KIPS Tr. Software and Data Eng.
Vol.6, No.4 pp.185~192 pISSN: 2287-5905



https://doi.org/10.3745/KTSDE.2017.6.4.185

# An Efficient Method for Finding Similar Regions in a 2-Dimensional Array Data

YeonJeong Choe† · Ki Yong Lee††

## ABSTRACT

In various fields of science, 2-dimensional array data is being generated actively as a result of measurements and simulations. Although various query processing techniques for array data are being studied, the problem of finding similar regions, whose sizes are not known in advance, in 2-dimensional array has not been addressed yet. Therefore, in this paper, we propose an efficient method for finding regions with similar element values, whose size is larger than a user-specified value, for a given 2-dimensional array data. The proposed method, for each pair of elements in the array, expands the corresponding two regions, whose initial size is 1, along the right and down direction in stages, keeping the shape of the two regions the same. If the difference between the elements values in the two regions becomes larger than a user-specified value, the proposed method stops the expansion. Consequently, the proposed method can find similar regions efficiently by accessing only those parts that are likely to be similar regions. Through theoretical analysis and various experiments, we show that the proposed method can find similar regions very efficiently.

Keywords : Array Data, Finding Similar Regions, Array Query Processing, Array Database

# 2차원 배열 데이터에서 유사 구역의 효율적인 탐색 기법

최 연 정† · 이 기 용††

## 요 약

여러 과학 분야에서 측정 또는 시뮬레이션의 결과로 2차원 배열 데이터가 활발히 생성되고 있다. 현재 배열 데이터에 대한 다양한 질의 처리 기법들이 연구되고 있으나 2차원 배열 데이터에서 크기가 미리 알려져 있지 않은, 값이 서로 유사한 구역을 찾는 문제는 거의 다루어지지 않았다. 따라서 본 논문에서는 주어진 2차원 배열 데이터에서 사용자가 지정한 값 이상의 크기를 갖는, 원소 값들이 서로 유사한 구역을 빠르게 찾는 방법을 제안한다. 본 논문의 제안 방법은 2차원 배열의 각 원소 쌍에 대해, 해당 원소로만 이루어진 크기가 1인 구역부터 시작하여 두 구역을 동일한 모양을 유지하면서 오른쪽 및 아래쪽으로 단계적으로 확장시켜나간다. 만약 두 구역의 값의 차이가 사용자가 지정한 값 이상으로 커지면 확장을 중단한다. 따라서 제안 방법은 배열에서 유사 구역이 될 가능성이 있는 부분들만 접근하여 유사 구역을 효율적으로 찾아낼 수 있다. 본 논문에서는 성능 분석과 다양한 실험을 통해 제안 방법이 매우 효율적으로 유사 구역을 찾을 수 있음을 보인다.

키워드 : 배열 데이터, 유사 구역 탐색, 배열 질의 처리, 배열 데이터베이스

## 1. 서 론

최근 여러 과학 분야에서 첨단 장비를 이용한 측정 또는 시뮬레이션 결과로 2차원 배열 데이터가 활발히 생성되고 있다. 크기가 $n \times m$인 2차원 배열 데이터 $A$는 $n \times m$개의 원소(element)로 구성되며, 각 원소는 $A[i][j]$로 나타낸다 ($1 \le i \le n$, $1 \le j \le m$). 과학 분야에서 생성되는 2차원 배열 데이터의 예로는 Sloan Digital Sky Survey(SDSS)[1]와 Palomar Transient Factory(PTF)[2]와 같은 천문학 프로젝트에서 매일 밤 촬영하는 별과 은하수 이미지, 각 지역의 해수 표면 온도를 측정한 데이터, 석유와 가스가 매립된 지역을 나타내는 데이터 등이 있다. Fig. 1은 지구 표면의 기온을 나타내는 래스터(raster) 데이터의 예이다. 래스터 데이터는 배열 데이터의 한 종류로, 영상을 일정 크기의 격자들로 나누고 각 격자 별로 집계 값을 저장하고 있는 2차원 배열 형태의 데이터이다.

본 논문은 앞서 설명한 2차원 배열 데이터에서 서로 유사

※ 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2015R1C1A1A02037071).
※ 이 논문은 2016년도 한국정보처리학회 추계학술발표대회에서 '이차원 배열 데이터에서 유사 구역의 효율적인 탐색 기법'의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : LG CNS 연구원
†† 종신회원 : 숙명여자대학교 컴퓨터과학부 부교수
Manuscript Received : December 15, 2016
Accepted : January 1, 2017
* Corresponding Author : Ki Yong Lee(kiyonglee@sookmyung.ac.kr)

한 구역을 찾는 문제를 다룬다. 2차원 배열 내의 직사각형 모양의 부분 배열을 구역(region)이라 부르자. 동일한 2차원 배열 내의 두 구역이 만약 그에 속한 원소들의 값의 차이가 주어진 범위 내의 값이라면 유사 구역이라고 부른다. 2차원 배열 데이터에서 유사 구역을 찾는 것은 다음과 같이 매우 유용하게 사용될 수 있다.

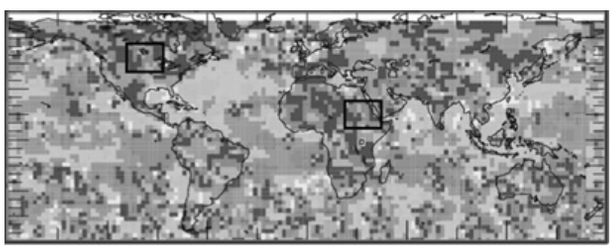


Fig. 1. Raster Data Representing Earth Surface Temperatures

**예 1.** 경찰은 범죄 위험요소와 환경이 비슷한 지역을 찾아 관리하기 위한 시스템이 필요할 수 있다. 만약 지역별 위험 요소가 Fig. 1과 같이 2차원 배열 데이터로 주어진다면, 해당 데이터에서 유사 구역을 찾는 일은 범죄 예방을 위해 도움이 될 것이다.

**예 2.** 산업 발달로 도시화가 진행됨에 따라 난방 시설과 자동차 열 등 인공열로 인해 특정 지역이 주변보다 온도가 3~4℃ 높은 열섬 현상이 일어나고 있다. 지역에 따른 온도 데이터가 2차원 배열 데이터의 형태로 주어졌을 때, 온도 분포가 유사한 구역을 찾아 열섬 현상 등을 파악하여 미리 대처할 수 있다.

하지만 지금까지는 2차원 배열 데이터에서 크기가 미리 알려지지 않은, 서로 유사한 구역을 찾는 방법은 [3] 외에는 거의 연구가 이루어지지 않았다. 유사한 연구로는 여러 이미지 중 주어진 이미지와 비슷한 이미지를 찾거나, 2차원 배열이 아닌 벡터로 표현된 여러 객체들 중 유사한 객체들을 찾는 연구들이 있으나 이들은 찾고자 하는 대상의 크기가 미리 알려져 있다는 점에서 본 연구의 문제와 다르다. 따라서 본 논문은 2차원 배열 데이터에서 크기가 미리 알려지지 않은, 서로 유사한 구역을 효율적으로 찾는 방법을 제안한다.

[3] 외에는 동일한 문제에 대한 기존 연구가 존재하지 않으므로 본 논문에서는 제안 방법의 성능을 다음과 같은 단순 방법과 비교한다. 단순 방법은 우선 유사 구역이 가질 수 있는 모든 크기의 가능한 후보 직사각형 모양들을 생성한다. 이후 각 후보 모양들에 대해 다음을 수행한다. 먼저 2차원 배열 데이터의 원소 $A[1][1]$을 왼쪽 상단 모서리로 하면서 해당 후보 모양과 동일한 모양의 구역을 잡는다. 그리고 $A[1][2]$부터 $A[n][m]$까지 이동하면서 각각을 왼쪽 상단 모서리로 하는 동일한 모양의 구역과 $A[1][1]$을 왼쪽 상단 모서리로 하는 구역의 원소 값을 비교한다. 만약 두 구역의 원소 값의 차이가 사용자가 지정한 값 이하이면 해당 두 구역을 유사 구역으로 반환한다. 이후에는 $A[1][2]$에서 $A[n][m]$까지 각각을 왼쪽 상단 모서리로 하면서 후보 모양과 동일한 모양을 가진 구역을 잡은 뒤 앞과 동일한 과정을 반복한다. 이렇게 한 후보 모양에 대한 탐색이 끝나면 다른 모든 후보 모양에 대해 동일한 과정을 반복한다. 3.2절에서 자세히 설명할 바와 같이, $N = n \times m$ 크기의 2차원 배열 데이터에 대해 단순 방법의 시간 복잡도는 $O(N^3 logN)$이며, $N$이 증가할수록 후보 모양의 개수가 급증하여 유사 구역을 탐색하는데 매우 비효율적이다.

따라서 본 논문은 모든 가능한 후보 직사각형 모양들을 고려하지 않고도 유사 구역을 효율적으로 탐색할 수 있는 방법을 제안한다. 본 논문은 [3]의 연구를 보다 완전한 방법으로 기술한 한편, 실험을 확장한 것이다. 제안 방법은 2차원 배열 데이터 원소들의 모든 쌍에 대해 다음을 수행한다. 우선 쌍을 이루는 두 원소 각각으로만 이루어진 크기가 1인 구역을 생성한다. 이후 두 구역을 동일한 모양을 유지하면서 오른쪽 또는 아래쪽으로 한 단계씩 확장을 진행한다. 만약 두 구역에 속한 원소들의 값 차이가 사용자가 지정한 값 이상으로 커지면 확장을 중단한다. 따라서 제안 방법은 단순 방법과 달리 모든 후보 모양을 고려하지 않고도 배열에서 유사 구역이 될 가능성이 있는 부분들만 접근하여 유사 구역을 효율적으로 찾아낼 수 있다. 4.3절에서 설명할 바와 같이, $N = n \times m$ 크기의 2차원 배열 데이터에 대해 단순 방법의 시간 복잡도는 $O(d \cdot N^2)$이다. 여기서 $d$는 주어진 두 구역에 대해 오른쪽 또는 아래쪽으로 확장이 진행된 평균 횟수를 나타낸다. 따라서 제안 방법은 단순 방법에 비해 효율적이며, 5장에서 실험을 통해 제안 방법의 효율성을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구들을 소개하며, 3장에서는 문제를 정의하고, 본 논문의 제안 방법의 비교 대상이 되는 단순 방법을 기술한다. 4장에서는 제안 방법을 상세히 설명하고 시간 복잡도를 분석한다. 5장에서는 실험 결과를 통해 제안 방법이 단순 방법보다 효율적임을 보이며, 6장에서 결론을 맺는다.

## 2. 관련 연구

본 장에서는 본 논문에서 다루는 유사 구역 탐색과 유사한 연구들을 살펴보고, 배열 데이터에 대한 질의 처리 연구에는 어떤 것들이 있는지 소개한다.

### 2.1 유사 구역 탐색 질의

본 논문에서 다루는 문제와 유사한 문제는 다양한 분야에서 연구되어 왔다[4, 5]. 특히 영상정보처리 분야에서 유사한 이미지를 찾는 문제가 있다. 하지만 영상정보처리 분야에서 유사한 이미지를 찾는 문제는 크기가 이미 알려진 어떤 이미지가 주어졌을 때, 주어진 다른 이미지들 중 그와 가장 유사한 이미지를 찾는 문제이다[6, 7]. 예를 들어 Fig. 2의 경우, 빨간색 테두리의 구역이 주어지면 다른 이미지들 중에서 그와 유사한 구역(파란색 테두리의 구역)을 찾는다. 따라서 본 논문에서 다루는, 크기가 미리 알려져 있지 않은 유사 구역을 탐색하는 문제와는 다르다. 더욱이 유사한 이

미지를 찾는 문제는 본 논문에서 다루는 문제와 같이 두 구역 내의 원소 값들을 바로 비교하기 보다는 컬러 히스토그램(color histogram), 에지 검출(edge detection)등 여러 기법을 이용해 해당 구역의 특징을 추출(feature extraction)하여 1차원 배열인 벡터로 변환하고, 벡터 간의 유사도를 계산하여 두 구역의 유사성을 판별한다. 따라서 2차원 구역 간의 유사도를 고려하는 본 논문의 문제와는 다르다.

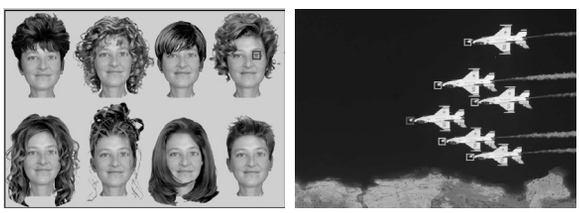


Fig. 2. An Example of Finding Similar Images

영상정보처리 분야 외에도 분자 생물학 분야에서 유사한 연구가 진행되었다. 예를 들어 Fig. 3과 같이 DNA, RNA, 단백질 시퀀스(sequence) 데이터가 주어졌을 때, 여러 시퀀스에서 유사 구역을 탐색하여 공통 염기 서열을 찾는 연구가 활발히 진행되었다[8]. Fig. 3에서 동일한 색으로 표시된 부분은 동일하거나 유사한 구역을 나타낸다. 하지만 분자 생물학에서 다루는 데이터는 Fig. 3에서 보는 바와 같이 1차원 배열로 표현되는 시퀀스 데이터로서, 2차원 배열 데이터를 고려하는 본 논문의 문제와는 다르다.

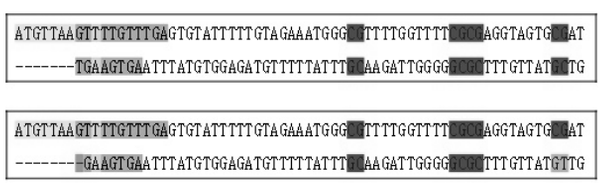



Fig. 3. Examples of Protein Sequence Data

마지막으로 지리공간정보 분야에서도 유사한 연구가 진행되었다. 지리공간정보 분야에서는 주로 기능적으로 유사한 지역을 탐색하는 연구가 진행되었다[9]. 예를 들어, 어떤 사람이 쇼핑하고 밥을 먹기 위해 자주 가는 곳 이외에 쇼핑몰과 음식점이 같이 있는 유사한 지역을 검색하고 싶다면, 자주 가는 지역과 유사한 구역을 검색할 수 있다. 하지만 이 경우에 유사 구역 검색은 각 구역을 그의 특징 및 기능을 TF-IDF 등의 기법을 통해 숫자로 나타내어 벡터로 표현한 뒤, 이들 벡터 간의 유사성을 계산하는 방식으로 이루어진다. 따라서 2차원 구역 간의 유사도를 고려하는 본 논문의 문제와는 다르다.

### 2.2 배열 데이터에서 질의 처리

최근 들어 배열 데이터에 대한 관심이 높아지면서 본 논문에서 다루는 문제 외에도 배열 데이터에 관한 다양한 질의 처리 기법들이 활발히 연구되고 있다. 예를 들어, 배열 데이터에 대한 윈도우 집계 질의(window aggregate query)는 배열 데이터 내부에서 윈도우를 이동시키며 윈도우 내에 있는 원소에 대한 집계 연산을 반복적으로 수행하는 질의이다[9]. 이에 대해서는 윈도우 간의 중복 계산을 제거하는 연구 등이 진행되고 있다[10]. 배열 데이터에 대한 조인(join) 연산도 활발히 연구되고 있다. 배열 간의 조인(join) 연산은 관계형 데이터베이스에서 릴레이션 간의 조인과는 달리 크게 세 가지 형태로 나뉜다[11]. 첫 번째는 두 배열에서 인덱스 값이 같은 원소를 조인하는 것이며, 두 번째는 두 배열에서 원소 값이 같은 원소를 조인하는 것이고, 세 번째는 두 배열에서 원소 값과 인덱스 값이 같은 원소를 조인하는 것이다. 이러한 새로운 형태의 조인은 아직 많은 연구가 필요하며, 현재 다양한 연구가 진행되고 있다. 배열 간의 조인 연산 중 특히 유사 조인(similarity join)에 대한 연구도 많이 진행되고 있다[12]. 이는 모양이 다르거나 원소의 값이 다소 다르더라도 배열의 비슷한 부분을 조인하는 연산으로, 현재 유사도를 다양하게 정의하여 여러 연구가 진행되고 있다. 외에도 샘플링(sampling), 감마연산(gamma operator) 등 배열 데이터에 고유한 다양한 질의에 관한 연구가 진행되어 왔다. 하지만 유사 구역을 탐색하는 질의에 대한 연구는 [3] 외에는 거의 진행된 바가 없다. 따라서 본 논문에서는 배열 데이터에서 유사 구역을 효율적으로 찾는 방법을 제안한다.

## 3. 문제 정의

본 장에서는 본 논문에서 다루는 질의를 정의하고, 해당 질의를 처리하는 단순 방법을 기술한다.

### 3.1 질의 정의

본 논문에서는 $n \times m$ 크기의 2차원 배열 데이터를 $A$라 했을 때, $A$에서 $i$번째 행의 $j$번째 열에 위치한 원소의 값을 $A[i][j]$로 나타낸다($1 \le i \le n$, $1 \le j \le m$). 또한 $A$에서 $i_1$번째 행부터 $i_2$번째 행의 $j_1$번째 열부터 $j_2$번째 열까지만을 포함하는 $A$의 부분 배열을 $A[i_1{:}i_2, j_1{:}j_2]$로 나타내고, 이를 $A$의 한 구역(region)이라 부른다 ($i_1 \le i_2, j_1 \le j_2$). $A$의 어떤 구역 $R = A[i_1{:}i_2, j_1{:}j_2]$에 대해 $R$의 가로 크기와 세로 크기를 각각 $W(R) = i_2 - i_1 + 1$와 $H(R) = j_2 - j_1 + 1$로 나타낸다. $A$의 두 구역 $R_1 = A[i_1{:}i_2, j_1{:}j_2]$, $R_2 = A[i'_1{:}i'_2, j'_1{:}j'_2]$이 다음을 만족하면 서로 유사하다고 한다.

① $W(R_1) = W(R_2)$이고 $H(R_1) = H(R_2)$

② $D(R_1, R_2) \le D_{max}$

$D(R_1, R_2)$는 두 구역 $R_1$, $R_2$의 원소 값의 차이를 나타내며, 이 값이 사용자가 지정한 값인 $D_{max}$를 넘지 않으면 유사하다고 한다. $W(R_1) = W(R_2)$이고 $H(R_1) = H(R_2)$일 때, $D(R_1, R_2)$는 다음과 같이 정의된다.

$$\sum_{i=i_1}^{i_2}\sum_{j=j_1}^{j_2}(A[i][j]-A[(i'_1-i_1)+i][(j'_1-j_1)+j])^2$$

이제 주어진 2차원 배열에서 유사 구역을 찾는 질의를 다음과 같이 정의한다. 이 때, 구해야하는 최소한의 구역 크기를 $S_{min}$이라 하고 사용자가 정의하며, 구역의 크기는 가로와 세로의 곱이다.

**질의 정의:** 2차원 배열 $A$와 $D_{max}$, $S_{min}$이 주어졌을 때, 다음을 만족하는 구역들의 쌍 ($R_1$, $R_2$)를 모두 찾아라.

① $W(R_1) = W(R_2)$이고 $H(R_1) = H(R_2)$
② $D(R_1, R_2) \leq D_{max}$
③ $W(R_1) \cdot H(R_1) = W(R_2) \cdot H(R_2) \geq S_{min}$

즉, 본 논문에서 다루는 질의는 주어진 2차원 배열 데이터에서 유사하면서도 사용자가 지정한 값 이상의 크기를 가지는 구역들의 쌍을 찾는 질의이다.

### 3.2 단순 방법

지금까지는 3.1절에서 정의한, 배열 데이터에서 크기가 미리 알려져 있지 않은 유사 구역을 탐색하는 질의에 대한 선행연구가 존재하지 않는다. 따라서 본 논문에서는 제안 방법의 성능을 다음과 같은 단순 방법과 비교하도록 한다.

단순 방법은 우선 사용자가 지정한 $S_{min}$ 이상의 크기를 가지는 모든 가능한 후보 직사각형 모양을 생성한다. 즉, 입력 배열 $A$의 크기가 $n \times m$이라면, $S_{min}$부터 $n \times (m - 1)$ 혹은 $(n - 1) \times m$까지의 크기를 가지는 모든 가능한 후보 직사각형 모양을 생성한다. 예를 들어, Fig. 4는 입력 배열 $A$의 크기가 8 × 8이고 $S_{min}$ = 4일 때 생성된 모든 가능한 후보 직사각형 모양들의 예를 보여주는 그림이다. 예를 들어, 크기가 4인 직사각형의 모양은 1 × 4, 2 × 2, 4 × 1로 총 3가지이며, 크기가 5인 직사각형의 모양은 1 × 5, 5 × 1로 2가지이고, 크기가 6인 직사각형의 모양은 1 × 6, 2 × 3, 3 × 2, 6 × 1로 총 4가지이다. 이러한 방법으로 크기가 7 × 8, 8 × 7인 직사각형의 모양까지 생성하면 총 87개의 후보 직사각형의 모양이 생성된다.

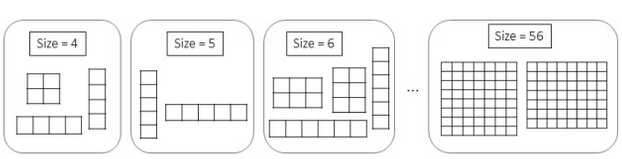



Fig. 4. Examples of Candidate Rectangle Shapes

이후 각 후보 직사각형 모양에 대해 다음을 수행한다. 먼저 $A[1][1]$을 왼쪽 상단 모서리로 하면서 해당 후보 직사각형 모양과 동일한 모양을 가지는 구역 $R_1$을 잡는다. 그리고 $A[1][2]$부터 $A[n][m]$까지의 각 원소에 대해, 각 원소를 왼쪽 상단 모서리로 하면서 해당 후보 직사각형 모양과 동일한 모양을 가지는 구역 $R_2$를 잡아 $D(R_1, R_2) \leq D_{max}$이면 ($R_1$, $R_2$)를 유사 구역으로 반환한다. 이후에는 $A[1][2]$에서 $A[n][m]$까지 각각에 대해 해당 원소를 왼쪽 상단 모서리로 하면서 해당 후보 직사각형 모양과 동일한 모양을 가지는 구역을 $R_1$으로 잡은 뒤 앞과 동일한 과정을 반복한다. 이렇게 한 후보 직사각형 모양에 대한 탐색이 끝나면 다른 후보 직사각형 모양에 대해서도 동일한 과정을 반복한다.

단순 방법의 경우, 유사 구역이 될 구역의 크기를 미리 알 수 없으므로 유사 구역이 될 가능성이 있는 후보 직사각형의 모양을 모두 생성한다. 후보 직사각형 모양의 총 개수를 $k$라고 하고, 입력 배열 $A$의 총 원소의 개수를 $N = n \times m$이라 하자. 단순 방법은 각 후보 직사각형 모양에 대해 중첩 반복(nested-loop) 방식으로 $A$를 검색하므로 총 시간 복잡도는 $O(kN^2)$가 된다. 여기서 $k$는 크기가 $S_{min}$부터 $N$까지인 모든 후보 직사각형 모양의 총 개수이다. 크기가 $s$인 직사각형의 개수는 직사각형의 가로 크기를 $w$라 하고 세로 크기를 $h$라 할 때 $s = w \cdot h$를 만족하는 모든 ($w$, $h$) 쌍의 개수와 동일하다. (단, $w$와 $h$는 양의 정수) $s = w \cdot h$를 만족하는 모든 ($w$, $h$) 쌍의 개수는 $s$의 약수 개수와 동일하므로, $s$의 약수 개수를 $d(s)$라 하면 $k = d(S_{min}) + d(S_{min} + 1) + ... + d(N) = O(N\log N)$이다[13]. 따라서 단순 방법의 총 시간 복잡도는 $O(kN^2) = O(N^3\log N)$가 된다. 따라서 단순 방법은 $N$이 증가할수록 후보 모양의 개수가 급증하여 유사 구역을 탐색하는 데 매우 비효율적이다.

## 4. 제안 방법

본 장에서는 본 논문에서 제안하는, 2차원 배열 데이터에서 크기가 미리 알려져 있지 않은 유사 구역을 효율적으로 탐색하는 방법을 자세히 설명한다.

### 4.1 개요

제안 방법은 모든 후보 직사각형 모양을 생성하는 대신 다음과 같은 방법으로 유사 구역을 탐색한다. 우선 주어진 2차원 배열 데이터 $A$에서 서로 다른 두 원소를 선택한다. 그리고 두 원소 각각으로만 이루어진 크기가 1 × 1인 구역 $R_1$과 $R_2$를 각각 생성한다. 이후 두 구역을 동일한 모양을 유지하며 오른쪽 또는 아래쪽으로 한 단계씩 확장을 진행해 나간다. 만약 확장을 진행하다가 두 구역의 원소 값의 차이 $D(R_1, R_2)$가 사용자가 지정한 값 $D_{max}$보다 커지면 확장을 중단한다. 만약 확장이 중단된 후에 $R_1$과 $R_2$의 크기가 $S_{min}$보다 크면 ($R_1$, $R_2$)를 유사 구역으로 사용자에게 반환한다. 이러한 확장 연산을 $A$의 서로 다른 원소 쌍에 대해 수행하면 유사 구역을 모두 찾을 수 있다. 따라서 제안 방법은 단순 방법과 달리 모든 후보 직사각형 모양을 고려하지 않고도 배열에서 유사 구역이 될 가능성이 있는 부분들만 접근하여 유사 구역을 효율적으로 찾아낼 수 있다.

Fig. 5는 앞서 설명한 확장 연산의 예를 보여주기 위한 그림이다. 만약 두 원소 $A[2][2]$와 $A[6][3]$가 주어졌다고 하자. 우선 각각으로부터 크기가 1인 두 구역 $R_1$ = $A[2{:}2, 2{:}2]$과 $R_2$ = $A[6{:}6, 3{:}3]$을 생성한다. 이후 $R_1$과 $R_2$를 동일한 방식으로 오른쪽과 아래쪽으로 확장시켜간다. Fig. 5에서 회색으로 표시된 구역은 $R_1$과 $R_2$가 각각 $R_1$ = A[2:3, 2:3], $R_2$ =

[6:7, 3:4]로 확장된 모습을 나타낸다. 만약 $D_{max}$ = 4이고 $S_{min}$ = 4라면, 두 구역은 크기가 $S_{min}$ 이상이고 $D(R_1, R_2)$ = 2 < $D_{max}$이므로 유사 구역으로 출력된다. 다음 절에서는 여기서 언급한 확장 연산을 상세히 설명한다.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

Fig. 5. Example of a 2-dimensional Array Data

### 4.2 확장 연산

본 절에서는 크기가 1 × 1인 두 구역 $R_1$과 $R_2$가 주어졌을 때 두 구역을 확장하는 연산을 상세히 설명한다. $R_1$과 $R_2$의 확장을 표현하기 위해 본 논문에서는 차이 그래프(difference graph)라는 개념을 사용한다. Fig. 6은 차이 그래프의 예를 보여주는 그림이다.

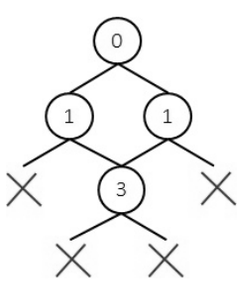



Fig. 6. Example of a Differentce Graph

차이 그래프의 루트(root) 노드는 $R_1$과 $R_2$의 크기가 모두 1 × 1일 때의 $D(R_1, R_2)$ 값을 나타낸다. 예를 들어, Fig 5에서 $R_1$ = $A$[2:2, 2:2], $R_2$ = $A$[6:6, 3:3]일 때 두 구역에 대한 차이 그래프의 루트 노드 값은 $D(R_1, R_2)$ = 0이 된다. 만약 루트 노드 값이 $D_{max}$보다 크면 더 이상 확장은 진행되지 않는다. 만약 루트 노드 값이 $D_{max}$보다 적으면 다음과 같이 확장이 진행된다. $R_1$과 $R_2$를 각각 아래쪽으로 크기 1만큼 확장시켜 크기가 각각 2 × 1인 $R_1$ = $A$[2:3, 2:2], $R_2$ = $A$[6:7, 3:3]를 만들고 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$를 계산한다. 만약 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$가 $D_{max}$보다 작으면 루트 노드에 왼쪽 자식 노드를 생성하고, 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$를 왼쪽 자식 노드에 기록한다. 동일한 방법으로 $R_1$과 $R_2$를 각각 오른쪽으로 크기 1만큼 확장시켜 크기가 각각 1 × 2인 $R_1$ = $A$[2:2, 2:3], $R_2$ = $A$[6:6, 3:4]를 만들고 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$를 계산한다. 만약 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$가 $D_{max}$보다 작으면 루트 노드에 오른쪽 자식 노드를 생성하고, 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$를 오른쪽 자식 노드에 기록한다. 만약 아래쪽이든 오른쪽이든 확장된 $R_1$과 $R_2$에 대한 $D(R_1, R_2)$가 $D_{max}$보다 크면 $X$로 표시된 노드를 생성한다. 이후에는 왼쪽 자식 노드와 오른쪽 자식 노드 각각에 대해 $X$로 표시된 노드가 아니면 재귀적으로 동일한 연산을 진행한다. 만약 모든 말단 노드가 $X$로 표시되면 확장 연산이 완료된다. 확장 연산이 진행되는 도중 차이 그래프에서 크기가 $S_{min}$보다 큰 구역을 나타내는 노드가 생성되면 해당 노드가 나타내는 구역 $(R_1, R_2)$을 유사 구역으로 출력한다.

Fig. 6은 Fig. 5의 $A$[2:2, 2:2]과 $A$[6:6, 3:3]가 각각 $R_1$과 $R_2$로 주어졌을 때 생성되는 차이 그래프를 나타낸다. 루트 노드는 $R_1$과 $R_2$의 크기가 1 × 1일 때의 $D(R_1, R_2)$를 나타낸다. 루트 노드의 왼쪽과 오른쪽 자식 노드는 각각 $R_1$과 $R_2$의 크기가 2 × 1과 1 × 2로 확장되었을 때의 $D(R_1, R_2)$를 나타낸다. 루트 노드의 왼쪽 자식 노드의 왼쪽 자식 노드가 $X$로 표시된 것은 $R_1$과 $R_2$의 크기가 3 × 1로 확장되면 $D(R_1, R_2)$가 $D_{max}$를 넘어감을 의미한다. 반면 루트 노드의 왼쪽 자식 노드의 오른쪽 자식 노드가 3으로 표시된 것은 $R_1$과 $R_2$의 크기가 2 × 2로 확장되어도 $D(R_1, R_2)$가 $D_{max}$를 넘지 않음을 나타낸다. 하지만 $R_1$과 $R_2$의 크기가 3 × 2, 2 × 3, 또는 1 × 3으로 확장되면 $D(R_1, R_2)$가 $D_{max}$를 넘어가므로 $X$로 표시된 노드가 생성된다.

Fig. 6에서 볼 수 있듯이 루트 노드의 왼쪽 자식 노드의 오른쪽 자식 노드와, 루트 노드의 오른쪽 자식 노드의 왼쪽 자식 노드는 모두 $R_1$과 $R_2$의 크기가 각각 2 × 2로 확장된 동일한 경우를 나타낸다. 이러한 경우 제안 방법은 동일한 노드를 중복해서 만들지 않고, 먼저 만들어진 노드를 공유한다. 예를 들어 Fig. 6에서 루트 노드의 왼쪽 자식 노드에서 3으로 표시된 오른쪽 자식 노드가 먼저 만들어진 경우, 루트 노드의 오른쪽 자식 노드는 왼쪽 자식 노드를 새로 만들지 않고 이미 만들어진 노드를 공유한다. 이로 인해 제안 방법은 중복된 계산을 피할 수 있다.

```
1: function findSimilarRegions(A, Smin, Dmax)
2:   for i1 = 1 to n
3:     for j1 = 1 to m
4:       R1 = A[i1:i1, j1:j1]; // a region of size 1 × 1
5:       for i2 = i1 to n
6:         for j2 = j1 + 1 to m
7:             R2 = A[i2:i2, j2:j2]; // a region of size 1 × 1
8:             if (D(R1, R2) ≤ Dmax)
10:              expandRegions(R1, R2);
11:            end if
12:        end for
13:      end for
14:    end for
15:  end for
16: end function
```

Fig. 7. Pseudo-code of the Proposed Method

### 4.3 전체 알고리즘 및 성능 분석

Fig. 7은 지금까지 설명한 제안 방법을 나타내는 의사 코드이다. 제안 방법은 $n \times m$ 크기의 2차원 배열 데이터 $A$, 유사 구역의 최소 크기 $S_{min}$, 유사 구역 간 최대 차이 $D_{max}$를 입력으로 받는다. 제안 방법은 $A$의 각 원소 $A[i_1][j_1]$에 대해 다음을 수행한다. 먼저 $A[i_1][j_1]$로만 구성된 크기가 1 × 1인 구역 $R_1 = A[i_1{:}i_1, j_1{:}j_1]$을 생성한다. 그리고 $A$에서 $A[i_1][j_1]$보다 오른쪽에 위치한 모든 원소 $A[i_2][j_2]$ ($i_2 \geq i_1$, $j_2 \geq j_1 + 1$) 각각에 대해 다음을 수행한다. 먼저 $A[i_2][j_2]$로만 구성된 크기가 1 × 1인 구역 $R_2 = A[i_2{:}i_2, j_2{:}j_2]$을 생성한다. 만약 $D(R_1, R_2) \leq D_{max}$이면 $R_1$과 $R_2$에 대해 확장 연산을 수행하는 expandRegions($R_1$, $R_2$)를 호출한다.

expandRegions($R_1$, $R_2$)은 주어진 두 구역 $R_1$과 $R_2$에 대한 확장 연산을 수행한다. 만약 $R_1$과 $R_2$이 확장되는 도중 $D(R_1, R_2) \leq D_{max}$과 $W(R_1) \cdot H(R_1) \geq S_{min}$ (또는 $W(R_2) \cdot H(R_2) \geq S_{min}$) 조건이 만족되면 해당 ($R_1$, $R_2$)를 유사 구역으로 반환한다. Fig. 8은 expandRegions() 함수의 의사 코드이다. exapndRegions() 함수는 두 구역 $R_1$과 $R_2$를 매개변수로 받는다. 만약 $R_1$과 $R_2$의 크기가 $S_{min}$을 넘는다면, expandRegions() 함수는 $D(R_1, R_2) \leq D_{max}$인 $R_1$과 $R_2$에 대해서만 호출되는 함수이므로 ($R_1$, $R_2$)를 유사 구역으로 출력한다. 그리고 $R_1$과 $R_2$를 아래쪽으로 크기 1만큼 확장시키고 확장된 $R_1$과 $R_2$에 대해 $D(R_1, R_2)$를 계산한다. 만약 $D(R_1, R_2) \leq D_{max}$이면 아래쪽으로 확장된 $R_1$과 $R_2$에 대해 expandRegions() 함수를 재귀적으로 호출한다. 또한 $R_1$과 $R_2$를 오른쪽으로도 크기 1만큼 확장시키고 확장된 $R_1$과 $R_2$에 대해 $D(R_1, R_2)$를 계산한다. 만약 $D(R_1, R_2) \leq D_{max}$이면 오른쪽으로 확장된 $R_1$과 $R_2$에 대해서도 expandRegions() 함수를 재귀적으로 호출한다.

```
1:  function expandRegions(R1, R2)
2:    // if expandRegions(R1, R2) has been already called,
3:    // the following is not executed.
4:    if (W(R1)·H(R1) ≥ Smin)
5:      output (R1, R2);
6:    end if
7:    R1' ← expand R1 downward by adding one row;
8:    R2' ← expand R2 downward by adding one row;
9:    if (D(R1', R2') < Dmax)
10:     expandRegions(R1', R2');
11:   end if
12:   R1'' ← expand R1 to the right by adding one column;
13:   R2'' ← expand R2 to the right by adding one column;
14:   if (D(R1'', R2'') < Dmax)
15:     expandRegions(R1'', R2'');
16:   end if
17: end function
```

Fig. 8. Pseudo-code of the Function expandRegions()

이제 제안 방법의 성능을 분석한다. 입력 배열 $A$의 총 원소 개수를 $N = n \times m$이라 하자. 제안 방법은 $A$의 서로 다른 원소 쌍들에 대해 각각 expandRegions()를 호출한다. 원소 쌍들의 개수는 $O(N^2)$이다. 또한 exapndRegions()의 시간 복잡도는 차이 그래프에서 생성된 노드 개수에 비례한다. 차이 그래프에서 생성된 평균 노드 개수를 $d$라고 하자. 이 경우 제안 방법의 총 시간 복잡도는 $O(d \cdot N^2)$가 된다. 차이 그래프에서 생성된 노드 개수는 두 구역 $R_1$과 $R_2$가 아래쪽과 오른쪽으로 확장된 횟수를 나타낸다. 따라서 $A$의 원소 값들이 서로 다른 일반적인 경우, 확장 횟수는 그리 크지 않을 것이므로 $O(d \cdot N^2) < O(N^3 \log N)$가 되어 제안 방법은 단순 방법에 비해 효율적이다. 그러나 $A$의 원소 값이 모두 서로 비슷한 최악의 경우, $d \approx N$이 되어 제안 방법의 성능도 $O(N^3)$으로 떨어지게 된다. 하지만 이 경우도 제안 방법은 단순 방법에 비해 효율적이다. 따라서 제안 방법은 모든 후보 직사각형 모양을 고려하는 단순 방법에 비해 매우 효율적으로 유사 구역을 찾을 수 있음을 알 수 있다.

## 5. 성능 실험

본 장에서는 실험을 통해 본 논문에서 제안한 방법의 효율성을 평가한다.

### 5.1 실험 방법

본 실험에서는 제안 방법의 성능을 3.2절에서 설명한 단순 방법과 비교하였다. 성능 척도로는 질의 처리에 걸리는 총 수행시간(초)를 사용하였다. 실험을 위해 2차원 배열 데이터를 임의로 생성하였으며, 배열의 각 원소는 [1, 100] 범위의 균등분포에서 임의로 추출한 값을 가진다. 실험에서는 성능에 영향을 줄 수 있는 다음 세 가지 변수들을 변화시키며 제안 방법과 단순 방법의 성능을 비교하였다.

① 배열 데이터 크기: 2차원 배열의 크기를 200 × 200, 300 × 300, 400 × 400, 500 × 500로 변화시켜가며 성능을 측정하였다.
② 사용자 정의 값 $D_{max}$의 크기: 두 구역이 유사하다고 판정하는데 사용되는 기준인 $D_{max}$의 값을 2, 3, 4, 5로 변화시켜가며 성능을 측정하였다. 일반적으로 $D_{max}$가 증가할수록 유사하다고 판정받는 구역이 증가한다.
③ 사용자 정의 값 $S_{min}$의 크기: 유사 구역으로 판정받기 위한 최소 크기인 $S_{min}$의 값을 2, 3, 4, 5로 변화시켜가며 성능을 측정하였다. 일반적으로 $S_{max}$가 증가할수록 유사하다고 판정받는 구역이 감소한다.

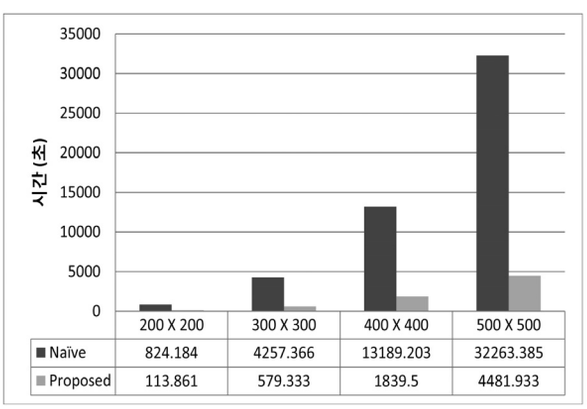
제안 방법과 단순 방법은 모두 Java로 구현하였으며, 실험은 윈도우 10 환경 기반의 CPU Intel 헥사 하스웰 i7 5820K, 메모리 8GB인 컴퓨터에서 수행하였다.

### 5.2 실험 결과

본 절에서는 배열 크기, $D_{max}$, $S_{min}$을 각각 변화시켜가며 제안 방법의 성능을 단순 방법과 비교한 결과를 보인다. Fig. 9는 배열의 크기를 200 × 200, 300 × 300, 400 × 400, 500 × 500으로 증가시키며 제안 방법(Proposed)과 단순 방법(Naive)이 유사 구역을 찾는데 걸리는 시간을 각각 측정한 결과이다. 이 때 사용자 정의 값 $D_{max}$와 $S_{min}$은 각각 4로 고정시켰다. 예상한 바와 같이 입력 데이터가 커질수록 두 방법 모두 탐색 시간이 증가한다. 하지만 제안 방법은 평균적으로 약 7 ~ 8배 단순 방법에 비해 좋은 성능을 보이고 있다. 이것은 제안 방법이 크기가 $S_{min}$ 이상인 모든 후보 직사각형 모양을 고려하지 않고도 필요한 부분만을 확장 연산을 통해 탐색하기 때문이다.

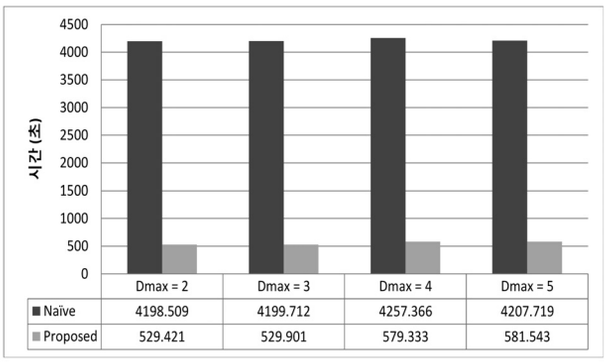
Fig. 10은 $D_{max}$ 값을 2, 3, 4, 5로 변화시켜가며 두 방법의 성능을 비교한 결과이다. 배열 데이터의 크기는 300 × 300로, $S_{min}$는 4로 고정시켰다. 단순 방법은 $D_{max}$ 값에 상관없이 모든 후보 직사각형 모양을 각각 탐색하기 때문에 성능은 $D_{max}$ 값에 별다른 영향을 받지 않는다. 그에 비해 제안 방법은 그래프로는 잘 드러나지 않지만 $D_{max}$ 값이 증가하면 성능이 약간 저하되는 것을 볼 수 있다. 이것은 제안 방법의 경우 $D_{max}$ 값이 클수록 확장이 더 많이 일어나 시간이 더 많이 걸리기 때문이다. 반면에 $D_{max}$ 값이 작을수록 확장이 적게 일어나 시간이 적게 걸린다. 하지만 $D_{max}$ 값에 상관없이 제안방법이 단순 방법보다 좋은 성능을 보여주고 있음을 알 수 있다.

Fig. 11은 $S_{min}$의 크기를 2에서 5까지 증가시키며 두 방법의 성능을 평가한 결과이다. 배열 데이터의 크기는 300 × 300로, $D_{max}$는 4로 고정시켰다. 단순 방법은 $S_{min}$ 값이 커질수록 생성해야 하는 후보 직사각형의 모양이 줄어들기 때문에 탐색시간이 감소된다. 제안 방법의 확장 연산은 $S_{min}$ 값에 큰 영향을 받지는 않으나 $S_{min}$ 값이 커질수록 출력하는 유사 구역의 수가 줄어들기 때문에 총 수행시간이 다소 감소하는 경향을 보인다. 하지만 이 실험에서도 $S_{min}$값에 관계없이 제안 방법이 단순 방법보다 좋은 성능을 보인다는 것을 알 수 있다. 따라서 탐색 성능에 영향을 미치는 3개 변수를 변화시켜가며 실험을 수행한 결과, 제안 방법은 효율적으로 유사 구역을 탐색함을 확인할 수 있었다.



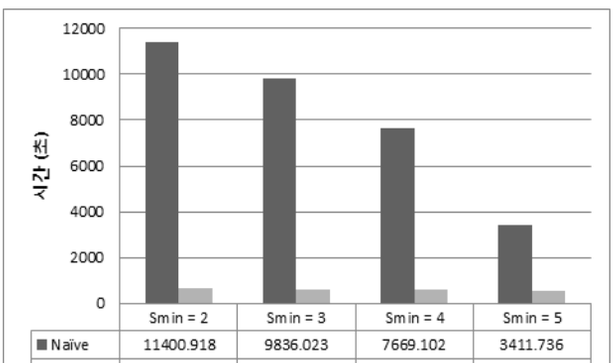

| | 200 X 200 | 300 X 300 | 400 X 400 | 500 X 500 |
|---|---|---|---|---|
| Naïve | 824.184 | 4257.366 | 13189.203 | 32263.385 |
| Proposed | 113.861 | 579.333 | 1839.5 | 4481.933 |

Fig. 9. Performance Evaluation for Difference Array Sizes




| | Dmax = 2 | Dmax = 3 | Dmax = 4 | Dmax = 5 |
|---|---|---|---|---|
| Naïve | 4198.509 | 4199.712 | 4257.366 | 4207.719 |
| Proposed | 529.421 | 529.901 | 579.333 | 581.543 |

Fig. 10. Performance Evaluation for Difference Values of $D_{max}$




| | Smin = 2 | Smin = 3 | Smin = 4 | Smin = 5 |
|---|---|---|---|---|
| Naïve | 11400.918 | 9836.023 | 7669.102 | 3411.736 |
| Proposed | 656.088 | 588.696 | 579.333 | 572.295 |

Fig. 11. Performance Evaluation for Difference Values of Size_min

## 6. 결 론

본 논문은 2차원 배열 데이터에서 크기가 미리 알려져 있지 않은, 서로 유사한 구역을 효율적으로 탐색하는 방법을 제안하였다. 본 논문은 2차원 배열 데이터에서 유사 구역을 찾는 문제를 정의하고, 동일한 문제에 대한 기존 연구가 없으므로 비교 대상이 되는 단순 방법을 제시하였다. 단순 방법은 가능한 크기의 후보 직사각형 모양을 모두 생성하고, 각 후보 직사각형 모양에 대해 배열을 검색하여 유사 구역을 탐색한다. 단순 방법은 가능한 크기의 후보 직사각형 모양을 모두 고려하므로 배열이 커질수록 매우 비효율적이며, 배열에 속한 원소의 개수가 $N$일 때 $O(N^3 \log N)$의 시간 복잡도를 가진다.

제안 방법은 모든 가능한 후보 직사각형 모양을 생성하는 대신, 배열의 각 원소 쌍에 대해 각 원소가 나타내는 1 × 1 크기의 구역을 점진적으로 확장해 나가는 방법을 사용한다. 이러한 확장은 확장된 두 구역 간의 원소 값의 차이가 $D_{max}$를 넘지 않을 때까지 진행된다. 확장이 종료되고 만약 확장된 두 구역의 크기가 $S_{min}$을 넘으면 해당 두 구역을 유사 구역으로 출력한다. 따라서 제안 방법은 유사 구역에 포함될 가능성이 있는 부분만 접근하므로 매우 효율적으로 유사 구역을 탐색할 수 있다. 1 × 1 크기로부터 시작된 두 구역의

평균 확장 횟수가 $d$인 경우, 제안 방법은 $O(d \cdot N^2)$의 시간 복잡도를 가진다. 따라서 제안 방법은 $d$가 그리 크지 않은 일반적인 경우, 단순 방법에 비해 매우 효율적이다.

실험에서는 탐색 방법의 성능에 영향을 미치는 배열 크기, $D_{max}$, $S_{min}$ 값을 변화시키며 제안 방법과 단순 방법의 성능을 비교하였다. 실험 결과 제안 방법이 단순 방법에 비해 약 7 ~ 8배 이상 좋은 성능을 보임을 확인하였다.

본 논문에서는 찾는 구역의 모양을 직사각형으로 제한하였다. 추후 연구로는 직사각형이 아닌 임의의 모양을 가지는 유사 구역을 탐색하는 방법을 연구할 계획이다.

## References


[1] The Sloan Digital Sky Survey [Internet], http://www.sdss.org/.

[2] Palomar Transient Factory [Internet], http://www.ptf.caltech.edu/.

[3] YeonJeong Choe and Ki Yong Lee, "Efficient Search of Similar Regions in Two-Dimensional Array Data," *KIPS Fall Conference*, November, 2016.

[4] Agrawal, Rakesh, Christos Faloutsos, and Arun Swami, "Efficient similarity search in sequence databases," *International Conference on Foundations of Data Organization and Algorithms,* Springer Berlin Heidelberg, 1993.

[5] Li, Ming, Bin Ma, and Lusheng Wang, "Finding similar regions in many strings," *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, ACM, 1999.

[6] N. Kumar, L. Zhang, and S. K. Nayar, "What is a good nearest neightbors algorithm for finding similar patches in images?" in *European Conference on Computer Vision (ECCV)*, II: 364-378, 2008.

[7] Q. Lv, M. Charikar, and K. Li, "Image Similarity Search with Compact Data Structures," in CIKM'04, November 8 - 13, 2004.

[8] Ming Li, Bin Ma, and Lusheng Wang, "Finding Similar Regions in Many Sequences," *Journal of Computer and System Sciences,* Vol.65, pp.73-96, 2002.

[9] C. Sheng and Y. Zheng, "Answering Top-k Similar Region Queries," Database Systems for Advanced Applications, Lecture Notes in Computer Science, 5981:186-201, 2010.

[10] L. Jiang, H. Kawashima, and O. Tatebe, "Incremental Window Aggregates over Array Database," *IEEE International Conference on Big Data*, 2004.

[11] D. V. Kalashnikov, "Super-EGOL fast multi-dimensional similarity join," *VLDB Journal,* Vol.4, No.2, pp.561-585, 2013.

[12] Weijie Zhao, Florin Rusu, Bin Dong Kesheng Wu, "Similarity Join over Array Data," in *Proceedings of ACM SIGMOD*, 2016.

[13] Divisor function [Internet], https://en.wikipedia.org/wiki/Divisor_function.



**최 연 정**

e-mail : cyj@sookmyung.ac.kr
2015년 숙명여자대학교 컴퓨터과학부(학사)
2017년 숙명여자대학교 컴퓨터과학과(석사)
2017년~현 재 LG CNS 연구원
관심분야 : 데이터베이스, 빅데이터, 배열데이터



**이 기 용**

e-mail : kiyonglee@sookmyung.ac.kr
1998년 KAIST 전산학과(학사)
2000년 KAIST 전산학과(석사)
2006년 KAIST 전산학과(박사)
2006년~2008년 삼성전자 책임연구원
2008년~2010년 KAIST 전산학과 연구조교수
2010년~현 재 숙명여자대학교 컴퓨터과학부 부교수
관심분야 : 데이터베이스, 빅데이터, 데이터마이닝, 질의처리