2차원 공간에서 효율적인 선형 스카이라인 알고리즘

이종욱
성균관대학교 소프트웨어학과
jongwuklee@skku.edu

임현승
강원대학교 컴퓨터과학과
hsim@kangwon.ac.kr

김성수
한국전자통신연구원
sungsoo@etri.re.kr

Efficient Linear Skyline Algorithm in Two-Dimensional Space

Jongwuk Lee
Dept of Software,
Sungkyunkwan University

Hyeonseung Im
Dept of Computer Science,
Kangwon National University

Sung-Soo Kim
ETRI

요 약

스카이라인 질의는 지배 개념을 이용하여 사용자가 선호하는 데이터를 효과적으로 찾아주는 방법이다. 본 논문에서는 스카이라인 질의의 결과가 지나치게 증가하는 문제를 해결하기 위해서, 사용자의 선호도 함수가 임의의 선형 함수라고 가정하였을 때, 2차원 공간에서 선형 스카이라인 질의를 효율적으로 계산하는 알고리즘을 제안한다. 제안한 알고리즘은 기존의 선형 스카이라인 알고리즘과 비교하였을 때, 최대 7배 이상 실행 시간이 향상되는 것을 확인할 수 있었다.

1. 서 론

스카이라인 질의(skyline queries)는 사용자의 특정한 선호도 함수를 알지 못하더라고, 지배(dominance) 개념을 이용하여 사용자가 선호하는 데이터를 효과적으로 찾아주는 방법이다[1-4]. 즉, 각 속성에 대해서 최댓값 또는 최솟값을 선호한다는 정보만으로 스카이라인 질의를 적용할 수 있다. 또한, 스카이라인의 결과는 단조 증가 형태의 선호도 함수(monotone function)에 대해서 가장 선호되는 후보 집합(top-1 candidates)을 의미하므로, 대규모의 데이터에서의 의사 결정에 도움을 줄 수 있다. 반면에 사용하는 속성들이 반상관(anti-correlated) 관계이거나 속성의 수가 증가할 경우 스카이라인 질의의 결과가 지나치게 증가하는 단점이 있다[5].

이와 같은 문제를 해결하기 위해서 본 논문에서는 선형 스카이라인 질의에 초점을 둔다. 선형 함수(linear combination function)는 가장 널리 사용되는 선호도 함수의 형태이며, 사용자 선호도 함수를 선형 함수로 가정할 경우, 선형 스카이라인은 임의의 선형 함수에 대해서 가장 선호되는 후보 집합을 의미한다. 이와 같은 선형 스카이라인은 기존의 스카이라인의 질의 결과를 효과적으로 줄일 수 있다. 그러나 선형 스카이라인 질의는 튜플(tuple) 간의 지배 관계 이외에도 여러 개의 튜플로 구성된 집합에 대해서 다른 튜플이 지배가 되는지의 여부를 판단해야 하므로 계산 복잡도가 증가할 수 있다.

본 논문에서는 2차원 공간에서 선형 스카이라인을 효율적으로 계산하는 알고리즘을 제안한다. 제안한 알고리즘은 기존 알고리즘과 달리 선형 스카이라인 계산을 위해 가상 튜플(virtual tuple)을 이용하여 스카이라인을 구하거나[6,7] 또는 컨벡스 헐(convex hull) 계산[8]에 기반을 두지 않는다. 즉, 가상 튜플을 추가하지 않고 선형 스카이라인의 특성을 고려하여 가장 기초가 되는 두 개의 선형 스카이라인 튜플을 먼저 찾는다. 다음으로 기초가 되는 선형 스카이라인 튜플을 선으로 구성하여 나머지 튜플과 지배 관계를 고려하며, 점진적으로 남은 선형 스카이라인을 찾는다. 이와 같은 방법은 가상 튜플로 인한 불필요한 계산을 피할 수 있다. 제안한 알고리즘을 기존의 선형 스카이라인 알고리즘[6-8]과 비교하였을 때, 최대 7배 이상 실행 시간이 향상되는 것을 실험을 통해 확인할 수 있었다.

2. 선형 스카이라인 알고리즘

전체 데이터의 집합 $S=\{p^1,...,p^n\}$는 n개의 튜플로 구성되며, 속성 집합 $A=\{A_1,A_2\}$는 2개의 속성을 가진다. p, q, r은 임의의 튜플을 나타내며, 튜플 $p=(p_1,p_2)$의 각 속성 값 p_i는 실수 값을 갖는다. 다음은 선형 스카이라인의 계산을 위해 사용되는 주요 개념이다[1-5].

<정의 1> 지배(dominance) 튜플 p가 모든 속성에 대해서 튜플 q보다 작거나 같고, 최소한 하나의 속성에 대해서 p가 q보다 작을 경우($\forall A_i \in A: p_i \le q_i \wedge \exists A_j \in A: p_j < q_j$), p는 q를 지배($p \prec q$)한다고 한다.

<정의 2> 스카이라인(skyline) 스카이라인은 전체 집합 S에서 다른 어떤 튜플에도 지배되지 않은 튜플들의 집합이다.

$$sky(S)=\{p \in S \mid \nexists q \in S: p \prec q\}$$

스카이라인 질의는 단조 증가 형태의 선호도 함수에 대해서 가장 선호되는 후보 집합을 제공한다는 장점이 있다. 그러나 사용하는 속성들이 반상관 관계이거나 속성의 수가 증가할 경우 스카이라인 질의 결과가 지나치게 증가한다는 단점이 있다[5]. 이와 같은 문제를 해결하기 위해서, 본 논문에서는 선호도 함수를 선형 함수라고 가정하고 선형 스카이라인 질의를 효율적으로 계산하는 방법에 초점을 둔다.

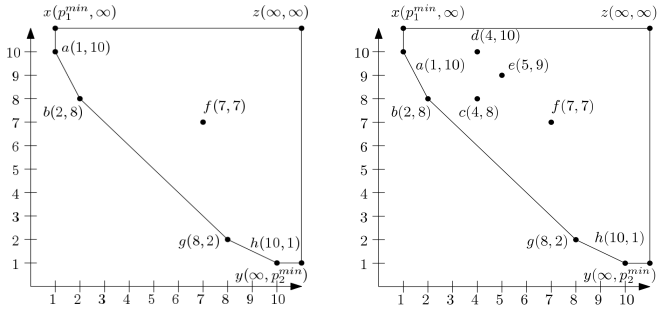
<정의 3> 선형 스카이라인(linear skyline) 2차원 공간에서의 선형 스카이라인은 임의의 선형 함수 $f(p)=\Sigma_{i=1}^{2} w_i \cdot p_i$에 대해서 가장 작은 값을 갖는 튜플들의 집합이다.

$$lsky(S)=\{p \in S \mid \nexists q \in S: f(q) < f(p)\}$$

2.1. 기존 선형 스카이라인 알고리즘

기존 연구[6-8]에서는 스카이라인과 컨벡스 헐(convex hull)을 계산하는 알고리즘을 조합하여 선형 스카이라인을 구하였다. 선형 스카이라인은 선형 함수만을 고려한 후보 집합을 고려하기 때문에 항상 스카이라인의 부분 집합이 된다. 또한 선형 스카이라인은 모든 튜플들을 둘러싼 다각형을 의미하는 컨벡스 헐의 부분 집합도 되는 특징이 있다.

이와 같은 특징을 이용하여 [6,7]에서는 스카이라인을 먼저 구한 후, 선형 함수를 고려했을 때 지배되는 튜플이 컨벡스 헐



(a) 기존 알고리즘 1[6,7] (b) 기존 알고리즘 2[8]

<그림 1> 선형 스카이라인을 구하기 위한 기존 알고리즘

에 포함되는 것을 막기 위해, 원점으로부터 컨벡스 헐의 바깥쪽 외곽 부분을 구성하는 가상 튜플(virtual tuple)을 추가한 뒤에 컨벡스 헐을 구하는 알고리즘을 제안하였다. <그림 1>(a)는 [6,7]에서 제안한 알고리즘이 동작하는 과정을 보여준다. 즉, 먼저 스카이라인 $\{a,b,f,g,h\}$를 구한 후, 가상의 튜플 집합 $\{x,y,z\}$를 추가한 다음 컨벡스 헐을 구한다. 이와 같은 경우, 기존의 컨벡스 헐에서 바깥쪽에 위치하는 튜플 f를 제거할 수 있다.

유사하게, [8]에서는 컨벡스 헐의 외곽 부분을 구성하는 가상 튜플을 추가한 뒤 컨벡스 헐 알고리즘을 그대로 적용하였다. <그림 1>(b)는 [8]에서 제안한 알고리즘이 동작하는 과정을 보여준다. 즉, 가상의 튜플 집합 $\{x,y,z\}$를 추가한 후 컨벡스 헐을 구한다. 이와 같은 기존의 알고리즘은 가상 튜플을 이용함으로써 선형 스카이라인을 정확하게 구할 수 있다. 그러나 가상 튜플을 추가한 뒤 기존 스카이라인 또는 컨벡스 헐 알고리즘을 활용함으로써 불필요한 계산이 발생하는 문제가 있다.

2.2 제안 알고리즘 QuickLS

본 논문에서는 가상 튜플을 추가하지 않고 선형 스카이라인의 특징만을 고려하여 효율적으로 선형 스카이라인을 계산하는 알고리즘을 제안한다. 구체적으로, 선형 스카이라인을 구하기 위해서 우선 선형 스카이라인에 속하는 외곽선을 효과적으로 찾는 것이 중요하다. 이를 위해, 각각의 속성에 대해서 가장 작은 값을 가지며 다른 튜플들에 의해 지배되지 않는 튜플들을 이용한다. 이와 같은 튜플들은 다음 <정리 1>에 의해 선형 스카이라인의 가장 바깥쪽에 위치하는 점이 될 수 있다. (<정리 1>은 속성 A_2에 대해서도 동일하게 적용된다.)

<정리 1> 다른 튜플에 의해 지배되지 않으면서 속성 A_1에서 가장 큰 값을 갖는 튜플 p는 속성 A_2에서 최솟값을 가진다. 즉, $p = \text{argmax}_{q\in sky(S)}\, q_1$이면 $p = \text{argmin}_{q\in S}\, q_2$을 만족한다.

<증명> 다른 튜플에 지배되지 않으면서, 속성 A_1에서 튜플 p보다 큰 값을 가지는 튜플 q가 존재한다고 가정하자. <정의 1>에 의해 q는 p보다 A_2에서 작은 값을 가지거나 같은 값을 가져야 한다. 그러나 p는 A_2에서 최솟값을 가지므로, q는 p보다 A_1에서 작은 값을 가질 수 없다. 따라서 가정은 위배되며, q는 존재할 수 없다.

본 논문에서는 <정리 1>을 이용하여 다음의 3 단계로 선형 스카이라인을 계산하는 알고리즘 QuickLS를 제안한다. (<그림 2>는 QuickLS의 의사코드이다.) 우선, <단계 1>로 선형 스카이라인의 기초 선에 해당되는 각 속성에서 어떤 튜플에도 지배되지 않으면서 가장 큰 값은 가지는 두 개의 튜플 p와 q를 찾는다. 찾은 두 개의 튜플이 동일한 경우 알고리즘은 그대로 종료한다. (즉, 다른 모든 튜플이 하나의 튜플에 의해 지배되는 경우를 의미한다.) 그렇지 않을 경우, 추가적인 선형 스카이라인

Algorithm 1: QuickLS(S)

```
1  LSKY(S) ← {}
   // step 1:  finding a skyline tuple with the maximum value
2  x ← argmax_{q∈SKY(S)} q_1
3  y ← argmax_{q∈SKY(S)} q_2
4  LSKY(S) ← LSKY(S) ∪ {x}
5  if x and y are same then
6      return LSKY(S) // degenerate case
7  LSKY(S) ← LSKY(S) ∪ {y}
8  S ← S − {x, y}
   // recursion for remaining tuples
9  LSKY(S) ← LSKY(S) ∪ FindLinearSkyline(x, y, S)
10 return LSKY(S)
```

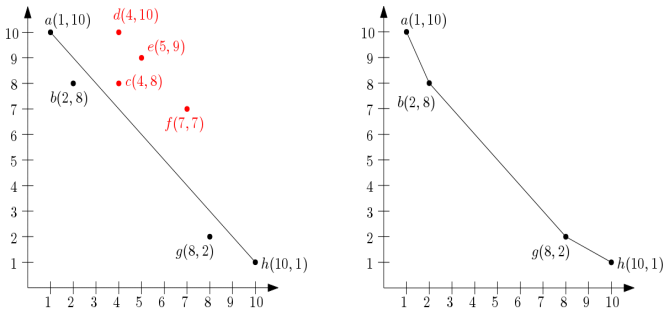
Algorithm 2: FindLinearSkyline(x, y, S)

```
1  LSKY(S) ← {}
   // step 2:  removing dominated tuples from dataset S
2  foreach p ∈ S do
3      if p is dominated by xy then
4          S ← S − {p}
   // termination condition
5  if S is empty then
6      return LSKY(S)
7  else
       // step 3:  finding a new linear skyline tuple
8      Find the furthest tuple z ∈ S from xy
9      LSKY(S) ← LSKY(S) ∪ {z}
10     S ← S − {z}
       // recursion for remaining tuples
11     LSKY(S) ← LSKY(S) ∪ FindLinearSkyline(x, z, S)
12     LSKY(S) ← LSKY(S) ∪ FindLinearSkyline(z, y, S)
13     return LSKY(S)
```

<그림 2> 제안 알고리즘 QuickLS의 의사 코드

을 찾기 위해서 두 개의 튜플을 이은 선 $\overline{pq}$가 나머지 튜플들을 지배하는지 확인한다 (<단계 2>). 즉, 선 $\overline{pq}$에 대해 어떤 튜플이 원점에서 먼 영역에 위치할 경우, 이 튜플은 두 개의 튜플 중 어느 하나의 튜플에 의해서 지배되거나 선형적으로 지배됨을 의미한다. <단계 3>에서는 지배되지 않은 튜플 중에서 선에서 가장 먼 튜플 r을 새로운 선형 스카이라인 튜플로 추가한다. 새롭게 추가된 튜플에 대해서 두 개의 선 $\overline{pr}$과 $\overline{rq}$을 구성하여 <단계 2>와 <단계 3>의 과정을 새로운 선형 스카이라인 튜플이 존재하지 않을 때까지 재귀적으로 반복한다.

<그림 3>은 제안한 알고리즘 QuickLS의 동작 과정을 보여준다. <단계 1>로 <정리 1>을 이용하여 외곽선에 해당하는 튜플 a와 h를 찾는다. <단계 2>로 두 튜플로 구성된 선 $\overline{ah}$에 대해서 지배되는 튜플(빨간색)들을 제거한다. <단계 3>으로 남은 튜플 중에서 선 $\overline{ah}$에서 가장 멀리 위치하는 튜플을 찾는다. 이 과정을 재귀적으로 반복하여 최종적으로 선형 스카이라인 $\{a,b,g,h\}$을 구할 수 있다.



(a) 외각선 $\overline{ah}$을 찾고, 지배되는 튜플 (빨간색) 제거 (b) 남은 점들에 대해서 선형 스카이라인을 추가

<그림 3> QuickLS를 이용해 선형 스카이라인을 구하는 과정

3. 실 험

3.1 실험 환경

본 장에서는 본 논문에서 제안한 QuickLS 알고리즘의 효율성을 실험을 통해 평가한다. 이를 위해 QuickLS와 기존 선형 스카이라인 알고리즘의 수행시간을 측정하고 비교 분석한다. 본 실험에서 고려한 알고리즘은 다음과 같다.

- SSkyline[3]: 스카이라인 알고리즘
- QHULL: 컨벡스 헐 알고리즘
- BSHELL[6,7]: SSkyline을 이용하여 스카이라인을 구한 후 선형 스카이라인을 계산하는 기존 알고리즘 1
- BSHELL2[8]: 가상 포인트를 추가한 후 QHULL을 이용하여 선형 스카이라인 계산하는 기존 알고리즘 2
- QuickLS: 제안한 알고리즘

SSkyline과 QHULL은 기존 알고리즘의 기반이 되는 알고리즘으로, 기존 알고리즘 BSHELL과 BSHELL2의 효율성이 특정 알고리즘에 얼마나 의존하는지를 확인하기 위해 비교하였다. 모든 알고리즘은 C++ 언어로 작성하였으며 g++ -O3 옵션을 이용해 컴파일 하였다. 모든 실험은 Intel Xeon E5-2630V3 2.4GHz, 64GB RAM 사양의 Ubuntu 14.04 워크스테이션에서 수행하였다. 실험 데이터로는 스카이라인 알고리즘의 성능을 측정하는데 주로 사용되는 독립(independent) 또는 반상관 분포[1]를 고려하였으며, 데이터는 1M, 2M, 4M, 8M, 16M 크기의 합성 데이터(synthetic data)를 이용하였다. 이와 같은 실험 환경은 기존의 스카이라인 알고리즘을 평가할 때 널리 사용되는 방법이다.

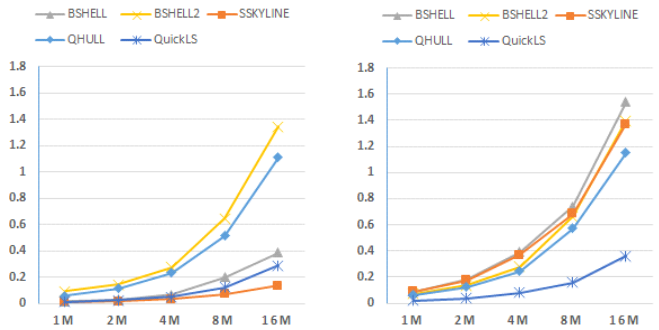
3.2 실험 결과

<그림 4>는 QuickLS와 다른 알고리즘 간의 성능을 비교한 결과이다. 각 알고리즘의 성능 척도로 각 데이터에 대한 수행시간을 10회 반복 측정하여 평균값을 이용하였다 (각 데이터 크기에 대해 10개의 실험 데이터를 생성하여 이용하였다). 그래프의 세로축은 실행 시간(초단위)이고 가로축은 데이터의 크기다. 실험 결과에 따르면 데이터의 분포와 크기에 상관없이 본 논문에서 제안한 QuickLS가 BSHELL[6,7], BSHELL2[8]보다 항상 우수함을 확인할 수 있다. QuickLS는 독립 분포 데이터에서는 BSHELL보다 최대 1.64배 (데이터 크기 8M), 평균 1.3배 빠르고, BSHELL2보다 최대 7.65배 (데이터 크기 1M), 평균 5.8배 빠르며, 반상관 분포 데이터에서는 BSHELL보다 최대 4.75배 (데이터 크기 8M), 평균 4.52배 빠르고, BSHELL2보다 최대 4.16배 (데이터 크기 8M), 평균 3.68배 빠름을 확인할 수 있었다.

<표 1>에는 각 실험 데이터에 대한 스카이라인과 선형 스카이라인의 개수를 요약했다. 특히 반상관 분포 데이터에서의 스카이라인 개수가 독립 분포 데이터에서의 스카이라인 개수보다 10배 정도 많음을 확인할 수 있는데, 이는 <그림 4>의 실험 결과와도 일치한다. 스카이라인을 구하는 SSkyline의 성능이 스카이라인의 개수가 적은 독립 분포 데이터에서 매우 우수하며, 따라서 내부적으로 SSkyline을 이용하는 BSHELL의 성능이 QHULL을 이용하는 BSHELL2보다 독립 분포 데이터에서는 더 우수하다. QHULL의 성능은 스카이라인의 개수와는 상관없으며 따라서 내부적으로 QHULL을 이용하는 BSHELL2의 성능은 데이터의 분포에 큰 영향을 받지 않음을 확인할 수 있었다. 마지막으로 QuickLS는 데이터의 분포(스카이라인의 개수)나 크기에 상관없이 항상 우수한 성능을 보임을 확인할 수 있었다.

4. 결 론

본 논문에서는 선형 스카이라인의 고유한 특징을 이용하여 2차원 공간에서 선형 스카이라인을 효율적으로 계산하는 알고리즘을 제안하였다. 제안한 알고리즘은 기존 알고리즘보다 데이터의 분포 및 크기에 상관없이 최대 7배 이상의 효율성 향상을 보였다. 후속 연구로는, (1) 병렬화 또는 분산 처리 기술을 이용하여 제안한 알고리즘을 최적화하는 방법과 (2) 3 차원 이상의 고차원 공간에서 선형 스카이라인을 효율적으로 계산하는 알고리즘에 대해 진행하고자 한다.



(a) 독립 분포 데이터 (b) 반상관 분포 데이터

<그림 4> 제안 알고리즘과 기존 알고리즘간의 효율성 비교

<표 1> 데이터의 크기와 데이터 분포에 따른 스카이라인과 선형 스카이라인의 개수 비교

(a) 독립 분포 데이터

	1M	2M	4M	8M	16M
스카이라인	14.9	16.2	16	16.7	18.6
선형 스카이라인	10	12.1	10.5	12.7	12.4

(b) 반상관 분포 데이터

	1M	2M	4M	8M	16M
스카이라인	150	156.3	168.6	174.3	184.3
선형 스카이라인	15.9	15.6	16.1	17.1	19

본 연구는 ETRI R&D 프로그램 (“듀얼모드 배치·쿼리 분석을 제공하는 빅데이터 플랫폼 핵심 기술 개발, 16ZS1410”)의 일환으로 수행하였습니다.

참 고 문 헌

[1] S. Börzsönyi, D. Kossmann, K. Stocker, “The skyline operator,” In IEEE ICDE, 2001, pp. 421-430.

[2] D. Papadias, Y. Tao, G. Fu, B. Seeger, “Progressive skyline computation in database systems,” ACM Trans. Database Syst., 30 (1), 2005, pp 41-82.

[3] S. Park, T. Kim, J. Park, J. Kim, H. Im, “Parallel Skyline Computation on Multicore Architectures,” In IEEE ICDE, 2009, pp. 760-771

[4] J. Lee, S. Hwang, “BSkytree: scalable skyline computation using a balanced pivot selection,” In EDBT, 2010, pp. 195-206.

[5] J. Lee, G. You, S. Hwang, “Personalized top-k skyline queries in high-dimensional space,” Inf. Syst., 34 (1), 2009, pp 45-61.

[6] Jun-Seok Heo, Junghoo Cho, Kyu-Young Whang, “Subspace top-k query processing using the hybrid-layer index with a tight bound,” Data Knowl. Eng., 83, 2013, pp 1-19

[7] Sun-Young Ihm, Ki-Eun Lee, Aziz Nasridinov, Jun-Seok Heo, Young-Ho Park, “Approximate convex skyline: A partitioned layer-based index for efficient processing top-k queries,” Knowl.-Based Syst., 61, 2014, pp 13-28

[8] Michael Shekelyan, Gregor Jossé, Matthias Schubert, “Linear path skylines in multicriteria networks,” In IEEE ICDE, 2015, pp. 459-470