이차원 배열 데이터에서 유사 구역의 효율적인 탐색 기법

최연정, 이기용
숙명여자대학교 컴퓨터과학부
e-mail : cyj@sookmyung.ac.kr, kiyonglee@sookmyung.ac.kr

Efficient Search of Similar Regions in Two-Dimensional Array Data

YeonJeong Choe, Ki Yong Lee
Division of Computer Science, Sookmyung Women's University

요 약

첨단 과학 장비를 이용한 시뮬레이션의 결과로 데이터의 정확도 및 정밀도가 향상되어 대용량의 이차원 배열 데이터가 생성되고 있다. 대용량의 이차원 배열 데이터에서 유사 구역(similar region)을 찾아내는 것은 매우 의미 있는 일이다. 따라서 본 논문에서는 대용량의 이차원 배열데이터에서 유사 구역을 찾는 단순 방법(naïve method)과 효율적으로 탐색할 수 있는 알고리즘을 제안한다. 또한 단순 방법과 제안 알고리즘의 시간 복잡도(time complexity)를 분석하고 실험을 통해 제안 방법이 단순 방법보다 더 빠르게 처리함을 보인다.

1. 서론

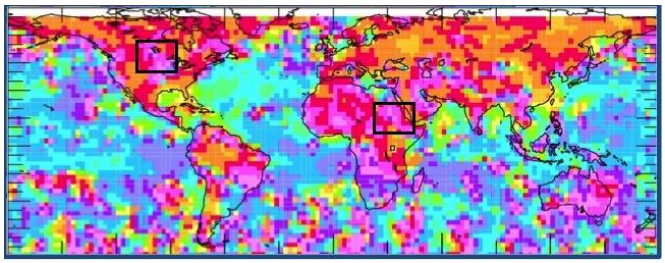
최근 과학 장비의 발전으로 데이터의 정확도 및 정밀도가 향상되었고 그에 따라 데이터 자체의 크기가 증가하였다. 많은 과학분야에서는 첨단 과학 장비를 이용한 시뮬레이션이 진행되고 있으며, 그 결과로 대용량의 이차원 배열 데이터가 생성되고 있다[1].

대용량의 이차원 배열 데이터에서 유사한 구역을 찾는 것은 여러 유용한 정보를 이끌어 낼 수 있다. 예를 들어, (그림 1)은 이차원 배열 데이터로, 지구 표면의 온도를 색으로 나타내고, 유사 구역을 표시한 예다. 온도가 유사한 구역을 찾는 것은 환경적 측면뿐 아니라 사회적 측면으로도 매우 유용한 정보이다.

이 때, 구역의 크기가 특정 값 이상인 모든 구역이라면 가능한 모든 구역 크기에 대해 전체 데이터를 살펴봐야 한다. 하지만 이는 대용량의 이차원 배열데이터에서 비효율적이다. 따라서 유사 구역을 효율적으로 검색하는 알고리즘을 제안한다.

본 논문에서는 이차원 배열 데이터 내에서 셀(cell) 값이 유사한 부분 배열(sub-array)을 유사 구역이라 하며, 이를 판별하기 위해 L^2 *norm* 을 이용하여 두 구역의 에러를 계산하고 사용자가 정의한 값보다 작으면 유사하다고 한다. 유사 구역을 효율적으로 찾기 위해 사용자가 정의한 에러 이하의 셀을 먼저 찾고, 셀 주위 값과 현재 구역의 에러 합이 사용자가 정의한 값보다 작을 때까지 구역을 확장하는 알고리즘을 제안한다.

논문의 구성은 다음과 같다. 2 장에서는 문제를 정의하고, 3 장에서는 유사 구역 검색에 대한 관련 연구



(그림 1) 지구 표면 온도의 유사 구역

를 살펴본다. 4 장에서는 본 논문에서 제안하는 2 차원 배열 데이터에서 유사 구역을 검색하는 알고리즘을 자세히 설명하고, 5 장에서는 실험 결과를 제시한다. 마지막으로 6 장에서 결론 및 추후 연구를 기술한다.

2. 문제 정의

본 논문에서는 $n \times m$ 크기의 2 차원 배열 데이터를 A 라하고, A 에서 (i,j) ($1 \le i \le n$, $1 \le j \le m$) 위치에 있는 셀 값을 A_{ij} 라고 한다. 배열 데이터 A 에서 부분 배열을 구역이라 하고, 부분 배열끼리 유사한지 판별하기 위해 사용자로부터 $Size_{min}$ 과 $Error_{max}$ 를 입력 받아 두 지역이 셀 수가 $Size_{min}$ 이상인 직사각형이면서 에러의 크기가 $Error_{max}$ 이하이면 유사 구역이라 한다. 에러의 크기는 대응되는 각 셀 값의 차를 제곱하여 합한 값을 에러의 크기라고 한다.

예를 들어, 유사 구역인지 판별하고 싶은 두 구역 R_1, R_2에서 왼쪽 상단 셀의 위치를 각각, (ux_1, uy_1), (ux_2, uy_2)라 하고, 두 구역의 크기는 가로 셀 수가 w, 세로

셀 수가 h 인 $w \times h \geq Size_{min}$ 을 만족하는 직사각형이라 하자. 이 때, 에러의 크기는 (식 1)이다. 이 값이 $Error_{max}$ 이하면 유사 구역이라 판별하고 $((ux_1, uy_1), (ux_2, uy_2), w, h)$를 결과값으로 내보낸다.

$$\sum_{j=uy_1}^{uy_1+h} \sum_{i=ux_1}^{ux_1+w} (A[i][j] - A[i + (ux_2 - ux_1)][j + (uy_2 - uy_1)])$$

(식 1) 두 구역간의 에러의 크기

3. 관련 연구

본 논문과 관련된 유사 구역 탐색(Similar Region Search)이나 유사 탐색(Similar Search)은 오래 전부터 여러 분야에서 연구되어 왔지만 배열 데이터에서는 생소하다. 따라서 3.1 절에서 여러 분야에서 연구된 유사 구역 탐색과 유사 탐색을 살펴보고, 배열 데이터에서 주로 연구되어온 질의는 무엇인지 3.2 절에서 서술한다.

3.1 유사 검색 질의

다양한 분야에서 유사 구역 탐색과 유사 탐색이 연구되어 왔다. 하지만 기존의 연구는 데이터 타입이 2 차원이 아니거나 배열 데이터일 경우 차수를 낮춰서 유사성(similarity)을 계산한다. 간단히 몇 가지 분야를 예로 살펴보고자 한다.

먼저 분자 생물학(Molecular Biology) 분야이다[2]. 분자생물학에서 유사 구역을 찾는 문제는 매우 중요한 연산이다. 특히 DNA, RNA, 단백질 시퀀스 데이터(Protein sequence data)에서 유사 구역을 탐색하여 공통 염기서열을 찾는 연구는 매우 활발히 진행되어왔다. 하지만 데이터 타입이 일차원 시퀀스 데이터로 이차원 배열데이터를 논의하는 본 논문과는 맞지 않는다.

다음은 지리 공간 데이터(Geo-spatial Data)에서 유사 구역 탐색에 대한 연구이다[3]. 이 분야에서 유사 구역 탐색은 기능적으로 유사한 지역을 탐색하는 것으로, PoI(Point of Interest)를 이용하여 기능을 카테고리별로 나눈다. 각 지역에서 카테고리 값을 TF-IDF 의 변형인 CF-IRF 로 구하고, 각 지역의 정보내용을 CF-IRF 값의 벡터(vector)로 표현하고 벡터간의 유사성을 계산하여 질의를 처리한다.

마지막으로 멀티미디어 데이터(Multimedia Data)이다[4]. 본 논문에서 논의하는 배열 데이터와 가장 유사해 보이지만 지리 공간 데이터와 유사하다. 이미지 데이터(image data)나 비디오 데이터(video data)에서 컬러 히스토그램(color histogram), 에지 검출(edge detection)등 여러 기법을 이용해 특징 추출(feature extraction)을 하고 차수를 낮춰 각 값을 벡터로 표현한다. 그리고 벡터간의 유사성을 계산한다.

지리 공간 데이터나 멀티미디어 데이터와 달리 본 논문에서 논의하는 배열 데이터는 전처리(pre-processing)를 거치지 않은 원시자료(raw data)로, 차수를 낮추지 않고 2 차원 배열의 행(row), 열(column)을 모두 고려하여 질의를 처리하고자 한다.

3.2 배열 데이터에서 질의 처리

최근 들어 배열 데이터에 대한 관심이 매우 높아져, 여러 질의에 대한 연구가 진행되어왔지만 유사 구역 탐색에 대한 연구는 진행된 적이 없다. 주로 어떤 질의에 대한 연구가 진행되어왔는지 알아보고자 한다.

배열 데이터에서 윈도우 집계 함수(window aggregate function)는 유용한 연산이다[5]. 윈도우 집계 함수는 윈도우(window)를 움직이면서 윈도우 내에 있는 모든 셀에 집계 연산을 하는 것이다. 이 때, 윈도우가 움직이면서 겹치는 셀의 중복연산을 피하기 위해 윈도우를 쪼개서 연산하고 그 값을 리스트(list)에 저장한 뒤 리스트를 슬라이딩(sliding)하여 계산하는 연구가 있다.

다음은 조인(join)에 관한 연구이다. 조인은 데이터베이스에서 매우 중요한 연산이고 배열 데이터에서도 많은 연구가 있다. 본 논문에서는 등가 조인(equi-join) 중 Dimension:Dimension 만 언급하고자 한다[6].

Dimension:Dimension 은 배열 데이터에서 배열의 인덱스(index)를 조인 조건으로 하는 것으로, SELECT * INTO T FROM α JOIN β where $\alpha.i = \beta.j$ 에서 where 절과 같이 두 배열 α 와 β 의 인덱스 값이 같은 것을 조인 조건(join condition)으로 한다.

배열 데이터에서 등가 조인 외에 유사 조인(similarity join)도 있다[7]. 배열데이터에서 유사 조인은 엡실론 조인(epsilon join)을 변형한 것으로 엡실론 조인은 d 차원의 두 데이터 셋(data set)에서 각각 임의의 d 차원의 점을 잡고 두 개의 d 차원 벡터의 거리가 엡실론(epsilon) 이하인 점의 쌍을 찾는 것이다. 엡실론 조인은 오직 같은 차수를 가진 점들의 집합에 대해서만 잘 정의되기 때문에 다른 차수의 배열로 확장은 불가능하다. 따라서 배열 유사 조인(array similarity join)이 제안되었는데 등가 조인의 Dimension:Dimension 와 관련이 깊다.

배열 유사 조인은 크게 두 가지의 기능으로 나눌 수 있다. 매핑 함수(mapping function)와 유사 모양(similarity shape)이다. 매핑 함수는 조인할 두 입력 배열(input array)에서 차수를 매핑할 조건을 표현하는 함수로 Dimension:Dimension 의 예에서 where 절 $\alpha.i = \beta.j$ 와 같이 표현하는 것이다. 유사 모양은 셀의 주위를 표현하는 식으로 배열 유사 조인은 엡실론 조인과 다르게 거리 대신 유사 모양에 기반하며, 매핑 함수와 모양 배열(shape array)의 조건 하에 두 배열이 유사 조인한다.

4. 제안 방법

이차원 배열 데이터에서 유사 구역을 탐색하는 연구가 존재하지 않으므로, 누구나 고안할 수 있는 단순 방법(naïve method)을 4.1 에서 기술하고, 보다 효율적으로 탐색하는 제안 방법을 4.2 에서 기술한다.

4.1 단순 방법(naïve method)

단순 방법은 $w \times h \geq Size_{min}$ 를 만족하는 후보 직사각형(candidate rectangle)을 모두 만들고, 그 중 하나

를 선택해서 피봇(pivot)으로 삼아 모든 데이터를 스캐닝(scanning)하는 방법이다. 피봇 또한 데이터 끝까지 슬라이딩하며 에러 크기를 구하고, 이를 모든 후보 직사각형에 대해 시행하는 방법이다. Algorithm 1 은 단순 방법을 의사 코드(pseudo-code)로 나타낸 것이다.

```
1: function naïve(A, Size_min, Error_max)
2: Input: 2-dimensional array data of n×m size A
          user-defined minimum size of regions Size_min
          user-defined maximum size of error Error_max
3: Output: (pivot_loc, current_loc, width, height)
          width and height of region
4:   C = all candidate rectangles (size ≥ Size_min)
5:   for all candidate rectangle c ∈ C
6:     for row = 1 to n
7:       for column = 1 to m
8:         /* Upper-left location of pivot */
           pivot_loc = (row, column);
9:         for i = row to n
10:          for j = column + 1 to m
11:          /* Upper-left location of current region */
             current_loc = (i, j);
12:          error = Calculate_error
                        (pivot_loc, current_loc, width, height);
13:          if(error ≤ Error_max)
14:            return (pivot_loc, current_loc, width, height);
15:          end if
16:        end for
17:      end for
18:    end for
19:  end for
20:  end for
21: end function
```

Algorithm 1: naïve method algorithm

이 방법에서 후보 직사각형은 넓이가 $Size_{min}$ 이상, $n \times m$ 이하인 모든 직사각형이다. 만약 넓이가 12 면 후보 직사각형의 집합 C = {1 × 12, 2 × 6, 3 × 4, 4 × 3, 6 × 2, 12 × 1}로 총 6 가지이다. 그런데 넓이가 $Size_{min}$ 이상인 후보 직사각형이므로 굉장히 많은 수의 후보 직사각형을 만들어야 한다.

여러 후보 직사각형 중 하나를 골라 피봇으로 삼고, 하나의 피봇당 $n \times m$ 개의 에러 크기를 구하고 난 뒤, 피봇 또한 데이터 끝까지 스캐닝 해야 한다. 따라서 하나의 후보 직사각형당 시간복잡도는 $O(n^4)$이므로 단순 방법의 시간복잡도는 $O(k \times n^4)$ (k = 후보 직사각형의 수, 즉 $|C|$)이다. 이 때 k가 굉장히 큰 수이므로, 대용량의 이차원 배열 데이터에서 유사 구역을 검색하는 단순 방법은 매우 큰 시간복잡도를 갖는다.

4.2 제안 방법

단순 방법에서 사용자가 정의한 $Size_{min}$ 이상인 넓이의 후보 직사각형 개수는 매우 많으며, 모든 후보 직사각형에 대해 동일한 일을 반복한다. 따라서 후보 직사각형을 만들지 않고 효율적으로 유사 구역을 탐색 하는 방법을 제안한다.

방법은 다음과 같다. 먼저 1 × 1 인 직사각형을 (1,1)에 피봇시키고, 슬라이딩 하면서 에러의 크기를 구한다. 에러의 크기가 $Size_{min}$ 보다 작으면 에러 그래프(error graph)를 생성한다. 에러 그래프는 피봇과 현재 위치의 에러를 그래프(graph)로 나타낸 것으로 두 위치에서 남쪽과 동쪽으로 지역을 확장시키면서 에러 크기를 구하고 에러의 총 합이 $Error_{max}$ 를 넘지 않을 때까지 그래프를 확장시킨다. 이 때, 구역의 넓이가 $Size_{min}$ 을 넘으면 (R_1, R_2, width, height)을 반환(return)한다. 이를 알고리즘으로 나타내면 Algorithm 2 와 Algorithm 3 이다.

```
1: function proposed(A, Size_min, Error_max)
2:     for row = 1 to n
3:       for column = 1 to m
4:         pivot_loc = (row, column);
5:         for i = row to n
6:           for j = column + 1 to m
7:             current_loc = (i, j);
8:             error = Calculate_error
                                (pivot_loc, current_loc, 1, 1);
9:             if(error ≤ Error_max)
10:              errorGraph(pivot_loc, current_loc, 1, 1);
11:            end if
12:          end for
13:        end for
14:      end for
15:    end for
16: end function
```

Algorithm 2: proposed method algorithm

```
1: function errorGraph(pivot_loc, current_loc, width, height)
2: Output: (pivot_loc, current_loc, width, height)
3:   left = Calculate_error
           (pivot_loc.south, current_loc.south, width, height);
4:   right = Calculate_error
           (pivot_loc.east, current_loc.east, width, height);
5:   if(width×height ≥ Size_min)
6:     output = (pivot_loc, current_loc, width, height);
7:   end if
8:   if(left ≤ Error_max)
9:     errorGraph
         (pivot.south, current region.south, width, height+1);
10:  end if
11:  if(right ≤ Error_max)
12:    errorGraph
         (pivot.east, current region.east, width+1, height);
13:  end if
14: end function
```

Algorithm 3: error graph algorithm

예를 들어, 2 차원 배열데이터 A 가 (그림 2)와 같고, $Size_{min}$ = 4, $Error_{max}$ = 4 라고 가정하자. (1,1)을 피봇으로

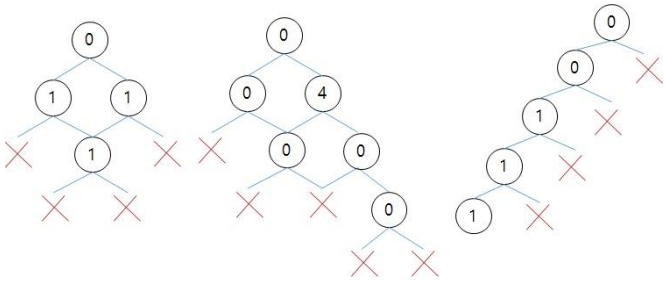
설정하고 (1,2)부터 데이터 끝까지 슬라이딩하며 에러의 크기를 구한다. (2,5)와 에러 크기가 4 보다 작지만 남쪽 방향과 동쪽 방향 셀의 에러 크기 합은 4 보다 커서 확장되지 않는다. (2,2)를 피봇으로 설정했을 때, (3,6)에서 에러 그래프는 (그림 3)의 (a)와 같고 크기가 $Size_{min}$ 이상이면서 모든 노드에서 에러의 합이 $Error_{max}$ 보다 작으므로 결과로 내보낸다.

1	21	78	63	41	14	4	55
15	100	78	52	2	59	21	64
92	47	100	90	43	100	77	11
20	37	66	4	79	46	99	81
7	44	17	21	58	13	32	36
6	62	100	76	52	2	3	3
99	70	47	100	5	77	83	10
56	32	50	33	23	27	100	10

(그림 2) 이차원 배열데이터 A 의 예

(그림 3)은 (그림 2)의 결과에 대한 에러 그래프로 (b)는 (2,2)와 (6,3), (c)는 (1,7)과 (4,4)의 에러 그래프이다.

제안 방법은 피봇과 에러가 $Error_{max}$ 이하인 셀에서 주변지역으로 확장시켜 나가면서 에러를 구하는 방식으로, 후보 직사각형의 개수만큼 오버헤드를 줄일 수 있어 더욱 효율적이다.



(a) (b) (c)

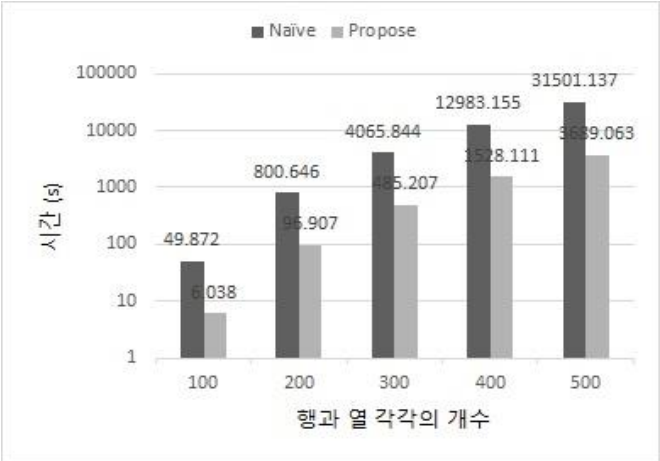
(그림 3) 에러 그래프의 예

5. 실험 결과

본 장에서는 단순 방법과 제안 방법의 성능을 비교한 실험 결과를 보인다. 성능 척도로는 $Size_{min}$ = 4, $Error_{max}$ = 2 인 유사 구역을 탐색하는데 드는 전체 시간을 기준으로 하였다.

실험 데이터로는 가상의 배열 데이터를 생성하였다. 배열데이터는 행과 열의 개수가 같고, 100 개에서 500 개까지 증가시켰으며, 각 셀 값은 [0,100] 사이의 자연수 중 임의로 선택하였다.

(그림 4)는 단순 방법과 제안 방법의 성능 비교 그림이다. 이 때, 단순 방법의 경우 $Size_{min}$ 가 4 이상인 모든 후보 직사각형을 생성해야 하지만 시간상의 문제로 넓이가 4 인 직사각형 $C = \{1 \times 4, 2 \times 2, 4 \times 1\}$ 만 계산하였다. 제안 방법은 넓이가 4 이상인 구역을 모두 탐색하고 있음에도 단순 방법보다 8 배 이상의



(그림 4) 단순 방법과 제안 방법의 성능 비교

좋은 성능을 보인다.

4 장에서 설명한 바와 같이 후보 직사각형의 개수는 매우 많으며 각각의 후보 직사각형에 대해 동일한 일을 반복한다. 그에 비해 제안 방법은 많은 후보 직사각형을 만들지 않고, 구역을 확장시켜 단순 방법에 비해 좋은 성능을 보인다.

6. 결론 및 추후 연구

본 논문은 이차원 배열 데이터에서 유사 구역의 효율적인 탐색 기법을 제안하였다. 제안 방법은 단순 방법처럼 많은 수의 후보 직사각형을 만들지 않으며, 구역을 확장시켜 나감으로써 성능이 향상된다. 실험을 통해 제안 방법이 단순 방법에 비해 유사 구역을 더 빠르게 탐색함을 확인하였다.

Acknowledgement

이 논문은 2016 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.NRF-2015R1C1A1A02037071)

참고문헌

[1] Array data, https://en.wikipedia.org/wiki/Array_data_type
[2] Ming Li, Bin Ma, Lusheng Wang “Finding Similar Regions in Many Sequences”, Journal of Computer and System Sciences 65, 73-96, 2002
[3] C. Sheng, Y. Zheng “Answering Top-k Similar Region Queries”, Database Systems for Advanced Applications, Lecture Notes in Computer Science, 5981:186-201, 2010
[4] Q. Lv, M. Charikar, K. Li “Image Similarity Search with Compact Data Structures”, In CIKM’04, November 8–13, 2004
[5] L. Jiang, H. Kawashima, O. Tatebe “Incremental Window Aggregates over Array Database”, IEEE International Conference on Big Data, 2004
[6] D. V. Kalashnikov. “Super-EGOL fast multi-dimensional similarity join”, VLDB Journal, 4(2):561-585, 2013
[7] Weijie Zhao, Florin Rusu, Bin Dong Kesheng Wu “Similarity Join over Array Data”, In Proceedings of ACM SIGMOD, 2016