공간자료구조를 활용한 단층구조분석 연구*)

방갑산
한성대학교 전자정보공학과
e-mail:ksbang@hansung.ac.kr

A Study on Fault Structure Analysis using Spatial Index Structure

Kap-San Bang
Dept of Electronic Information, Hansung University

요 약

지진 활동의 결과로 발생하는 피해를 감소시키기 위해 지층의 단층구조를 파악하는 것이 중요하다. 지층 탄성파 데이터를 공간자료구조를 활용하여 분석하고 효율적인 공간정보를 생산하는 시스템 개발과 관련된 연구를 제안한다. 공간자료구조는 방대한 탄성파 데이터를 지층을 이루는 암석층을 공간데이터로 저장하고, 분석하여 단층대를 이루는 공간을 검색 가능하게 한다. 검색된 단층가능 구간은 전문가의 판단을 통해 주변의 다른 단층구간과 다양한 요소들과 결합하여 지진 활동 연구에 중요한 판단 자료로 사용될 수 있다. 최근의 불안정한 지진활동 피해에서도 알 수 있듯이, 단층대의 확인 작업은 다양한 지상/지하 구조물의 안정성에 중요한 역할을 하는 자료가 된다.

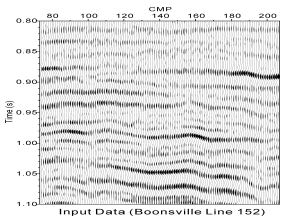
1. 서론

최근 세계적으로 지진활동이 활발해 지고 있고, 백두산 역시 대규모의 잠재적 지진 발생 위험 지역으로 구분되어 더 이상 한반도가 지진 안전지역이 아님을 알 수 있다. 지진 자체에 의한 피해와 부수적으로 수반되는 위험지역을 파악하기 위해 단층대의 분석이 중요하다. 지하구조 가운데서 불안정한 단층상태가 지진에 가장 취약한 위험요소인데, 이러한 단층상태를 정보시스템에 축척하고, 분석하고 인식하는 작업은 많은 연구가 필요한 상태이다. 일부 지역에 제한된 수의 시추공을 굴착하거나 갱도를 통해 직접 지하 광물과 일부 구조를 파악 할 수도 있으나 이는 매우 제한적인 범위에서 행하여지므로 전반적인 구조를 파악 하는데 한계가 있다. 보다 넓은 지역의 지하구조를 파악하기 위해 탄성파가 많이 활용되고 있다. 탄성파를 통해 얻어진 자료를 처리하여 다양한 분야에 필요한 많은 정보를 얻을 수 있으나 처리과정에서 지질전문가의 해석이 요구된다. 또한 얻어지는 정보의 양이 방대하고 3차원의 정보로 확대하여 해석할 경우 전문가의 주관적인 해석에 의한 데이터 처리는 단편적이고 국부적인 한계를 가질 수밖에 없다. 본 논문은 지하 지층을 공간자료구조를 활용하여 인식하는 시스템 개발과 관련된 기술적인 방법을 제안하고 이와 관련된 사항을 기술하였다.

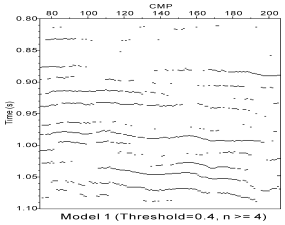
*) 본 연구는 2016년도 한성대학교 연구년 지원과제임.

2. 단층인식

이 논문에서 제안하는 지하단층 인식 시스템에 가장 적절한 공간자료구조는 공간상에서 포인트 데이터를 구성하고 검색하는데 가장 효과적인 자료구조로서 알려져 있는 R^{+}-tree[5, 6]이다.



(그림 1)탄성파 데이터



(그림2) 그림 1(threshold= 0.4) 대한 thining

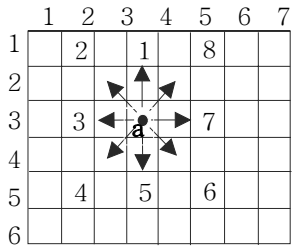
본 논문에서 사용된 입력 데이터는 Boonsville 지역에서 얻어진 광범위한 3차원 탄성파 데이터로서 그림 2는 3차원 데이터중의 하나의 수직 2차원 단면 데이터 중의 일부만을 보여주고 있다. 실제로 얻어진 데이터는 광범위한 지역에 걸친 3차원의 데이터로 구성되어있다. 그림 2에서의 데이터는 래스터 형태로서 351 X 70 개의 셀로서 구성되어있다. 각 셀은 0또는 1의 값을 갖는데, 해당 셀의 위치에서 탄성파의 강도가 주어진 threshold이상인 값을 가질 때 1로서 표현된 것이다. 주어진 입력데이터(그림 2)는 아래와 같이 정의된 2차원의 구조체 배열에 저장한다.

```
struct cell_type {
    unsigned value:1;   //cell의 값을 표현
    unsigned flag: 1;   //cell의 방문 여부를 표현
};
```

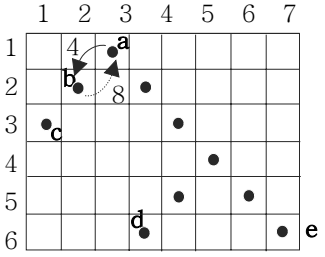
2.1 지층 object의 검색

단층인식의 첫 번째 단계로서 주어진 입력 데이터(그림 2)로부터 지층을 구성하는 object를 인식하여 파악된 object들을 리스트로 구성하는 것이다. 즉, 연속된 1의 값을 가진 셀 들을 파악하여 그것을 하나의 지층 object로서 인식하는 것이다. 각각의 지층 object에 대하여 말단점들을 검색한다. 검색된 object와 그에 속한 말단점의 좌표값 들은 연결리스트에 저장되고 또한 차후에 파일로 저장된다.

각 셀에 flag값을 사용하여 이미 검색되어 하나의 object의 구성원으로 사용된 셀을 중복하여 방문/검색하는 비효율을 방지한다. 셀 값으로 구성된 구조체 배열을 행위주(row major) 방식으로 읽으면서 0이 아니면서 flag값이 0인 셀을 만나면 해당 셀의 flag을 1로 설정하고 그 셀의 위치에서 이동 가능한 8가지 방향으로 검색을 한다.



(그림 3) 말단점 검색방향



(그림 4) object 인식 경로

그림 3에서 셀 a의 flag이 0인 경우 셀 a의 연속한 셀들이 연결되어 있는지를 검사하기 위한 방향은 모두 8가지이다. 셀 a에서 말단점 발견을 위한 이동방향을 왼쪽 또는 오른쪽으로 분리하여 한쪽 방향으로만 후속 셀의 검색을 진행시킬 경우 검색과정에서 역방향의 edge가 검색되지 못하는 경우가 발생할 수 있다. 그림 4에서 셀 a를 방문한 다음 오른쪽 검색을 위해 3가지 방향(6, 7, 8번 방향)으로 진행시킬 경우 오른쪽 끝점인 e를 발견할 수 있으나 진행방향과 반대편으로 구성된 edge d를 검색할 수 없다. 하나의 지층 object가 여러 갈래의 경로를 취할 수 있는 가능성이 있기 때문이다. 따라서 모든 방문한 셀에서 이동 가능한 모든 8방향으로 검색 알고리즘을 재귀적으로 수행하여 모든 말단 점을 검색하여야 한다. 그림 4의 경우 셀 a에서 모든 가능한 8방향의 검색을 반복적으로 진행하다가 4번 방향에 셀 b가 연결되어 있음을 발견하면 검색 알고리즘은 셀 b에 대하여 재귀적으로 검색알고리즘을 수행한다. 이 경우 셀 b는 부모 셀의 방향을 검색에서 제외하여야만 검색경로를 거슬러 올라가는 경우를 방지할 수 있다. 따라서 자식 셀의 검색경로는 모두 7가지 경우가 된다. 그림 4에서 셀 b의 검색경로 중에 8번 방향은 제외된다. 부모 셀에서 자식 셀로의 검색방향에 대한 자식 셀에서 부모 셀의 방향은 아래와 같다.

<표 1>부모 cell 방향과 자식 cell 방향 변환

부모방향	1	2	3	4	5	6	7	8
자식방향	5	6	7	8	1	2	3	4

그림 4에서 셀 a가 최초로 방문되었을 때 왼쪽과 오른쪽으로 이동하면서 말단 셀(c, d, e)을 발견하는 알고리즘은 다음과 같다.

발견된 말단 셀들은 아래와 같이 정의된 구조체 리스트에 저장한다:

```
struct Head {
   object_count;
    struct object_node *object_pointer;
};
struct object_node {
    int object_id;
    int terminal _count;
    struct terminal_node *terminal_pointer;
    struct object_node *next;
};
struct terminal_node {
    int r;          //row: y coordinate
    int c;          //column: x coordinate
    int terminal_id:
    struct terminal_node *next;
};
```

Object_Detect()

1. 각각의 cell에 대하여 반복적으로 수행한다
2. if cell 값이 1이고 flag 값이 0 이면
    flag= 1 //방문설정
    Create_object() //object 생성
    부모 cell 방향을 0으로 설정
    Scan_Terminal()

Scan_Terminal( )

1. 자식 cell 카운터를 0으로 설정한다
2. 현재 cell의 8방향 각각에 대해 반복한다
    if 자식 cell 방향이 부모 cell 방향이 아니면
    자식 cell 방향의 좌표값을 생성한다
    if 생성된 좌표가 적절하면
        자식 cell의 flag= 1로 설정
        자식 cell 카운터 1 증가
        자식 cell 방향을 표1을 참조로 변환
        재귀적으로 Scan_Terminal( )을 호출
    else 자식 cell 카운터 1 증가

3. if 자식 cell 카운터 < 2 인 경우 말단점이 된다
   Create_Terminal() //말단점 노드 생성
   해당 말단점이 속한 object 구조체에 연결한다

Create_Object( )

1. 하나의 구조체 object_node를 생성하고 구성원의 설정한다.

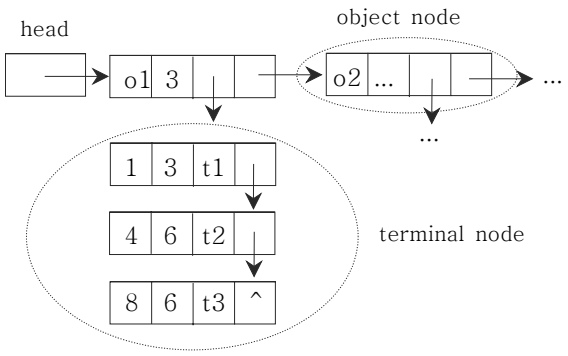
```
object_id= Head->object_counter++;
terminal _count= 0;
terminal_pointer= NULL;
next= NULL;
```

2. Head 리스트에 생성된 Object 노드를 연결한다.

Create_Terminal( )

1. 하나의 구조체 terminal_node를 생성하고 주어진 cell의 행/열 좌표를 구조체 terminal_node의 구성원으로 설정한다.
2. 구조체 object_node의 terminal_counter를 terminal_id에 설정하고 1 증가 시킨다.
3. 구조체 object_node의 terminal_node 포인터에 생성된 노드를 연결시킨다.

그림 5는 구조체 Head, Object, Terminal 노드간의 관계를 보여준다.



(그림 5) object와 말단 셀 저장

각 object는 ID가 부여되어 식별되며(그림 5의 o1, o2, ...), 해당 object의 말단 셀의 정보를 연결리스트로서 저장한다. 말단 셀을 표현하는 정보인 좌표값, 말단셀 ID(그림 4의 말단 셀 c, d, e는 그림 5의 t1, t2, t3로 표현)가 저장된다. 동일한 object에 소속된 말단 cell들은 서로 단층을 이루는 쌍을 구성할 수 없기 때문에 object ID를 사용하여 불필요한 검색을 피할 수 있다. 예를 들어, 그림 5의 object o1의 말단 셀 t1과 t2는 같은 object의 구성원 이므로 단층대를 구성하는 말단 셀 쌍이 될 수 없다. 단층대를 구성하는 말단 셀은 서로 다른 object의 말단 셀 2개가 근접함으로서 구성될 수 있다.

2.2 공간자료구조 구성과 단층 검색

단층검색의 두 번째 단계는 첫 번째 지층 object 검색과정에서 발견된 말단점의 좌표값을 활용하여 공간자료구조인 R^+-tree를 구성한다. 세 번째 단계에서는 각 각의 말단점에 대하여 가장 인접한 다른 object에 소속된 말단점을 검색하여 그중에 가장 높은 확률을 가진 object를 결과값으로 반환한다.

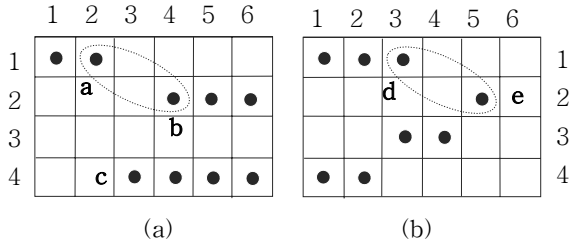
발견된 말단 셀의 좌표로서 R^+-tree를 구성한다. 일반적인 2차원 공간자료구조에서 하나의 대상을 나타내기 위해서는 4개의 좌표값을 사용하지만 단층검색을 위한 R^+-tree에는 단순히 말단점의 좌표값 만으로 구성되므로 셀의 위치는 x, y 두 개의 좌표로서 구성될 수 있다. R^+-tree의 구성이 매우 간단해지고 R^+-tree의 공간분할이 이러한 포인트 데이터를 구성하는데 최적의 성능을 발휘할 수 있다. R^+-tree의 루트노드와 중간 레벨 노드들은 하위 레벨 노드를 포함하는 MBR(Minimum Bounding Rectangle)을 나타내야 하므로 여전히 4개의 좌표로 구성이 되어야 하고 전체 R^+-tree의 노드 구조를 하나의 단일 구조로 써야하므로 아래와 같이 R^+-tree의 노드 구조를 구성한다.

```
struct R_node {
    int object_id;  //중간노드의 경우 -1로 설정
    int x1;         //column
    int y1;         //row
    int x2;         //column
    int y2;         //row
    struct R_node *son;//최하위 레벨의 경우 NULL
};
```

루트노드와 모든 중간 레벨 노드의 경우는 실제의 말단점을 가리키는 것이 아니라 인덱스의 역할을 하므로 object_id 값이 -1로 NULL 값으로 설정한다. 최하위 레벨 노드의 하나의 말단점은 두 개의 좌표 값만 필요하므로 실제로 x1=x2, y1=y2 로 설정한다. 또한 최하위 레벨 노드는 하위레벨의 노드가 존재하지 않으므로 하위레벨 노드를 나타내는 포인터 son은 NULL 값으로 설정한다.

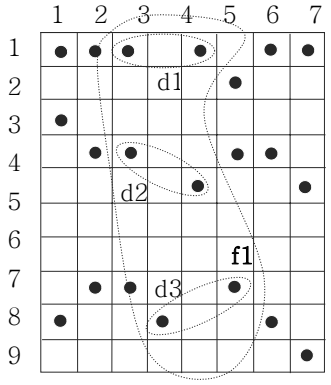
구성된 R^+-tree에 대해 검색 대상이 되는 말단셀 리스트에 저장된 각 셀 에 대하여 가장 근접한 셀을 검색한다. 검색방식은 주어진 셀의 위치에서 가장 인접한 이웃(nearest neighbor) 셀을 발견하는 방식이다. 최초의 검색범위 크기는 전체 object의 평균길이를 사용하여 설정하고 최소검색에서 검색범위에 포함되는 셀이 발견되지 않을 경우 검색범위를 확장시켜 검색을 진행한다. 검색범위의 확장범위도 object의 평균길이와 축적된 실험의 결과에서 얻어진 값을 사용하여 조정한다. 또한 검색범위가 일정크기 이상으로 확대되어도 검색에 실패하는 경우 해당 지층과 match되는 지층이 존재하지 않는 것으로 한다. 검색범위에서 발견된 셀 중에서 object ID를 비교하여 동일한 ID를 가진 object를 제거한다. 이것은 동일한 지층 object

의 시작과 종료좌표가 하나의 단층구간으로 인식되는 것을 방지하기 위한 것이다. 발견된 object 쌍은 각 object의 접근 방향에 따라 두 가지 경우로서 구분된다. 첫째, 그림 6(a)의 셀 a, b와 같이 접근방향이 서로 다른 경우 둘째는 그림 8(b)의 셀 d, e와 같이 접근방향이 동일한 경우이다. 첫 번째 경우는 단층의 가능성이 매우 높으나 두 번째 경우는 단층을 이루는 가능성이 상당히 희박하다. 그러나 접근방향이 서로 같은 경우라 할지라도 그림 6(b)에서 보는 바와 같이 단층이 발생한 다음 지층변화에 의해 방향이 변화되는 경우도 발생할 수 있기 때문에 이러한 검색결과를 하나의 단층구간을 구성하는 후보로서 리스트에 저장한다.



(그림 6) 검색 결과의 해석

그림 7에서는 발견된 단층구간 object 리스트를 검색하여 연속된 단층구간을 단층대로 인식하는 과정을 보여준다. 즉 여러 개의 단층구간 d1, d2, d3가 인접한 공간에서 발생하면 이 구간에 지층의 단층대가 형성되어 있음을 확인하는 것이다. 따라서 단층대 f1의 존재를 인식한다. 이를 위해 R^+-tree를 사용하여 단절구간의 공간값을 모두 저장한다. 단절구간의 공간값은 두개의 말단점 중에 하나의 좌표값을 사용하여 저장한다. 저장된 단층구간 각각에 대하여 인접한 단층구간을 검색한다. 최초의 검색범위 크기는 평균적인 단절구간 길이로 설정하고 최소검색에서 검색범위에 포함되는 단절구간이 발견되지 않을 경우 검색범위를 확장시켜 검색을 진행한다. 검색범위의 확장은 축적된 실험의 결과에서 얻어진 값을 사용하여 조정한다. 또한 검색범위가 일정크기 이상으로 확대되어도 검색에 실패하는 경우 해당 단절증과 단층대를 형성하는 인접 단층구간이 존재하지 않는 것으로 한다. 하나의 단층대를 형성하는 인접한 단층구간은 데이터구조에 저장한다



(그림 7). 단층대 인식

최종적인 검색이 종료된 후에 Arc/view[3]와 같은 시각화 도구를 사용하여 해당구간을 표시하여 지질 전문가의 최종적인 해석을 위한 전처리 과정이 된다.

3. 결론

지하 단층 구조를 체계적으로 분석하기 위한 연구는 지진 활동에 대한 지상의 구조물의 안전성 평가에 대단히 중요한 역할을 한다. 본 논문은 지하 단층을 공간자료구조를 활용하여 인식하는 시스템 개발 방법을 제안하고 이와 관련된 사항을 기술하였다. 이러한 시스템의 개발은 지진 전문가의 지질모델 작성 및 단층활동 지역을 분석할 수 있는 전처리 시스템으로서 중요한 역할을 할 것이다.

참고문헌

[1]Tor Bernhardsen, Geographic Information System, Norwegian Mapping Authority, 1992.

[2]Guttman, A., R-trees: A Dynamic Index Structure for Spatial Searching, Proc. of the ACM SIGMOD, pp. 47-57, 1984.

[3]Understanding GIS, The Arc/Info Method, 1994.

[4]Michael F Goodchild, Introduction to GIS, National Center for Geographic Information and Analysis Univ. of California, 1991.

[5]Hoel, E. G., and Samet, H., A Qualitative Comparison Study of Data Structures for Large Segment Databases, ACM SIGMOD, pp. 205-214, 1992.

[6]Sellis, T., Roussopoulos, T. and Faloutsos, C., R^+-tree: A Dynamic Index for Multi-dimensional objects, Proc. of the 13th VLBD Conference, pp. 507-518, 1987.

[7]Zhexuan Song, Nick Roussopoulos, Hashing Moving Objects, International Conference on Mobile Data Management, pp. 161-172, 2001

[8]Robinson, J.T, The K-D-B tree: A search structure for large multiple-dimensional dynamic indexes. In Proceedings of the ACM SIGMOD Conference, pp. 10-18, 1981