거리의 합이 최소가 되는 λ 이산 2-중심 문제
(Min-Sum λ Discrete 2-Center Problem)

신 찬 수 † Alexahder Wolff ††
(Chan-Su Shin)

요 약 본 논문에서는 기관위치설정 문제 중에서, 이차원 평면에 주어진 n개의 점들로 구성된 집합 P에 대한 이산 2-중심 문제를 다룬다. P의 있는 서로 다른 두 점을 중심으로 선택하는 데, P의 다른 점들은 두 중심 중에서 가까운 중심까지의 거리의 최대값과 두 중심 사이의 거리의 합이 최소가 되도록 선택하는 것이 목적이다. 본 논문에서는 $O(n^2\log n)$ 시간에 이산 2-중심을 구하는 알고리즘을 제시한다.

키워드 : 계산기하학, 기하 알고리즘, 이산 2-중심 문제

Abstract In this paper, we deal with the following facility location problem. Given a set P of n points in the plane, find two (discrete) centers p and q in P that minimize the sum of there distance plus the distance of any other point to the closer center. In this paper, we propose an $O(n^2\log n)$-time algorithm to compute the two centers.

Key words : compuational geometry, geometric algorithm, discrete 2-center problem

1. 서 론

이차원 평면에 주어진 n개의 점들의 집합을 P라 하고, 서로 다른 두 점 p, $q \in P$를 고려하자. $P\setminus\{p, q\}$의 점들에 대해, q보다 p에 가까운 점들 중에서 거리가 가장 먼 점을 f_p라 하고, p보다 q에 가까운 점 중에서 거리가 가장 먼 점을 f_q라 하자. 그러면, $\delta(p,q)=\lambda\cdot d(p, q)+\max(d(p,f_p), d(q,f_q))$라 정의하자. 여기서 $d(a, b)$는 평면에 주어진 두 점 a, b사이의 유클리디언 거리를 의미하고, λ는 음이 아닌 실수를 나타낸다. P에 속한 모든 두 점 p, q에 대해, $\delta(p, q)$가 가장 작은 두 점을 P의 *λ 이산 2-중심*(discrete 2-center)이라 한다. 특별히, $\lambda=0$인 경우에 대부분의 연구가 집중되었는데, λ를 생략하여 이산 2-중심 문제라 불린다.

이러한 λ 이산 2-중심 문제는 서비스 제공을 위해 특정한 지점까지 도착하는 데 걸리는 시간이 중요한 소방서나 경찰서 등과 같은 공공 기관의 위치를 결정하는 데 응용된다. 어떤 마을의 중요 시설이나 주택 등을 이차원 평면의 점들로 표현하고, 두 개의 소방서를 그 점들 가운데 두 곳을 결정하여 설치한다고 가정하자. 만약, 어떤 점에서 화재가 발생했다면, 그 곳으로부터 가까운 곳에 있는 소방서에서 화재 진압을 위해 출동해야 한다. 즉, 두 소방서의 위치 p, q는 $\max(d(p,f_p),d(q,f_q))$가 최소화되는 두 곳에 위치하는 것이 바람직하다. 이것은 점 집합 P에 대해 $\lambda=0$인 경우에 이산 2-중심을 구하는 것과 동일하다. 그러나 소방서와 다른 성질의 서비스 기관을 설치한다면 고려 사항이 달라질 수도 있다. 예를 들어, 두 기관의 협동 작업이 요구된다면, 두 서비스 기관 사이의 거리도 함께 고려되어야 한다. 서비스 기관 사이의 거리에 λ 만큼의 가중치를 부과하여 두 기관의 위치를 결정한다면, 이 문제가 바로 λ 이산 2-중심 문제이다. 본 논문에서는 점 집합 P에 대한 λ 이산 2-중심을 구하는 효율적인 알고리즘을 제시한다.

이와 관련된 다양한 문제들을 살펴보자. 이산 2-중심 문제를 확장하여 2개보다 많은 k개의 중심의 위치를 찾는 이산 k-중심 문제를 생각해 볼 수 있다. 또한, k개의 중심이 P의 점들 중에서 선택되는 것이 아니라 평면 위의 임의의 점이 될 수 있다고 가정할 수도 있다. 이 문제를 *연속 k-중심*(continuous k-center) 문제라 부

· 본 연구는 한국과학재단 목적기초연구 R05-222-00780-0 지원으로 수행되었음.

† 학생회원 : 한국외국어대학 전자정보학부 디지털정보공학 교수
cssin@hufs.ac.kr
†† 비 회 원 : Institute of Mathematics and Computer Science
awolff@uni-greifswald.de

논문접수 : 2001년 12월 1일
심사완료 : 2002년 3월 7일

른다. 일반적으로 이산 또는 연속 k-중심 문제에서 k가 문제의 입력으로 주어진 경우에는 NP-hard라는 것이 증명[1]되어 있다. 그래서, 지금까지의 연구 방향은 k가 상대적으로 작은 경우 (주로, k=1, 2, 3 인 경우)일 때 빠른 시간에 동작하는 알고리즘의 개발이나 근사적으로 중심의 위치를 구하는 근사 알고리즘의 개발에 집중되었다. 일반적인(λ=0) 이산 2-중심 문제에 대해서는 $O(n^{4/3}\log^5 n)$ 시간에 동작하는 알고리즘[2]이 가장 빠른 알고리즘이고, 연속 2-중심 문제에 대해서는 $O(n\log^2 n)$시간의 랜덤(randomized) 알고리즘[3]과 $O(n\log^2 n\log^2\log n)$ 시간의 결정(deterministic) 알고리즘[3]이 현재까지 알려진 가장 좋은 알고리즘이다. 이 외에도 트리를 포함한 그래프에서의 k-중심 문제나 k-서버와 같은 온라인(online) 문제, 또는 다른 거리 모델에서의 k-중심 문제 등이 연구되었다. 그러나, 본 논문에서 살펴볼 λ>0인 경우에 이산 2-중심 문제에 대해서는 알려진 결과가 없다.

마지막으로 $d(p,q)+\max(d(p,f_p),d(q,f_q))$를 최소화하는 대신에 $d(p,q)+d(p,f_p)+d(q,f_q)$ 값이 최소가 되는 두 점 p, q를 구하는 문제를 생각할 수 있다. 이 문제는 점 집합에 대한 *최소 지름 신장 트리*(minimum-diameter spanning tree)를 구하는 문제를 풀면 된다. P에 대한 신장 트리 T의 지름은 임의의 두 점을 트리의 에지를 따라가면서 트리 경로로 연결했을 때 경로를 구성하는 에지의 길이의 합 중에서 가장 큰 값으로 정의된다. 그런데, 최소 지름을 갖는 신장 트리는 P의 한 점 p에 다른 모든 점들이 연결된 *단항 트리*(monopolar tree)이거나 P의 두 점 p, q 점을 중심으로 p, q를 제외한 다른 점들이 각각 p, q 중 하나에 연결된 *이항 트리*(dipolar tree) 가운데 하나라는 사실이 Ho 등[4]에 의해 증명되었다. 그래서 P에 대해 최소 지름을 갖는 이항 트리를 구하면, 그 트리의 두 중심 p, q가 바로 $d(p,q)+d(p,f_p)+d(q,f_q)$ 값을 최소로 하는 점이 된다. [4]의 알고리즘에 따르면 $O(n^3)$시간에 계산할 수 있다. (이 시간보다 더 빠른 시간에 동작하는 알고리즘은 현재까지 알려져 있지 않고, 최소 지름의 이항 트리보다 지름이 5/4배 이내의 지름을 갖는 근사(approximted) 이항 트리를 $O(n^2\log n)$ 시간에 계산하는 근사 알고리즘만이 최근에 제시되었다[5].)

용어 정의

이제, 본 논문에서 사용할 몇 가지 정의 및 기호에 대해 설명한다. P의 속한 n개의 점들은 이차원 평면에 주어진다. P의 서로 다른 두 점 p, q에 대해, p, q를 연결하는 선분의 이등분선(bisector)을 $b_{p,q}$라 하고, $h_{p,q}$를 $b_{p,q}$에 의해 나뉜 두 반평면(half-space) 중에서 p를 포함하는 반평면으로 정의한다. 반평면 $h_{q,p}$는 q를 포함하는 반평면으로 정의된다. 그러면 이차원 평면은 $h_{p,q}$, $b_{p,q}$, $h_{q,p}$로 분할된다. P의 한 점 $f_{p,q}$는 $P\backslash h_{q,p}$에 있는 P의 점들 중에서 p로부터 가장 먼 점으로 정의한다. (여기서, $P\backslash h_{q,p}$라는 것은 P의 점들 중에서, 반평면 $h_{q,p}$에 포함되지 않는 점들을 의미한다.) 만약, $P\backslash h_{q,p}$에 어떠한 점도 놓여 있지 않다면, $f_{p,q}$는 p 자체가 된다. 즉, $f_{p,q}=p$가 된다. 반대로 $f_{q,p}$는 $P\backslash h_{p,q}$에 있는 점 중에서 q로부터 가장 먼 점으로 정의한다. $P\backslash h_{p,q}$에 점이 없다면, $f_{q,p}=q$가 된다. 그러면, $\delta(p,q)=\lambda d(p,q)+\max(d(p,f_{p,q}),d(q,f_{q,p}))$로 정의된다.

디스크 $D(c,r)$은 점 c를 중심으로 하고, 실수 $r\geq 0$을 반지름으로 한 원의 경계와 내부로 정의된다.

2. λ 이산 2-중심을 구하는 알고리즘

알고리즘은 P의 두 점으로 구성된 모든 쌍 (p,q)에 대해 $f_{p,q}$와 $f_{q,p}$를 $O(n^2\log n)$ 시간에 계산한다. 그러면, $\delta(p,q)$를 상수시간에 계산할 수 있고, $O(n^2)$개의 (p,q) 쌍에 대해서 가장 작은 $\delta(p,q)$ 값을 갖는 쌍을 λ 이산 2-중심으로 출력하면 된다.

이를 위해, P의 한 점 p를 고정한다. p를 제외한 P의 $(n-1)$개의 점 q에 대해 $f_{p,q}$를 모두 계산한다. 이 과정이 $O(n\log n)$에 수행된다면, 모든 (p,q)쌍에 대해 $f_{p,q}$, $f_{q,p}$는 $O(n^2\log n)$에 계산할 수 있다. 이 과정을 지금부터 이 절의 끝까지 설명한다.

다음의 소정리 1이 중요한 역할을 한다.

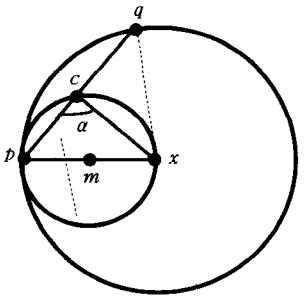

그림 1 소정리 1의 증명

소정리 1. 두 점 $p, q\in P$을 고려하자. 점 p로부터 가장 먼 점을 $x\in P(x\neq q)$라 하자. 점 $f_{p,q}=x$라면

$q \notin D(x, d(x,p))$이다. 그 역도 성립한다.

(증명) 점 x가 p로부터 가장 먼 점이기 때문에, $x = f_p$, q라면 $x \notin h_{q,p}$이고 그 역도 성립한다. (그림 1 참조) 이것은 각 $\alpha = \angle pcx$가 90도보다 작다는 조건과 동일하다. 여기서 c는 선분 pq의 중점으로 정의된다. 만약, 점 m이 선분 px의 중점이라면, 위의 조건은 다시 $c \notin D(m, d(m,p))$이라는 조건과 일치한다. 즉, $d(m,c) > d(m,p)$이다. 삼각형 Δpxq의 두 변 pq, px의 중점을 연결하여 삼각형 Δpmc이 정의되므로 두 삼각형은 서로 닮은꼴이다. 이 성질과 $d(p,q) = 2d(p,c)$라는 사실을 이용하면, $d(x,q) > d(x,q)$임을 쉽게 알 수 있다. 즉, $q \notin D(x, d(x,p))$이 성립한다. (QED)

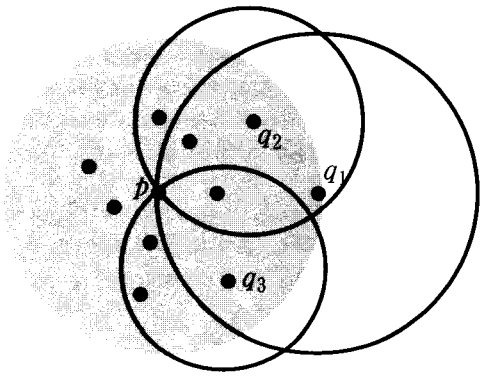

그림 2 고정된 점 p에 대한 $f_{p,q}$

이제, P의 한 점 p를 고정하자. 그림 2 참조. 각 점 $q \in P \setminus \{p\}$에 대해 $f_{p,q}$를 구하는 알고리즘을 설명한다. 우선, 점 p로부터 거리가 먼 순서대로 P의 점들을 나열하여, $q_1, q_2, \cdots, q_n$이라 하자. 그러면 q_1은 p에서 가장 먼 점이고 $q_n = p$이다. 점 q_1부터 차례로 고려한다. $P_1 = P \setminus \{p\}$이라 하자. P_1의 점들 중에서 디스크 $D(q_1, d(p, q_1))$ 외부에 있는 점이 있다고 가정하고, 그러한 점 중의 하나를 z라 하자. 그러면 정의에 의해, q_1은 p로부터 가장 먼 점이고, $D(q_1, d(p, q_1))$ 외부에 점 z가 존재하므로, 소정리 1에 의해 $f_{p,z}$는 바로 q_1이 된다. P의 부분집합 $Q_1 = \{q_i \in P_1 \mid q_i \notin D(q_1, d(p, q_1))\}$이라 정의하면, Q_1에 속하는 모든 점 z에 대해, $f_{p,z} = q_1$ 임을 알 수 있다. 이제, $P_2 = P_1 \setminus Q_1$으로 정의한다. 두 번째 점 q_2를 중심으로 한 디스크 $D(q_2, d(p,q_2))$ 외부에 존재하는 P_2의 점들을 Q_2라 하면, 소정리 1은 점 $z \in Q_2$에 대해 $f_{p,z} = q_2$가 됨을 말해준다. 같은 방법으로 $P_i = P_{i-1} \setminus Q_{i-1}$을 정의하고, 디스크 $D(q_i, d(p,q_i))$ 외부에 있는 P_i의 점 집합을 Q_i로 정의하면, Q_i에 속하는 모든 점 z에 대해 $f_{p,z} = q_i$가 된다.

디스크 $D_i = D(q_i, d(p,q_i))$로 정의하자. 주의할 점은 디스크 $D_1, \cdots, D_n$의 경계 위에 공통적으로 점 p가 놓여 있다는 것이다. 즉 모든 디스크들의 경계는 점 p에서 서로 교차한다. 위에서 설명한 알고리즘을 다시 설명하면 다음과 같다. Q_i에 속하는 점들은 $D_1, \cdots, d_{i-1}$에 각각 포함되지만, D_i에 대해서는 외부에 존재한다. 결국, Q_i의 점들은 $D_1 \cap \cdots \cap D_{i-1}$에 포함되지만, $D_1 \cap \cdots \cap D_i$에는 포함되지 않는다. 그래서, 각 점 $q \in P \setminus \{p\}$에 대해, $q \in D_1 \cap \cdots \cap D_{i-1}$이지만 $q \in D_1 \cap \cdots \cap D_i$가 성립하는 최소 인덱스 i를 찾으면 $f_{p,q} = q_i$가 됨을 알 수 있다. 점 q에 대한 최소 인덱스를 $\sigma(q)$라 표기한다. 이하에서는 점 $q \in P \setminus \{p\}$에 대한 $\sigma(q)$를 $O(\log n)$ 시간에 찾을 수 있는 자료구조인 탐색 트리 T를 $O(n \log n)$ 시간에 구성하는 방법을 제시한다.

트리 T의 구성 방법

트리 T는 높이가 $O(\log n)$인 균형 이진 트리(balanced binary tree)로써, 가장 왼쪽 리프 노드에 D_1을 저장하고, 오른쪽 리프 노드들에는 차례대로 $D_2, \cdots, D_n$까지 저장한다. T의 노드 v에 대해, v를 루트로 하는 T의 부트리(subtree)를 T_v로 표기하자. T의 리프 노드가 아닌 내부 노드 v에는 T_v에 속한 리프 노드에 저장되어 있는 디스크들의 교차영역 $J(T_v)$이 저장된다. 모든 내부 노드 v에 대한 $J(T_v)$는 리프 노드들의 높이 (또는 레벨(level))로부터 루트의 높이까지 차례로 올라가면서 상향식으로 계산한다. 두 자식 노드 u와 w를 가진 내부 노드 v에 대해, $J(T_v) = J(T_u) \cap J(T_w)$가 된다. 교차영역 $J(T_u)$와 $J(T_w)$는 이미 계산되어 있으므로 두 영역의 교차 부분을 계산하면 된다.

우선, 몇 가지 용어를 정의하자. 이차원 평면에 주어진 면적이 있는 닫힌 영역 J의 경계(boundary)를 ∂J로 표기한다. $|T|$는 트리 T의 리프 노드의 개수를 의미하며, $|J|$는 교차영역 J의 경계에 참여하는 원호의 개수로 정의된다.

소정리 2. 한 점 p를 공유하고 있는 디스크들의 집합 $D_1, \cdots, D_m$의 교차영역의 경계를 구성하는 원호는 모두 서로 다른 디스크의 원호이다. 그래서, 교차영역을 구성하는 원호의 개수는 최대 m개이다.

(증명) (그림 3 참조) $J = D_1 \cap \cdots \cap D_m$이라 하고, D_i의 경계를 이루는 원호 C_i가 J에 참여한다고 하자. (여기서, $1 \le i \le m$이다.) ∂J에서 원호 C_i에 시계반대방향으로

인접한 원호를 C_j라 하고 시계방향으로 인접한 원호를 C_k라 하자. (여기서 $1\le j, k\le m$ 이다.) C_j와 C_k는 각각 디스크 D_j와 D_k의 경계의 일부라고 가정한다. 교차점 $s_j = C_i \cap C_j$라 하고, $s_k = C_i \cap C_k$라 하자. 두 가지 경우로 나눈다. s_j와 s_k 모두 p가 아닌 경우부터 고려하자. 그러면, D_i와 D_j는 s_j와 p에서 두 번 교차한다. 모든 원은 최대 두 점에서 만나기 때문에, 그 두 점 이외의 곳에선 전혀 만나지 않는다. 결국, 그림 3에서 보는 것처럼, 점 p에서 시계반대방향으로 점 s_j까지의 D_i의 경계 원호는 J의 경계를 구성하지 못한다. 같은 이유로, 점 p부터 시계방향으로 점 s_k까지의 D_i의 경계 원호 또한 J의 경계를 이루지 못한다. 결국, D_i의 원호 C_i만이 오직 한 번 ∂J에 참여함을 알 수 있다. 두 번째 경우는 s_j와 s_k 중 하나가 p와 일치하는 경우인데, 첫번째 경우와 비슷하게 증명할 수 있다. 결국, D_i의 원호는 최대 한번만 ∂J에 참여할 수 있다. 그래서, $|J| \le m$이 된다. (QED)

이제, $J(T_v) = J(T_u) \cap J(T_w)$를 계산하는 방법을 설명한다. 위의 소정리 2에 의해, $\partial J(T_u)$ (또는 $\partial J(T_w)$) 속하는 하나의 원호가 $\partial J(T_v)$에 참여한다면 오직 한 번만 참여하게 된다. 이것은 기존의 두 볼록 다각형의 교차영역을 계산하는 선형시간 알고리즘[6]을 그대로 적용할 수 있음을 의미한다. 즉, $\partial J(T_u)$와 $\partial J(T_w)$의 원호를 차례대로 방문하면서 원호끼리의 교차점을 찾아 $\partial J(T_v)$를 구성하면 된다. 결국, $|J(T_v)| \le |J(T_u)| + |J(T_w)| \le |T_u| + |T_w| = |T_v|$이므로 $J(T_v)$는 $O(|T_v|)$ 시간에 계산할 수 있다. 트리 T의 특정 높이에 존재하는 내부 노드 v에 대해, $\sum_v |T_v| = |T|$이므로 같은 높이에 있는 모든 노드의 $J(T_v)$를 구하는 데 걸리는 시간은 $O(|T|) = O(n)$이면 충분하다. T의 높이가 $O(\log n)$이므로 T를 구성하는 데 필요한 시간은 $O(n \log n)$이 된다. 같은 높이에 있는 노드 v에 저장되는 $J(T_v)$의 경계를 구성하는 원호의 개수 또한 $O(n)$이 되어, 전체적으로는 $O(n \log n)$공간 복잡도가 요구된다.

지금까지는 T의 각 내부 노드 v에는 $\partial J(T_v)$를 구성하는 원호들이 시계반대방향의 순서로 저장되어 있다고 가정하고 알고리즘을 서술하였다. 그러나, $\sigma(q)$의 탐색을 효과적으로 수행하기 위해 $\partial J(T_v)$를 저장하는 방법을 다음과 같이 수정한다. $\partial J(T_v)$의 왼쪽 끝 점과 오른쪽 끝 점을 기준으로 $\partial J(T_v)$를 위쪽 체인과 아래쪽 체인으로 나누어 배열(array) 형태로 저장한다. (여기서 주의할 것은 두 체인 모두 x축에 대해 단조(monotone)하다는 것이다.) 위쪽 체인과 그 체인의 수직한 방향으로 아래쪽에 있는 영역을 $J^+(T_v)$라 표기하고, 아래쪽 체인과 그 체인의 위쪽 영역을 $J^-(T_v)$라 표기하면, $J(T_v) = J^+(T_v) \cap J^-(T_v)$가 된다. 이제는 내부 노드 v에서 $\partial J(T_v)$을 저장하는 대신 $\partial J^-(T_v)$와 $\partial J^-(T_v)$를 저장한다. 정의에 의해, $J^+(T_v) = J^+(T_u) \cap J^+(T_w)$이고 $J^-(T_v) = J^-(T_u) \cap J^-(T_w)$이므로 볼록 다각형 교차 알고리즘을 적용하여 $\partial J^+(T_v)$는 $\partial J^-(T_u)$, $\partial J^-(T_u)$로부터 계산하고, $\partial J^-(T_v)$는 $\partial J^+(T_w)$, $\partial J^-(T_w)$로부터 계산할 수 있다. T의 구성 시간은 $O(n \log n)$으로 동일하다.

T를 탐색하여 $\sigma(q)$ 계산하기

마지막으로 트리 T를 이용하여, 주어진 점 $q \in P \setminus \{p\}$에 대한 인덱스 $\sigma(q)$를 $O(\log n)$ 시간에 구하는 방법을 설명한다. 우선, 내부 노드 v에 저장된 $J(T_v)$에 q가 포함되는지 아닌지를 $O(\log n)$ 시간에 결정하는 방법을 설명한다. 노드 v에는 $\partial J^-(T_v)$와 $\partial J^-(T_v)$를 구성하는 원호들이 배열에 저장되어 있다. 만약, q가 $J(T_v)$에 포함된다면, 정의에 의해, $q \in J^+(T_v) \cap J^-(T_v)$이고 그 역도 성립해야 한다. 이 조건은 다시 q가 $\partial J^+(T_v)$의 아래쪽에 $\partial J^-(T_v)$에 대해서는 위쪽에 존재하는 것과 같다. 여기서는 q가 $\partial J^+(T_v)$의 아래쪽에 있는지를 검사하는 방법만을 설명한다.

점 q를 지나는 수직선을 h라 하자. 만약 h가 $\partial J^+(T_v)$의 왼쪽 끝점보다 왼쪽에 있거나 오른쪽 끝점보다 오른쪽에 있다면 체인과 만나지 않는다는 것이 되어, $q \notin J^+(T_v)$임을 의미한다. $q \in J^+(T_v)$이기 위해서는 h가 반드시 $\partial J^+(T_v)$와 교차해야 한다. h와 교차하는 $\partial J^+(T_v)$의 원호를 C라 하자. 원호 C의 바로 아래쪽에 q가 놓이면 $q \in J^+(T_v)$가 되고 위쪽에 놓이면 $q \notin J^+(T_v)$가 된다. 그런데, C는 x축에 단조하기 때문에 q가 C의 어느 쪽에 놓이는지는 산술 계산에 의해 상수시간에 알 수 있다. 이제 남은 문제는 h와 교차하는 원호 C를 찾는 것이다. 체인 $\partial J^+(T_v)$ 또한 x축에 단조하기 때문에 체인을 구성하는 원호들에 대해 이진탐색 기법을 사용하면 $O(\log n)$ 시간에 쉽게 찾을 수 있다.

이제, 점 $q \in P \setminus \{p\}$에 대해, $q \in D_1 \cap \cdots \cap D_{\sigma(q)-1}$ 이지만 $q \notin D_1 \cap \cdots \cap D_{\sigma(q)}$가 성립하는 최소 인덱스 $\sigma(q)$를 찾아야 한다. 이를 위해, T의 루트 노드부터 시작하여 $D_{\sigma(q)}$가 저장되어 있는 리프 노드까지 하향식으로 탐색해간다. 우선, 루트 노드 v에 대해 $J(T_v)$에 q가 포함되는지 판단한다. 그러나, 루트 노드 v에서 $J(T_v) = D_1 \cap \cdots \cap D_n$으로 정의되고 $D_n = D(q_n, d(p, q_n)) = D(p, d(p, p)) = D(p, 0)$가 되어 $J(T_v) = \varnothing$이 된다. 그래서 루트 노드 v에

대해서는 항상 $q \notin \int(T_v)$이 된다. q가 $\int(T_v)$에 포함되지 않았기 때문에 v의 왼쪽 자식 노드 u와 오른쪽 자식 노드 w 중의 하나로 내려가서 탐색을 계속해야한다. v의 왼쪽 노드 u에 대해 $q \in \int(T_u)$인지 아닌지 판단한다. 만약 $q \notin \int(T_u)$이라면 $D_{\alpha(q)}$가 T_u에 존재하게 되어 u로 내려가 탐색을 똑같이 반복하면 된다. 그러나 $q \in \int(T_u)$라면 $D_{\alpha(q)}$는 오른쪽 부트리 T_w에 존재함을 의미하므로 오른쪽 자식 노드 w로 내려가 탐색을 해야 한다. 이젠, w로 내려와 탐색을 한다고 가정하자. w의 왼쪽 자식노드를 l, 오른쪽 자식노드를 r이라 하자. 우선, $q \in \int(T_l)$인 경우를 고려하자. 이미 앞에서 비교를 통해 $q \in \int(T_u)$임이 판명되었기 때문에 $q \in \int(T_u) \cap \int(T_l)$가 되어 $D_{\alpha(q)}$는 T_r에 있게 된다. 이제는 노드 r로 내려가 탐색을 반복한다. 만약, $q \notin \int(T_l)$이라면, 당연히 $D_{\alpha(q)}$는 T_l에 존재하므로 노드 l로 내려가 탐색을 계속하면 된다. 이렇게 루트에서부터 시작하여 한 번의 (q가 해당 노드에 저장된 교차영역의 내부에 있는지 외부에 있는지에 관한) 비교로 왼쪽과 오른쪽 자식 노드 중의 한 곳으로 내려갈 수 있기 때문에, 가장 마지막 리프 노드까지는 $O(\log n)$번의 비교가 필요하고 한 번의 비교는 $O(\log n)$시간이 필요하므로 전체적으로는 $O(\log^2 n)$시간이 소요된다. 우리가 원하는 수행시간을 위해선 $O(\log n)$ 시간을 더 줄여야 한다.

어떤 노드 v에서의 비교가 $O(\log n)$ 시간이 필요한 이유는 점 q를 지나는 수직선 h와 교차하는 $\partial\int^+(T_v)$ (또는 $\partial\int^-(T_v)$)의 원호 C를 이진 탐색으로 찾기 때문이다. 그러나 fractional cascading 방법[7]을 적용하면 루트 노드에서만 C를 찾는 데 $O(\log n)$ 시간이 필요하고 그 이후에는 한 레벨 내려갈 때마다 $O(1)$ 시간에 C를 찾을 수 있음이 알려져 있다. 이 방법을 간략히 설명하면 다음과 같다. (루트노드에서는 앞에서 설명한 이진탐색 방법을 이용하여 원호 C를 찾는다.) 임의의 노드 v에 대해서, v의 왼쪽과 오른쪽 자식 노드를 각각 u, w이라 하고, 수직선 h와 교차하는 $\partial\int^+(T_v)$의 원호 C를 이미 찾았다고 가정하자. 이제는 u 또는 w중 하나의 자식노드로 내려가서 h와 교차하는 새로운 원호 C'을 ($O(\log n)$ 시간이 아닌) $O(1)$ 시간에 찾아야 한다. 그런데, $\int^+(T_v) = \int^+(T_u) \cap \int^+(T_w)$이므로 $\partial\int^+(T_v)$을 구성하는 각 원호의 바로 위에 있는 $\partial\int^+(T_u)$와 $\partial\int^+(T_w)$의 원호들이 포인터에 의해 연결되어 있다면, 원호 C'은 원호 C에 연결된 포인터를 이용하여 $O(1)$ 시간에 쉽게 찾을 수 있다. 이렇게 부모노드와 자식노드 사이의 포인터 관계를 미리 구성해주는 기법을 fractional cascading이라 한다. 이러한 구성을 위해 부가적으로 소요되는 시간은 트리 T에 저장된 정보의 양에 비례한다. 이 정보의 양은 각 노드 v에 저장된 $\partial\int^+(T_v)$과 $\partial\int^-(T_v)$의 원호의 개수에 의해 결정되며, 모든 노드에 대해 이 개수를 합하면 $O(n\log n)$이므로, $O(n\log n)$ 시간이면 충분하다. 결국, 다음의 사실이 성립한다.

소정리 3. 한 점 $p \in P$를 고정하자. $P \setminus \{p\}$에 속하는 모든 점 q에 대해 $f_{p,q}$을 $O(n\log n)$ 시간에 계산할 수 있다.

정리 1. 이차원 평면에 주어진 n개의 점들로 구성된 집합 P에 대한 λ 이산 2-중심을 $O(n^2\log n)$ 시간, $O(n^2)$ 공간을 사용하여 계산할 수 있다.

(증명) P의 각 점 p에 대해, 소정리 3을 이용하여, $P \setminus \{p\}$에 속하는 모든 점 q에 대해, $f_{p,q}$을 $O(n\log n)$ 시간에 계산한다. 결국, P의 모든 쌍 p, q에 대해 $f_{p,q}$를 $O(n^2\log n)$ 시간에 계산할 수 있다. 이때 $O(n^2)$개의 $f_{p,q}$를 저장해야 하므로 $O(n^2)$ 공간이 필요하다. (QED)

3. 결 론

본 논문에서는 이차원 평면에 주어진 n개의 점들로 구성된 집합 P에 대한 λ 이산 2-중심을 $O(n^2\log n)$ 시간, $O(n^2)$ 공간을 사용하여 계산할 수 있음을 보였다. 앞으로의 가장 흥미로운 미해결 문제는 알고리즘의 시간복잡도 또는 공간복잡도를 선형(linear)에 가깝게 줄이는 것이다.

참 고 문 헌

[1] D. Hochbaum and D. B. Shmoys, "A best possible heuristic for the k-center problem", *Mathematics of Operations Research*, Vol 10:180--184, (1985).

[2] P. K. Agarwal, M. Sharir, and E. Welzl, The discrete 2-center problem, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 147--155, (1997).

[3] T. M. Chan. More planar two-center algorithms. *Comput. Geom. Theory Appl.*, 13:189-198, (1999).

[4] J.-M. Ho, D. T. Lee, C.-H. Chang, C. K. Wong: Minimum Diameter Spanning Trees and Related Problems. *SIAM J. Comput.* 20(5): 987-997 (1991).

[5] C.-S. Shin, S.-M. Park, Approrximation algorithms for a minimum-diameter spanning tree, submitted to *Journal of KISS*, (2001).

[6] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New

York, (1985).
[7] B. Chazelle and L. J. Guibas, Fractional cascading: I. A data structuring technique, *Algorithmica*, 1:133--162, (1986).

신 찬 수

1991년 서울대 계산통계학과 학사. 1993년 한국과학기술원 전산학 석사. 1998년 한국과학기술원 전산학 박사. 1999년 ~ 2000년 홍콩과기대 전산학과연구원. 2000년 ~ 2001년 한국과학기술원 전산학과 연구교수. 2001년 ~ 현재 한국외국어대학 전자정보학부 디지털 정보공학 조교수. 관심분야는 계산기하학, GIS 알고리즘, 컴퓨터 그래픽스, 그래프 드로잉