정지점의 활용을 극대화한 효율적인 스카이라인 계산법

고철우○ 이윤준
KAIST 전산학과
cwkoh0117@hotmail.com, yoonjoon.lee@kaist.ac.kr

Maximized Utilization of Stop Point for Efficiently Computing the Skyline

Chol Woo Koh○ Yoon Joon Lee
Department of Computer Science, KAIST

요 약

스카이라인 질의는 사용자의 선호도를 고려하여 무수히 많은 데이터로부터 사용자에게 유용한 정보만을 반환한다. 스카이라인을 효율적으로 계산하기 위한 많은 방법들이 연구 되었지만, 그 중에서도 스카이라인 질의 기능이 제공되지 않는 일반적인 데이터베이스로부터 스카이라인을 계산할 수 있는 최적의 알고리즘인 Sort and Limit Skyline Algorithm(SaLSa)이 있다. SaLSa는 정렬된 데이터와 정지점의 활용으로 전체 데이터 중 일부만 읽으며 스카이라인을 구할 수 있다. 정지점을 중간에 계산하는 SaLSa는 정지점의 기능을 충분히 활용하지 못한다. 본 논문에서는 정지점을 미리 계산하여 정지점의 제거기능을 최대화시킨 효율적인 스카이라인 알고리즘 Skyline with Transformation(SWT)을 제안하고, 실험을 통해 SWT가 SaLSa에 비해 데이터 제거 효과 및 스카이라인 질의 처리 속도가 우수함을 검증한다.

1. 서 론

정보 기술의 발달과 함께 데이터베이스의 크기가 급속도로 증가하고 있다. 따라서 데이터를 저장하는 것 이상으로 사용자가 필요로 하는 정보를 효과적으로 제공할 수 있는 기술이 중요하다. 예를 들어, 하와이에 있는 사용자가 가격이 저렴하고 해변으로부터 가까운 호텔을 찾을 경우, 데이터베이스 시스템은 다중 속성을 고려한 사용자에게 최적의 호텔을 정해줄 수는 없지만, 최소한으로 유용한 정보만을 제공함으로써 사용자의 의사 결정을 도울 수 있다. 여기서 유용한 정보란, 다른 호텔에 의해 지배(dominate)당하지 않는 호텔의 집합이고, 이들을 스카이라인(skyline)이라고 한다. 스카이라인에서 지배는 다음과 같이 정의 된다. 두 개의 데이터 d와 d'을 비교하여, 모든 속성에서 d가 d'보다 우수하거나 동등하고, 적어도 하나의 속성에서 d가 d'보다 우수할 경우, d는 d'을 지배한다고 표현한다.

그림 1은 가상의 호텔들을 2차원 상의 그래프로 표현한 것으로, x축은 해변으로부터의 거리, y축은 가격을 나타낸다. 사용자가 가격이 저렴하고 해변으로부터 가까운 호텔을 찾는 상황을 가정해보자. 호텔 a를 보면, 호텔 f, g와 비교하여 해변으로부터의 거리도 가깝고 가격 또한 저렴하다. 따라서 a는 f와 g를 지배한다. 같은 원리로, b는 e와 f를 지배한다. 호텔 c와 d를 비교하면, 해변으로부터의 거리는 같지만 가격이 더 저렴한 c가 d를 지배한다. 이렇게 사용자의 선호도를 기준으로 다른 호텔에 의해 지배당하지 않는 호텔의 집합{a, b, c, h}가 사용자의 의사결정을 돕기 위해 스카이라인 결과로 제공 된다.

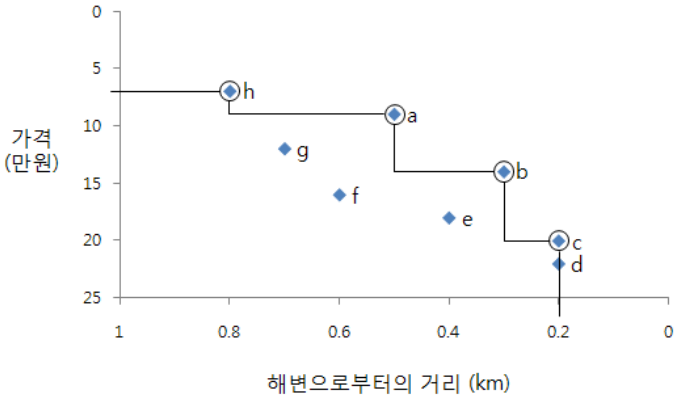

그림 1. 호텔 스카이라인

스카이라인은 최근 들어 데이터베이스 연구 분야에서 많은 관심을 받고 있으며, 이미 다양한 계산 방법이 연구 되었다. 대표적인 알고리즘으로 Block Nested Loop(BNL)[1], Divide and Conquer(D&C)[1], Nearest Neighbor(NN)[2], Branch and Bound Skyline Algorithm(BBS)[3], Z-Sky[4] 등이 있다. 하지만 이들 대부분은 데이터베이스 서버 내에서 스카이라인을 효율적으로 계산할 수 있도록 제안된 알고리즘으로, 서버 내에 별도의 자료구조가 필요하다. 따라서 스카이라인 계산 기능이 없는 일반적인 데이터베이스 서버로부터 사용자가 스카이라인 결과를 얻고자 할 때에는 사용할 수 없다. 본 논문에서는 이러한 한계를 극복하고자 한다.

데이터베이스 서버에 스카이라인 계산 기능이 없다면, 클라이언트 쪽에서 스카이라인을 계산해야 한다. 클라이언트는 먼저 데이터베이스 서버로부터 모든 데이터를 전송 받고, 이로부터 스카이라인을 계산한다. 하지만 이 경우 스카이라인을 계산하는 방법이 아무리 효율적이어도, 모든 데이터를 전송 받기위해 소비되는 전송 비용으로 인해 전체적인 과정이 매우 비효율적이다. 이러한 비효율성을 보완한 알고리즘으로 Sort and Limit Skyline Algorithm(SaLSa)[5]이 있다. SaLSa는 정렬된 데이터와 정지점(stop point)의 활용으로 읽어 들일 데이터를 제한한다. 하지만 알고리즘 중간에 정지점을 계산하는 SaLSa는 정지점의 능력을 충분히 활용하지 못 한다.

본 논문에서는 SaLSa의 단점을 보완한 알고리즘 Skyline with Transformation(SWT)을 제안한다. SWT는 SaLSa와 달리 정지점을 미리 계산하여 정지점의 활용을 극대화시킴으로써 SaLSa에 비해 읽어 들이는 데이터를 대폭 감소시킨다. 실제 실험 결과, 17531개의 데이터를 갖는 6-차원 NBA 데이터 집합으로부터 59개의 스카이라인 결과를 얻기 위해 SaLSa는 10209개의 데이터를 읽어야 하지만 SWT는 897개의 데이터만 읽으면 된다.

본 논문의 구성은 다음과 같다. 2장에서는 현존하는 스카이라인 알고리즘에 대해 살펴본다. 3장에서는 본 논문에서 제안하는 SWT를 소개한다. 4장에서는 실험을 통해 SWT의 데이터 제거 효과 및 스카이라인 질의 처리 속도가 기존 연구에 비해 우수함을 검증하고, 5장에서 결론을 도출한다.

2. 배경지식 및 관련연구

d-차원 공간 $S = s_1, s_2, ..., s_d$ 와 S에 존재하는 데이터 집합 $P = p_1, p_2, ..., p_n$ 가 주어질 때, 스카이라인 L과 지배를 의미하는 '$\gg$'는 다음과 같이 정의된다.

$$L = \{p \in P \mid \nexists p' \in P - \{p\}, p' \gg p\} \quad (1)$$

$$p \gg p' \Leftrightarrow \forall s_i \in S, p.s_i \geq p'.s_i \wedge (\exists s_j \in S, p.s_j > p'.s_j) \quad (2)$$

스카이라인을 계산하는 가장 기본적인 알고리즘은 Block Nested Loop(BNL)[1]이다. BNL은 메인 메모리에 스카이라인 후보들을 저장할 윈도우 w를 할당하고 모든 데이터를 순차적으로 읽어 들인다. 하나의 데이터 p를 읽을 때, p는 w에 존재하는 데이터와 비교된다. 만약 p가 w에 존재하는 데이터에 의해 지배당하지 않을 경우, p는 w에 추가되고 p가 지배하는 w내의 모든 데이터는 제거 된다. 만약 메모리 부족으로 인해 w에 더 이상 데이터를 추가할 수 없는 상황이 발생하면, p는 임시파일에 저장된다. 이 과정을 모든 데이터에 대해 반복하면, w에는 스카이라인 데이터만 남게 되고, 임시파일에 저장된 데이터를 다시 읽으며 앞의 과정을 되풀이 하면 최종적인 스카이라인을 구할 수 있다.

Sort Filter Skyline(SFS)[6]는 단조함수 F를 이용하여 데이터를 내림차순으로 정렬 시킨 후에 읽어 들인다. 단조함수는, $F(p) \geq F(p')$ 이면, $p' \not\gg p$ 를 만족하는 함수로 정의된다. 스카이라인을 계산하는 방법은 BNL과 동일하다. 하지만 정렬된 데이터를 사용함으로써 뒤에 읽어 들일 데이터가 앞에서 이미 읽어 들인 데이터를 지배하는 상황이 발생하지 않는다. 이로 인해 w에는 스카이라인 데이터만 포함되므로, 보다 용이한 w의 관리가 가능하고, 용량 초과로 인한 반복과정을 최소화 시킬 수 있다.

Sort and Limit Skyline Algorithm(SaLSa) [5]은 SFS를 발전시킨 알고리즘이다. 정렬된 데이터를 읽어 들임으로써 SFS의 모든 장점을 가짐은 물론, 정지점(stop point) P_{stop}을 활용하여 읽어 들일 데이터를 제한한다. P_{stop}은 현재까지 읽어 들인 데이터 집합 R 중에, 가장 작은 속성 값이 가장 큰 데이터로 정의 된다.

$$P_{stop} = \{p \in R \mid \nexists p' \in R - \{p\}, Min(p') > Min(p)\} \quad (3)$$

$$Min(p) = Min(p.s_1, p.s_2, ..., p.s_d) \quad (4)$$

[5]에 따르면, SaLSa는 가장 큰 속성 값을 반환하는 함수로 데이터를 정렬할 때 가장 좋은 성능을 발휘한다. F는 다음과 같다.

$$F(p) = Max(p) = Max(p.s_1, p.s_2, ..., p.s_d) \quad (5)$$

하나의 데이터 p를 읽을 때, p는 먼저 P_{stop}과 비교한다. 만약 $F(p) \geq Min(P_{stop})$ 이고, p가 w에 존재하는 데이터에 의해 지배당하지 않을 경우, p는 w에 추가 된다. 이때, $Min(p) > Min(P_{stop})$ 이면, p를 새로운 P_{stop}으로 갱신한다. 이에 반해, 만약 $F(p) < Min(P_{stop})$ 일 경우, 다음 식에 의해 p는 P_{stop}으로부터 지배당한다.

$$\forall s_i \in S, p.s_i \leq Max(p) = Min(P_{stop}) \leq P_{stop}.s_i \quad (6)$$

또한, p 뒤에 읽어 들일 모든 데이터 p'은, $F(p') \leq F(p) \leq Min(P_{stop})$ 에 의해, P_{stop}으로부터 지배당함을 알 수 있다. 따라서 SaLSa는 남은 데이터를 모두 제거시키며 알고리즘을 종료한다.

그림 2는 SaLSa의 효과를 그래프로 표현한 것이다. 그래프 상에서 어둡게 표현된 영역은 스카이라인 판별을 위한 비교 과정 없이 바로 제거가 가능한 데이터를 포함한다. 따라서 이를 제거영역(pruning region)[7] 이라고 부른다. 제거영역은, 한 변의 길이를 $Min(P_{stop})$ 으로 하는 d-차원 큐브(cube)로 형성 되고, $F(p) < Min(P_{stop})$ 을 만족하는 모든 p를 포함 한다.

SaLSa는 제거영역에 존재하는 모든 데이터를 제거함으로써 SFS보다 우월한 성능을 발휘한다. 하지만, 일반적으로 각 차원의 값이 상이함에도 가장 작은 값을 갖는 차원에 따라 제거영역이 결정되는 SaLSa는 P_{stop}의 효과를 충분히 활용하지 못한다. 이 경우, P_{stop}이 지배하는 모든 데이터를 포함하는 지배영역(dominance region)에 비해 제거영역이 상대적으로 작기 때문이다. 위의 그래프에서 P_{stop}이 지배하는 데이터는 13개이지만 P_{stop}에 의해 제거되는 데이터는 4개에 불과하다.

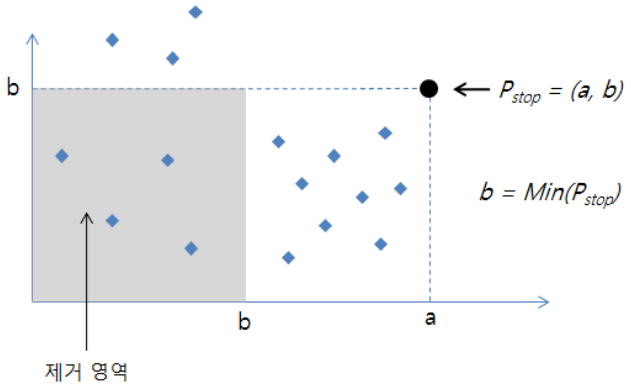

그림 2. SaLSa의 효과

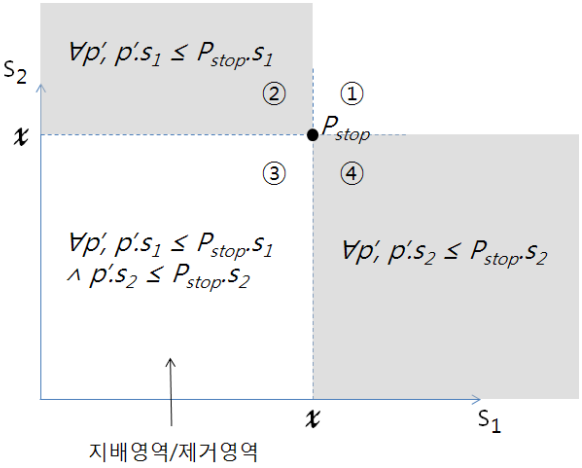

그림 3. 데이터 변환의 효과

3. Skyline with Transformation(SWT)

SWT는 단조함수 F로 정렬된 데이터와 정지점 P_{stop}을 활용한다는 측면에서 SaLSa와 동일하다.

P_{stop}을 활용한 데이터의 제한이 효과를 보기 위해서는 F가 $\forall s_i \in S, p.s_i \le F(p)$을 만족해야 한다. F가 이를 만족할 때, $F(p) < Min(P_{stop})$인 모든 p에 대해, $P_{stop} \gg p$가 성립하기 때문이다. 또한, $F(p)$의 값은 작을수록, $Min(P_{stop})$은 클수록 $F(p) < Min(P_{stop})$를 만족하는 p가 증가되어 더 많은 데이터를 제거할 수 있게 된다. 위의 조건을 만족하는 F중, 가장 작은 값을 반환하는 함수는 $Max(p)$이다. 또한, $Min(p)$값이 가장 큰 p가 P_{stop}이 될 때 $Min(P_{stop})$은 가장 큰 값을 갖는다. 따라서 최적의 F는 $Max(p)$, 최적의 P_{stop}은 $Min(p)$가 가장 큰 데이터로 정의 된다.

SWT는 정지점의 활용을 극대화함으로써 SaLSa의 단점을 보완한다. 정지점의 극대화 된 활용은 정지점이 지배하는 모든 데이터를 제거할 때 이루어진다.

SWT는 먼저 하나의 데이터 p를 P_{stop}으로 지정하고, 데이터 변환을 통해 P_{stop}의 모든 속성 값이, 동일한 값, $x\,(x > 0)$가 되도록 만든다. 이 경우 다음과 같은 두 가지 효과를 얻을 수 있다.

정리 1.

1) p가 스카이라인일 경우, p는 최적의 P_{stop}이 된다.
2) P_{stop}의 지배영역과 제거영역이 일치되어 P_{stop}의 활용이 극대화 된다.

증명.

그림3은 데이터 변환으로 얻을 수 있는 효과를 그래프로 나타낸 것이다. P_{stop}으로 지정된 p가 스카이라인일 경우, 스카이라인의 정의에 의해, 영역 ①에는 데이터가 존재할 수 없다. 또한, 영역②와 ③에 존재하는 모든 데이터 p'은 $p'.s_1 \le p.s_1$을 만족하고, 영역③과 ④에 존재하는 모든 데이터 p''은 $p''.s_2 \le p.s_2$을 만족한다. 따라서 모든 p', p''에 대해 $Min(p') \le Min(p), Min(p'') \le Min(p)$가 성립하고, 이로써 p는 최적의 P_{stop}이 된다.

모든 속성 값이 x인 P_{stop}의 지배영역은 x를 한 변으로 하는 d-차원 큐브로 이루어진다. P_{stop}의 제거영역 또한 $Min(P_{stop}) = x$ 를 한 변으로 하는 d-차원 큐브로 이루어진다. 따라서 지배영역과 제거영역은 일치하게 되고, P_{stop}이 지배하는 모든 데이터를 제거할 수 있게 된다. ■

변환 과정이 끝나면 $F = Max(p)$를 이용하여 데이터를 내림차순으로 정렬시킨 후에 읽어 들인다. 하나의 데이터 p를 읽을 때, p는 w에 존재하는 데이터와 비교된다. 만약 p가 w에 존재하는 데이터에 의해 지배당하지 않을 경우, p는 w에 추가된다. 이때, p가 P_{stop}이면 남은 모든 데이터가 P_{stop}에 의해 지배당하므로, 알고리즘은 종료된다.

알고리즘 1. Skyline with Transformation

```
1: w ← empty list, U ← data list
2: preselect P_stop from the data list
3: transform U
4: sort U according to F
5: while U is not empty
6:     p ← get next point from U, U ← U − {p}
7:     if w ⊁ p, then w ← w + {p}
8:     if p = P_stop then break
9: return w
```

알고리즘 1은 SWT의 의사코드이다. SWT는 알고리즘 1의 2,3번째 줄에 해당하는 P_{stop}지정 방법과 데이터 변환 방법에 따라 성능에 차이를 보인다.

지정된 P_{stop}이 스카이라인일 경우에 최적의 P_{stop}이 된다는 것을 앞의 증명을 통해 알아보았다. 따라서 전체 데이터 중 유일하게 스카이라인임을 보장받는, F 값이 가장 큰 데이터가 P_{stop}으로 지정된다. 또한, P_{stop}의 역할이 최대한 많은 데이터를 제거하는 것임을 감안할 때, 모든 속성에서 우수한 값을 갖는 데이터를 P_{stop}으로 지

정 하는 것이 바람직하다. 이를 위해 SWT는 모든 속성에 동등한 가중치가 부여되도록 정규화 과정을 수행하여 P_{stop}을 계산한다. 정규화는 다음과 같이 이루어진다. ($Max(s_i)$와 $Min(s_i)$는 속성 s_i의 최대값과 최소값을 나타낸다.)

$$\forall p \in P, \forall s_i \in S, \quad p.s_i = (p.s_i - Min(s_i)) / (Max(s_i) - Min(s_i)) \tag{7}$$

(8), (9), (10)은 SWT에서 사용 가능한 P_{stop}으로, 각각 $F(p) = Min(p)$, $F(p) = \sum_{i=1}^{d} p.s_i$, $F(p) = \prod_{i=1}^{d} p.s_i$의 값이 가장 큰 데이터로 정의 된다.

$$P_{MaxMin} = \{p \in P \mid \nexists p' \in P - \{p\}, Min(p) < Min(p')\} \tag{8}$$

$$P_{MaxSum} = \left\{p \in P \mid \nexists p' \in P - \{p\}, \sum_{i=1}^{d}(p.s_i) < \sum_{i=1}^{d}(p'.s_i)\right\} \tag{9}$$

$$P_{MaxVol} = \left\{p \in P \mid \nexists p' \in P - \{p\}, \prod_{i=1}^{d}(p.s_i) < \prod_{i=1}^{d}(p'.s_i)\right\} \tag{10}$$

다음으로 데이터 변환을 위해 사용 가능한 두 가지 방법을 알아본다.

첫 번째 변환 방법은 스케일링(scaling) 이다. 스케일링은 P_{stop}의 모든 속성 값이 x가 되도록 각 속성을 일정 비율로 늘리거나 줄이는 방법으로, 다음과 같은 과정으로 수행된다.

$$\forall p \in P, \forall s_i \in S, p.s_i = \frac{p.s_i}{P_{stop}.s_i} \times x \tag{11}$$

두 번째 변환 방법은 원점 평행이동(translation) 이다. 원점 평행이동은 P_{stop}의 모든 속성 값이 x가 되도록 원점을 평행이동 시키는 방법으로, 다음과 같은 과정으로 수행된다.

$$\forall p \in P, \forall s_i \in S, p.s_i = p.s_i + (x - P_{stop}.s_i) \tag{12}$$

SWT의 중요한 특성 중 하나는 데이터 변환이 일어난 후에도 스카이라인 결과가 변하지 않는다는 것이다. 데이터 변환 과정이 데이터 사이의 지배 관계에 영향을 주지 않기 때문이다.

정리 2.

데이터 변환은 데이터 사이의 지배 관계에 영향을 주지 않는다.

증명.

$$p \gg p' \Leftrightarrow (\forall s_i \in S, p.s_i \geq p'.s_i) \wedge (\exists s_j \in S, p.s_j > p'.s_j) \tag{13}$$

$$p \gg p' \Leftrightarrow (\forall s_i \in S, (\alpha.s_i \times p.s_i) \geq (\alpha.s_i \times p'.s_i)) \wedge (\exists s_j \in S, (\alpha.s_j \times p.s_j) > (\alpha.s_j \times p'.s_j)) \quad \forall s_k \in S, \alpha.s_k = (x / P_{stop}.s_k) \tag{14}$$

$$p \gg p' \Leftrightarrow (\forall s_i \in S, (p.s_i + \alpha.s_i) \geq (p'.s_i + \alpha.s_i)) \wedge (\exists s_j \in S, (p.s_j + \alpha.s_j) > (p'.s_j + \alpha.s_j)) \quad \forall s_k \in S, \alpha.s_k = (x - P_{stop}.s_k) \tag{15}$$

(13)은 데이터 변환이 이루어지기 전의 지배 관계를 정의한다. (14)와 (15)는 각각 스케일링과 원점 평행이동 후에도 지배관계가 유지됨을 보인다. (14)에서 x는 정의에 의해, $P_{stop}.s$는 정규화 과정에 의해 $x > 0$, $P_{stop}.s > 0$ 을 만족하므로, $\forall s \in S, \alpha.s > 0$가 성립한다. 따라서 (13)의 지배 관계가 (14)와 (15)에 동등하게 적용되고, 이는 변환 과정이 지배 관계에 영향을 주지 않음을 증명한다. ■

	s_1	s_2	s_3
p_1	1.0	0.3	0.5
p_2	0.4	1.0	0.2
p_3	0.6	0.4	0.0
p_4	0.0	0.6	0.2
p_5	0.8	0.5	1.0
p_6	0.2	0.4	0.4
p_7	0.3	0.3	0.4
p_8	0.3	0.0	0.2

① →

	s_1	s_2	s_3
p_1	1.2	0.8	0.5
p_2	0.6	1.5	0.2
p_3	0.8	0.9	0.0
p_4	0.2	1.1	0.2
p_5	1.0	1.0	1.0
p_6	0.4	0.9	0.4
p_7	0.5	0.8	0.4
p_8	0.5	0.5	0.2

② →

	s_1	s_2	s_3
p_2	0.6	1.5	0.2
p_1	1.2	0.8	0.5
p_4	0.2	1.1	0.2
p_5	1.0	1.0	1.0
p_3	0.8	0.9	0.0
p_6	0.4	0.9	0.4
p_7	0.5	0.8	0.4
p_8	0.5	0.5	0.2

③

그림 4. SWT 알고리즘의 실행 과정 예제

그림 4는 SWT의 실행 과정을 보여준다. ①이 데이터베이스에 존재하는 데이터일 때, P_{MaxSum}을 P_{stop}으로 지정하면, p_5가 P_{stop}이 된다. ②는 원점 평행이동 방식으로 P_{stop}의 모든 속성 값이 1이 되도록 변환한 데이터이고, ③은 ②를 $F = Max(p)$로 정렬한 데이터이다. 클라이언트는 ③의 데이터를 하나씩 읽어 들인다. 첫 번째 데이터 p_2는 바로 w에 추가되고, 두 번째 데이터 p_1 또한 다른 데이터에 의해 지배당하지 않으므로 w에 추가된다. 세 번째 데이터 p_4는 w에 존재하는 p_2에 의해 지배당하므로 w에 추가되지 않는다. 네 번째 데이터 p_5는 P_{stop}이다. 따라서 p_5는 w에 추가되고 p_5뒤의 모든 데이터는 제거된다.

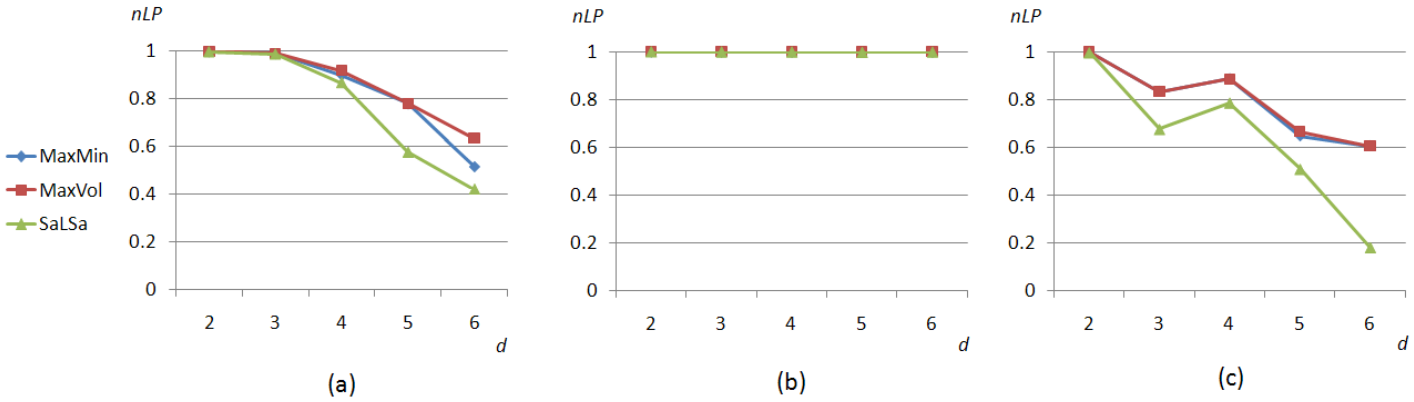

그림 5. (a) independent, (b) correlated, (c)anti-correlated 분포에 대한 Normalized Liming Power

4. 실험 결과

본 실험의 목적은 SWT의 데이터 제거 효과 및 질의 처리 속도가 SaLSa에 비해 우수함을 검증하는 것이다. 실험에는 실제 데이터와 데이터 생성기로 만든 가상의 데이터를 사용하였다.

실제 데이터는 [8]로부터 1950~2008시즌 사이에 활약한 모든 NBA 선수들의 기록을 시즌 단위로 추출하였다. 총 17531개의 데이터에 대해, 경기 수(gp), 득점(pts), 리바운드(reb), 도움(ast), 필드 골(fg), 자유투(ft)를 속성으로 사용하였다.

가상의 데이터는 [1]에서 사용한 데이터 생성기를 이용하여 각각 independent, correlated, anti-correlated 분포를 갖는 세 종류의 데이터 집합을 생성하였다. 각 데이터 집합은 6차원 데이터 200,000개를 갖는다.

가상의 데이터를 이용한 실험에서는 다양한 데이터 분포에서의 SWT의 데이터 제거 효과를 검증하기 위해 normalized limiting power (nLP)[5]를 척도로 성능을 측정하였고, 위에서 제시한 정지점 지정 방식과 데이터 변환 방식에 따른 데이터 제거 효과를 비교하였다. nLP는 스카이라인이 아닌 데이터 중에서 제거된 데이터의 비율로, 불필요한 정보를 얼마나 효과적으로 제거하는지를 측정하며, 값이 클수록 좋은 성능을 나타낸다. 전체 데이터를 r, 스카이라인을 w, 제거된 데이터를 U라 할 때, nLP는 다음과 같이 계산된다.

$$nLP = \frac{|U|}{|r| - |w|} \quad (16)$$

그림 5는 각 데이터 집합에서의 속성 수에 따른 nLP 값을 나타낸 것으로, SWT와 SaLSa를 비교한다. SWT는 정지점이 지배하는 모든 점을 제거하기 때문에 제거되는 데이터의 양적인 측면에서는 두 데이터 변환 방식이 동일하다. 또한, MaxSum을 이용한 정지점 지정 방식은 대부분의 실험에서 MaxVol 방식과 동일한 효과를 발휘하였다. 따라서 SWT를, 데이터 변환 방식에 상관없이, 정지점 방식에 따라 MaxMin과 MaxVol로만 구분하였다.

그림 5를 통해 모든 경우에 대해서 SWT가 SaLSa보다 많은 데이터를 제거함을 알 수 있다. 속성의 수가 증가할수록 SWT의 상대적인 효과는 더욱 상승한다. SaLSa의 경우 속성의 수가 증가함에 따라 정지점의 속성 간에 편차가 커지므로, 제거영역이 지배영역보다 작아지는데 반해, SWT는 언제나 제거영역과 지배영역을 동일하게 유지하기 때문이다.

이와 같은 상황이 independent와 anti-correlated 분포를 갖는 데이터에서 잘 나타났다. 하지만 데이터가 correlated 분포를 갖는 경우, 정지점의 속성 간에 편차가 크지 않고, 정지점이 대부분의 데이터를 지배하기 때문에 모든 실험에서 SaLSa와 SWT 모두 좋은 성능을 발휘하였다.

SWT의 두 방식 중에서는 MaxVol 방식이 MaxMin 방식에 비해 조금 더 좋은 성능을 발휘하였다. 이는 MaxVol 방식이 지배영역이 가장 큰 데이터를 정지점으로 지정함으로써, 보다 많은 데이터를 지배할 확률이 높기 때문이다.

실제 데이터를 이용한 실험에서는 SWT의 실제 효과 및 스카이라인을 계산하기 위해 소비되는 시간을 비교해 보았다. 실험은 2GB 메모리, 코어2듀오 2.4GHz의 성능을 갖는 데이터베이스 서버와, 2GB 메모리, 코어2쿼드 2.83GHz의 성능을 갖는 클라이언트 컴퓨터로 진행하였다.

SWT와 SaLSa는 스카이라인을 계산하기 위해 세 가지 과정을 수행한다. 첫 번째 과정은 데이터 전송이 일어나기 전에 서버에서 수행하는 전처리 과정으로, 정렬 등이 이에 해당한다. 두 번째 과정은 데이터를 서버에서 클라이언트로 전송하는 과정이다. 세 번째 과정은 전송 받은 데이터가 스카이라인인지 판별하는 과정이다.

SWT의 목적은 최대한 많은 데이터를 제거하여 두 번째 과정의 비용을 줄임으로써 전체적인 스카이라인 계산 시간을 단축시키는 것이다.

이와 같은 SWT의 효과가 그림 6을 통해 나타난다. 그림 6은 데이터 전송 속도에 따른 스카이라인 계산 시간을 나타낸 것으로, NBA 데이터를 이용한 실험 결과이다.

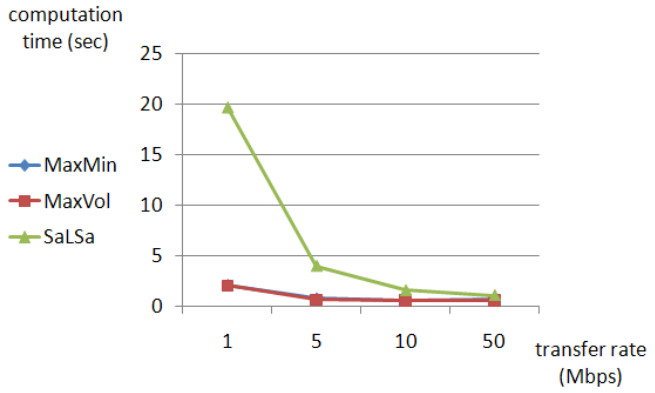

그림 6. NBA 데이터에 대한 스카이라인 계산 시간

총 17531개의 NBA 데이터집합으로부터 59개의 스카이라인 결과를 얻기 위해 SaLSa는 10209개의 데이터를, SWT는 897개의 데이터를 읽는다. 이를 각각 1Mbps, 5Mbps, 10Mbps, 50Mbps 의 전송 속도를 갖는 환경에서 실험한 결과 모든 경우에서 SWT가 SaLSa에 비해 우월한 성능을 발휘하였고, 두 번째 과정의 중요도가 높은, 전송 비용이 큰 환경에서는 SWT가 SaLSa에 비해 압도적인 성능을 발휘하였다.

5. 결론

본 논문에서는 스카이라인 질의 기능을 제공하지 않는 일반적인 데이터베이스 서버로부터 스카이라인을 효율적으로 계산하는 방법인 SWT를 제안하였다. SWT는 정지점의 활용을 극대화함으로써 스카이라인을 계산하기 위해 읽어 들이는 데이터를 최소화 한다.

정지점의 극대화된 활용은 정지점이 지배하는 모든 데이터를 제거할 때 이루어진다. SWT는 이를 위해 정지점을 미리 지정하여, 정지점을 기준으로 모든 데이터를 변환하는 작업을 수행한다.

실험을 통해서 SWT가 모든 경우에 대해, 기존 연구인 SaLSa보다 뛰어난 데이터 제거 효과 및 스카이라인 질의 처리 속도를 보임을 검증하였다. 속성의 수가 증가될수록 SWT의 효과는 커지고, 전송 비용이 큰 경우 SWT는 기존 연구에 비해 압도적인 성능을 발휘하는 것으로 나타났다.

Acknowledgement

이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임 (No. R01-2007-000-20135-0)

참고문헌

[1] Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of ICDE, pp. 421-430 (2001)

[2] Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedongs of VLDB Conference, pp. 275-286 (2002)

[3] Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM TODS 30(1), 41-82 (2005)

[4] K. C. K. Lee, B. Zheng, H. Li, and W. C. Lee. Approaching the skyline in z order. In VLDB, 2007.

[5] Bartolini, I., Ciaccia, P., Patella, M.: SaLSa:Computing the skyline without scanning the whole sky. In: Proceedings of CIKM,

[6] Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with Presorting. In: Proceedings of ICDE, pp. 717-816 (2003)

[7] Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient computation of skylines in subspaces. In ICDE 2006, Atlanta, GA, Apr. 2006.

[8] http://www.basketballreference.com/