One-to-One 최단경로 알고리즘의 성능 평가
(Performance Evaluation for One-to-One Shortest Path Algorithms)

심 충 섭 † 김 진 석 ††
(Chung Sup Sim) (Jin Suk Kim)

요 약 최단 경로 탐색 알고리즘 (Shortest Path Algorithm) 은 출발지에서 목적지에 이르는 여러 경로 중에서 가장 경제적이고 효율적인 경로를 찾는 알고리즘으로 레이블링 기법에 기초하고 있다. 레이블링 기법에는 레이블 고정 (Label-Setting) 기법과 레이블 수정 (Label-Correcting) 기법이 있다. One-to-One 최단 경로 탐색 알고리즘에서 레이블 고정 기법이 빠르다고 알려져 왔으나 최근 연구에서 대용량 도로 데이타에 대한 실험을 통해 레이블 수정이 레이블 고정보다 탐색 시간이 빠름을 보였다[1,2]. 레이블 수정 기법 중에서 가장 속도가 빠른 것은 그래프 성장 (Graph Growth) 알고리즘인데, 이 알고리즘은 One-to-All 방식을 사용하고 있으므로 One-to-One 최단 경로 탐색에는 적합하지 않다. 본 논문에서는 One-to-One 방식을 사용하는 새로운 알고리즘을 제안하였고, 실험결과 그래프 성장 알고리즘의 성능에 비해 새로 제안된 알고리즘의 성능이 30~40% 향상되었음을 알 수 있었다.

키워드 : 최단경로알고리즘, Graph Growth, One-to-One, Label-Correcting

Abstract A Shortest Path Algorithm is the method to find the most efficient route among many routes from a start node to an end node. It is based on Labeling methods. In Labeling methods, there are Label-Setting method and Label-Correcting method. Label-Setting method is known as the fastest one among One-to-One shortest path algorithms. But Benjamin[1,2] shows Label-Correcting method is faster than Label-Setting method by the experiments using large road data. Since Graph Growth algorithm which is based on Label-Correcting method is made to find One-to-All shortest path, it is not suitable to find One-to-One shortest path. In this paper, we propose a new One-to-One shortest path algorithm. We show that our algorithm is faster than Graph Growth algorithm by extensive experiments.

Key words : Shortest Path Algorithm, Graph Growth, Label-Correcting

1. 서 론

최단 경로 알고리즘 (Shortest Path Algorithm) 은 교통 및 네트워크 분석 관련 문제에서 매우 중요하게 사용된다. 최단 경로는 출발지와 목적지 사이의 여러 경로 중에서 가장 경제적이고 효율적인 경로를 나타낸다.

최단 경로 알고리즘은 크게 하나의 출발 노드에서 다른 모든 노드까지의 최단 경로를 구성하는 방법과 모든 노드 사이의 최단 경로를 구성하는 방법으로 구분한다. 이러한 최단 경로 알고리즘을 풀기 위한 방법으로는 그래프에서 링크로 연결된 노드들의 비용을 추적하여 최단 경로를 찾아내는 레이블링 기법에 기초하고 있다[3].

레이블링 기법은 크게 레이블 고정 (Label-Setting) 기법과 레이블 수정 (Label-Correcting) 기법으로 나눌 수 있다. 레이블 고정 기법은 출발 노드에서 목적 노드에 이르기까지 목적 노드가 찾아지고 영구적인 표지가 되었을 때 최단 경로가 결정되며, 레이블 수정 기법은 출발 노드에서 목적 노드에 이르는 마지막 단계에 임시로 표지된 모든 노드들을 평가하여 최단 경로가 결정된다[3,4]. Benjamin은 노드의 개수가 많은 GIS (Geographic Information System) 도로 데이타를 사용할 경우에 레이블 수정 기법이 레이블 고정 기법보다

† 비 회 원 : (주)디지웨이브
cssim95@venus.uos.ac.kr
†† 종신회원 : 서울시립대학교 컴퓨터통계학과 교수
jskim@venus.uos.ac.kr
논문접수 : 2002년 3월 22일
심사완료 : 2002년 9월 11일

더 빠르게 수행된다는 것을 실험으로 보였다[1].

본 논문에서는 One-to-One 최단 경로 탐색에 적합한 알고리즘을 제안한다. 본 논문에서 제안한 알고리즘은 레이블 수정 기법 중에서 가장 좋은 성능을 보이는 Graph Growth 알고리즘을 기반으로 하였다.

본 논문은 다음의 순서로 구성되어 있다. 제 2장에서는 최단 경로 알고리즘에 대해서 알아본다. 제 3장에서는 본 논문에서 제안한 알고리즘을 소개하며 제 4장에서는 제안한 알고리즘과 기존의 알고리즘의 성능을 실험을 통하여 비교 분석하였다. 제 5장에서 결론을 맺고 논문을 마친다.

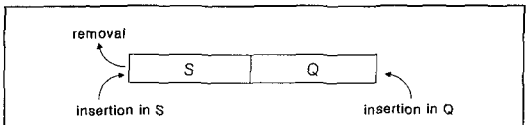
2. 두 개의 큐를 이용한 Graph Growth 알고리즘

Graph Growth[5] 알고리즘은 <그림 1>에 나타나 있다(여기서 ***d(s,i)*** : 노드 ***s***에서 노드 ***i***까지의 거리값, ***link(i,j)*** : 노드 ***i***와 노드 ***j***가 연결된 링크값이다).

```
Input : s(start node), t(target node)
Output : shortest path lists

Procedure ShortestPath_GraphGrowth(s,t)
begin
  Queue_Initialization(Q);
  for i=1 to n do
    d(s,i) = ∞;
  d(s,s) = 0;
  Queue_Insertion(Q,s);

  while (Queue ≠ Null) do
  begin
    Queue_Removal(Q,i);
    for each successor node j of node i do
      if d(s,j) > d(s,i) + link(i,j) then
      begin
        d(s,j) = d(s,i) + link(i,j);
        Queue_Insertion(Q,j);
      end
  end
end

Queue_Initialization(Q) : 큐를 초기화한다.
Queue_Removal(Q,i)      : 큐에서 노드를 제거하고 i에
                          저장한다.
Queue_Insertion(Q,j)    : 큐에 노드 j를 추가한다.
```

그림 1 Graph Growth 알고리즘[1]

Graph Growth 알고리즘이 사용한 자료구조는 deque이다. deque 는 스택과 큐로 구성되어 있으며 <그림 2>과 같은 구조로 만들어져 있다. deque 에서는 앞에 있는 스택에서 삽입과 삭제가 일어나며 뒤에 있는 큐에서 삽입만이 일어난다.

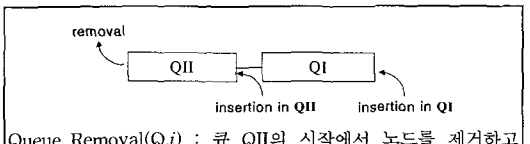


Queue_Removal(Q,*i*) : 스택 S의 top에서 노드 *i*를 제거한다. 만일 스택이 비었으면 큐 Q의 시작에서 제거한다.

Queue_Insertion(Q,*j*) : 만일 한번이라도 deque안에 있었으면 스택 *S*에 삽입하고 그렇지 않으면 큐 *Q*에 삽입한다.

그림 2 deque 자료구조[1,3]

Pallottino는 두 개의 큐를 이용한 Graph Growth 알고리즘을 제안하였다[6]. Pallottino 가 사용한 자료구조는 두 개의 큐로 구성되어 있으며 <그림 3>와 같은 구조로 만들어져 있다. 첫 번째 큐에서는 삭제만 가능하고 첫 번째 큐와 두 번째 큐에서는 삽입이 일어난다.



Queue_Removal(Q,*i*) : 큐 QII의 시작에서 노드를 제거하고 제거된 노드를 ***i***에 저장한다. 만일 큐 QII가 비었으면 큐 QI의 시작에서 제거한다.

Queue_Insertion(Q,*j*) : 만일 ***j***가 한번이라도 QI이나 QII에 있었으면 ***j***를 큐 QII에 삽입하고 그렇지 않으면 ***j***를 큐 QI에 삽입한다.

그림 3 two-queue 자료구조[1,3]

3. One-to-One 최단 경로 알고리즘

본 절에서는 새로운 최단 경로 알고리즘을 제안한다. 기존의 Graph Growth 알고리즘은 One-to-All 방식으로 설계되었기 때문에 시작 노드에서 다른 모든 노드까지의 최단 경로 탐색한다. 즉 목적 노드가 시작 노드와 가깝다 하더라고 최단 경로를 알아내기 위해서는 전체 노드를 탐색해야 한다는 단점이 있다.

본 논문에서 제안한 최단 경로 알고리즘은 시작 노드에서 목적 노드까지의 경로 값을 Threshold 값으로 줌으로써 Threshold 값 이상으로의 노드 탐색을 줄였다. 제안된 알고리즘의 자료구조는 Graph Growth 알고리즘에서 사용하였던 것과 같이 두 개의 큐를 이용하였으며, 알고리즘은 <그림 4>에 나타나 있다.

본 논문에서 제안된 알고리즘에서는 목적 노드가 검색되면 현재까지의 비용이 Threshold 값보다 작은지를 비교한다. 만일 현재까지의 비용이 Threshold 값보다 크

```
Input : s(start node), t(target node)
Output : shortest path list

Procedure ShortestPath_Algorithm(s,t)
begin
  Queue_Initialization(Q);
  for i=1 to n do
    d(s,i) = ∞;
  Threshold = ∞;
  d(s,s) = 0;
  Queue_Insertion(Q,s);

  while (Q ≠ Null) do
  begin
    Queue_Removal(Q, i);
    for each j adjacent to i do
      if d(s,j) > d(s,i) + link(i,j) then
      begin
        d(s,j) = d(s,i) + link(i,j);
        [redacted]
      end
  end
end
```

그림 4 새로운 최단 경로 알고리즘

면 더 이상 확장을 막기 위해 현재 노드를 큐에 삽입하지 않는다. 그렇지 않고 현재 노드의 비용이 Threshold 값보다 작고 우리가 찾는 목적 노드면 기존의 Threshold 값을 현재 노드의 비용으로 대치한다. 하지만 목적 노드를 찾았으므로 확장을 막기 위해 현재 노드를 큐 안에 삽입하지 않는다. 만일 현재 노드의 비용이 Threshold 값보다 작고 우리가 찾는 목적 노드가 아니라면 목적 노드를 아직 찾지 못했기 때문에 현재 노드를 큐 안에 삽입하여 그래프를 확장한다. 본 논문에서 제안한 알고리즘이 최단 경로를 찾는다는 것은 다음과 같이 증명될 수 있다[7].

S를 최단 경로가 이미 찾아진 정점들의 집합이라 하고 S에 있지 않은 정점 w에 대해 DIST(w)는 시작노드(v) 에서 출발하여 S에 있는 정점들만을 거쳐 w에서 도달하는 최단 경로라 하자.

(*i*) 만약 최단 경로가 정점 u로 가는 것이라면, 그 경로는 v에서 시작하여 u에서 끝나며 그 경로는 S에 있는 정점들만을 통과하게 된다. 이것을 증명하기 위해서는 u로 가는 최단 경로 상에 있는 중간 정점들이 모두 S에 있음을 보여야 한다. 만약 S에 포함되지 않은 정점 w가 이 경로에 있다고 가정해 보자. 본 논문에서 제안한 알고리즘은 최단 경로 값인 Threshold 값 이하의 모든 노드들을 탐색하기 때문에 정점 w를 지나는 경로가 이미 발견되어 있어야 한다. 그러므로 S에 없는 중간 정점은 존재할 수 없다.

(*ii*) 생성된 다음 경로의 종착점은 S에 없는 모든 정점 중에서 DIST(u)가 최소인 그런 정점 u가 되어야 한다. 이것은 DIST 의 정의와 (*i*)에 의해서 유도된다. S에 없는 정점 중에서 DIST가 서로 같은 정점들이 있다면, 그 중에서 아무거나 선택 할 수 있다.

(*iii*) Graph Growth 알고리즘은 알고리즘 수행을 마치는 순간 모든 노드들의 최단 경로를 찾게 된다. 본 논문에서 제안한 알고리즘도 Graph Growth 알고리즘을 이용하기 때문에 Threshold 값 이하로 검색되는 노드들에 대해서는 최단 경로를 찾게 된다.

본 논문에서 제안한 알고리즘과 Graph Growth 알고리즘은 둘 다 두 개의 큐를 이용하였기 때문에 Time Complexity는 $O(n^2m)$으로 동일하다. 여기서 n은 노드의 수이며 m은 링크의 수이다[1].

<그림 5>은 Graph Growth 알고리즘과 본 논문에서 제안한 알고리즘의 방문 노드 수를 비교해 본 것이다.

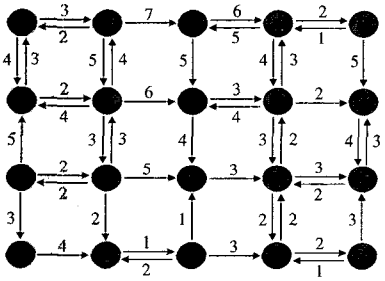


그림 5 네트워크 구성

<그림 5>에 나타난 그래프에서는 시작 노드를 2로 하고 목적 노드를 13과 19로 하였다. Graph Growth 알고리즘은 시작 노드 2에서 어떠한 목적 노드로의 최단 경로를 구성하더라도 노드 탐색 횟수는 22번이다 (2→1→3→7→6→4→8→12→5→9→13→11→17→10→14→16→18→13→14→15→19→20). 하지만 본 논문에서 제안된 알고리즘을 사용하면 목적 노드 13의 최단 경로를 구성하기까지는 13번의 노드 탐색이 필요하며 목적 노드 19까지의 최단 경로를 구성하기까지는 20번의 노드 탐색이 필요하다. 이것을 그림으로 표현하면 <그림 6>, <그림 7>와 같다.

<그림 6>의 최단 경로 탐색 과정은 다음과 같다. 먼저 모든 노드의 거리값과 Threshold 를 초기화한다(**for** $1 \le i \le 20$ **do** $d(2,i)=\infty$, $d(2,2)=0$, $Threshold=\infty$). 그리고 시작 노드 2를 큐($Q1$)에 삽입한다. 최단 경로 탐색은 큐에서 노드 2를 제거하여 노드 2에서 갈 수 있

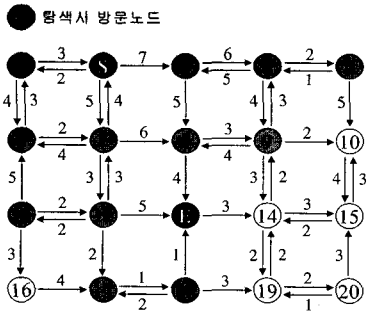


그림 6 최단 경로 탐색(2 ⇒ 13)

는 다음 노드 1,3,7의 거리값을 ***d(2,1)=2, d(2,3)=7, d(s,3)=5***로 갱신하면서 거리값이 수정된 노드 1,3,7들을 큐(***QI***)에 삽입한다. 다음으로 큐에서 한 노드를 제거한 후 제거된 노드 1에서 갈 수 있는 노드 2, 6을 비교한다. 노드 2의 거리값은 기존의 값보다 현재의 값이 크므로 수정하지 않고 노드 6의 거리값을 ***d(2,6)=6***으로 수정한다. 수정된 노드 6은 큐(***QI***)에 삽입하고 최단 경로 탐색 과정을 반복한다. 이러한 최단 경로 탐색 방법을 계속 반복하여 큐 안에 더 이상 탐색할 노드가 없으면 최단 경로 탐색을 중지한다. 노드 2에서 노드 13까지의 방문 노드 순서는 2→1→3→7→6→4→8→12→5→9→11→17→18이며 방문 노드를 그림으로 표현하면 <그림 6>와 같다.

노드 2에서 노드 19까지의 방문 노드 순서는 2→1→3→7→6→4→8→12→5→9→13→11→17→10→14→16→18→13→14→15이며 방문 노드를 그림으로 표현하면 <그림 7>과 같다.

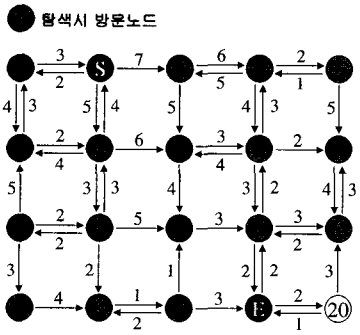


그림 7 최단 경로 탐색(2 ⇒ 19)

4. 실험 및 평가

4.1 실험 조건

본 논문에서 제안한 알고리즘을 테스트하기 위해 본 논문에서는 Pentium III 600MHz 프로세서와 메모리 128 MByte를 가진 컴퓨터에 RedHat Linux 6.2 기반의 gcc 컴파일러를 사용하여 알고리즘을 평가하였다. 실험한 테스트 셋은 크게 두 개로 나누었다. 첫 번째 테스트 셋은 노드 수가 7,506개이며, 링크의 수가 18,936개의 경주시지도이며, 두 번째 테스트 셋은 임의로 4,900개에서 40,000개까지의 노드를 갖는 지도이다. 본 실험에서는 최단 경로 탐색을 하기 위해 주어진 노드에서 임의로 100개의 노드를 시작 노드로 선택하고 각 시작 노드 당 각각 100개의 노드를 목적 노드로 하여 선택하여 10,000번의 최단 경로 탐색을 수행하였다. 그리고 최단 경로 탐색 수행 시간의 평균을 내어 제안된 알고리즘과 기존의 알고리즘을 비교하였다.

4.2 경주시 지도에서의 최단 경로 탐색

Graph Growth 알고리즘, Dijkstra 알고리즘, 본 논문에서 제안한 알고리즘의 실행시간을 비교한 것이 <표 1>에 나타나 있다.

표 1 최단 경로 탐색 비교 표(경주시 지도)

알고리즘	GG	AD	DD	OA
시간(μs)	7,031	5,151	5,475	4,497

- GG : Graph Growth Algorithm(Two Queue)
- AD : Dijkstra Algorithm(Approximate Bucket)
- DD : Dijkstra Algorithm(Double Bucket)
- OA : Our Algorithm

본 논문에서 제안된 알고리즘이 기존의 Dijkstra 알고리즘과 Graph Growth 알고리즘보다 빠르게 수행됨을 볼 수 있다.

4.3 임의로 구성한 지도에서의 최단 경로 탐색

본 절에서는 노드의 수가 많은 지도를 대상으로 각 알고리즘을 수행시켜 수행시간을 비교하려고 한다. 본 절에서 사용한 지도는 <그림 5>와 유사한 임의의 생성된 격자 그래프이다.

4.3.1 모든 인접한 노드사이에는 링크가 존재

노드 수는 4,900-40,000개이며, 링크 값은 5-300사이의 값을 임의로 정하였다. 실험 결과는 <그림 8>에 있다. <그림 8>에서 보면 앞 절의 결과와는 다르게 Graph Growth 알고리즘이 Dijkstra 알고리즘보다 빠르게 수행된다는 것을 알 수 있다. 또한 본 논문에서 제안된 알고리즘이 기존의 알고리즘보다 약 30-40% 더 좋은 성능을 보임을 알 수 있다.

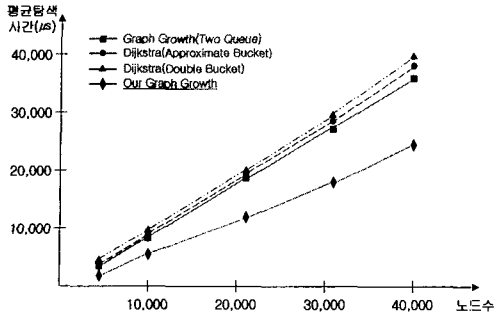


그림 8 모든 인접한 노드 사이에 링크 존재

4.3.2 인접한 노드사이에는 약 70%의 확률로 링크가 존재

본 절에서는 실제 도로 네트워크와 비슷한 조건을 만들기 위해 4.3.1절에서 사용한 격자 그래프에서 노드 사이의 링크를 임의로 제거하여 일방통행이나 삼거리 등을 가진 그래프로 새롭게 생성하였다. 노드 수는 4,900-40,000개이며, 링크값은 5-300사이의 값을 임의로 정하였다. 실험 결과는 <그림 9>에 나타나있다. <그림 9>에서 보면 Dijkstra 알고리즘이 Graph Growth 알고리즘보다 적은 노드 수에서는 더 좋은 성능을 보임을 알 수 있으며 노드 수가 많아질수록 Graph Growth 알고리즘이 더 좋은 성능을 보임을 알 수 있다. 또한 본 논문에서 제안된 알고리즘이 기존의 알고리즘들보다 약 30% 좋은 성능을 보임을 알 수 있다.

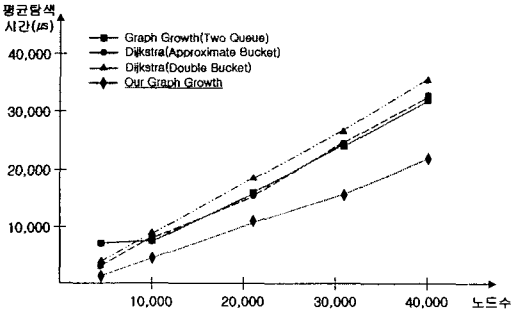


그림 9 인접한 노드 사이에는 약 70%의 확률로 링크 존재

5. 결 론

최단 경로를 찾는 알고리즘은 교통 및 네트워크 관리 등의 분야에서 중요하게 사용되고 있다. 최단 경로를 찾는 문제는 크게 All-to-All 방식, One-to-All 방식 One-to-One 방식으로 나뉘어져 해결되고 있다. 본 논문에서는 One-to-All 방식인 Graph Growth 알고리즘을 기반으로 One-to-One 방식의 알고리즘을 제안하였다. 즉, 본 논문에서 제안된 알고리즘은 기존의 One-to-All 방식에서 최단 경로를 구하기 위해 모든 노드를 탐색하는 시간을 줄였으며 본 논문에서 제안된 알고리즘은 기존의 알고리즘보다 성능이 더 뛰어남을 실험을 통하여 보였다.

참 고 문 헌

[1] F. B. Zhan and C. E. Noon, "A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths," Journal of Geographic Information and Decision Analysis, vol. 4, no. 2, pp. 1-11, 2000.

[2] F. B. Zhan, "Three Fastest Shortest Path Algorithms on Real Road Networks : Data Structures and Procedures," Journal of Geographic Information and Decision Analysis, vol. 1, no. 1, pp. 69-82, 1997.

[3] B. V. Cherkassky, A. V. Goldberg and T. Redzik, "Shortest Paths Algorithms : Theory and Experimental Evaluation," Mathematical Programming, vol. 73, no. 2, pp 129-174, 1996.

[4] E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," Numerische Mathematik, I, pp. 269-271, 1959.

[5] U. Pape, "Implementation and Efficiency of Moore Algorithms for the Shortest Root Problem," Mathematical Programming, vol. 7, pp. 212-222, 1974.

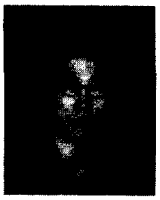
[6] S. Pallottino, "Shortest-Path Methods : Complexity, Interrelations and New Propositions," Networks, vol. 14, pp. 257-267, 1984.

[7] 이석호역, "자료구조론," 홍릉과학출판사, pp. 223-224, 1991.

[8] 최기주, "U-Turn을 포함한 가로망 표현 및 최단경로의 구현", 대한교통학회지, vol. 13, no. 3, pp. 35-52, 1995.

[9] 노정현, 남궁성, "도시가로망에 적합한 최단경로탐색 기법의 개발", 대한국토도시계획학회지, vol. 30, no. 5, pp. 153-168, 1995.

[10] 최재원, "시변 교통량을 고려한 차량 경로계획 시스템의 구현," 공학석사 학위논문, 서울대학교 전기공학부 대학원, 2000.



심 충 섭

1995년 3월 ~ 2000년 2월 서울시립대학교 전산통계학 학사. 2000년 3월 ~ 2002년 2월 서울시립대학교 전산학 석사. 2002년 2월 ~ 현재 (주)디지웨이브 근무. 관심분야는 최단경로알고리즘, GIS

김 진 석

1986년 3월 ~ 1990년 2월 과학기술대학 전산학 학사. 1990년 3월 ~ 1992년 2월 KAIST 전산학 석사. 1992년 3월 ~ 1997년 2월 KAIST 전산학 박사. 1997년 3월 ~ 1999년 2월 KAIST 인공지능연구센터 Postdoc 연구원. 1997년 10월 ~ 1998년 8월 미국 M.I.T. Laboratory for Computer Science Postdoc Fellow. 1998년 10월 ~ 1999년 2월 전자통신연구원 (ETRI) 슈퍼컴퓨터센터 초빙 연구원. 1999년 3월 ~ 현재 서울시립대학교 컴퓨터통계학과 조교수. 2002년 2월 ~ 현재 서울시립대학교 컴퓨터통계학과 학과장. 관심분야는 병렬알고리즘, GRID, GIS