구간 데이타에 대한 최대 에러 히스토그램 구축

이호석
서울대학교
hslee@kdd.snu.ac.kr

심규석°
서울대학교
shim@ee.snu.ac.kr

이병기
포항공과대학교
bkyi@postech.ac.kr

Max Error Histogram Construction for Interval Data

Hoseok Lee
Seoul Nat'l University

Kyuseok Shim°
Seoul Nat'l University

Byoung-kee Yi
Postech

요 약

히스토그램은 원본 데이타를 효과적으로 요약하는 기법중의 하나 이며, 선택도 측정과 근사 질의 처리 등에 널리 사용되고 있다. 기존의 히스토그램 구축 알고리즘들은 하나의 값으로 표현되는 점 데이타에 대하여 적용 가능한 알고리즘 이었다. 그러나 일상생활에서는 하루 동안의 온도, 주식 가격과 같은 구간 데이타들도 점 데이타만큼 흔하게 접할 수 있다. 본 논문에서는 기존의 Max 에러에 대한 히스토그램 구축 알고리을 구간 데이터에 대하여 확장한다. 합성 데이타를 사용한 실험을 통하여 기존의 점 데이타에 대한 히스토그램을 초보적으로 확장하는 방법보다 본 논문에서 제시된 알고리즘의 성능이 좋다는 것을 보였다.

1. 서 론

웨이블릿[2]과 함께 히스토그램[5, 7]은 데이타를 요약하는 일반적인 도구이며, 질의 최적화를 위한 선택도 측정[6]및 근사 질의 처리[1] 등에 흔히 사용되고 있다.

지금까지 연구되어 온 히스토그램 구축 알고리즘들은 대부분 점 데이타를 다루고 있다. 그러나 최고점과 최저점으로 나타내어지는 구간 데이타를 여러 분야에서 흔하게 접할 수 있는데 예를 들어, Yahoo 웹 사이트[10]에서는 일별, 주별, 월별로 각 주식의 최대 가격, 최소 가격을 포함한 데이타를 제공하고 있으며, 병원에서는 최고 혈압과 최저 혈압의 구간으로 환자의 혈압데이타를 저장하고, 기상청에서는 하루 동안의 온도를 최고 기온, 최저 기온의 구간으로 제공한다. 그러나 구간 데이타를 일상생활에서 흔하게 접할 수 있는데도 불구하고 우리가 아는 한, 현재까지 구간 데이타에 대한 히스토그램 구축 알고리즘은 연구되지 않았으며, 기존의 점 데이타에 대한 히스토그램 구축 알고리즘을 구간 데이타에 초보적으로 확장하여 적용하는 것은 에러가 많이 발생할 수 있다. 본 논문에서는 [9]에서 제시된 두 구간 사이의 L_1에러의 정의를 사용하여 최대 L_1 에러를 최소화 시키는 구간 데이타에 대한 히스토그램 구축 알고리즘을 제시한다.

2. 관련 연구

히스토그램에 대한 연구는 데이타베이스 분야에서 오래전부터 연구되어왔으며, 대부분의 경우에 기존의 히스토그램 구축 알고리즘은 점 데이타를 다루고 있다. [7]에서 제시된 V-Optimal 히스토그램은 절대 오차 제곱의 합을 에러 척도로 사용하는 최적 히스토그램으로써 동적 프로그래밍(Dynamic Programming)을 이용하여 구축된다. V-Optimal 히스토그램 구축 알고리즘은 n개의 데이타가 주어지고, B개의 버킷을 사용하여 히스토그램을 구축할 때, 시간 복잡도와 공간 복잡도는 각각 $O(n^2B)$과 $O(nB)$이다. 그렇기 때문에 V-Optimal 히스토그램 구축 알고리즘은 데이터의 크기가 큰 경우에 히스토그램을 구축하기 위해 매우 많은 시간이 소요된다. 이러한 단점을 해결하고자 참고 문헌 [3,4]에서는 V-Optimal 히스토그램의 에러보다 $(1+\varepsilon)$배 만큼 에러를 허용하면서 더 빠르게 구축될 수 있는 히스토그램 구축 알고리즘을 제시하였다.

위의 히스토그램 구축 알고리즘들을 비롯하여 대부분의 히스토그램 구축 알고리즘은 절대 오차에 대한 히스토그램을 다루는 알고리즘이었다. 그러나 많은 응용 분야에서는 절대 오차보다 상대 오차를 중요시 한다. 이러한 점에 착안하여 [5]에서 상대 오차를 기준으로 하는 히스토그램 구축 알고리즘 REHIST가 제시되었다.

3. 기본 개념

점 데이타에 대한 히스토그램 구축 문제의 정의는 다음과 같다.

정의 3.1. [히스토그램 구축 문제]
n개의 점 데이타 $x_1, x_2, \cdots, x_n$, S개의 숫자를 저장할 수 있는 히스토그램 저장 공간, 에러 척도 E()가 주어졌을 때, 히스토그램 구축 문제는 최대 S개의 숫자를 저장하면서 주어진 에러 척도 E(H)를 최소화 시키는 히스토그램 H를 찾는 것이다.

히스토그램은 보통 전체 데이터 구간을 여러 개의 구간으로 분할한 후에 각 구간을 하나의 버킷으로 표현한다. 이 때, 분할된 하나의 구간에 속하는 연속적인 점들 $x_i(i\in[s_r, e_r])$를 하나의 대표값 x_r를 이용하여 나타낸다. 여기서 하나의 버킷을 표현하기 위해서는 버킷의 시작점 s_r, 끝점 e_r, 버킷의 대표값 x_r에 대한 정보가 필요하다.

그림 1은 8개의 데이타를 3개의 버킷을 사용하여 구축한 히스토그램을 보여주고 있다. 그림 1.(a)와 같이 8개의 데이타가 있을 때, 그림 1.(b)가 하나의 히스토그램이며 각 컬럼이 하나의 버킷이다. 이 히스토그램은 1, 2번 데이타를 대표값 5로 나타내고 3부터 6번 까지의 데이타는 대표값 21로 나타내며 7,

· 본 연구는 2006년도 정보통신부 지원 대학 IT 연구센터 육성지원사업의 연구비 지원으로 수행하였습니다.

8번 데이타는 대표값 10으로 나타내는 히스토그램이다.

그림 1에서 보듯이 j번째 버킷의 시작점은 (j-1)번째 버킷의 끝점 다음점이라는 것을 알 수 있으므로, 저장 공간을 효율적으로 사용하기 위하여 시작점 또는 끝점 중 하나와 대표값을 저장함으로써 하나의 버킷을 표현할 수 있다. 따라서 S개의 숫

i	1	2	3	4	5	6	7	8
xi	6	4	20	22	18	24	8	12

(a) 데이타 분포

구 간	[1,2]	[3,6]	[7,8]
대표값	5	21	10

(b) 히스토그램

그림 1. 히스토그램의 예

자를 저장할 수 있는 저장 공간이 주어지면 ⌊S/2⌋(앞으로는 B라 부르겠음)개의 버킷을 사용할 수 있게 된다. 히스토그램 H가 구축된 후에는 원본 데이타의 갯수인 n대신 O(B)만큼의 공간을 사용하여 원본 데이터에 근사한 값을 저장 할 수 있으며, i번째 데이타 값인 $x_i(1\le i\le n)$를 묻는 질의에 대한 답을 이미 구축된 히스토그램 H를 이용하여 구할 수 있다. 이 때 히스토그램을 구축한 후에는 근사적인 값만이 저장 되어있기 때문에, 실제 데이타 x_i와 대표값 x_r 사이에 오차가 발생한다.

[1,n]의 구간 중에서 임의의 구간 [i,j]에 해당하는 x_i, x_{i+1}, …, x_j의 최대값과 최소값을 구하려면 O(n)시간이 필요한데, 아래에 정의된 구간 트리를 이용하면 O(logn)시간 안에 구할 수 있다는 것이 [5]에서 제시하고 증명하였다. 본 논문에서 제시하는 구간 데이터에 대한 히스토그램 구축 알고리즘도 이와 비슷한 일을 하기 위하여 구간 트리를 사용하기 때문에 아래의 구간 트리를 소개한다.

정의 3.5 [구간 트리]
주어진 구간 [1, n]에 대해 다음의 4 가지 조건을 만족하는 이진 트리를 구간 트리라 정의한다.

- 트리의 루트는 전체 구간 [1, n]에 대응한다.
- 트리의 리프 노드들은 길이가 1인 구간, 즉, [i, i]에 대응한다.
- 구간 [i, j]에 대응하는 노드의 왼쪽 자식 노드는 구간 [i, ⌊(i+j+1)/2⌋-1]에 대응하고, 오른쪽 자식 노드는 구간 [⌊(i+j+1)/2⌋, j]에 대응한다.
- 구간 [i, j]에 대응하는 트리의 노드에 대해, x_i, x_{i+1}, …, x_j의 최대값과 최소값인 max, min을 저장한다.

구간 트리를 이용하여 임의의 구간 [i,j]에 속하는 데이터들의 min과 max를 구하는 방법을 다음의 예제를 통하여 살펴보자.

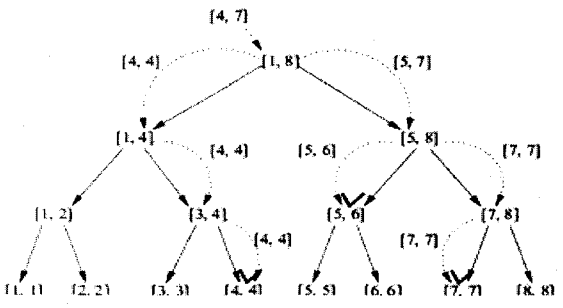


그림 2. [4, 7]의 min, max계산 과정

예제 3.3. 그림 2는 [5]에서 제시된 방법을 기반으로 하여 구간 [4, 7]를 분해하는 과정을 그림으로 보여주고 있다. 루트 노드에 대응하는 구간이 [1,8]이므로 구간 [1,4]의 부분 구간에 대응하는 노드는 왼쪽 자식을 루트로 하는 트리에서 찾을 수 있으며 [5, 8]의 부분 구간에 대응하는 노드는 오른쪽 자식을 루트로 하는 트리에서 찾을 수 있다. 따라서 구간 [4,7]을 두 개의 부분 구간 [4,4], [5,7]로 분해하여 각각을 왼쪽 자식과 오른쪽 자식을 루트로 하는 트리에서 찾도록 한다. 위와 같은 방법을 재귀적으로 적용함으로써 구간 [4,7]의 분할 {[4,4], [5,6], [7,7]}의 각각의 원소에 대응하는 노드를 구간 트리에서 찾을 수 있으며, 이러한 정보를 이용하여 구간 [4, 7]에 속하는 데이타들의 max와 min을 구할 수 있다.■

또한, [5]에서 알고리즘의 효율성을 위하여 아래의 정리를 이용하였다. 본 논문에서 제시하는 구간 데이터에 대한 히스토그램 구축 알고리즘도 이 정리를 사용하기 때문에 아래의 정리를 소개한다.

정리 3.4. F(x)와 G(x)가 각각 단조 증가, 단조 감소 함수일 때, F(i+1) > G(i+1)와 F(i) ≤ G(i)을 만족하는 i에 대해서 $\min_x\{\max(F(x), G(x))\} = \min\{G(i), F(i+1)\}$이다.(그림 3참고)

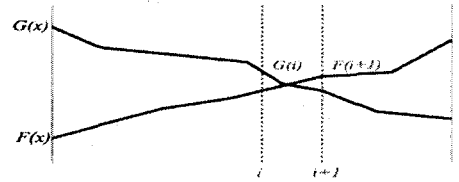


그림 3. 이진 탐색을 이용한 최대 L_1에러 계산

4. 구간 데이타에 대한 최적 히스토그램 구축 알고리즘

이 장에서는 구간 데이타에 대한 히스토그램의 소개와 기존의 히스토그램의 초보적인 확장을 통한 구간 데이타에 대한 히스토그램을 구축하는 방법을 소개한 후 구간 데이타에 대한 최적 히스토그램 구축 알고리즘을 제시할 것이다.

4.1 구간 데이타에 대한 히스토그램의 개요

구간 데이타는 일상생활에서 흔히 접할 수 있는 데이타 형태로, 예로는 하루 동안의 주식 가격, 하루 동안의 온도 등이 있으며, $x_i = [lx_i, hx_i]$ $(lx_i \le hx_i)$로 표현할 수 있다. 구간 데이타에 대한 히스토그램 문제의 정의는 정의 3.1과 크게 다른 것이 없으며, 두 개의 구간 데이타 간의 에러는 [9]에서 정의한 것을 이용하여 구간 데이타에 대한 히스토그램에서 사용하는 에러 척도를 정의할 수 있다. 본 논문에서는 구간 데이타에 대한 히스토그램의 에러 척도로 최대 L_1 에러를 사용할 것이며, 하나의 버킷에서의 최대 L_1 에러는 다음과 같이 정의된다.

정의 4.1. [구간 데이타에 대한 버킷의 최대 L_1에러]
구간 $[s_r, e_r]$에 속하는 구간들을 하나의 대표 구간 $[l_r,h_r]$로 나타낼 때, 버킷의 최대 L_1 에러는

$$\mathrm{Err}_{\mathrm{ML}_1}(s_r, e_r) = \max\left(|lx_i - l_r| + |hx_i - h_r|\right)$$

로 정의된다.

$\mathrm{Terr}_{\mathrm{ML}1}[j, k]$는 구간 데이타 x_1, x_2, …, x_j에 대하여 최대 L_1에러를 에러 척도로 사용하고 k개의 버킷을 사용하는 최적 히스토그램의 에러라 정의하자. 그러면 최대 L_1 에러를 최소화

하는 히스토그램 문제는 $Terr_{ML1}$[n, B]를 에러로 가지는 히스토그램을 찾는 문제가 된다.

구간 데이타에 대한 히스토그램에서의 하나의 버킷 b_r은 버킷의 시작점 s_r, 버킷의 끝점 e_r, 버킷의 대표 구간 $[l_r,h_r]$등의 정보를 가지고 있어야 하며, (s_r, e_r, l_r, h_r)로 표현할 수 있다. 점 데이타에 대한 히스토그램처럼 j번째 버킷의 시작점은 (j-1)번째 버킷의 끝점 바로 다음점이기 때문에 s_r과 e_r중 하나만 저장하면 되고, 따라서 구간 데이타에 대한 히스토그램에서 하나의 버킷은 3개의 숫자를 저장함으로 표현할 수 있다. 즉, 점 데이타에 대한 히스토그램 구축 알고리즘에서는 S개의 숫자를 저장할 수 있는 저장 공간이 주어졌을 때, $\lfloor S/2 \rfloor$개의 버킷을 사용할 수 있었지만, 구간 데이타에 대해서는 $\lfloor S/3 \rfloor$개의 버킷만을 사용할 수 있게 된다.

4.2. 점 데이타 히스토그램 구축 알고리즘의 초보적인 확장

구간 데이타를 기존의 점 데이타에 대한 히스토그램 구축 알고리즘에 초보적으로 확장 적용시키는 방법을 먼저 설명한다.

n개의 구간 데이타 x_1(=[lx_1, hx_1]), x_2(=[lx_2, hx_2]), …, x_n(=[lx_n, hx_n])를 L=lx_1, lx_2, …, lx_n과 H=hx_1, hx_2, …, hx_n의 n개로 이루어진 두 개의 점 데이타 시퀀스로 나타낼 수 있다. 가장 쉽게 생각할 수 있는 방법은 첫 번째 점 데이타 시퀀스 L에 $\lfloor B/2 \rfloor$개의 버킷을, 두 번째 점 데이타 시퀀스 H에 B-$\lfloor B/2 \rfloor$개의 버킷을 사용하여 최대 절대 오차를 최소화 하는 히스토그램을 구축하는 것이다.

다음으로 L와 H를 접합하여 2n개로 이루어진 점 데이타 시퀀스 lx_1, lx_2, …, lx_n, hx_1, hx_2, …, hx_n에 B개의 버킷을 사용하여 최대 절대 오차를 최소화 하는 히스토그램을 구축하는 방법을 생각해 볼 수 있다. 두 번째 방법은 저장 공간을 L과 H에 대해서 S/2씩 사용하는 것이 아니라 S를 양쪽에 최적으로 나누어 줄 수 있으므로 히스토그램의 에러가 더 줄을 수 있다. 5장의 실험에서 본 논문에서 제시하는 알고리즘의 우수성을 보이기 위해 위의 두 가지 방법과 비교 실험을 수행하였다.

구간 데이타 [l, h]는 $l \le h$라는 성질이 있다. 그러나 위의 두 가지 방법으로 히스토그램을 구축하면, 어떤 구간 데이타 x_i=[lx_i, hx_i]의 대표 구간 [l_r, h_r]이 $l_r > h_r$인 경우가 존재할 수도 있다는 문제점이 있다. 이러한 경우는 구간 데이타의 작은 값이 큰 값보다 커지는 경우로 다음의 예를 통해 이러한 경우가 발생한다는 것을 확인할 수 있다.

예제 4.2. 4개의 구간을 가진 데이타 x_1=[2, 4], x_2=[2, 100], x_3=[10, 100], x_4=[100, 104]가 있다고 하자. 이 데이타를 두 개의 시퀀스 L=2,2,10,100와 H=4, 100, 100, 104로 나타낼 수 있다. 4개의 버킷을 사용하여 최대 L_1에러를 최소화 하는 히스토그램을 구축하는데 위의 방법을 사용해 보도록 하자. 첫 번째 방법은 L과 H에 각각 2개의 버킷을 사용함으로써 히스토그램 구축 알고리즘을 적용하는 것이다. 먼저, L에 2개의 버킷을 사용하여 히스토그램을 구축 구축하면, 2, 2, 10이 대표값 6으로 하나의 버킷이 되고, 100이 대표값 100으로 다른 하나의 버킷이 된다. H에 2개의 버킷을 사용하여 똑같이 적용하게 되면, 4가 대표값 4로 하나의 버킷이 되고, 100, 100, 104이 대표값 102로 하나의 버킷이 된다. 여기서 x1=[2, 4]에 대한 대표값을 확인해 보면, 최저점(lx_1) 2은 6으로, 최고점(hx_1) 4는 4로 표현됨에 따라 최저점이 최고점보다 더 커지게 되는 현상이 발생하게 된다. 두 번째 방법을 사용해도 같은 결과가 나온다. ■

4.3. 최대 L_1에러를 최소화 하는 히스토그램 구축 알고리즘

최대 L_1에러를 최소화 하는 히스토그램을 구축하는 알고리즘을 만들기 위해서는 먼저 하나의 버킷에서 최대 L_1에러를 최소화하는 대표 구간과 그 때의 최대 L_1에러를 구할 수 있어야 한다. 버킷 b_r=(s_r, e_r, l_r, h_r)의 최대 L_1에러는 정의 4.1에서 보았듯이 $Err_{ML_1}(s_r, e_r) = \max(|lx_i - l_r| + |hx_i - h_r|)$ 이며, 구간 데이타를 2차원 평면위의 점으로 생각함으로써 최대 L_1 에러를 최소화하는 대표 구간과 그 때의 최대 L_1 에러를 계산할 수 있다.

4.3.1. 최대 L_1 에러를 최소화 하는 대표 구간과 최대 L_1 에러의 계산

최대 L_1 에러를 최소화 하는 대표 구간과 최대 L_1 에러를 구하는 기본적인 아이디어는 구간 데이타를 2차원 상의 점으로 생각하는 것이다. 다음 몇 가지의 정리와 보조 정리를 이용하여 이를 구할 수 있다.

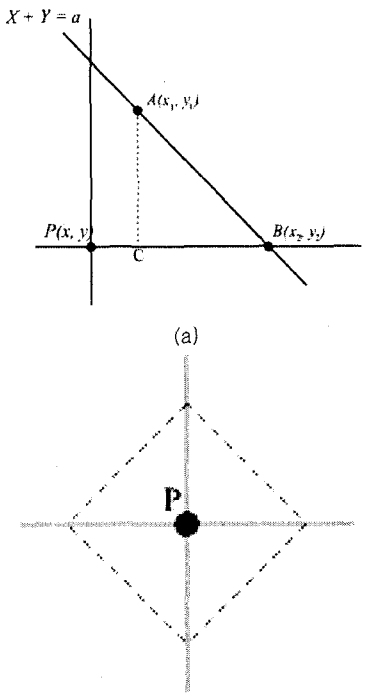


그림 4. L_1 거리의 특성

보조 정리 4.3. 2차원 상의 점 P(x, y)를 기준으로 하여 4분면으로 나누었을 때, 그림 4.(a)와 같이 점 P를 기준으로 1사분면을 지나는 직선 x+y=a가 있다고 하자. 그러면 P에서 직선 x+y=a위에 있으면서 1사분면에 속하는 모든 점까지의 L_1 거리는 모두 같다.

보조정리 4.4. 2차원 상의 점 P(x, y)에서부터 L_1 거리가 같은 점들의 모임은 점 P를 중심으로 하는 45도 회전된 정사각형 모양이다.(그림 4.(b)참고)

정리 4.5. k개의 구간 데이타 x_i(=[lx_i, hx_i]), x_2(=[lx_2,

hx_2]), …, x_k(=[lx_k, hx_k])가 주어졌을 때, A, B, C, D를 각각 $max_{1\le i\le k}\{hx_i+lx_i\}$, $min_{1\le i\le k}\{hx_i+lx_i\}$, $max_{1\le i\le k}\{hx_i-lx_i\}$, $min_{1\le i\le k}\{hx_i-lx_i\}$이라 하자. 그러면 k개의 구간 데이타를 고려한 최대 L_1 에러를 대표 구간은 [A+B-C-D/4, A+B+C+D/4] 이며, 그 때의 최대 L_1에러는 $max\{(A-B)/2, (C-D)/2\}$가 된다.

증명 구간 데이타를 2차원 평면상의 점으로 생각을 하면, 두 개의 구간 데이터 간의 L_1 에러는 2차원 평면상의 두 개의 점 간의 L_1 거리가 되고, k개의 구간 데이타에 대하여 최대 L_1 에러를 최소화 하는 대표 구간을 찾는 것은 k개의 점으로부터 대표점까지의 L_1거리 중에서 최대인 것을 최소화하는 대표점 (x^*, y^*)를 구하는 것이 된다.

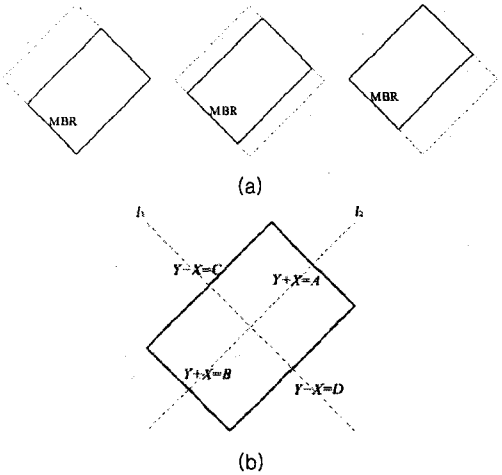


그림 5. 최대 L_1 거리를 최소화하는 대표값

점 (x,y)를 대표점으로 하였을 때, (x,y)로부터 L_1거리가 가장 큰 점을 z라 하자. 그리고 (x,y)로부터 z까지의 L_1거리와 같은 L_1거리를 가지는 점들의 집합을 고려해 보자. 이 집합은 보조정리 4.4에 의해서 점 (x,y)를 중심으로 하고 45도 회전된 정사각형임을 알 수 있다. 이 정사각형을 R이라 하자. 그리고 45도 회전된 최소 경계 직사각형 (MBR:Minimum Bounding Rectangle)을 45도 회전 MBR이라고 부르기로 하겠다. 점 (x,y)에서부터 R까지의 L_1거리가 모든 점을 고려한 최대 L_1 거리임을 알 수 있다. 여기서 주의하여 살펴 볼 것은 R이 (x,y)를 중심으로 하는 정사각형 중에서 k개의 점을 포함하는 45도 회전 MBR을 포함하는 가장 작은 정사각형이라는 것이다. 이러한 이유는 MBR의 정의상 MBR의 네 개의 변 각각에는 적어도 k개의 점 중에 하나의 점이 존재하는데[8], R이 MBR의 일부를 포함하지 않게 되면 MBR의 어떤 한 변을 모두 포함하지 않게 되고 그러면 R이 k개의 점 중 적어도 하나의 점을 포함하지 못하게 되기 때문이다. 또한 (x,y)로 부터 L_1거리가 가장 큰 점 z가 MBR의 네 변중 하나의 변 위에 있기 때문에 R보다 더 작으면서 (x,y)를 중심으로 하는 45도 회전된 정사각형들은 45도 회전 MBR을 포함하지 못하기 때문이다. 즉, 임의의 점 (x,y)을 k개의 점들에 대한 대표점으로 하였을 때 (x,y)를 중심으로 하며 45도 회전 MBR을 포함하는 가장 작은 45도 회전된 정사각형 R까지의 거리가 최대 L_1거리가 된다. (x,y)에서 R까지의 거리는 R의 대각선의 거리에 비례하며 따라서 변의 길이에 비례한다. 그렇기 때문에 최대 L_1거리를 최소화하기 위해서는 45도 회전 MBR을 포함하는 45도 회전된 정사각형의 변의 길이가 최소화 되어야한다. 여기서 최대 L_1거리를 최소로 만드는 대표점 (x^*,y^*)는 45도 회전 MBR을 포함하는 가장 작은 정사각형의 중심점으로 하면 된다.

그러나 실제로 45도 회전 MBR을 포함하는 가장 작은 45도 회전된 정사각형은 여러 개가 존재할 수 있으며(그림 5.(a)참고), 여러 개의 정사각형 중 하나를 찾아서 그 정사각형의 중심점을 k개의 점에 대한 대표점으로 하면 된다. 여러 가지 45도 회전 MBR을 포함하는 가장 작은 45도 회전된 정사각형 중에서 비교적 쉽게 찾을 수 있는 것은 45도 회전 MBR의 중심을 중심점으로 하고 45도 회전 MBR을 포함하는 가장 작은 45도 회전된 정사각형 R*이다. R*의 중심점은 그림 5.(b)에서의 직선 l_1 : Y+X=(A+B)/2와 l_2 : Y-X=(C+D)/2의 교점이므로, (x^*, y^*) = ((A+B-C-D)/4,(A+B+C+D)/4)가 되며, (x^*, y^*)로 부터 정사각형 R*까지의 거리는 max{(A-B)/2, (C-D)/2}이 된다. ■

4.3.2. 구간 트리와 이진 탐색의 적용

최대 L_1 에러를 최소화 하는 히스토그램을 구축하기 위해서 $Terr_{ML1}[j,k] = min_{1\le i\le j-1}\{max(Terr_{ML1}[i,k-1], Err_{ML1}(i+1, j))$와 같이 동적 프로그래밍을 위한 재귀 식을 세울 수 있다.

정리 4.5에 의해 Err_{ML1}를 계산하기 위해서는 버킷의 범위에 속하는 데이타들에 대한 변수 A, B, C, D를 구해야 하며, 변수 A, B, C, D는 [5]에서 사용된 구간 트리를 구간 [i, j]에 해당하는 노드에서 그 범위에 속하는 데이타들의 최대값과 최소값인 max, min 대신에 A, B, C, D를 저장하도록 변형함으로써 O(logn) 시간 안에 구할 수 있다.

$Terr_{ML1}[i, k-1]$은 x_1, x_2, …, x_i의 데이타를 (k-1)개의 버킷을 사용하였을 때, 최적 히스토그램의 에러이므로 i가 증가할수록 (k-1)개의 버킷으로 포함해야 할 데이타의 갯수가 많아진다. 따라서, $Terr_{ML1}[i, k-1]$은 i에 대해 단조 증가 한다는 것을 알 수 있다. 반면, $Err_{ML1}(i+1, j)$는 x_{i+1}, …, x_j를 하나의 버킷으로 나타낼 때의 최대 L_1 에러를 나타내고, i가 증가할수록 포함해야 하는 데이타의 갯수는 적어진다. 따라서 REHIST[5]에서 사용하였던 이진 탐색을 이용할 수 있다.

```
Procedure OptErr_ML1()
 1. Create an IntervalTree
 2. for j:=1 to n do {
 3.   Terr_ML1[j, 1] := Err_ML1(1, j)
 4.   for k:=1 to B do {
 5.     Terr_ML1[j, k] := ∞
 6.     low := 1
 7.     high := j
 8.     while high - low > 1 do {
 9.       mid := ⌊(high+low)/2⌋
10.       if Terr_ML1[mid, k-1] < Err_ML1(mid+1, j)
11.         low := mid
12.       else
13.         high := mid
14.     }
15.     temp := Err_ML1(low+1, j)
16.     if Terr_ML1[low+1, k-1] < temp
17.       Terr_ML1[j, k] := min(Terr_ML1[j, k], Terr_ML1[low+1, k-1])
19.     else
20.       Terr_ML1[j, k] := min(Terr_ML1[j, k], temp)
21.   }
22. }
end
```

그림 6 $OptErr_{ML1}$ 알고리즘

지금까지의 내용을 기반으로 최대 L_1 에러를 최소화 하는 히스토그램 구축 알고리즘이 그림 6에 제시되어 있다. 1번째 줄에서 구간 트리를 구축하여 3, 10, 15번째 줄에서 Err_{ML1}를 계산하는데 사용되고 있으며 nB개의 엔트리를 가지는 테이블 $Terr_{ML1}$를 채우기 위해 8~14번째 줄에서 이진 탐색을 수행하고 있다.

구간 트리를 이용하여 임의의 구간 [i, j]에 대한 $Err_{ML1}(i, j)$는 구간 트리를 이용하면 [5]에서 처럼 O(logn) 시간 안에 구할 수 있으므로, 결론적으로 n개의 구간 데이타에 대한 B개의 버킷을 사용하여 최대 L_1 에러를 최소화하는 히스토그램은 구간 트리와 이진 탐색을 이용하여 $O(nB\log^2 n)$의 시간과 O(nB)의 공간을 이용하여 찾을 수 있다.

5. 실험 결과

이 장에서는 점 데이터에 대한 히스토그램 구축 알고리즘의 초보적인 확장을 통한 알고리즘과 본 논문이 제시하는 알고리즘을 합성 데이타와 실생활 데이타를 가지고 비교 실험 하였다. 구현은 C++로 하였으며 펜티엄 4 2.8GHz의 CPU와 512MB의 메모리가 장착된 컴퓨터에서 운영체제는 Linux를 구동시켜 실험하였다.

5.1. 실험 데이타

실험 결과를 보기에 앞서 실험에 사용한 합성 데이타에 대하여 설명할 것이다.

합성 데이타는 Zipf 분포를 따르는 점 데이타를 생성한 후 점 데이타를 구간 데이타로 변형하는 방법을 사용하여 생성하였다. Zipf 파라미터는 0.3 부터 2.0까지 변화시켰고, 데이타 수가 $128(=2^7)$부터 $16384(=2^{14})$인 데이타 집합을 생성하였다. 각 데이타 집합에 속하는 모든 데이타의 합은 데이타의 수와 관계없이 1,000,000으로 하였다. 그림 7에 Zipf 파라미터 1.0, 데이타 수 2,048인 점 데이타 집합에 대한 그래프가 있으며, 그래프는 로그 스케일로 나타내었다.

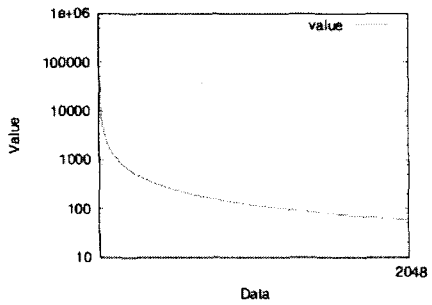


그림 7. 점 데이타의 형태

위와 같이 점 데이타를 생성한 후, 각각의 점 데이타를 구간 데이타로 바꾸기 위해서 1에서 10사이의 난수 r을 발생시켜 점 데이타 x를 [x-r, x+r]과 같은 식으로 변환하여 구간 데이타 집합을 생성하였다.

5.2. 비교 알고리즘

각각의 에러 척도에 대하여 다음의 세 가지 히스토그램 구축 알고리즘을 비교하였다. n개의 구간 데이타 $X=\{x_1(=[lx_1, hx_1]), x_2(=[lx_2, hx_2]), \cdots, x_n(=[lx_n, hx_n])\}$ 에 대하여, S개의 숫자를 저장하는 히스토그램을 구축할 때,

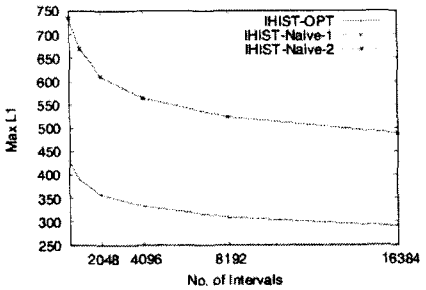
- IHIST-Naive-1 : X를 두 개의 점 데이타 시퀀스 L과 H로 분리하여 L뒤에 H를 접합하여 2n개의 점 데이타 시퀀스로 만들어 $\lfloor S/2 \rfloor$개의 버킷을 사용하여 히스토그램을 구축.
- IHIST-Naive-2 : X를 두 개의 점 데이타 시퀀스 L과 H로 분리하여 L에 $\lfloor S/4 \rfloor$개의 버킷을 사용하고, H에 $\lfloor S/2 \rfloor - \lfloor S/4 \rfloor$개의 버킷을 사용하여 히스토그램을 구축.
- IHIST-OPT : 구간 데이타 자체에 $\lfloor S/3 \rfloor$개의 버킷을 사용하여 히스토그램을 구축.

5.3. 합성 데이타 집합의 실험 결과

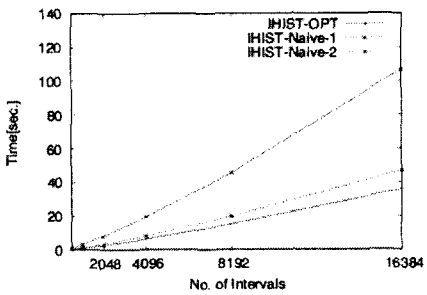
실험은 Zipf 파라미터, 데이타 집합의 크기, 히스토그램의 저장 공간을 변화시키면서 4가지의 데이타 형태를 사용였으며, 디폴트 세팅은 Zipf 파라미터를 1.0으로, 데이타 집합의 크기를 2,048로, 히스토그램의 저장 공간을 100으로 하여 실험을 수행하였다.

5.3.1. 데이타 집합 크기의 영향

데이타 집합의 크기는 $128(=2^7)$부터 $16384(=2^{14})$까지 변화시키면서 실험을 하였으며, 데이타 집합의 영향을 보여주는 실험 결과가 그림 8에 있다. 그래프에서는 데이타 집합의 크기가 128, 256일 때까지 하나의 그래프에 나타내면, y축 값의 범위가 너무 커져서 그래프를 확인하는데 어려움이 있기 때문에 데이타 집합의 크기를 512부터 그래프에 나타내었다.



(a) 최대 L_1에러



(b) 수행시간

그림 8. 최대 L_1 에러에 대한 데이터 집합 크기의 영향

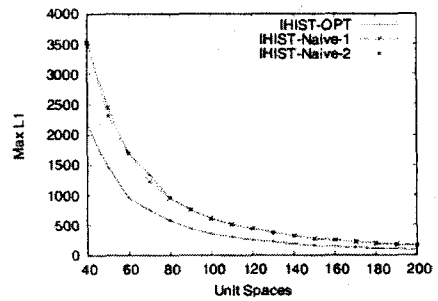
실험 결과 평균적으로 40%정도 개선시키는 것으로 확인되었다. 그래프에서 볼 수 있듯이, 모든 형태의 데이타 집합에서 데이타 집합의 크기가 커질수록 세 가지 에러 척도의 에러가 작아졌다. 이유는 데이타 집합의 크기에 관계없이 데이타들의 합이 1,000,000으로 유지하였으므로, 데이타 집합의 크기가 커질수록 하나의 데이타 값은 작아지기 때문에, 에러도 작아지는

것으로 판단된다. 또한 IHIST-Naive-1와 IHIST-Naive-2의 에러는 대부분 같기 때문에, 그래프에서 겹쳐져 보인다. 앞으로의 합성 데이타 집합에 대한 실험에서도 대부분 IHIST-Naive-1와 IHIST-Naive-2의 실험 결과가 같게 나오는 것을 확인할 수 있었다.

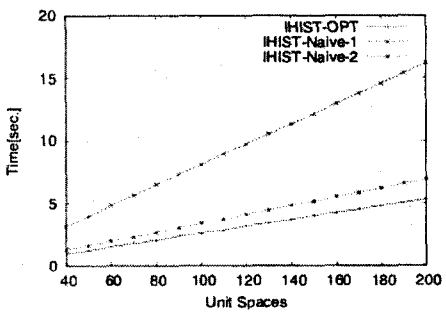
알고리즘의 수행 시간에 대해서는 예상대로 대부분의 경우에 IHIST-Naive-1가 가장 오래 걸리고, IHIST-OPT가 가장 적게 걸리는 것을 확인할 수 있다.

5.3.2. 히스토그램의 저장 공간의 영향

히스토그램의 저장 공간 S는 10부터 200까지 변화시키면서 실험을 하였다. 앞에서도 말했듯이, 히스토그램의 저장 공간이 100으로 주어진다면 점 데이타에 대한 히스토그램(IHIST-Naive-1과 IHIST-Naive-2)은 50개의 버킷을 사용할 수 있고, 구간 데이타에 대한 히스토그램(IHIST-OPT)은 33개의 버킷을 사용할 수 있다. 그림 9의 그래프에서는 데이타 집합 크기의 영향에 대한 그래프와 마찬가지로, 히스토그램의 저장 공간이 40이하일 때까지 하나의 그래프에 나타내면, y축 값의 범위가 너무 커져서 확인하는데 어려움이 있기 때문에 히스토그램의 저장 공간을 40부터 나타내었다.



(a) 최대 L_1에러



(b) 수행시간

그림 9. 최대 L_1 에러에 대한 저장 공간의 영향

저장 공간이 커질수록 많은 수의 버킷을 사용할 수 있으므로 에러는 줄어들게 될 것이라는 예상을 할 수 있으며 그래프에서 확인 할 수 있다. 최대 L_1 에러를 최대 45%정도 개선시키는 것을 확인할 수 있었다. 각 알고리즘의 시간 복잡도가 저장 공간에 비례한다는 것이 수행 시간의 그래프에 잘 나타나있다.

5. 결론

히스토그램은 하나의 값으로 표현되는 점 데이타를 효과적으로 요약하는 기법중의 하나 이며, 선택도 측정과 근사 질의 처리 등에 널리 사용되고 있다. 하지만 일상생활에서는 하루 동안의 온도, 주식 가격과 같은 구간 데이타들도 흔하게 접할 수 있다. 본 논문에서는 기존의 최대 에러에 대한 히스토그램 구축 알고리즘을 구간 데이터에 대하여 확장한 IHIST-OPT를 제안 하였다. 이 알고리즘은 n개의 구간 데이타에 대한 B개의 버킷을 사용하여 최대 L_1 에러를 최소화하는 히스토그램을 구간 트리와 이진 탐색을 이용하여 $O(nB\log^2 n)$의 시간과 $O(nB)$의 공간을 이용하여 찾는다. 합성 데이타를 사용한 실험을 통하여 기존의 점 데이타에 대한 히스토그램을 초보적으로 확장하는 방법보다 본 논문에서 제시된 알고리즘의 성능이 좋다는 것을 보였다.

참고문헌

[1] P. B. Gibbons, Y. Matias, and V. Poosala. "Fast Incremental Maintenance of Approximate Histograms." In Proc. of VLDB, Athens, Greece, Aug. 1997.

[2] S. Guha, C. Kim, and K. Shim. "XWAVE: Approximate Extended Wavelets for Streaming Data." In Proc. of VLDB, Toronto, Canada, Sep. 2004.

[3] S. Guha, and N. Koudas. "Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation." In Proc. of ICDE, San Jose, California, USA, Feb. 2002.

[4] S. Guha, N. Koudas, and K. Shim. "Data Streams and Histograms." In Proc. on STOC, Heraklion, Crete, Greece, Jul. 2001.

[5] S. Guha, K. Shim, and J. Woo. "REHIST : Relative Error Histogram Construction Algorithms." In Proc. of VLDB, Toronto, Canada, Sep. 2004.

[6] Y. E. Ioannidis. "Universality of serial histograms." In Proc. of VLDB, Dublin, Ireland, Aug. 1993.

[7] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. "Optimal Histograms with Quality Guarantees." In Proc. of VLDB, New York City, New York, USA, Aug. 1998.

[8] N. Roussopoulos, S. Kelley, and F. Vincent. "Nearest Neighbor Queries." In Proc. of SIGMOD, San Jose, Calfornia, USA, May 1995.

[9] B. Yi, and J. Roh. "Similarity Search for Interval Time Sequences." DASFAA, Jeju Island, Korea, Mar. 2004.

[10] http://biz.yahoo.com/r.

[11] http://www.fi.edu/weather/data2/index.html.