z in V, w(y, z) is defined by
      w(y, z) = 0, if y=z,  ∞, when (y, z) is not an edge, weight of the edge from y
                to z, otherwise.
Define pathto(z) to be the list of vertices in the shortest current path from A to z.  */
begin
 for each x ∈ V do
   begin
     d[x] ← w(A, x)
```

```
    pathto(x) ← A
  end
Mark vertex A
While unmarked vertices remain that
  are a finite distance from A do
  begin
    x ← one of the unmarked vertices whose distance from A is minimal
    Mark vertex x
    for each unmarked y∈V such that (x, y)∈E do
    begin
      d' ← d[x] + w(x, y)
      if d' < d[y] then
      begin
        d[y] ← d'
        pathto(y) ← pathto(x), x
      end
    end
  end
end.
```

2. 최적 여행 경로 구하기 알고리즘

다음은 방문지를 한바퀴 돌고 종착지로 가는 최적 경로를 구하는 방법이다. 이러한 문제는 Traveling Sales Person probelm(TSP)이라는 이름으로 컴퓨터 과학 분야에 널리 알려진 문제이다. 대표적인 NP-complete 문제로써, 많은 사람들이 TSP 문제를 푸는 효율적인 방법을 찾아내기 위하여 노력하여 왔다. 가장 대표적인 TSP 문제를 접근하는 방법은 그래프 탐색 알고리즘의 하나인 A^* 방법이다.

그래프 탐색이란 상태를 정점(node라고도 함)으로 하고 합법적인 상태 변화를 간선으로 하며 상태 변화에 드는 비용을 간선의 weight로 하는 그래프에서, 초기 상태로부터 목적 상태에 도착하기 위한 최적 경로를 찾는 문제이다. 그래프 탐색 문제에서 상태의 수는 보통 기하급수로 증가한다. TSP 문제에서 여행자가 현재 위치한 도시를 상태로 하고, 도시에서 다른 도시까지 직접 연결된 국도를 합법적인 상

태 변화로 하면, TSP 문제도 그래프 탐색 문제로 변환된다. 문제에 주어진 도시의 수가 n이라 하고, 어떤 도시에 이웃한 도시의 수를 평균 m이라 하면, TSP의 그래프 탐색 문제를 구성하는 상태의 수는 약 n^m이 된다. 따라서 초기 상태로부터 모든 가능한 상태를 차례로 섭렵하면서 목적 상태에 도달하는 최적 경로를 찾는 방법은 도시의 수가 커짐에 따라 메모리 부족으로 수행이 불가능하게 된다.

<표 2> A^* 알고리즘의 골격

Procedure A^*

1. Create a search graph, G, consisting solely of the start node, s. Put s on a list called OPEN.
2. Create a list called CLOSED that is initially empty.
3. LOOP: if OPEN is empty, exit with failure.
4. Select the first node on OPEN, remove it from OPEN, and put it on CLOSED. Call this node n.
5. If n is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from n to s in G. (Pointers are established in step 7.)
6. Expand node n, generating the set, M, of its successors and install them as successors of n in G.
7. Establish a pointer to n from those members of M that were not already in G(i.e., not already on either OPEN or CLOSED). Add these members of M to OPEN. For each member of M that was already on OPEN or CLOSED, decide whether or not to redirect its pointer to n. For each member of M already on CLOSED, decide for each of its descendants in G whether or not to redirect its pointer.
8. Reorder the list OPEN, either according to some arbitrary scheme or according to huristic merit.
9. Go LOOP.

이와 같이 상태의 수가 다룰 수 없을 만큼 많은 문제를 접근하는 방법으로, 각각의 상태에서 목적 상태에 도달하기까지의 비용을 예측하고, 가장 비용을 절약할 수

있으리란 상태를 선택하여 나아가는 방법을 알고리즘 A라고 하며 인공지능 분야에서 널리 사용된다. 더 나아가서 예측 비용이 언제나 실제 비용보다 작을 때, 이러한 예측 비용을 사용하는 알고리즘을 인공지능 분야에서는 A*라고 한다. 알고리즘 A*는 항상 최적해를 찾으며 (Gelperin, 1977; Nilsson, 1968), A*의 골격은 <표2>와 같다 (Nilson, 1980).

TSP의 경우에는 여행자가 출발지에 있는 상태가 1번 라인에서 탐색그래프의 root node가 된다. A* 알고리즘에서 노드는 OPEN과 CLOSE라는 두 개의 리스트에 나뉘어 기록된다. OPEN 리스트의 노드는 아직 자손 노드를 생성하지 않은 노드들이다. 4번 라인에서 OPEN 리스트의 첫 번째 노드를 선택하여 CLOSE 리스트로 보내면서 이 노드의 자식 노드들을 생성하고, 7번 라인에서 이들을 연결한다. A* 알고리즘에서 가장 중요한 역할을 하는 것은 8번 라인에서 OPEN 리스트에 있는 노드들을 가장 promising한 것부터 차례로 정렬하는 것이다. 초기 노드에서 그 노드에 이르는데 소요된 비용과, 그 노드에서 목적 노드에 이르는데 드는 비용의 예측치의 합이 가장 적은 노드가 가장 promising한 노드이다. 이때, A* 알고리즘의 효율성을 결정하는 것은 예측치를 산출하는 방법의 정확성이다.

TSP는 네트워크 상에 포함된 모든 도시를 한번씩 방문하고 제자리로 돌아오는 것을 원칙으로 한다. 그러므로 여행 경로는 방문지 각각에 대하여 한번씩은 들어가고, 한번씩은 떠난다. 따라서, 여행에 드는 비용은 거리 행렬의 각 행의 최소값과 각 열의 최소값의 총합 보다는 작지 않다. 이러한 성질을 이용하여 8번 라인에서 노드의 promising 정도를 평가하는 방법이 Horowitz(1984)에 소개되었다.

TSP 문제는 기본적으로 출발 도시와 종착 도시가 같다고 가정한다. 그러나 실생활에서는 출발지와 종착지가 다른 경우가 발생할 수 있다. 예를 들면 서울로 입국하여 렌트카를 빌려 관광을 하다가 김해 공항에서 출국하는 수가 있을 수 있다. 본 시스템은 출발지와 종착지가 동일할 경우는 물론 다를 경우에도 최적 경로를 제공하여 준다. 전자의 경우에는 기존 (Horowitz, 1984)의 방법을 그대로 적용한다. 후자의 경우에는 출발 도시로 들어올 수 없도록, 종착 도시에서 나갈 수 없도록, 그리고 여행 도중에 종착 도시로 들어가는 일이 없도록 미리 방지 함으로써 최적 경로를 찾아 준다.

제안된 방법을 예를 들어 설명하면, 출발 도시 A, 종착 도시 E, 경유 도시 B, C, D, 각 도시간의 거리는 표3의 무게행렬과 같다고 하자. 제안된 방법은 우선 <표

4>에 보이는 바와 같이 A열과 E행에 ∞를 넣어 A도시로 들어오는 경로나 E 도시에서 나가는 경로를 선택할 수 없도록 한다.

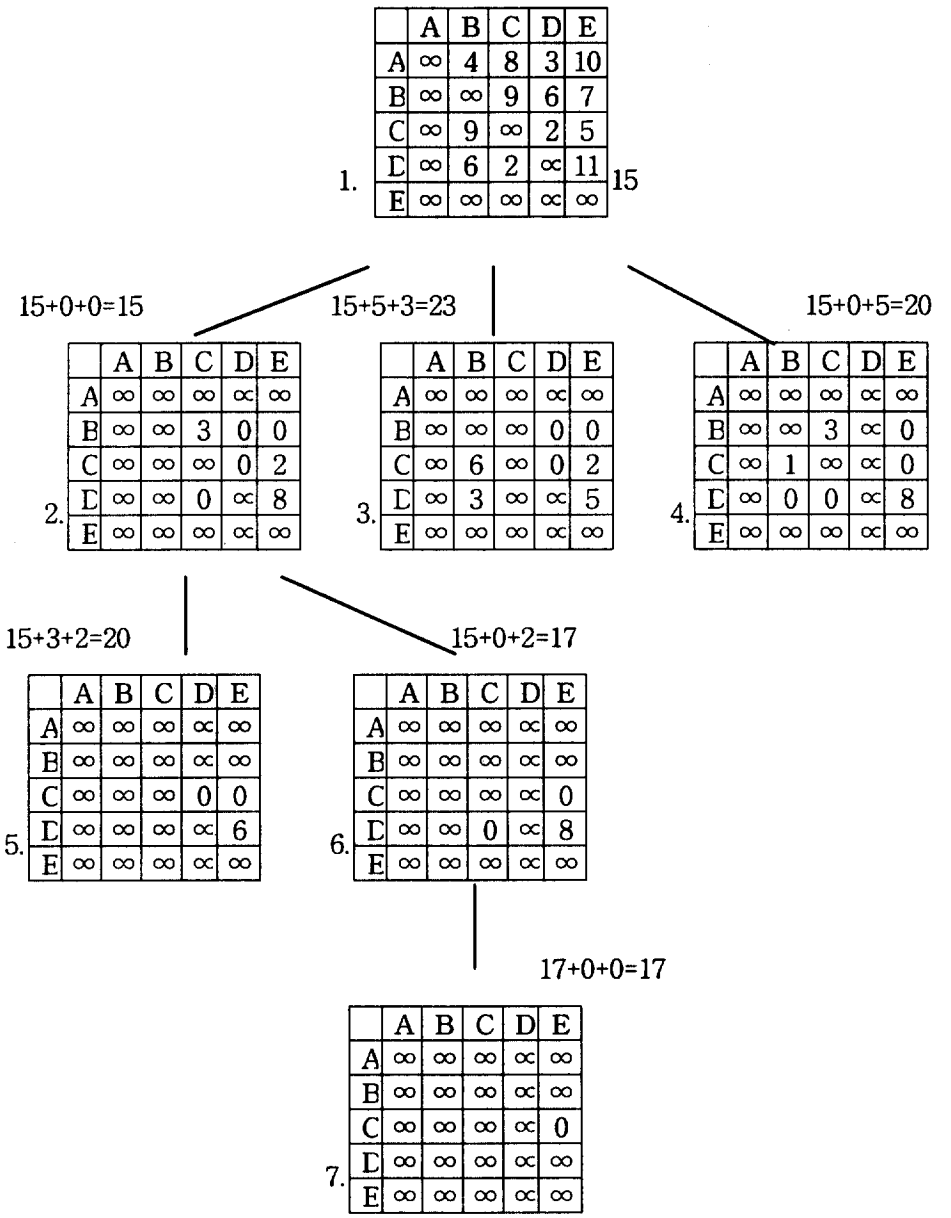
	A	B	C	D	E
A	∞	4	8	3	10
B	4	∞	9	6	7
C	8	9	∞	2	5
D	3	6	2	∞	11
E	10	7	5	11	∞

<표 3> 가상의 다섯 개 도시의 거리 행렬

	A	B	C	D	E
A	∞	4	8	3	10
B	∞	∞	9	6	7
C	∞	9	∞	2	5
D	∞	6	2	∞	11
E	∞	∞	∞	∞	∞

<표 4> 출발 도시와 종착 도시 표시.

모든 도시를 한번씩 반드시 통과하는 경로의 총비용은 최소한 각 도시에서 다른 도시로 나가는 거리 중 최단거리들과 각 도시로 들어오는 거리중 최단 거리의 합보다 작지않다. 즉, 도시 A에서 출발하여 다른 도시까지 가려면 최소 3 단위의 비용이 들고, 이 최소한의 비용을 사용하였다고 가정하면 도시 A에서 B, C, D, E까지 가는 거리로 각각 1, 5, 0, 7이 남는다. 마찬가지 방법으로 B에서 나가는 길의 거리는 최소 6, 이 비용을 사용하고 나면 B에서 C, D, E로 가는 비용은 각각 3, 0, 1이 남는다. C에서 나가는 길은 최소 비용이 2이고, 이 비용을 소비하고 나면 C에서 B, D, E로 가는 비용은 각각 7, 0, 3이 남는다. 마찬가지로 D에서 나가는 길은 최소 비용이 2이고, 이 비용을 소비하고 나면 B, C, E로 가는 비용은 각각 4, 0, 9가 남는다. 다음에는 각 도시로 들어오는 길의 최소 비용을 고려하여 보자. 위의 과정을 마치고 나면 B열의 entry는 각각 1, ∞, 7, 4, ∞로 B로 들어오는 길의 비용은 1이 되고, 이 비용을 소비하였다고 가정하면 남는 비용은 각각 0, ∞, 6, 3, ∞로 된다. 이 방법을 C, D, E에 적용하면 이들 도시에 들어가는 비용이 각각 최소 0, 0, 1인 것을 알수 있다. 그러므로 <표4>의 행렬에서 원하는 경로의 최소 비용은 3+6+2+2+1+0+0+1 = 15이다. 이 비용을 사용하고 나면 각 비용은 <표5>의 root node(node 1)에 보이는 바와 같아진다.



<표 5> 출발 도시와 종착 도시가 상이한 경우 제안된 방법에 의한 최적 경로 탐색 공간.

도시 A에서 갈 수 있는 곳은 B, C, D, E 네 곳이지만 E는 반드시 다섯 번째 도시어야 함으로 제외한다. 노드1에 나타난 도시 A와 B간의 거리는 0이다. 도시 A에서 B로 가는 경로가 선택되었음으로 A행과 B열에 ∞를 넣음으로써 또 다시 나가(들어오)는 경우를 선택하지 않도록 한다. 이렇게 얻은 행렬이 노드2이다. B, C, D 행과 C, D, E 열에 0 entry가 있으므로 이들 도시로 들어가거나 나가는 데 드는 최소비용의 총합은 0이다. 그러므로 노드2의 경로를 선택한 경우 원하는 경로의 비용은 최소 15로 예상된다 비슷한 방법으로 C나 D를 선택한 경우가 각각 노드 3과 4이며 이때의 최소 비용은 각각 15+5+3=23과 15+0+5=20이다.

예상되는 최소비용이 가장 작은 노드 2를 expand하면 노드 5와 6이 생성된다. 이때도 역시 도시 E로 가는 선택은 제외한다. 노드5와 6에서 예상되는 최소 비용은 각각 15+3+2=20과 15+0+2=17이다. 지금까지 OPEN 리스트에 수록된 노드는 노드 3, 4, 5, 6이며, 이들 노드에서 예상되는 최소비용들 중 최소값은 17이다. 따라서, 노드6을 expand하면 예상되는 비용이 17인 노드 7을 생성하며, 이것을 다시 expand하면 A, B, D, C, E의 순서로 여행하라는 경비가 17인 최적해를 구하게된다.

본 시스템은 사용자의 출발지, 종착지 및 방문지를 대화식으로 입력 받고, 최적경로를 지도위에 그려 주어야 한다. 지도를 동적으로 그리려면 자바애플릿을 사용하여야 한다.

Ⅲ. 연구방법

1. 연구방법 및 도구

연구 방법과 도구로는 그래프 이론과 JAVA 언어를 들 수 있다. 그래프 이론을 이용하여 각 도시간의 최단 경로 및 거리를 구하고 또한 주어진 도시를 한번씩 방문하고 돌아 오는 최단 거리 여행 경로를 구한다. 한편 이러한 이론을 인터넷 상에서 서비스를 제공하도록 구현하기 위하여 JAVA 언어를 사용한다.

즉, 지도에 나타나는 각 시.군에 대해 바로 이웃한 도시간의 거리를 측정하였고, 이 데이터로부터 Dijkstra's Algorithm을 사용하여 모든 쌍의 도시간의 최단.최소비

용거리를 구하였으며, 방문하고자 하는 도시가 주어지면, 이들 도시간의 최단.최소 비용거리에 A* 알고리즘을 적용하여 주어진 도시들을 방문하는 최적 경로를 찾는 프로그램을 JAVA 언어를 사용하여 구축함으로써 인터넷 상에서 활용 가능토록 하였다.

2. 연구내용

연구내용은 시군간의 거리구하기와 사용자의 출발 도시와 도착 도시, 그리고 방문지를 질의하고, 이들 간의 거리로부터 최적 경로를 구하고, 이 경로를 동적으로 화면에 표현하여 주는 JAVA 프로그램으로 구분된다.

Ⅳ. 시스템의 구현 및 결과 해석

1. 시스템 구현

1) 시군간 거리 구하기.

각 도시간의 거리를 구하기 위하여 우선 각 도시에 대한 이웃 도시까지의 거리를 지도에서 찾았다. 이렇게 구한 거리의 일부를 <표6>에 보인다. <표6>에서 보는 바와 같이 소수 첫째 자리까지 km를 단위로 구하였다. 거리 행렬을 float type으로 하면 기억 장소를 너무 많이 필요로 함으로, 정수 type으로 선언하고 거리의 단위를 100m로 표시한다.

<표 6> 이웃하는 도시 및 관광지 간의 거리

울진	평해 44.9	울진 불영사 15.4	울진 봉화 90.5		
평해	영덕 13.3	평해 영양 52			
영덕	주왕산 35	영덕 청송 54.9	영덕 홍해 38.7		
청송	주왕산 5.2	청송 영양 36.6	청송 안동 51.6	청송 의성 49	청송 안강 60

주왕산 안강 54.8
포항 흥해 9 포항 안강 16.8 포항 구룡포 24.1 포항 경주 29.9
경주 건천 11.4 경주 불국사 15.4
건천 운문사 45.9 건천 영천 26
운문사 청도 55.9
청도 경산 24.7 청도 고령 78
대구 경산 17.2 대구 성주 28.2 대구 왜관 19.6 대구 군위 48.3
성주 고령 34.2 성주 왜관 26.1
왜관 구미 19.4
선산 구미 32.8 선산 김천 22.1 선산 상주 30.7 선산 군위 61.7 선산 예천 70.4
상주 김천 27.3 상주 점촌 22 상주 예천 50.2
점촌 문경 22.7 점촌 예천 28
예천 봉정사 30 예천 영주 26.9 예천 안동 32.1
안동 봉정사 18.1 안동 도산서원 18.3 안동 의성 31.4
영주 봉화 15.1
봉화 부석사 22
... 이하 생략

<표1>에 소개된 Dijkstra algorithm에 <표6>을 입력으로 사용하여 각 시군간의 거리를 구한다. 이 프로그램의 결과는 JAVA 프로그램의 인수로 사용된다. 그러나 이 결과는 시간에 따라 변화하는 것이 아니므로 굳이 JAVA로 구현되어야 할 필요가 없다. 그러므로 본 알고리즘은 C 언어로 구현하였다.

2) JAVA 프로그램 구현

본 시스템은 초기화면에 한국 지도, 출발지, 목적지, 경유지들을 선택 할 수 있는 Checkbox와, 선택이 끝났으니 계산 결과를 보여달라고 요구하는 버튼들을 보인다. 애플릿은 우선 init를 호출하고, 계속해서 start와 paint메세지를 비동기적으로 호출한다. 그럼으로 init에서 다음과 같은 명령으로 그림 파일을 image로 받고 이를 paint에서 화면에 출력시키도록 한다.

Image img = getImage(getDocumentBase(),"jido.gif");

여기서 getDocumentBase는 현재 애플릿을 호출하는 HTML파일의 URL(Universal

Resource Locator)을 return하는 함수로 "()"는 인수가 없음을 뜻하는 데, image가 HTML파일과 같은 URL에 있으므로 이 메소드를 사용한다.

지도를 paint에서 출력시키는 이유는 지도위에 도시의 위치와 경로를 덧 그릴 때 repaint하기 위함이다. 다음은 choiceTest라는 이름의 panel을 만들어 출발지와 목적지를 표시할 수 있도록 하고, 도시명을 레이블로 하는 Checkbox의 배열을 한 개의 panel에 담아 보임으로써 경유지들을 선택 할 수 있도록 한다. 그리고 버튼을 사용하여 계산을 요구하도록 한다. 이들을 BorderLayout 클래스에 <표7>과 같은 방법으로 배치시킨다. 여기에서 p2는 계산을 요구하는 버튼이고, choiceTest는 출발지와 목적지를 선택하는 Flowlayout이며 출발도시 이름과 목적지 이름을 선택 대상으로 각각 나열하여 주는 두 개의 Choice로 구성된다. 여기에서 "North"는 이것을 컨테이너의 위쪽에 위치시키라는 것이다. CheckboxTest는 150여개 도시 이름 각각을 레이블로 하는 Checkbox를 같은 크기의 격자 안에 배치시켜 만들어진 GridLayout인 panel이다.

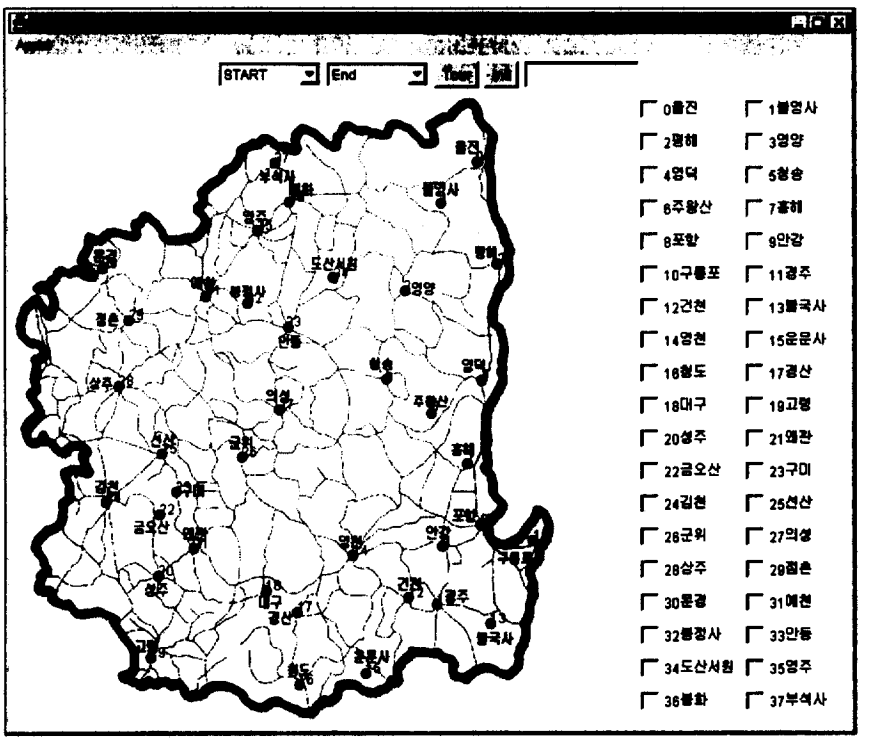
<표 7> 초기화면

```
setLayout(new BorderLayout());
add("South", p2);
add("North", choiceTest());
add("East", CheckboxTest());
```

애플릿에서, 미리 준비된 이미지(gif/jpeg)를 출력하기 위해서는 우선 Applet 클래스에 있는 getImage()메쏘드로 이미지 파일을 로드한 뒤에 Graphics 클래스에 있는 drawImage()를 사용한다. paint() 메쏘드내에서 화면출력 부분은 크게 아래와 같이 초기화면 그리기와 계산 결과 최적 경로를 그려주는 부분으로 나누어진다.

```
public void paint(Graphics g)
 {    // 함수의 시작이라는 표시
   if(init_flag == 0){  // block의 시작이라는 표시
    ...../// 초기 또는 initialize 버튼이 눌려진
   }          // 경우에 출력되어 그려지는 부분
   else if(init_flag == 1){
```

```
.....// touring 버튼이 눌려져 입력 데이타의
}       // 처리결과를 그려주는 부분
} // 함수의 끝이라는 표시
```



자바 프로그램은 사용자가 컴포넌트를 선택함에 따라 이를 처리하는 action method가 작동함으로써 수행된다. instanceof는 어떤 컴포넌트 객체가 활동을 시작하였는지 판별하여 준다.

```
public boolean action(Event evt, Object arg)
{
  if(!(evt.target instanceof Checkbox) && !(evt.target instanceof Button) &&
     !(evt.target instanceof Choice)){
          return false;
  }
  else if(evt.target instanceof Choice){
```

```
            ... // 초이스버튼 선택 이벤트 처리 부분
    }
    else if(evt.target instanceof Button){
            ... // 최적 경로 구하라는 버튼 선택 이벤트 처리 부분
    }
  return true;
}
```

버튼, 메뉴와 같은 컴포넌트들은 레이블을 가지는 데, 이 레이블로 이벤트 발생 객체를 참조할 수 있다. 레이블은 이벤트 객체의 인스턴스 변수 arg에 지정된다. 따라서 equals() 메쏘드를 이용하여 비교한다.

```
public boolean action(Event evt, Object arg)
{
 .....
 else if(evt.target instanceof Choice) //초이스버튼에서 event가 발생했을 경우
 {
     if(evt.target.equals(choice1)){          // 출발도시를 선택하는 초이스일 경우
     .....
  }
  else if(evt.target.equals(choice2)){    // 도착도시를 선택하는 초이스일 경우
      .....
  }
```

초이스 버튼과 체크박스에서 선택된 값은 keyPath[]에 저장 되어 있다. 이를 이용해, <표1>의 알고리즘 적용 결과 만들어진 전체 150여개 도시간 최단 거리가 저장되어진 157×157 행렬인 matrix[][]에서 해당되는 row를 먼저 차례대로 고른 다음, 선택된 row에서 keyPath[]에 해당하는 column을 다시 선택한다. 이렇게 해서 선택된 출발지와 경유지, 그리고 도착지의 거리 data가 만들어진다.

```
select_row[i] = matrix[keyPath[i]-1];
    .....
select_col[j][i] = select_row[j][keyPath[i]-1];
```

이때 출발지와 목적지가 다를 경우 출발지에서 들어오는 값과 도착지에서 나가는 값을 없애기 위해, 해당 출발지의 column과 해당 목적지의 row에 -1값을 넣어준다.

2. 결과해석

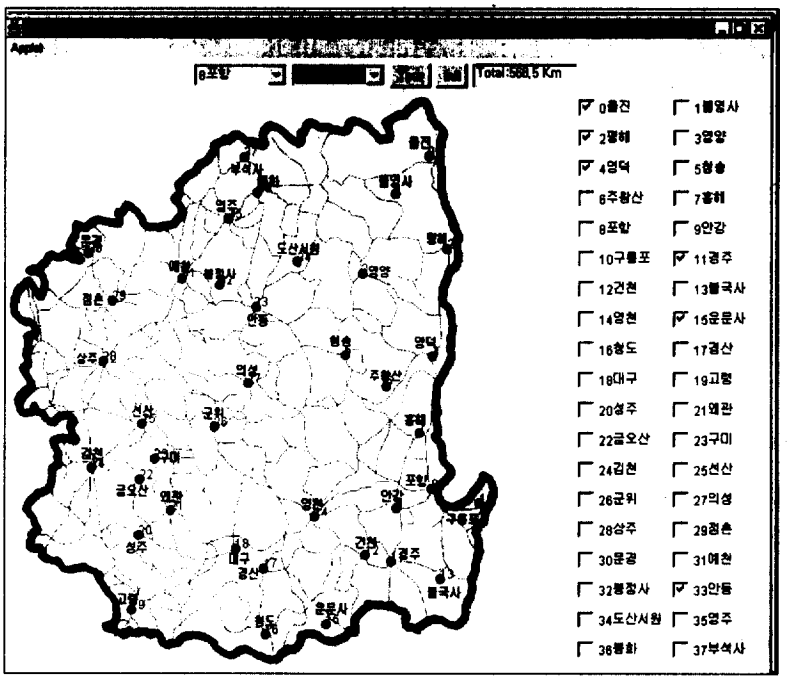
<그림1>은 본 APPLET WWW 페이지인 http://wwwcs.dongguk.ac.kr/~yim /tsp.html을 UNIX상에서 Netscape웹브라저를 이용해 불러들인 후 초기화가 끝난 모습이다. 왼쪽에는 우리나라 지도가 보이고 오른쪽에는 모든 도시의 이름이 나열되어 있다.

이 APPLET을 사용하기 위해 먼저 APPLET의 최상단에 위치한 버튼 중 왼쪽의 START 초이스버튼을 누르면 도시명들이 나타나는데 그 중에서 출발지를 선택한다 .비슷한 방법으로 START 옆의 DESTINATION이라고 레이블된 초이스버튼을 클릭하여 목적지를 선택한다.

오른쪽에는 157개의 도시들이 GridLayout으로 배치되어 있는데 그 중에서 여행하고 싶은 도시들을 두 곳 이상 선택한다. <그림 2>는 위의 모든 설정이 끝난 모습을 보여주는 그림이다. <그림 2>에서 경유지로 선택된 36 Namwon(남원), 47 Mokpo(목포), 102 Ulsan(울산), 124 Jindo(진도)그리고 145 Pohang(포항)은 체크박스가 까맣게 칠해져 있음을 알 수 있다.

최종 결과를 보기 위해 touring버튼을 누른 결과가 <그림 3>에 보인다. 선택된 도시들이 선으로 연결되어 있음을 알 수 있다. 이로부터 서울을 출발하여 목포, 진도, 남원, 포항, 울산을 차례로 거쳐 부산으로 가는 것이 최적 경로임을 알 수 있다.

본 논문에서는 최적 경로를 제작하는 방법을 소개한다. 소개된 방법은 전자계산학 분야에서 널리 알려진 (Skvarcius, 1980)과 (Horowitz, 1984)을 기반으로 한다. 단 (Horowitz, 1984)에 소개된 방법은 출발지와 목적지가 같은 경우에만 최적해를 구하는 데 반하여, 본 논문에 제안된 방법은 출발지와 목적지가 다를 경우에도 최적해를 구 할 수 있다. 본 논문에 제안된 방법을 요약하면 표8과 같다.



<표 8> 본 논문은 최적 경로를 찾는 제안된 방법

1. 원시데이터로부터 도시간의 근접행렬을 구한다. (근접행렬: 이웃한 도시의 거리가 명시된 행렬)
2. 근접행렬에 <표1>의 알고리즘을 적용하여 모든 쌍의 도시간의 최단거리 및 최적경로를 구한다. 최단거리를 기록한 행렬을 무게행렬이라 한다.
3. 사용자로 하여금 방문을 원하는 n 개의 도시를 선택하도록 하고, 선택된 도시만의 무게행렬을 구한다.
4. 출발지와 목적지가 같으면 단계 3에서 구한 무게행렬을 reduce한 결과를 근노드로 하고, 여기에 (Nilsson, 1980)에 소개된 A^* 알고리즘을 적용함으로써 최적경로를 구한다. 이때 (Horowitz, 1984)에 소개된 reduced matrix 방법을 heuristic 함수로 사용한다.
5. 출발지 A와 목적지 B가 같지 않으면 단계3에서 구한 무게행렬의 A 열과 B 행을 무한대로 초기화하여 여기에 단계4를 수행한다. 단 A^* 알고리즘은 도시 B로 가는 자식노드를 생성하지 않도록 변형한다.

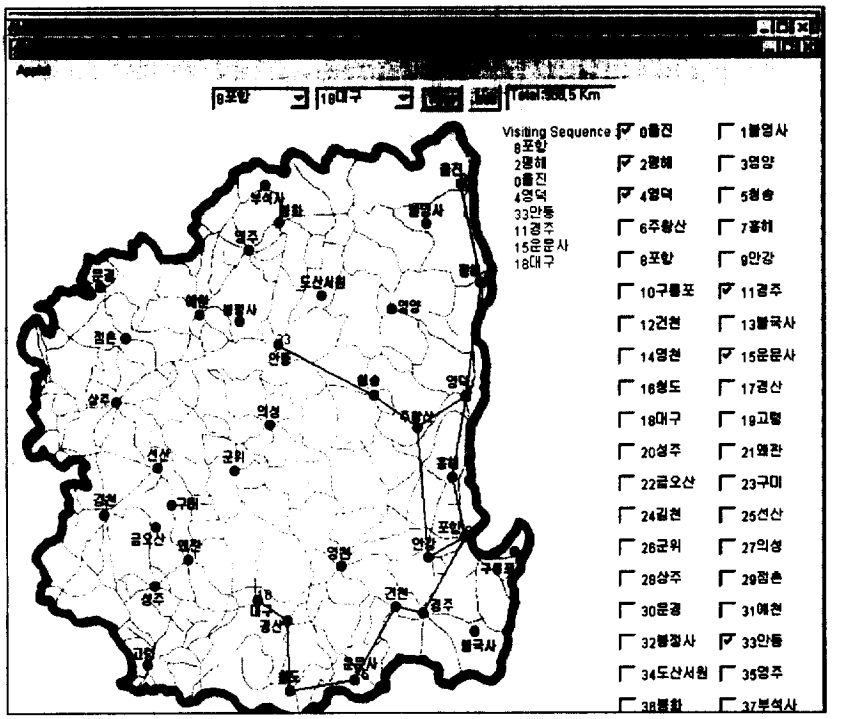
<정리 1> <표 8>에 제안된 방법은 최적경로를 구한다.

<증명> 도시간의 근접행렬에서 무게행렬을 구하는 소개된 방법의 타당성은 참고문헌 (Skvarcius, 1980)에 소개되어있다. 참고문헌 (Nilsson, 1968)은 사용되는 heuristic 함수가 실제치보다 항상 작을 때 A^* 알고리즘은 최적해를 찾는다는 사실을 증명하였으며, 출발지와 목적지가 같은 경우에 사용할 수 있는 heuristic 함수로 reduced matrix 방법이 (Horowitz, 1984)에 소개되었으므로 제안된 방법의 단계4는 출발지와 목적지가 같은 경우에 최적해를 찾는다.

출발지 A와 목적지 B가 다른 경우에, 제안된 방법은 A 열과 B 행를 무한대로 초기화 한다. 이것은 출발지 A로 들어오는 모든 길의 비용을 무한대로 하여 주고, 목적지 B에서 나가는 비용을 무한대로 하여 줌으로써 출발지 A로 돌아오거나 목적지 B로 들어오는 일이 없도록 함으로 단계 5는 출발지와 목적지가 다른 경우에 최적해를 구한다.

제안된 방법의 실용성을 보이기 위하여 본 논문은 국도를 여행하는 여행자에게 최적 여행 경로를 제공하는 인터넷 홈페이지 작성을 예로 보였다. 그러나 제안된 방법이 국도 데이터만 다룰 수 있는 것은 아니다. 국도 뿐만 아니라 고속도로나 지방도까지 고려한 자동차 여행에 필요한 시간을 원시데이터로 하면, 제안된 방법으로 시간을 최적화하는 최적경로를 구 할 수 있다. 원시데이터로 여행에 필요한 경비를 사용하면 제안된 방법은 경비를 최적화하는 최적 경로를 구한다.

정보화 사회에서 가장 유망한 산업 형태의 하나가 정보제공 산업이다. 정보 제공 산업 형태 중 가장 손쉬운 것은 유용한 정보를 인터넷에 올려 놓고 유료 서비스를 제공하는 것이다. 이를 위하여 다량의 원시데이터를 매력있게 인터넷에 올려 놓고, 사용자의 필요에 맞게 원시데이터를 가공하는 기술은 더욱 필요하다. 본 논문은 원시데이터를 최적화라는 측면에서 가공하는 방법을 소개하였다.



V. 결론 및 시사점

본 논문은 출발지와 종착지 그리고 중간에 방문할 도시를 선택하면 자동차로 국도를 이용하여 가장 짧은 거리를 달려 선택된 도시들을 모두 한번씩 방문하고 종착지에 도착하는 경로를 발견하여 주는 WWW homepage 작성 사례를 발표하였다. 이를 위하여 우선 각 도시에 대한 이웃 도시까지 국도의 거리를 모두 조사하였다. 그리고 이 거리 데이터에 두 정점간의 최단경로를 찾는 알고리즘을 적용하여 모든 도시간의 최단 경로를 구하였다. 그리고 출발지와 목적지가 같거나 다르거나 두 가지 경우를 모두 다룰 수 있는 최적 경로를 찾는 알고리즘을 제시하였다.

다가오는 21세기는 정보화 사회임을 부정하는 사람은 아무도 없다. 따라서 관광분야의 정보화 또한 시급한 문제임은 누구나 공감하는 바이다. 정보화의 최첨단 기술은 인터넷을 이용한 정보 공유 방법인데, 이러한 기술 중에서 관광 분야 종사

자가 사용할 수 있는 기술은 인터넷을 통한 관광 정보 제공이다. 인터넷 상의 관광 정보 제공의 예로 http://www.kyoungbuktour.or.kr과 http://www.bulguksa.or.kr을 들 수 있다. 관광 정보를 인터넷에 제공 할 때, 가공되지 않은 원시데이터 (예를들어 한국지도, 각 고장의 명승지 그림, 명승지 설명서 등의 각종 멀티미디어 데이터)를 올리는 것은 그리 큰 문제가 되지 않는다. 그러나 원시데이터를 조작하여 사용자의 요구에 맞는 가공 데이터를 제작하여 제공하는 것은 매우 어려운 일이다. 이러한 가공데이터 중 가장 많이 사용되는 것이 최적 문제와 관련이 있다.

정보화 시대를 맞이하여 정보가 곧 경제력인 시대가 되었다. 이에 부응하여 관광 정보를 산업화하려는 국가적인 시도가 태동하고 있다 (예를들어, "동국대학교 관광경영학과 경주 관광 정보 시스템"을 경상북도 지정 우수 창업동아리로 선정하여 금전적으로 지원.) 관광 정보를 보유하고 있는 쪽은 당연히 관광 종사자들이다. 그러나 관광 종사자는 정보를 가공하는 능력이 부족하여 원시 데이터를 매력있게 제공하는 데에만 노력을 경주하여 왔다. 본 논문에서는 원시데이터에 최적화 기법을 적용하여 사용자의 요구에 맞게 가공하는 방법을 소개하였다. 또한 소개된 방법을 국도 여행에 적용하여 보았으나 다음 단계에서는 모든 교통수단을 동원하였을 경우, 즉 철도, 항공까지 연구영역을 확대하여 연계한다면 가장 단시간에 여행할 수 있는 경로 및 수단을 찾아 줄 수 있을 것으로 기대된다. 앞으로 관광 정보의 산업화가 활성화될 것을 예상하며, 본 논문이 여기에 일조 하기 바란다.

참고문헌

경상북도 지정 우수 창업동아리 (1998). "동국대학교 관광경영학과 경주 관광 정보 시스템"

김병문 (1986). 『관광지리학』. 서울: 형설출판사.

김영문 · 채수원(1996). 관광지선택에 있어서 AHP의 활용에 관한 연구. 『관광학연구』, 20(1): 63~81.

김원인(1994). 관광지선호에 관한 실증적 연구. 『관광학연구』, 18(1): 1~22.

김종은 · 길용현(1988). 『관광지리학』. 서울: 집문당.

김향자 · 엄서호(1997).휴가목적지 선택결정 요인으로서 지각행동조절에 관한 연구. 『관광학연구』, 21(1): 11~29.

서태양 (1996). 『관광학원론』, 서울: 법문사.

이태희 (1998). 관광객의 신기성 욕구수준에 따른 관광목적지 선호도 차이에 관한 연구. 『관광학연구』, 22(1): 9~20.

Ahmed, Z.U. (1991). The influence of the components of a state's tourist image on product positioning strategy. *Tourism Management*, 12(4).

Frank, G.B. (1984). Positioning as a probrem-solving tool. *HSMA Marketing Review*.

Gelperin, D. (1977). On the optimality of A*. *Artificial Intelligence*, 8(1): 69-76.

Haahti, A. J. (1986). Finland's competitive position as a destination. *Annals of Tourism Research*.

Hart, P.E., Nilsson, N.J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Science and Cybernetics*, SSC-4(2): 100-107.

Horowitz, E. & Sahni, S. (1984). *Fundamentals of Computer Algorithms*, Computer Science Press.

http://www.kyoungbuktour.or.kr

http://www.bulguksa.or.kr

Kaspar, C. (1984). Strategy of success position for tourism. review de tourism, AIEST, No. 2.

Nilsson, Nils J. (1980). *Principles of Artificial Intelligence*, Tioga Publishing Company.

Skvarcius & Robinson (1980). *Discrete Mathematics with Computer Science Applications*, The Benjamin/Cummings Publishing Company, Inc..

ABSTRACT

A study on Implementation of a Web Site Providing Optimal National Road Touring Routes Using a Heuristic Search Algorithm

Seo, Tai-yang · Yim, Jae-geol

We have implemented a WWW homepage which finds an optimal touring route for users assuming the user would drive for himself on the national roads. Our home page asks a user to select all the cities he/she wants to visit(including starting city and destination city). It finds a sequence of the cities so that the user would travel minimum distance if he/she visits cities in the order of the sequence. This paper presents algorithms used in the homepage. Existing heuristic algorithms to find optimal path assume that the starting city and the destination city are the same. Our algorithm is also a heuristic algorithm, yet it is free from the assumption.

3인 익명 심사 필
2000년 5월 31일 논문접수
2000년 8월 2일 최종심사