구간 데이타에 대한 히스토그램 구축 알고리즘의 확장

(Extensions of Histogram Construction Algorithms for Interval Data)

이 호 석 † 심 규 석 †† 이 병 기 †††
(Hoseok Lee) (Kyuseok Shim) (Byoung-Kee Yi)

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

키워드 : 히스토그램, 구간 데이타, 다이나믹 프로그래밍

Abstract Histogram is one of tools that efficiently summarize data, and it is widely used for selectivity estimation and approximate query answering. Existing histogram construction algorithms are applicable to point data represented by a set of values. As often as point data, we can meet interval data such as daily temperature and daily stock prices. In this paper, we thus propose the histogram construction algorithms for interval data by extending several methods used in existing histogram construction algorithms. Our experiment results, using synthetic data, show our algorithms outperform naive extension of existing algorithms.

Key words : Histogram, Interval Data, Dynamic Programming

1. 서 론

웨이블릿[1]과 함께 히스토그램[2,3]은 데이타를 요약하는 일반적인 도구이며, 질의 최적화를 위한 선택도 측정[4]및 근사 질의 처리[5] 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 있다. 예를 들어, Yahoo 웹 사이트[6]에서는 일별, 주별, 월별로 각 주식의 최대 가격, 최소 가격을 포함한 데이타를 제공하고 있으며, 병원에서는 최고 혈압과 최저 혈압의 구간으로 환자의 혈압데이타를 저장한다. 본 논문에서는 [7]에서 제시된 두 구간 사이의 L_1, L_2에러의 정의를 사용하여 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합을 최소화 시키는 구간 데이타에 대한 히스토그램 구축 알고리즘을 제시한다.

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성 · 지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음

† 정 회 원 : 서울대학교 전기컴퓨터공학부
hslee@kdd.snu.ac.kr
†† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
shim@ee.snu.ac.kr
††† 종신회원 : 포항공대 컴퓨터공학과 교수
bkyi@postech.ac.kr

논문접수 : 2007년 1월 22일
심사완료 : 2007년 4월 28일

2. 관련 연구

기존의 히스토그램 구축 알고리즘은 점 데이타를 다루고 있다. [3]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. [8,9]에서는 V-Optimal 히스토그램의 에러보다 $(1+\varepsilon)$배 만큼 에러를 허용하면서 더 빠른 히스토그램 구축 알고리즘을 제시하였다. 또한 많은 응용 분야에서는 절대 오차보다 상대 오차를

중요시하기 때문에 [2]에서는 상대 오차를 기준으로 하는 히스토그램 구축 알고리즘 REHIST가 제시되었다.

3. 기본 개념

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 데이타 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.

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 이용하여 하나의 버킷을 표현할 수 있다. 따라서 S개의 숫자를 저장할 수 있는 저장 공간이 주어지면 $\lfloor S/2 \rfloor$ (앞으로 B라 부르겠음)개의 버킷을 사용할 수 있게 된다. 히스토그램 H가 구축된 후에는 원본 데이타의 갯수인 n대신 B만큼의 공간을 사용하여 원본 데이타의 요약정보를 저장하며, 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 요약정보만이 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, $x_{i+1}, \cdots, x_j$의 최대값과 최소값을 구하려면 아래에 정의된 구간 트리를 이용하면 O(logn)시간 안에 구할 수 있다는 것이 [2]에서 제시되고 증명되었다. 본 논문에서 제시하는 구간 데이타에 대한 히스토그램 구축 알고리즘도 이와 비슷한 일을 하기 위하여 구간 트리를 사용하기 때문에 아래의 구간 트리를 소개한다.

i	1	2	3	4
x_i	6	4	20	22
i	5	6	7	8
x_i	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

정의 3.2 [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 $[i, \lfloor (i+j+1)/2 \rfloor -1]$에 대응하고, 오른쪽 자식 노드는 구간 $[\lfloor (i+j+1)/2 \rfloor, j]$에 대응한다.
- 구간 [i, j]에 대응하는 트리의 노드에 대해, x_i, x_{i+1}, $\cdots$, x_j의 최대값과 최소값인 max, min을 저장한다.

예제 3.3. 그림 2는 구간 [4,7]를 분해하는 과정을 보여준다. 루트 노드에 대응하는 구간이 [1,8]이므로 구간 [1,4]의 부분 구간에 속하는 노드는 왼쪽 자식을 루트로 하는 트리에서 찾을 수 있으며 [5,8]의 부분 구간에 속하는 노드는 오른쪽 서브트리에서 찾을 수 있다. 따라서 구간 [4,7]을 두 개의 부분 구간 [4,4], [5,7]로 분해하여 각각을 왼쪽과 오른쪽 서브트리에서 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 데이타의 *max*와 *min*을 구할 수 있다. □

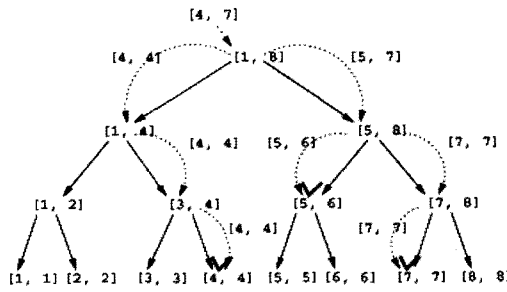


그림 2 [4, 7]의 min, max 계산 과정

마지막으로 [2]에서는 상대 오차의 합을 최소화 시키는 히스토그램 구축 알고리즘에서 알고리즘의 효율성을 위하여 이진탐색트리를 사용하였다. 데이타의 구간 [sr, er]을 하나의 대표값 x_r로 표현할 때 상대 오차의 합은 $P(x_r) \cdot x_r + Q(x_r)$이며 여기서

$$P(x_r) = 2 \sum_{x_i : x_i \le x_r} \frac{1}{\max c, |x_i|} - \sum_{i=s_r}^{e_r} \frac{1}{\max c, |x_i|}$$

$$Q(x_r) = \sum_{i=s_r}^{e_r} \frac{x_i}{\max c, |x_i|} - 2 \sum_{x_i : x_i \le x_r} \frac{x_i}{\max c, |x_i|}$$

이다. [2]에서는 구간 [sr, er]의 데이타 중 $P(x) \ge 0$을 만족하는 가장 작은 데이타를 대표값으로 하였을 때의

상대 오차의 합이 가장 작아진다는 것을 증명하고 그러한 데이타와 그 때의 상대 오차의 합을 구하기 위하여 이진탐색트리를 이용하였다.

4. 구간 데이타에 대한 최적 히스토그램 구축 알고리즘

4.1 구간 데이타에 대한 히스토그램의 개요

구간 데이타는 $x_i = [lx_i, hx_i]$ $(lx_i \le hx_i)$로 표현할 수 있다. 구간 데이타에 대한 히스토그램 문제의 정의는 정의 3.1과 크게 다른 것이 없으며, 두 개의 구간 데이타 간의 에러는 [7]에서 정의한 것을 이용하여 구간 데이타에 대한 히스토그램에서 사용하는 에러 척도를 정의할 수 있다. 구간 데이타에 대한 히스토그램의 에러 척도로 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합을 사용할 것이다.

정의 4.1. [구간 데이타에 대한 버킷의 에러] 구간 $[s_r, e_r]$에 속하는 구간들을 하나의 대표 구간 $[l_r,h_r]$로 나타낼 때, 버킷의 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합은 각각

$$ERR_{ML_1}(s_r,e_r) = \max(|lx_i - l_r| + |hx_i - h_r|)$$

$$ERR_{SL_1}(s_r,e_r) = \sum_{i=s_r}^{e_r}(|lx_i - l_r| + |hx_i - h_r|)$$

$$ERR_{SQL_2}(s_r,e_r) = \sum_{i=s_r}^{e_r}((lx_i - l_r)^2 + (hx_i - h_r)^2)$$

로 정의된다.

앞으로 $TERR_{ML1}[j, k]$, $TERR_{SL1}[j, k]$, $TERR_{SQL2}[j, k]$를 구간 데이타 $x_1, x_2, \cdots, x_j$에 대하여 각각 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합을 에러 척도로 사용하고 k개의 버킷을 사용하는 최적 히스토그램의 에러라 정의하고 사용할 것이다.

구간 데이타에 대한 히스토그램에서의 하나의 버킷 br은 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 된다. 따라서 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

n개의 구간 데이타 $x_1(=[lx_1, hx_1])$, $x_2(=[lx_2, hx_2])$, $\cdots$, $x_n(=[lx_n, hx_n])$를 $L=lx_1, lx_2, \cdots, lx_n$과 $H=hx_1, hx_2, \cdots, hx_n$의 n개로 이루어진 두 개의 점 데이타 시퀀스로 나타낼 수 있다. 첫 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 $lx_1, lx_2, \cdots, lx_n, hx_1, hx_2, \cdots, hx_n$에 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 좋을 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.

4.3 최대 L_1 에러를 최소화 하는 히스토그램 구축 알고리즘

최대 L_1에러를 최소화 하는 히스토그램을 구축하려면 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 2차원상의 점으로 생각하는 것이다. 다음 몇 가지의 정리와 보조 정리를 이용하여 이를 구할 수 있다.

보조 정리 4.3. 2차원 상의 점 P(x, y)를 기준으로 하여 4분면으로 나누었을 때, 그림 3(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 3(b)참고).

정리 4.5. k개의 구간 데이타 $x_1(=[lx_1, hx_1])$, $x_2(=[lx_2, hx_2])$, $\cdots$, $x_k(=[lx_k, hx_k])$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

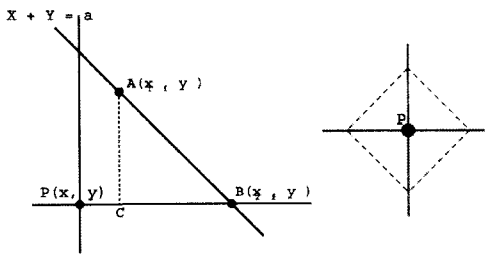


그림 3 L_1 거리의 특성

구간 데이타를 고려한 최대 L_1 에러를 대표 구간은 [A+B-C-D/4, A+B+C+D/4] 이며, 그 때의 최대 L_1 에러는 max{(A-B)/2, (C-D)/2}가 된다.

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

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 보조 정리 4.4에 의해서 점 (x,y)를 중심으로 하고 45도 회전된 정사각형임을 알 수 있다. 이 정사각형을 R이라 하자. 그리고 모든 k개의 점을 포함하는 가장 작은 45도 회전된 직사각형을 45도 회전 최소 경계 직사각형(MBR: Minimum Bounding Rectangle), 줄여서 MBR이라고 부르기로 하겠다. 그러면, R은 (x,y)를 중심으로 하는 정사각형 중에서 MBR을 포함하는 가장 작은 정사각형임을 알 수 있다. 이러한 이유는 MBR의 중에 하나의 점이 존재하기 때문에 R은 MBR을 포함해야 한다[10]. 따라서, 임의의 점 (x,y)을 k개의 점들에 대한 대표점으로 하였을 때 (x,y)를 중심으로 하며 MBR을 포함하는 가장 작은 45도 회전된 정사각형 R까지의 거리가 최대 L_1거리가 된다. 최대 L1거리를 최소화하기 위해서는 MBR을 포함하는 45도 회전된 정사각형의 크기가 최소화 되어야한다. 여기서 최대 L_1거리를 최소로 만드는 대표점 (x^*,y^*)는 45도 회전 MBR을 포함하는 가장 작은 정사각형의 중심점으로 하면 된다.

그러나 실제로 MBR을 포함하는 가장 작은 45도 회전된 정사각형은 여러 개가 존재할 수 있으며(그림 4(a) 참고), 여러 개의 정사각형 중 하나를 찾아서 그 정사각형의 중심점을 k개의 점에 대한 대표점으로 하면 된다. MBR을 포함하는 여러 45도 회전된 최소 정사각형 중

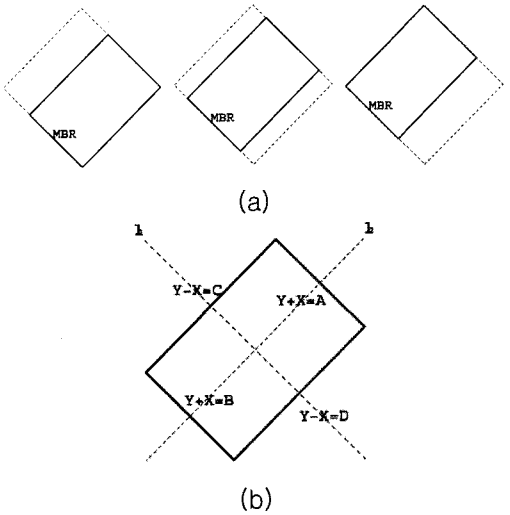


그림 4 최대 L_1 거리를 최소화하는 대표값

에서 비교적 쉽게 찾을 수 있는 것은 MBR의 중심을 중심점으로 하고 MBR을 포함하는 45도 회전된 최소 정사각형 R^*이다. R^*의 중심점은 그림 4(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 재귀식을 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 [2]에서 사용된 구간 트리를 구간 [i, j]에 해당하는 노드에서 그 범위에 속하는 데이타들의 최대값과 최소값인 max, min 대신에 A, B, C, D를 저장하도록 하여 O(logn) 시간 안에 구할 수 있다.

$TERR_{ML1}[i, k-1]$은 $x_1, x_2, \cdots, x_i$의 데이타를 (k-1)개의 버킷을 사용하였을 때, 최적 히스토그램의 에러이므로 i가 증가할수록 (k-1)개의 버킷으로 포함해야 할 데이타의 개수가 많아진다. 따라서, $TERR_{ML1}[i, k-1]$은 i에 대해 단조 증가 한다는 것을 알 수 있다. 반면, $ERR_{ML1}(i+1, j)$는 $x_{i+1}, \cdots, x_j$를 하나의 버킷으로 나타낼 때의 최대 L_1 에러를 나타내고, i가 증가할수록 포함해야 하는 데이타의 갯수는 적어진다. 따라서 REHIST[2]에서 사용하였던 이진 탐색을 이용할 수 있다. 최대 L_1 에러를 최소화 하는 히스토그램 구축 알고리즘이 그림 5에

```
Procedure OptERR_ML1()
begin
 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 5 OptERR$_{ML1}$ 알고리즘

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 [2]에서처럼 O(logn) 시간 안에 구할 수 있으므로, n개의 구간 데이타와 B개의 버킷에 대해 최대 L_1 에러를 최소화하는 히스토그램은 구간 트리와 이진 탐색을 이용하여 O(nB)의 공간과 O(nBlog^{2}n)의 시간을 사용하여 찾을 수 있다.

4.3 L_1 에러의 합을 최소화하는 히스토그램 구축 알고리즘

버킷 b_r=(s_r, e_r, l_r, h_r)에서 L_1 에러의 합은 $ERR_{SL1}(s_r, e_r) = \sum_{i=s_r}^{e_r} (|lx_i - l_r| + |hx_i - h_r|)$이며, 다음과 같이 정리할 수 있다.

$$ERR_{SL1}(s_r, e_r) = \sum_{i=s_r}^{e_r} |lx_i - l_r| + \sum_{i=s_r}^{e_r} |hx_i - h_r|$$

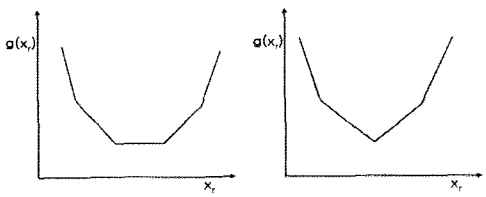
위의 정리된 식의 첫 번째 항과 두 번째 항은 모두 점 데이타에 대한 절대 오차의 합임을 알 수 있다. 따라서 버킷의 시작점과 끝점이 주어지면 구간 데이타를 두 개의 점 데이타 시퀀스 L=$l_{x_{sr}}$, $l_{x_{sr+1}}$, …, $l_{x_{er}}$와 H=$h_{x_{sr}}$, $h_{x_{sr+1}}$, …, $h_{x_{er}}$로 분해하여 각각에 대해 절대 오차의 합을 독립적으로 계산한 후 합함으로써 구간 데이타에 대한 ERR$_{SL1}$(s_r, e_r)을 구할 수 있다. L에 대한 절대 오차의 합을 $ERR_s^H(s_r, e_r)$로, H에 대한 절대 오차의 합을 $ERR_s^H(s_r, e_r)$로 표현하면, 위의 식을

$$ERR_{SL1}(s_r, e_r) = ERR_s^L(s_r, e_r) + ERR_s^H(s_r, e_r)$$

로 다시 쓸 수 있다. 즉, ERR$_{SL1}$(sr, er)은 두 절대 오차의 합을 이용하여 계산할 수 있으므로, 점 데이타에 대한 절대 오차의 합을 계산하는 방법에 초점을 맞출 것이다.

4.3.1 절대 오차의 합

k개의 점 데이타 $x_1, \cdots, x_k$가 주어졌을 때, $\sum_{i=1}^{k} |x_i - x_r|$을 계산하기 위해서는 먼저 점 데이타에 대한 절대 오차의 합을 최소화하는 대표값 x_r부터 구해야 한다. 점 데이타가 고정되어 있다고 생각하면, $\sum_{i=1}^{k} |x_i - x_r|$를 x_r에 대한 함수 $g(x_r)$이라고 생각할 수 있고 그래프로 그리면 그림 6의 형태가 된다. 그래프의 형태는 k가 짝수냐 홀수냐에 따라 다르며, k개의 데이타를 정렬된 형태를 $x'_1, x'_2, \cdots, x'_k$ 라 하면 그래프에서 k가 짝수일 경우에는 $[x'_{k/2}, x'_{k/2+1}]$에서 $g(x_r)$이 최소가 되고, k가 홀수일 경우에는 $x'_{\lceil k/2 \rceil}$일 때, $g(x_r)$이 최소가 된다는 것을 예상할 수 있으며, 정리 4.6에서 이를 증명할 것이다.



(a) k가 짝수일 때 (b) k가 홀수 일때

그림 6 g(x_r)의 그래프

정리 4.6. k개의 점 데이타 X = $x_1, x_2, \cdots, x_k$가 주어졌을 때, X의 정렬된 형태를 $X_s = x'_1, x'_2, \cdots, x'_k$로 나타내자. 그러면 k가 짝수일 경우에는 대표값이 $[x'_{k/2}, x'_{k/2+1}]$에 속하는 값일 때, k가 홀수일 경우에는 대표값이 $x'_{\lceil k/2 \rceil}$일 때, 절대 오차의 합이 최소가 된다.

증명. 증명은 수학적 귀납법을 이용하여 할 것이며, k가 짝수일 경우와 홀수일 경우의 증명이 거의 같으므로 짝수일 경우만 증명을 할 것이다.

1) k=2일 때 데이타는 x'_1, x'_2 두 개가 있고, 대표값이 $[x'_1, x'_2]$에 속할 때 절대 오차의 합이 가장 작다.

2) k=2m일 때, 즉, 2m개의 데이타의 정렬된 형태

$x'_1, x'_2, \cdots, x'_{2m}$가 있을 경우, 대표값이 $[x'_m, x'_{m+1}]$에 속하는 값일 때 절대 오차의 합이 최소가 된다고 가정하자.

k=2(m+1)일 경우, 즉, 정렬된 데이타 $X_s = x'_1, \cdots, x'_{2(m+1)}$이 있을 경우, 대표값이 $[x'_{2(m+1)/2}, x'_{2(m+1)/2+1}]$에 속하는 값일 때 절대 오차의 합이 최소가 됨을 증명해야 한다. 먼저 2(m+1)개의 데이타를 $X_{s1} = x'_2, \cdots, x'_{2(m+1)-1}$과 $X_{s2} = x'_1, x'_{2(m+1)}$로 나누자. 그러면 X_{s1}은 2m개의 데이타로 이루어져 있으므로, 가정에 의해 대표값을 $[x'_{2(m+1)/2}, x'_{2(m+1)/2+1}]$에 속하는 값으로 했을 때, 절대 오차의 합(S_1이라 하자)이 최소가 되고, X_{s2}는 2개의 데이타로 이루어져 있으므로, 대표값을 $[x'_1, x'_{2(m+1)}]$에 속하는 값으로 했을 때, 절대 오차의 합(S_2라 하자)이 최소가 된다. 또한 X_s에 대한 절대 오차의 합(S라 하자)은 $S_1 + S_2$ 이므로, S_1과 S_2 모두 최소일 때, S가 최소가 된다. S_1과 S_2 모두를 최소로 만드는 대표값의 범위는 $[x'_{2(m+1)/2}, x'_{2(m+1)/2+1}]$와 $[x'_1, x'_{2(m+1)}]$의 교집합이고, $[x'_{2(m+1)/2}, x'_{2(m+1)/2+1}] \subseteq [x'_1, x'_{2(m+1)}]$이므로, 대표값이 $[x'_{2(m+1)/2}, x'_{2(m+1)/2+1}]$에 속할 때, 최소가 된다. □

정리 4.6에 의하여 절대 오차의 합에 대한 대표 값을 구할 수 있다. 이제는 그러한 대표 값으로 하였을 때 실제 절대 오차의 합을 어떻게 구하는지 설명할 것이다.

k개의 데이타 $x_1, \cdots, x_k$이 있을 때, $\sum_{i=1}^{k}|x_i - x_r|$를 정리하면 $\sum_{i=1}^{k}|x_i - x_r| = P(x_r)x_r + Q(x_r)$이 된다. 여기서

$$P(x_r) = 2\sum_{i:x'_i \le x_r} 1 - k,\quad Q(x_r) = \sum_{i=1}^{k} x'_i - 2\sum_{i:x'_i \le x_r} x'_i$$

이다. $P(x_r) \cdot x_r + Q(x_r)$를 $g(x_r)$라 하면, 위의 정리 4.6에 의해서 k이 짝수일 경우에 $x_r = x'_{k/2}$일 때 (실제로는 x_r가 $[x'_{k/2}, x'_{k/2+1}]$사이의 임의의 수이면 상관 없지만, $x'_{k/2}$를 사용하겠음) $g(x_r)$가 최소가 되며, $P(x'_{k/2})=0$이 된다. k가 홀수일 경우에는 $x_r = x'_{\lceil k/2 \rceil}$일 때 $g(x_r)$가 최소가 되며, $P(x'_{\lceil k/2 \rceil})=1$이 된다. 즉, k가 짝수일 경우에 $P(x'_i)=0$을 만족하는 x'_i를 홀수일 경우에 $P(x'_i)=1$을 만족하는 x'_i를 찾아 $g(x'_i)$을 계산하면 되지만, 실제로는 같은 값을 가지는 데이타가 존재할 수 있기 때문에, 짝수인 경우에 $P(x'_i)=0$를, 홀수인 경우에 $P(x'_i)=1$를 정확하게 만족하는 x'_i가 존재하지 않을 수 있다. 따라서 x'_i의 값을 가지는 데이타가 여러 개일 때, x'_i는 $P(x'_i) \ge 0$을 만족하고, x'_i보다 작은 값들은 $P(x'_i) < 0$을 만족하므로, $P(x'_i) \ge 0$을 만족하는 가장 작은 첨자 i에 대한 데이타 x'_i를 찾아서 $g(x'_i)$를 계산하면 된다.

4.3.2 이진 탐색 트리의 적용

우리는 REHIST[2]에서 하나의 버킷에 대하여 최소화된 상대 오차의 합을 계산하는데 사용하였던 이진 탐색 트리를 변형하여 사용하면 O(logn) 시간 안에 $\sum_{i=1}^{k}|x_i - x_r|$을 계산할 수 있다.

이진 탐색 트리의 노드 t는 키로 데이타 값 x_t을 가지고 부가적으로 x_t와 같은 값을 가지는 데이타의 수를 저장하는 변수 n, 그리고 노드 t를 루트로 하는 부분 트리 T라고 할 때 다음과 같은 변수 m과 sum을 유지한다.

$m = \sum_{N \in T} N.n$, $sum = \sum_{N \in T} N.n \times N.x_N$이진 탐색 트리에 데이타 x가 삽입될 때, 변수 n, m, sum의 값들은 적절하게 갱신되어야 한다.

이진 탐색 트리를 이용하여 $\sum_{i=1}^{k}|x_i - x_r|$를 계산하기 위해서는 트리의 루트부터 시작해서 $P(x) \ge 0$을 만족하는 가장 작은 x를 키로 가지는 노드를 찾을 때까지 순회 해야한다. 그러한 x를 찾기 위해서 현재 x_t를 키로 가지는 노드 t에 있다고 하고, t를 루트로 가지는 트리를 T라고 했을 때, [2]에서와 같이 두 개의 변수 ex.m과 ex.sum을 유지한다. ex.m은 트리 T에 속하지 않으면서 x_t보다 작은 데이타의 갯수를 나타내는 변수이고, ex.sum은 x_t보다 작은 데이타들의 합을 나타내는 변수로 식으로 나타내면 다음과 같다.

$$ex.m = \sum_{\substack{N \notin T,\\ N.x_N < x_t}} N.n,\quad ex.sum = \sum_{\substack{N \notin T,\\ N.x_N < x_t}} N.n \times N.x_N$$

이렇게 ex.m과 ex.sum을 유지할 수 있다면 [2]에서 제시된 방법을 사용하여 $\sum_{i=1}^{k}|x_i - x_r|$을 O(logn) 시간 안에 계산할 수 있다.

4.3.3 히스토그램 구축 알고리즘

지금까지의 내용을 적용하여 $OptERR_{SL1}$ 알고리즘을 만들 수 있으며, $OptERR_{SL1}$ 알고리즘은 그림 7에 제시되어 있다. 데이타들의 정렬된 형태를 이진 탐색 트리를 이용하여 점증적으로 유지하기 위해서 이진 탐색 트리를 이용하고 있으며, 이진 탐색 트리는 구간 [i+1, j]에 속하는 데이타들을 포함하고 있기 때문에, i를 1부터 (j-1)까지 증가시키는 것은 x1을 제외한 모든 데이타를 포함하는 이진 탐색 트리를 만든 후에 데이타를 하나씩 삭제하는 것을 의미한다. 이러한 방법보다 i를 (j-1)에서 1까지 감소시키면서 이진 탐색 트리에 데이타를 하나씩 삽입하는 것이 효율적이기 때문에, 11번째 줄의 i는 j부터 2까지 감소하고 있으며, 12, 13번째 줄에서 xi를 이진 탐색 트리에 삽입하여 14번째 줄에서 이를 이용해 $ERR_{SL1}(i, j)$를 계산한다.

4.4 L_2에러 제곱의 합을 최소화하는 히스토그램 구축 알고리즘

```
Procedure OptERR_SL1()
begin
 1. for j:=1 to n do {
 2.   BST_L.Insert(lxj)
 3.   BST_H.Insert(hxj)
 4.   TERR_SL1[j, 1] := ERR_SL1(1, j)
 5. }
 6. delete BST_L, BST_H
 7. for j:=1 to n do {
 8.   for k:=2 to B do {
 9.     TERR_SL1[j, k] := ∞
10.   }
11.   for i:=j to 2 do {
12.     BST_L.Insert(lxj)
13.     BST_H.Insert(hxj)
14.     tmp_err := ERR_SL1(i, j)
15.       for k:=2 to B do {
16.         temp := TERR_SL1[i-1, k-1] + tmp_err
17.         if temp < TERR_SL1[j, k]
18.           TERR_SL1[j, k] = temp
19.       }
20.   }
21.   delete BST_L, BST_H
22. }
end
```

그림 7 $OptERR_{SL1}$ 알고리즘

버킷 $b_r=(s_r, e_r, l_r, h_r)$에 대한 L_2 에러 제곱의 합은 $ERR_{SQL_2}(s_r,e_r)=\sum_{i=s_r}^{e_r}((lx_i-l_r)^2+(hx_i-h_r)^2)$이며, 다음과 같이 정리할 수 있다.

$$ERR_{SQL_2}(s_r,e_r)=\sum_{i=s_r}^{e_r}(lx_i-l_r)^2+\sum_{i=s_r}^{e_r}(hx_i-h_r)^2$$

위의 정리된 식에서, 첫 번째 항과 두 번째 항 모두 점 데이타에 대한 절대 오차 제곱의 합임을 알 수 있다. 따라서 L_1 에러의 합과 마찬가지로 버킷의 시작점과 끝점이 주어지면 구간 데이타를 두 개의 점 데이타 시퀀스 $L=lx_{sr}, lx_{sr+1}, \cdots, lx_{er}$와 $H=hx_{sr}, hx_{sr+1}, \cdots, hx_{er}$로 분해하여 각각에 대해 절대 오차 제곱의 합을 독립적으로 계산한 후 합함으로써 구간 데이타에 대한 $ERR_{SQL2}(s_r, e_r)$을 구할 수 있다. L에 대한 절대 오차 제곱의 합을 $ERR_{SQ}^{L}(s_r, e_r)$로, H에 대한 절대 오차 제곱의 합을 $ERR_{SQ}^{H}(s_r, e_r)$로 표현하면, $ERR_{SQL2}(s_r, e_r) = ERR_{SQ}^{L}(s_r, e_r) + ERR_{SQ}^{H}(s_r, e_r)$로 다시 쓸 수 있다. $ERR_{SQ}^{L}(s_r, e_r)$와 $ERR_{SQ}^{H}(s_r, e_r)$는 V-Optimal 히스토그램 구축 알고리즘[7]에서처럼

$$ERR_{SQ}(s_r,e_r)=\sum_{k=s_r}^{e_r}x_k^2-\frac{1}{e_r-s_r-1}\left(\sum_{k=s_r}^{e_r}x_k\right)^2$$

을 이용하여 계산할 수 있으며, 두 개의 배열 $Sum[i]=\sum_{l=1}^{i}x_l$과 $SqSum[i]=\sum_{l=1}^{i}x_l^2$를 전처리 단계에 계산함으로써 O(1)시간 안에 계산할 수 있다.

L_2 에러 제곱의 합을 최소화 하는 히스토그램 구축 알고리즘의 의사 코드가 그림 8에 제시되어 있다. 1번째 줄의 Preprocessing()은 ERR_{SQ}^{L}과 ERR_{SQ}^{H}를 O(1) 시간 안에 계산하기 위해서 분해된 점 데이타의 시퀀스 L과 H에 대하여 Sum과 SqSum 배열을 계산하는 프로시저이다. 3, 7번째 줄의 ERR_{SQL2}는 두 개의 점 데이타에 대한 절대 오차 제곱의 합을 이용하여 계산된다.

```
Procedure OptERR_SQL2()
begin
 1. Preprocessing()
 2. for j:=1 to n do {
 3.   TERR_SQL2[i, 1] := ERR_SQL2(1, i)
 4.   for k:=2 to B do {
 5.     TERR_SQL2[i, j] := ∞
 6.     for i:=1 to (j-1) do {
 7.       temp := TERR_SQL2[i, k-1] + ERR_SQL2(i+1, j)
 8.       if temp < TERR_SQL2[j, k]
 9.         TERR_SQL2[j, k] := temp
10.     }
11.   }
12. }
end
```

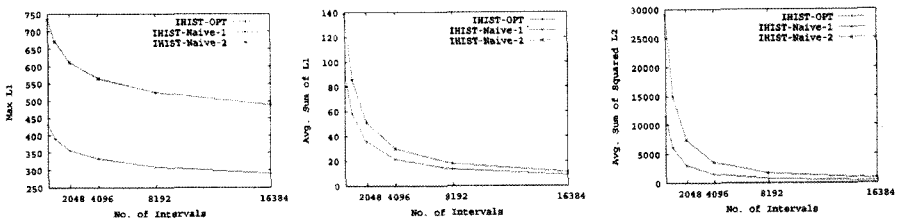
그림 8 $OptERR_{SQL2}$ 알고리즘

5. 실험 결과

이 장에서는 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.

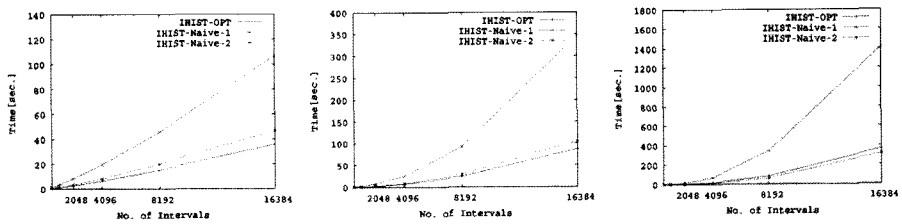
5.1 실험 데이타

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. 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.



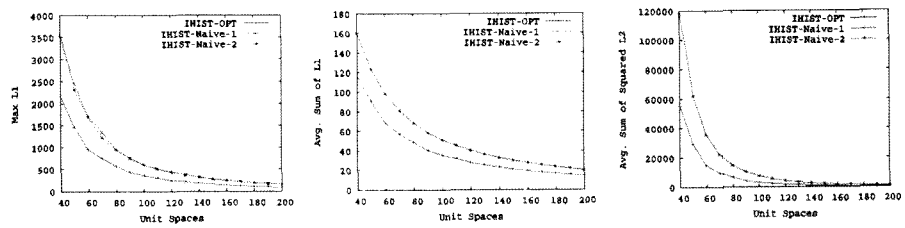
(a) 최대 L_1 에러 (b) 평균 L_1 에러의 합 (c) 평균 L_2 에러 제곱의 합

그림 9 히스토그램 에러 척도에 대한 데이타 집합의 크기의 영향



(a) 최대 L_1 에러 (b) L_1 에러의 합 (c) L_2 에러 제곱의 합

그림 10 히스토그램 구축 시간에 대한 데이타 집합의 크기의 영향



(a) 최대 L_1 에러 (b) 평균 L_1 에러의 합 (c) 평균 L_2 에러 제곱의 합

그림 11 히스토그램 에러 척도에 대한 히스토그램 저장 공간의 영향

5.2 비교 알고리즘

- **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, H에 $\lfloor S/2 \rfloor - \lfloor S/4 \rfloor$ 개를 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 하였으나, 그래프 식별을 어려움 때문에 데이타의 집합의 크기가 512일 때부터 그래프를 그렸다. 결과는 그림 9, 10에 있다. 그림 9의 L_1 에러의 합과 L_2 에러 제곱의 합은 히스토그램의 에러를 데이타의 개수로 나누어 평균적인 에러를 그래프로 나타내었다.

실험 결과 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합 각각을 평균적으로 40%, 30%, 55%정도 개선시키는 것으로 확인되었다. 그래프에서 볼 수 있듯이, 모든 형태의 데이타 집합에서 데이타 집합의 크기기 커질수록 세 가지 에러 척도의 에러가 작아졌다. 이유는 데이타 집합의 크기에 관계없이 데이타들의 합이 1,000,000

으로 유지하였으므로, 데이타 집합의 크기가 커질수록 각각의 데이타 값은 작아지기 때문에, 에러도 작아진다. 또한 IHIST-Naive-1와 IHIST-Naive-2의 에러는 대부분 같기 때문에, 그래프가 겹쳐져 보인다. 앞으로의 실험에서도 대부분 IHIST-Naive-1와 IHIST-Naive-2의 실험 결과가 같게 나오는 것을 확인할 수 있었다.

알고리즘의 수행 시간에 대해서는 L_2 에러 제곱의 합을 제외하고 항상 IHIST-Naive-1가 가장 오래 걸리고, IHIST-OPT가 가장 적게 걸리는 것을 확인할 수 있다. L_2 에러 제곱의 합에 대해서는 IHIST-Naive-1이 IHIST-OPT보다 상수 배 빠른 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 예를 들어 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 11에서 그래프 식별 문제 때문에 히스토그램의 저장 공간을 40부터 나타내었다.

저장 공간이 커질수록 많은 수의 버킷을 사용할 수 있으므로 에러는 줄어들게 될 것이라는 예상을 할 수 있으며 그래프에서 확인할 수 있다. 실험 결과, 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합 각각을 최대 45%, 30%, 60% 정도 개선시키는 것을 확인할 수 있었다. 또한 지면 부족으로 나타내지는 않았지만 각 알고리즘의 시간 복잡도는 저장 공간에 비례한다는 것을 확인하였다.

6. 결 론

본 논문에서는 기존의 점 데이타에 대한 히스토그램 구축 알고리즘을 구간 데이타에 대하여 확장한 IHIST-OPT를 제안하였다. 본 논문에서 사용하고 있는 히스토그램의 에러 척도로는 최대 L_1 에러, L_1 에러의 합, L_2 에러 제곱의 합이며 각각의 시간 복잡도와 공간 복잡도는 표 1에 제시되어 있다. 또한, 합성 데이타를 사용한 실험을 통하여 기존의 점 데이타에 대한 히스토그램을 초보적으로 확장하는 방법보다 본 논문에서 제시된 알고리즘의 성능이 좋다는 것을 보였다.

표 1 세 알고리즘의 시간, 공간 복잡도

에러 척도	시간 복잡도	공간 복잡도
최대 L_1 에러	$O(nB\log^2 n)$	$O(nB)$
L_1 에러의 합	$O(n^2(B+\log n))$	$O(nB)$
L_2 에러 제곱의 합	$O(n^2B)$	$O(nB)$

참 고 문 헌

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

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

[3] 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.

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

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

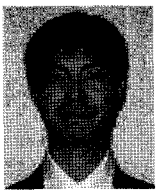
[6] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," In Proc. of SIGMOD, San Jose, Calfornia, USA, May 1995.

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

[8] 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.

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

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



이 호 석

2004년 성균관대학교 수학과, 정보통신공학부 졸업(학사). 2006년 서울대학교 전기컴퓨터공학부 졸업(석사). 관심분야는 데이타마이닝, 데이타베이스

심 규 석

정보과학회논문지 : 데이타베이스 제 34 권 제 2 호 참조

이 병 기

1989년 서울대학교 계산통계학과 학사 1993년 서울대학교 계산통계학과 석사 2000년 University of Maryland, Ph.D. in Computer Science. 2000년~2001년 New Jersey Institute of Technology, Asst. Prof. 2001년~현재 포항공과대학교 컴퓨터공학과 조교수. 관심분야는 데이타베이스, 데이타마이닝, 멀티미디어, 바이오인포매틱스, LBS