Towards Efficient Index Construction and Approximate Nearest Neighbor Search In High-Dimensional Spaces

연세대학교 컴퓨터과학과 안성현

2024년 12월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

Towards Efficient Index Construction and Approximate Nearest Neighbor Search In High-Dimensional Spaces

양현서

Table of Contents

1. Abstract & Introduction

- Nearest Neighbor Search Problem
  - 데이터셋 & 거리 함수가 주어졌을 때,
  - 주어진 쿼리 포인트와의 거리(distance)가 가장 짧은 데이터셋 포인트를 찾는 것이 목표
  - 한계: 고차원 대규모 데이터셋, 정확한 NN 검색은 계산 비용이 매우 크며, time-consuming

- 고차원 공간에서 Approximate Nearest Neighbor Search 대안으로 제시
  - LSH(Locality-Sensitive Hashing) Based Method
    - Locality-Sensitive 해시 함수 데이터의 유사성 보존
      - 고차원 공간의 데이터 포인트를 저차원 공간으로 매핑하고, 유사한 데이터가 동일한 해시 버킷으로 매핑되도록 설계됨
    - Hash-boundary 문제 때문에 높은 쿼리 품질에 도달하기 위해 비용이 많이 듦

  - Graph-Based Methods
    - Approximate Proximity Graph, APG를 이용하여 더 나은 쿼리 성능을 제공
    - LSH 기반 방법에 비해 쿼리 정확도가 더 높음
    - APG 구축 비용이 해시 기반 인덱스 생성 비용보다 1~2 order 정도 더 높음
    - 기본 데이터셋이 변화할 때 점진적으로 유지하는 데 한계가 있음

1. Abstract & Introduction

- **새로운 제안 방법: LSH-APG**
  - 경량화된 LSH 인덱스 프레임워크 사용하여 APG 구축, 효율적인 ANN 검색 쿼리 처리 지원
    - LSH(Locality-Sensitive Hashing) Based Method 단점: hash-boundary issues 완화
    - Graph-Based Methods 단점: 높은 그래프 구축 시간, 동적 데이터셋 유지 관리 어려움 완화

- LSH 인덱스로 APG 탐색 entry point 결정
  - LSH 인덱스를 사용해 초기 쿼리 결과 빠르게 검색하고 이를 APG 탐색의 entry point로 활용
  - 이후, 그래프 기반 기술을 사용해 쿼리 결과의 정확도를 더욱 향상시킴

- 쿼리 효율성 향상을 위한 가지치기(pruning) 전략
  - LSH 기반 가지치기 조건(pruning condition) 개발
  - 쿼리 포인트에서 멀리 떨어진 이웃을 필터링 → 그래프 탐색 중 접근해야 할 데이터 포인트 수를 줄여서 검색 공간 줄임

- APG 구축 과정에서 연속 삽입(consecutive insertion) 전략 활용
  - 모든 포인트를 순차적으로 APG에 삽입, 각 포인트를 쿼리 포인트로 간주하여 해당 포인트의 최근접 이웃을 그래프 인덱스에 삽입
  - LSH 프레임워크를 활용하여 검색 효율성을 개선하여 그래프 구축 비용을 낮춤
  - 정확성 보장(formal correctness)과 복잡성 분석(complexity analysis)도 가능하게 함

1. Abstract & Introduction

- **논문의 주요 기여**

- 최신 LSH 기반 및 그래프 기반 방법을 포괄적으로 분석하여 LSH-APG 새로운 솔루션 개발
  - LSH-APG는 낮은 구축 비용을 달성하지만, 쿼리 효율성 또는 쿼리 품질을 희생하지 않음

- LSH-APG의 효과를 입증하기 위한 비용 모델(cost model) 제공
  - LSH-APG의 예상 쿼리 비용이 데이터셋 Cardinality에 거의 독립적임
  - LSH 프레임워크의 효과 또한 이론적으로 증명됨

- 데이터베이스가 변화함에 따라, 인덱스 구조를 유지하기 위한 효율적인 업데이트 전략 설계
  - 유지보수 비용과 쿼리 비용(삽입/삭제)이 데이터셋 Cardinality에 크게 영향을 받지 않음

- 실험 결과, LSH-APG는 인덱싱 비용을 크게 줄이고, 기존 방법들과 비교했을 때 쿼리 효율성과 정확성 간의 최적의 균형을 달성함
  - 실제 데이터셋과 합성 데이터셋에서 실험한 결과, 기존의 그래프 기반 방법보다 훨씬 낮은 구축 비용, 더 나은 쿼리 성능을 달성

2. Related Work

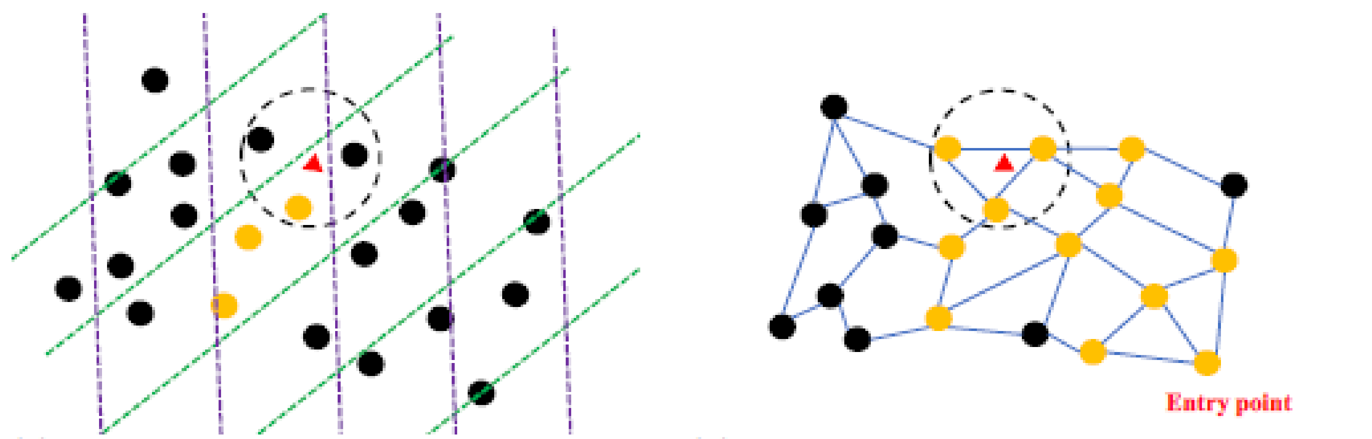
2.1 LSH-based Methods

- LSH 함수를 사용하여 고차원 공간의 데이터 포인트를 여러 저차원 hash bucket으로 매핑
- 쿼리가 속한 버킷을 확인하여 ANN(근사 최근접 이웃) 쿼리에 답 제공함

- 쿼리 결과 정확도에 대한 견고한 이론적 보장을 제공, 구현 단순하고 효율적임

- 한계: 높은 쿼리 정확도, 서브-선형(sub-linear) 쿼리 비용 보장하려면 서로 다른 버킷 너비를 가진 여러 LSH 인덱스 준비 필요 → 비효율적으로 큰 인덱스 크기
- hash boundary 문제 때문에 높은 쿼리 품질을 달성하기 어려움
    - 두 데이터 포인트가 실제로는 가까운 거리에 위치해 있음에도 불구하고, 해시 함수의 특성상 다른 해시 버킷으로 매핑되는 경우

- 동적 LSH(dynamic LSH) 방법 설계
    - 각 쿼리 포인트에 대해 쿼리 중심의 해시 버킷을 동적으로 생성
    - collision counting-based strategy, metric-based strategy 새로운 LSH 프레임워크
    - 동적 버킷화(dynamic bucketing)의 오버헤드(overhead)
    - 동적 쿼리 전략과 정적 LSH 프레임워크를 결합한 DB-LSH 제안



(a) Search in the LSH-Based Method (b) Search in the Graph-Based Method

Figure 1: Illustration of LSH-based and graph-based methods. The red triangle is the query point q. The three points in the black dashed circle are the 3NN of q. The orange points denote those that are accessed during the search.

2. Related Work

2.2 Graph-based Methods

- Approximate Proximity Graph(APG)을 활용해 ANN 검색 지원
  - 벡터 데이터셋에서 최근접 이웃(Nearest Neighbor)을 효율적으로 검색하기 위해 사용되는 그래프 기반 데이터 구조
  - 정확성과 효율성 측면에서 다른 방법들보다 더 나은 쿼리 성능을 보임

- 한계: APG의 구축 비용이 O(n^2), 대규모 데이터셋에 대해 사용하기 어려움

- 구축 비용 감소 시도, 약간의 쿼리 품질을 희생하여 APG를 구축하는 여러 인덱싱 전략이 개발됨
  - NN-Descent: APG 구축 복잡도를 O(n~1.14)로 낮춤
  - 하지만 high-quality 이웃을 찾기 위해 약 10번의 반복(iteration) 필요 → 여전히 시간 많이 소요됨

- NSW (Navigable Small World): Consecutive Insertion Strategy 사용해서 Approximate KNN Graph와 Delaunay Graph(DG) 구축
  - hubness 문제: 특정 정점들의 out-degree가 높아 쿼리 성능을 저하시킬 수 있는 현상
    - HNSW (Hierarchical Navigable Small World): NSW와 동일한 연속 삽입 전략을 사용하지만, 각 포인트의 최대 degree를 제한하여 hubness 문제를 완화
    - HCNNG (Hierarchical Clustering and NN Graph), VRLSH: 데이터를 여러 subgroup으로 clustering하거나 분할하고, 각 하위 그룹에서 APG를 구축
    - 그래프 인덱스의 구축 비용을 O(n)로 줄일 수 있음
    - 단점: 좋은 성능을 달성하려면 그래프를 여러 번 구축해야 함

2. Related Work

2.3 Other Methods

- **Tree-based Methods**
  - M-Tree, R-Tree, KD-Tree 및 그 변형들
  - pivot 또는 hyperplane을 사용하여 공간을 여러 하위 공간으로 분할, 중복되는 하위 공간(overlapping subspaces)만을 고려 -> 검색 공간을 줄임
  - 한계: 데이터의 차원이 증가할수록 분할 전략의 효과가 감소, 고차원 공간에서 트리 기반 방법은 효율적인 ANN 검색에 적합하지 않음

- **Quantization-based Methods**
  - VA-file, Product Quantization(PQ)
  - 데이터를 quantize하고, quantization value에 따라 클러스터링, 쿼리 포인트와 동일한 양자화 값을 가진 후보 포인트 검색
  - 한계: quantization errors로, 특히 고차원 공간에서 높은 쿼리 정확도를 달성하기 어려움

- Tree-based Methods: 다차원 공간에서 exact nearest neighbors을 찾는데 적합
- Quantization-based Methods: well-clustered datasets에서 좋은 성능을 보임
- 하지만 고차원 공간에서는, LSH 기반 방법과 그래프 기반 방법이 모든 데이터셋에 대해 높은 쿼리 품질을 보장할 수 있음

3. Preliminaries

3.1 Problem Definition

- 유클리드 공간에서 c-ANN 및 (c, k)-ANN 쿼리를 연구
- Definition 1 : (c, k)-ANN 쿼리
  - 주어진 조건: q(쿼리 포인트), c>1(approximation ratio, 근사 비율), 양의 정수 k
  - 쿼리가 반환하는 근사 최근접 이웃의 거리는 정확한 최근접 이웃의 거리의 c배를 초과하지 않음
  - (c, k)-ANN 쿼리: k개의 포인트 $o1, \dots, ok$를 반환
    - $o1, \dots, ok$: q와의 거리에 따라 오름차순으로 정렬됨, 근사 최근접 이웃
    - $\| q, oi \| \leq c \cdot \| q, oi \|$

Table 1: List of Key Notations.

Notation	Description
$\mathbb{R}^d$	d-dimensional Euclidean space
$\mathcal{D}$	The dataset
n	The cardinality of dataset
LID	The local intrinsic dimensionality of dataset
o, v, u	A data point
q	A query point
$\|o_1, o_2\|$	The distance between o_1 and o_2
$e = (o_1, o_2)$	The directed edge from o_1 to o_2
$h^*(o), h(o)$	Hash function
$\chi^2(m)$	The χ^2 distribution with freedom m
C_Q	The expected number of points accessed per query

- Remark 1. c-ANN 쿼리는 (c,k)−ANN 쿼리에서 k=1인 경우, 가장 가까운 포인트 하나를 반환함
- Remark 2. 그래프 기반 방법에서는 일반적으로 c를 명시적으로 사용하여 쿼리 품질을 제어하지 않음, 혼동을 피하기 위해 (c,k)−ANN을 간단히 kANN으로 줄여 표현함

3. Preliminaries

3.2 Locality Sensitive Hashing, LSH

- Definition 2: Locality Sensitive Hashing, LSH
  - 가까운 점들은 높은 확률로 같은 해시 값에 배치, 먼 점들은 낮은 확률로 같은 해시 값에 배치

- 유클리드 공간에서 일반적인 LSH 정의

$$h^*(o) = \vec{a} \cdot \vec{o}, \qquad (1)$$

- 유클리드 공간에서 자주 사용되는 또 다른 LSH 패밀리 정의

$$h(o) = \left\lfloor \frac{h^*(o) + b}{w} \right\rfloor, \qquad (2)$$

Definition 2 (Locality Sensitive Hashing (LSH) [42, 47]). *Given a distance r and an approximation ratio $c > 1$, a family of hash functions $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{R}\}$ is called (r, cr, p_1, p_2)-locality-sensitive, if for $\forall o_1, o_2 \in \mathbb{R}^d$, it satisfies both conditions below:*

(1) If $\|o_1, o_2\| \le r$, $\Pr[h(o_1) = h(o_2)] \ge p_1$;
(2) If $\|o_1, o_2\| > cr$, $\Pr[h(o_1) = h(o_2)] \le p_2$,

where $h \in \mathcal{H}$ is chosen at random, p_1, p_2 are collision probabilities and $p_1 > p_2$.

3. Preliminaries

3.3 Graph-Based Methods

- 그래프 기반 방법의 기본 구조: 근접 그래프(Proximity Graph), G=(V,E)
  - V: D(데이터셋)에 있는 모든 데이터 포인트를 나타내는 Vertex Set
  - E: 원래 공간에서 충분히 가까운 포인트들 간 모든 Edge를 포함하는 Edge 집합

- **Cluster & Merge (클러스터링 및 병합)**
- **Iteration (반복 갱신)**
- **Consecutive Insertion (연속 삽입)**

- Edge의 수는 쿼리 성능에 직접적으로 영향을 미침
  - vertex의 out-edge 수가 많을수록 많은 후보군과 계산이 수행되고, query 처리 속도가 느려짐

  - 이웃 선택 전략: edge 수와 분포를 제어하기 위함
  - Simple Selection Strategy
    - 가장 가까운 M개의 이웃 선택 (M은 미리 정의된 임계값)

  - Heuristic Selection Strategy
    - HNSW, NSG에서 o가 edge (o,v), (o,u)를 가지는 경우
      - $\|o,u\| < \|u,v\| < \|o,v\|$일 때, edge (o,v)는 (o,u)와 충돌한다고 간주
      - (o,u)와 (o,v)가 너무 유사하면, 둘 다 저장할 필요가 없음, 더 긴 edge (o,v)가 삭제됨

3. Preliminaries

3.4 기존 ANN 방법의 한계

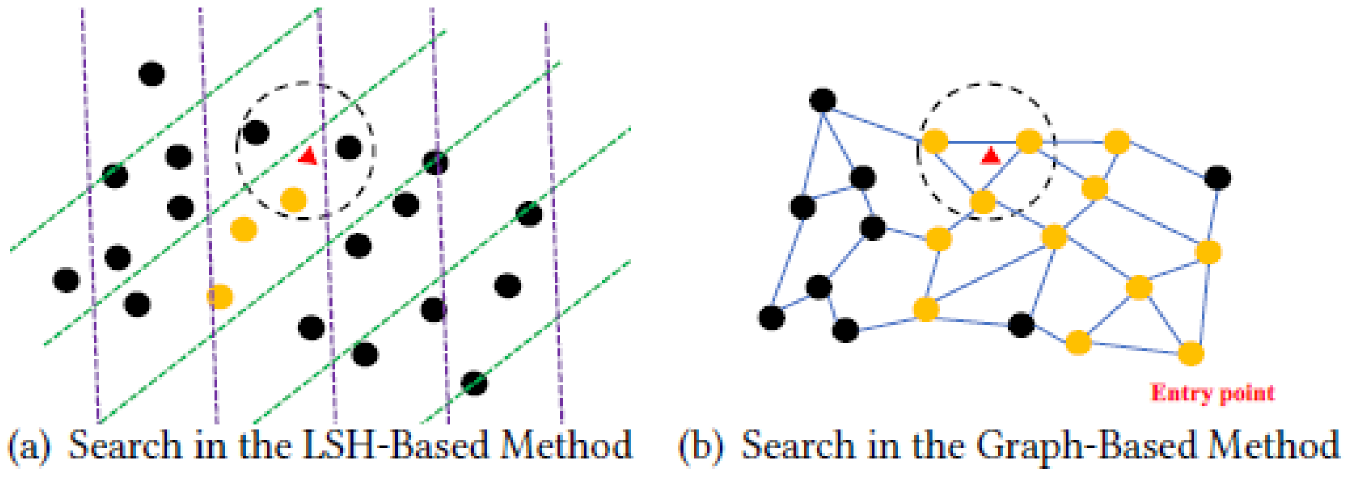


Figure 1: Illustration of LSH-based and graph-based methods. The red triangle is the query point q. The three points in the black dashed circle are the 3NN of q. The orange points denote those that are accessed during the search.

- LSH Based Method
  - 데이터 포인트가 여러 해시 버킷(hash bucket)에 매핑됨
  - 2차원의 투영된 공간에서 hash table, B+-Tree, R-Tree와 같은 간단한 구조로 인덱싱됨
  - LSH 인덱스는 동적 데이터셋에 대해 유지 관리하기가 용이함

- ANN 쿼리 처리
  - 쿼리 포인트가 속한 해시 버킷의 점들만 검사함 (Figure 1(a)의 3개 주황색 점들)
  - Equations 1과 2의 LSH 계열: 점들 간 거리에 따라 collision probability이 monotonically decreasing함을 보장함
  - LSH 기반 방법은 쿼리 품질에 대한 보장을 제공

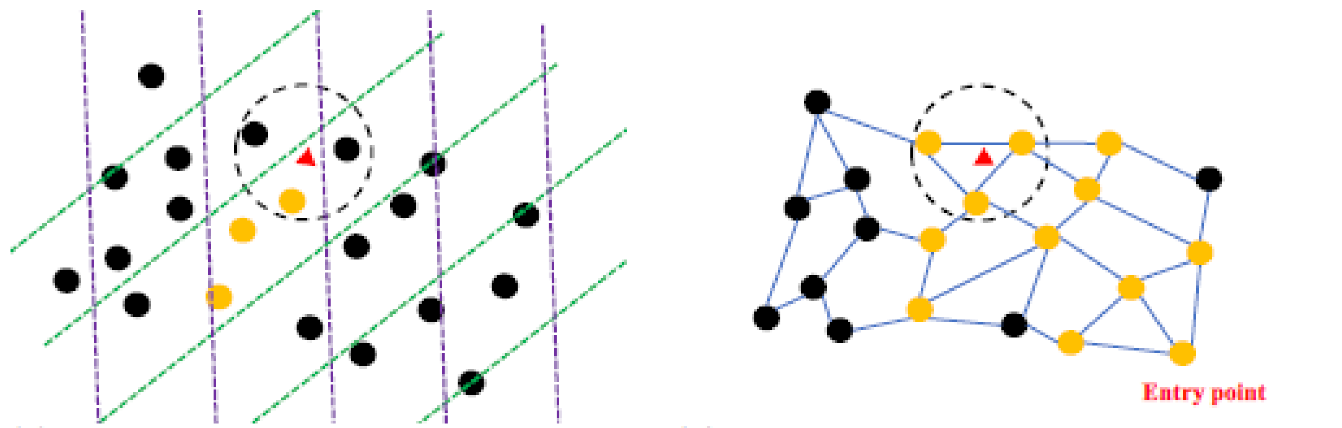
- 한계
  - 단순한 query strategy와 hash boundary 문제로 high recall을 달성하기 어려움
  - Figure 1(a), 예시에서 3개 중 2개의 이웃이 LSH index에 의해 검색되지 않았음
  - 누락된 이웃을 찾기 위해서는 더 많은 LSH index 구축 필요 → 더 높은 쿼리 비용이 발생

3. Preliminaries

3.4 기존 ANN 방법의 한계

- Graph Based Method
  - 각 정점(vertex)은 2~4개의 이웃을 가짐

- ANN 쿼리 처리
  - 임의의 entry point(아래 오른쪽 점)에서 시작, APG에서 greedy search을 통해 올바른 결과에 접근
  - 3ANN(3개의 근사 최근접 이웃)을 검색하는 동안 접근한 점들이 주황색으로 표시됨
  - 이론적인 보장은 제공하지 않지만, 항상 LSH 기반 방법보다 성능 우수



(a) Search in the LSH-Based Method (b) Search in the Graph-Based Method

Figure 1: Illustration of LSH-based and graph-based methods. The red triangle is the query point q. The three points in the black dashed circle are the 3NN of q. The orange points denote those that are accessed during the search.

- 한계: 막대한 구축 비용(huge construction cost)에서 발생
- Cluster & Merge strategy: 각각의 cluster가 분리되어 있고, subgraph 품질이 낮아서 작업을 여러 번 반복해야 함 → time-consuming

- Iteration strategy: 모든 vertex의 이웃을 매 iteration마다 업데이트 해야 함

- Consecutive Insertion: Construction cost와 APG의 품질이 쿼리 전략에 크게 의존함
  - Heuristic Selection Strategy: 모든 이웃 쌍을 비교해야 해서 계산 비용 크게 증가
  - Simple Selecting Strategy: 더 효율적이지만, 데이터가 밀집된 영역에서는 유사한 edge가 선택될 가능성 높아 쿼리 처리 시 불필요한 계산 비용 초래

4. LSH-APG Framework

- ANN(근사 최근접 이웃) 검색에서 쿼리 성능을 희생하지 않으면서 구축 비용을 줄일 수 있는 새로운 프레임워크인 LSH-APG 제안

- 주요 아이디어
  - Consecutive insertion과 Simple selection strategy을 통해 APG(Approximate Proximity Graph) 빠르게 구축
  - 이를 기반으로 LSH framework를 추가로 활용하여 APG 구축 및 쿼리 처리를 가속화
    - LSH framework: 더 가까운 entry point를 제공하고 관련 없는 edge를 필터링
    - Fig2. LSH-APG는 쿼리 포인트 q(빨간 삼각형)가 속한 버킷에서 가장 가까운 점(파란색 점)을 entry point로 선택하여 탐색 단계(hops) 수를 2로 줄임

  - LSH 기반 pruning condition(가지치기 조건)을 채택해 q와의 거리가 지나치게 먼 점은 탐색 경로에서 제외함

- **Low construction cost**
  - 간선(edge)의 분포를 고려하지 않기 때문에 거리 계산 비용이 크게 감소함

- **Guaranteed query cost and quality**
  - LSH-APG는 가벼운 LSH 인덱스들을 사용해 그래프에서 쿼리에 더 적합한 진입점(entry point)을 빠르게 찾음

- **Accurate pruning condition**
  - LSH 기반 가지치기 조건(pruning condition)을 사용하여 검색 중 일부 간선들을 필터링함

- **Incremental index maintenance**
  - LSH-APG는 데이터가 변화함에 따라 저비용으로 점진적 인덱스 유지 관리를 지원

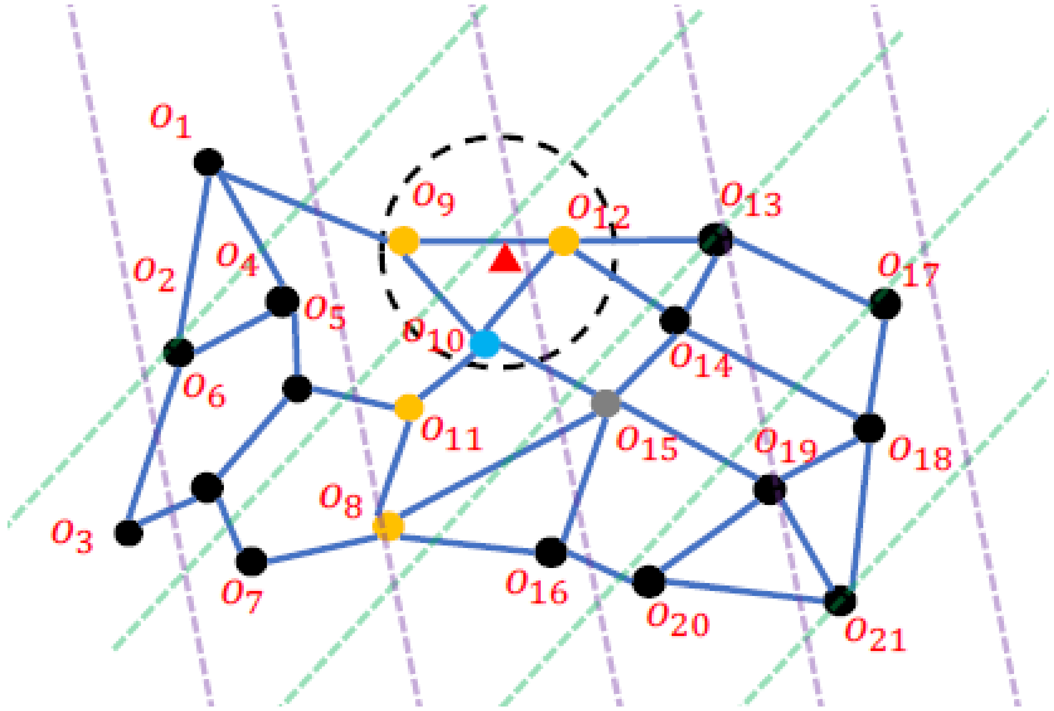


Figure 2: Search in LSH-APG

4. LSH-APG Framework

4.1 Naive-APG: the Basic Structure

- Naive-APG의 인덱스, IG=(V,E)는 directed NN graph

- 기존 NN 그래프: vertex가 고정된 수의 edge로 연결됨
- APG: 데이터 분포에 더 적합하도록 각 정점의 간선 수(degree)를 [T,T´]범위 내에서 가변적으로 설정
  - 데이터가 밀집된 영역에서는 희소한 영역에 있는 데이터 포인트보다 더 많은 간선이 필요
  - 각 데이터 포인트가 주변 데이터 포인트들과 구별하기 어려워지기 때문

- Consecutive insertion strategy(연속 삽입 전략)을 통해 I_G를 구축 (Algorithm 1)

- $T' = T$일 때, 대부분의 NN 그래프에서와 마찬가지로 I_G 의 모든 정점은 동일한 수의 간선을 가지게 됨
- 성능이 낮음
  - 실제 데이터셋은 보통 불균등하게 분포, 이 설정은 이러한 차이를 전혀 고려하지 않음
  - 먼저 삽입된 점들은 이후에 포인트들을 삽입 할 때 자주 접근됨, 이 점들에 더 많은 간선을 제공하면 이후 삽입되는 점들이 더 나은 ANN 결과를 찾는데 도움이 됨
  - T' 값을 지나치게 크게 설정하면 검색 중 계산해야 할 거리 수가 증가하여 높은 쿼리 비용이 발생
  - 기본값 $T' = 2T$로 설정

Algorithm 1: Building Naive-APG($\mathcal{D}, T, T'$)

Input: Dataset $\mathcal{D}$ and parameter T, T'
Output: $\mathcal{I}_G$

$\mathcal{I}_G \leftarrow \emptyset$;
for *each point* $o \in \mathcal{D}$ **do**
  $candidates \leftarrow T$ ANN results of o found in $\mathcal{I}_G$;
  for *each* $e \in candidates$ **do**
    $\mathcal{I}_G \leftarrow \mathcal{I}_G \cup \{(o, e), (e, o)\}$;
    if $e.degree > T'$ **then**
      $o_f \leftarrow e$'s furthest neighbor in $\mathcal{I}_G$;
      $\mathcal{I}_G \leftarrow \mathcal{I}_G - \{(e, o_f)\}$;
return $\mathcal{I}_G$;

4. LSH-APG Framework

4.2 LSH-APG: Optimized by LSH Indexes

- Algorithm 1
  - o의 ANN 결과를 찾는데 필요한 거리 계산만 수행, 낮은 비용으로 APG를 구축, 낮은 query 비용
  - 그러나 그래프 품질이 보장되지 않음
  - o에 대해 충분히 좋은 ANN 결과를 찾지 못하면 -> o의 edge 품질이 제한됨

- 그래프에서 쿼리에 대한 더 나은 entry point을 빠르게 찾기 위해 가벼운 LSH 인덱스 한 세트를 채택하여 쿼리 품질을 보장
  - 해시 인덱스 I_H: 빠르게 구축되어야 함, NN 쿼리에 효율적으로 응답 가능해야 함
  - 최종 ANN 결과는 그래프 인덱스에서 세부적으로 조정되어, 쿼리 품질이 매우 높을 필요는 없음

- Quality and efficiency in high dimensional nearest neighbor search, SIGMOD, 2009 논문의 아이디어 가져옴

1. K개의 LSH 함수 $h_1 \dots h_k$ 무작위 선택
2. o의 K개의 해시 값 $h_1(o) \dots h_k(o)$ 를 계산
3. $H(O)$를 Zorder curve를 통해 1차원 값 $z(H(o))$로 변환, 이를 B+-Tree 같은 정렬된 index로 저장
4. 이 과정을 L번 반복하여 L개의 B+-Tree를 생성, 이들로 I_H 를 구성

- Algorithm 2: I_H 와 I_G 를 동시에 구축하는 방법 설명

Algorithm 2: Building LSH-APG($\mathcal{D}, T, T'$)

Input: Dataset $\mathcal{D}$ and parameter T, T'
Output: LSH-APG index $\mathcal{I}_G$ and $\mathcal{I}_H$

```
I_H ← ∅, I_G ← ∅;
for each point o ∈ D do
    candidates ← call
     kANN-Query(o, I_G, I_H, p_τ = 0.95, T);
    for each e ∈ candidates do
        I_G ← I_G ∪ {(o, e), (e, o)};
        if e.degree > T' then
            o_f ← e's furthest neighbor in I_G;
            I_G ← I_G − {(e, o_f)};
    Insert o into the corresponding LSB-Tree in the I_H;
return I_H and I_G;
```

4. LSH-APG Framework

4.3 ANN Query in LSH-APG

- Algorithm 3: LSH-APG의 ANN query processing 알고리즘

- 두 가지 주요 단계 포함
  - LSH 인덱스 I_H 를 사용하여 entry points 찾기 (Line 1-4)
    - 해시 함수들을 사용하여 해당 버킷(hash bucket)을 결정
    - I_H에서 q의 kANN 검색 실행
  - 그래프 인덱스 I_G 를 사용하여 쿼리 품질 향상 (Line 5-21)
    - EPs 점들을 I_G에서 kANN 쿼리 진입점으로 사용
    - k번째 최근접 이웃 R_K
    - 쿼리 효율성 향상, LSH 기반의 pruning condition

- LSH framework를 통해 entry points 생성 -> 초기 탐색 반경을 크게 줄임
- 알고리즘 종료에 필요한 hop 수 줆 -> 쿼리 효율성을 향상
- 더 가까운 entry point는 쿼리가 탐색 반경이 큰 local minimal에서 종료될 확률을 줄임

Algorithm 3: kANN Query($q, \mathcal{I}_G, \mathcal{I}_H, p_\tau, k$)

Input: A query point q, LSH-APG index $\mathcal{I}_G$ and $\mathcal{I}_H$, p_τ, k
Output: k nearest points to q

1 Compute q's projected values $h_1^*(q), \ldots, h_{L\times K}^*(q)$;
2 $EPs \leftarrow$ the set of k approximate nearest points to q in $\mathcal{I}_H$;
3 $V \leftarrow$ the set of visited points during the above search;
4 $R \leftarrow EPs$; //the result set of k best results found so far
5 $m \leftarrow K$, $P(q) \leftarrow (h_1^*(q), \ldots, h_m^*(q))$;
6 $t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}$;
7 **while** $|EPs| > 0$ **do**
8     $e_p \leftarrow$ pop the nearest element in EPs to q;
9     $R_k \leftarrow$ the furthest points in R to q;
10   **if** $\|e_p, q\| > \|q, R_k\|$ **then**
11   **break**;
12   **for** *each* $o \in N(e_p)$ **do**
13   **if** $o \notin V$ **then**
14             $V \leftarrow V \cup \{o\}$;
15   **if** $\|P(q), P(o)\| < t \cdot \|q, R_k\|$ **then**
16   Compute $\|q, o\|$;
17   Insert o into EPs;
18   Insert o into R;
19   **if** $|R| > k$ **then**
20   Remove the furthest point to q in R;
21 **return** R;

4. LSH-APG Framework

4.4 Cost Model of LSH-APG

- LSH-APG의 성능을 입증하기 위해, 쿼리 비용 및 쿼리 품질을 분석하기 위한 비용 모델을 설계
  - 쿼리 비용 및 쿼리 품질이 데이터의 cardinality에 거의 영향을 받지 않음을 입증 (Theorem 2)
  - 쿼리가 종료될 때 반환된 점과 쿼리 포인트 간의 거리가 충분히 작음을 증명
  - LSH 인덱스가 쿼리 비용에 미치는 이점 계산 (Lemma 2)

- 쿼리 비용과 품질 분석
  - 쿼리 비용(CQ)은 주로 그래프에서 검색 중 발생하는 계산 비용에서 비롯됨
  - 쿼리 품질, 쿼리가 종료될 때의 검색 반경(search radius) s가 제한됨을 증명
  - 정리 1. LSH-APG의 쿼리 비용과 쿼리 품질은 data cardinality n과 독립적
  - 정리 2. ep의 예상 간선 길이가 ro이고 이웃들이 ep 주위에 균일하게 분포한다고 가정할 때, 최종 검색 반경 s는 $(1+\gamma)ro$로 제한될 것으로 예상됨, (γ<1)
    - 검색 반경($s(r)$)과 쿼리 종료 조건이 이웃의 분포와 평균 거리의 함수로 효과적으로 제어 가능

$$C_Q = lT_e$$

- p(r): 쿼리가 en에서 종료될 확률, *δ(r) = r-r'*: hop length일 때, l(r), s(r)은 다음 방정식 만족함

$$\begin{aligned} l(r) &= p(r)\cdot 1 + [1-p(r)][l(r-\delta(r))+1], \\ s(r) &= p(r)\cdot r + [1-p(r)]s(r-\delta(r)). \end{aligned} \quad (3)$$

$$p(r) = \Pr[r' > r] = \Pr[\delta(r) > 0]$$

- I_H에서 얻는 이점: 더 가까운 entry point oh를 제공하여 초기 검색 반경을 줄임

$$B_{I_H} = l(r_2) - l(r_1)$$

- 공간 복잡도: O(n)
- 구축 시간 복잡도: O(ndC_Q) (C_Q 가 n에 독립적)
- LSH-APG 쿼리 비용: O(dC_Q))

5. LSH-Based Pruning Condition

- 그래프 기반 방법에서 쿼리 비용은 q와 ep의 이웃 간 거리 계산 횟수에서 발생
  - 기존의 쿼리 전략은 ep의 모든 이웃을 검사 → 불필요, 시간 소모적
- 멀리 있을 가능성이 있는 일부 이웃을 필터링
- o를 삭제하고 $\| q, o \|$의 거리를 계산하지 않는 게 합리적 (식 (4))
- Lemma 4. LSH 기반 검색에서 투영된 거리 $\| P(q), P(o) \|$와 실제 거리 $\| q, o \|$의 관계를 확률적으로 보장
- Lemma 5. LSH-based pruning condition 적용, $\| q, o \|$가 증가할수록 포인트 o가 필터링될 확률이 높아짐
- 멀리 있는 점들을 사전에 필터링하면 불필요한 거리 계산을 줄일 수 있어 쿼리 수행 비용이 감소하여 전체 검색 과정이 더 효율적이게 됨
  - 가지치기 조건은 $p\tau$ 값을 기반으로 작동, $p\tau$ 값은 특정 점이 필터링되지 않을 확률 결정

- $P(o)$: Lemma 1에서 정의된 m-차원으로 투영된 벡터
- $\chi^2_{p_\tau}(m)$: $\chi^2(m)$ 분포에서 p_τ에 해당하는 분위값(quantile)
- d_k: 현재 발견된 k-번째 가장 가까운 NN(nearest neighbor) 결과

$$\|P(q), P(o)\| < \sqrt{\chi^2_{p_\tau}(m)} \cdot d_k, \tag{4}$$

LEMMA 4 (LEMMA 4 IN [47]). *Given a query q, an approximation ratio c and parameter t, we define the following two events:*

- **E1:** *For a point o that $\|q, o\| \leq r$, its projected distance to q, $\|P(q), P(o)\|$, is smaller than tr.*
- **E2:** *There are fewer than βn ($\beta > \alpha_2$) points whose distances to q exceed cr but projected distances to q are smaller than tr.*

LEMMA 5. *With the LSH-based pruning condition, the probability that a point o is filtered increases with $\|q, o\|$. Assume $p_\tau = \frac{1}{2}$ and the current search radius is r, for any $c > 1$, the point whose distance to q is less than r will not be filtered with at least the probability of $\frac{1}{2}$ and we access at most $O(n^\alpha)$ points whose distance to q is greater than cr, where $\alpha = 1 - \frac{9\kappa(c^{-2/3}-1)^2}{4}$ and $\kappa = \frac{m}{\log n}$.*

5. LSH-Based Pruning Condition

- LSH 기반 가지치기 조건을 사용하는 LSH-APG에서 k=2인 kANN 쿼리 실행

- Naïve-APG
  - 무작위 entry point o21을 사용하여 검색
  - o21, o19, o15, o10, o12, o9를 순서대로 접근
  - 계산비용 12

- LSH-APG
  - 쿼리 포인트 q(빨간 삼각형)와 충돌하는 점으로 o8, o10, o11을 찾음
  - o10에 먼저 접근, o10의 이웃인 o9, o12, o15를 Eps에 추가
  - o9에 접근, o9는 q에서 2번 째로 가까운 점(R_2=o10)보다 더 멀리 떨어져 있으므로, 쿼리 o9에서 종료
  - o12에 접근 o13, o14
  - 계산비용 8

- LSH-APG + Pruning condition
  - 원래는 LSH-APG에서 o8, o10, o11 찾고, o10, o12, o9을 순서대로 모두 접근해야 함
  - o10에 접근할 때, o15 필터링 되고, o9, o12만 EPs에 추가
  - o12에 접근할 때, o13, o14 필터링 됨
  - o9에 접근하고 쿼리 종료
  - 계산 비용 5

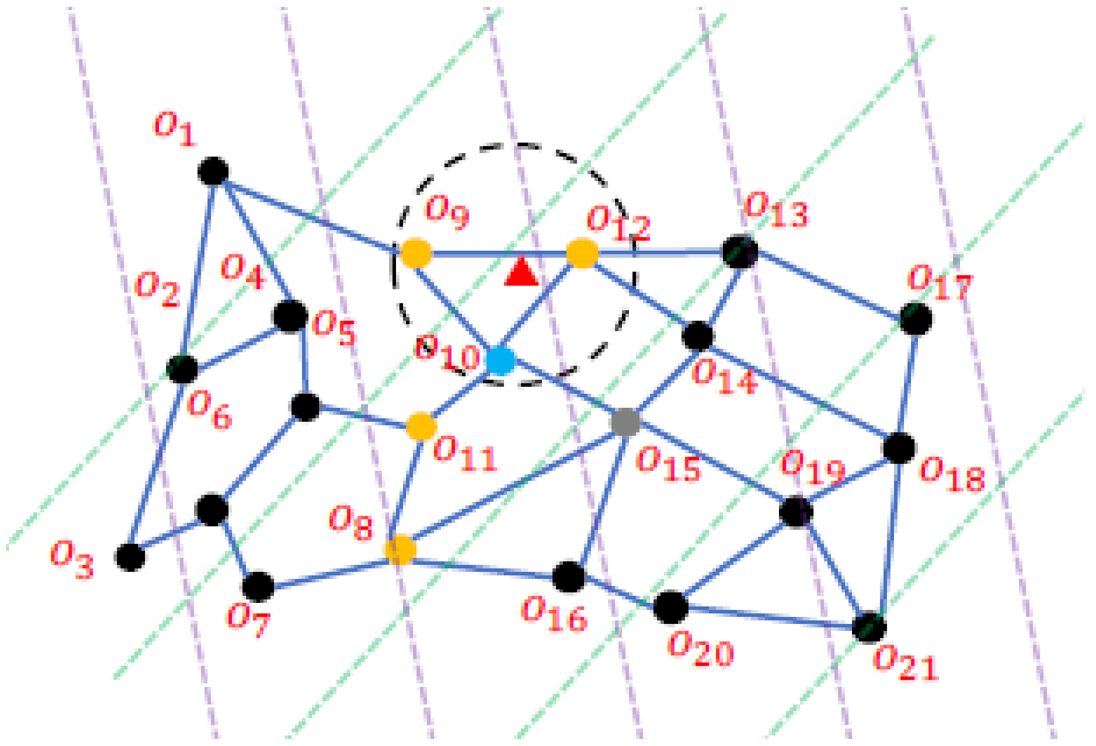


Figure 2: Search in LSH-APG

6 Index Maintenance

- LSH-APG는 데이터가 변할 때, 관련된 점들에 대한 간선을 재구성해야 함

- 삽입 (Insertion)
  - LSH-APG는 순차 삽입 전략(Consecutive Insertion Strategy)을 기반으로 구축
  - 새로운 점을 삽입하는 작업은 간단하고 자연스럽게 수행됨

- 삭제 (Deletion)
  - I_G에서 o의 모든 out-edges와 in-edges를 제거하고 o를 I_H 에서 삭제해야 함
  - I_H 에서 삭제하는 것은 간단하지만 I_G 에서 in-edges를 삭제하는 것은 그래프에 기록되어 있지 않음

  1. o와 o의 모든 out-edges를 Deleting 상태로 표시 (Line 4)
  2. LSH-APG에서 o의 가장 가까운 이웃을 기준으로, 검색 반경 dm 내에서 Range Search를 수행 (Line 9-15)
  3. 점 u를 찾으면, u가 RN(o)에 있는지 확인 (Line 16-24)
     - u∈RN(o)이면, (u, o) 간선을 제거, o의 in-degree를 1 감소시킴
     - u의 차수가 T 미만이면, u의 이웃의 이웃(neighbors of neighbors)에서 추가 이웃 찾아 T′까지 이웃의 수 늘림

- Range Search의 비용을 제어하기 위해 최대 검색 비용C_{DM}을 설정

Algorithm 4: Delete-Point $(o, \mathcal{I}_G, \mathcal{I}_H, C_{Dm})$

Input: A point to be deleted o, $\mathcal{I}_G$, $\mathcal{I}_H$, C_{Dm}
Output: The updated indexes $\mathcal{I}_G$ and $\mathcal{I}_H$

1 $RN(o) \leftarrow \{v|(v,o) \in E\}$;
2 $d_m \leftarrow \max_{v \in RN(o)} \|o, v\|$;
3 Delete o from $\mathcal{I}_H$;
4 Mark o and all the out-edges of o as the Deleting status;
5 $EPs \leftarrow \{v|(o,v) \in E\}$;
6 $V \leftarrow \varnothing$ stores the set of visited points;
7 $m \leftarrow K, t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}, cnt \leftarrow 0$;
8 $t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}$;
9 $d_k \leftarrow$ the current k-th **while** $|EPs| > 0 \&\& cnt < C_{Dm}$ **do**
10 　$cnt \leftarrow cnt + 1$;
11 　$e_p \leftarrow$ pop the nearest element in EPs to q;
12 　**for each** $u \in N(e_p)$ **do**
13 　　**if** $u \notin V$ **then**
14 　　　$V \leftarrow V \cup \{u\}$;
15 　　　**call** Access(u);

16 **Function** Access(u) **is**
17 　**if** $\|P(q), P(u)\| < t \cdot d_m$ **then**
18 　　Compute $\|q, u\|$;
19 　　Insert u into EPs;
20 　　**if** $u \in RN(o)$ **then**
21 　　　Remove the edge (u, o) from $\mathcal{I}_G$;
22 　　　**if** $|N(u)| < T$ **then**
23 　　　　$N(u) \leftarrow N(u) \cup \{y|y \in N(N(u))\}$;
24 　　　　$N(u) \leftarrow$ The T' closest points in $N(u)$ to u;

7. Experimental Study

7.1 Experimental Settings

- **Dataset**
  - 6개의 실제 데이터셋
  - 2개의 합성 데이터셋: Rand10M, Gauss10M
- **비교한 알고리즘**
  - LSH based Method
    - DB-LSH: 현재까지 쿼리 복잡도가 가장 낮은 것으로 입증된 방법
  - Graph based Method
    - HNSW, HCNNG, NSG
- **Evaluation Metric**
  - Index size (IS)
  - The normalized maximum common subgraph (NMCS): the similarity between a graph index G for the ANN query and the exact NN graph
  - Maximum common subgraph (MCS): similarity of two graphs
  - Indexing Time (IT), Query time (QT)

Table 2: Summary of Datasets

Datasets	Cardinality	Dim.	LID	Size (GB)
MNIST*	60,000	784	12.7	0.184
Deep1M*	1,000,000	256	26.0	1.00
Gauss10M†	10,000,000	32	26.3	1.19
Rand10M†	10,000,000	32	23.9	1.19
Gist1M*	1,000,000	960	36.2	3.58
SIFT10M*	10,000,000	128	22.0	4.77
SIFT100M*	100,000,000	128	23.7	47.7
Tiny80M*	79,302,017	384	44.6	113

*Real-world Datasets; †Synthetic Datasets

7. Experimental Study

- **인덱스 크기 (Index Size)**
  - LSH-APG가 그래프 기반 알고리즘 중 가장 큰 인덱스 크기를 가지지만 DB-LSH보다는 작음
  - HNSW: Gist1M과 Tiny80M 데이터셋에서 매우 작은 인덱스 크기를 보임
    - 휴리스틱 neighbor selection 전략 사용
    - 단점:쿼리 성능(query performance)에 부정적인 영향을 미칠 수 있음
- **NMCS (Normalized Mean Cosine Similarity)**
  - LSH-APG: 그래프 기반 알고리즘 중 가장 높은 NMCS를 기록
  - 쿼리 품질(query quality)에 유리
- **IT (Indexing Time)**
  - DB-LSH: 모든 알고리즘 중 가장 낮은 IT를 기록.
  - 이유: 적은 수의 해시 함수만 계산하므로 간선(edge)을 찾는 시간보다 훨씬 빠름.
  - LSH-APG: 그래프 기반 알고리즘 중 가장 낮은 IT 기록, HNSW와 유사
- **Data Cardinality n의 영향**
  - 모든 알고리즘의 쿼리 시간(QT)은 증가하고 재현율(recall)은 감소
  - LSH-APG의 QT 증가는 상대적으로 작음

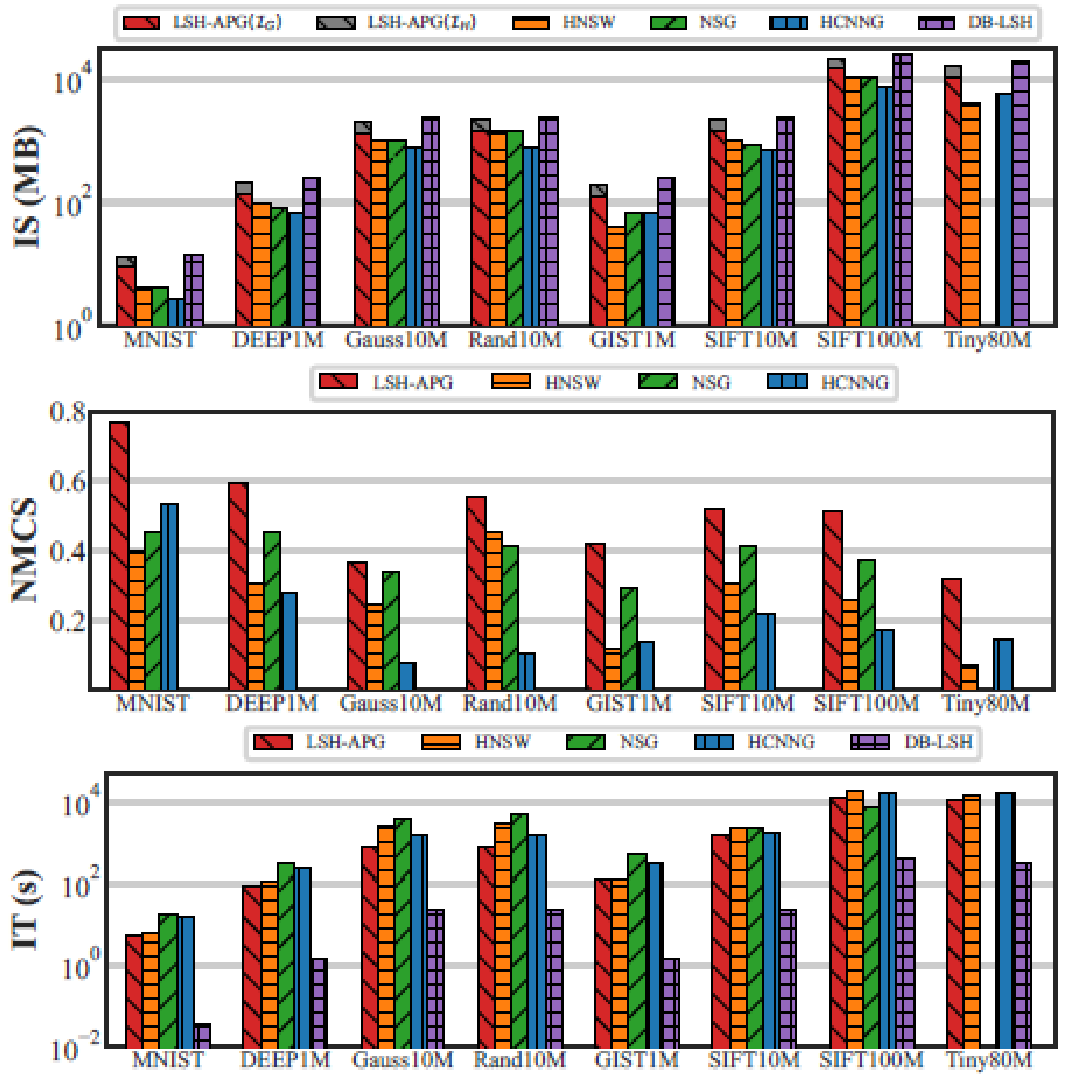


Figure 8: Indexing Performance in All Datasets

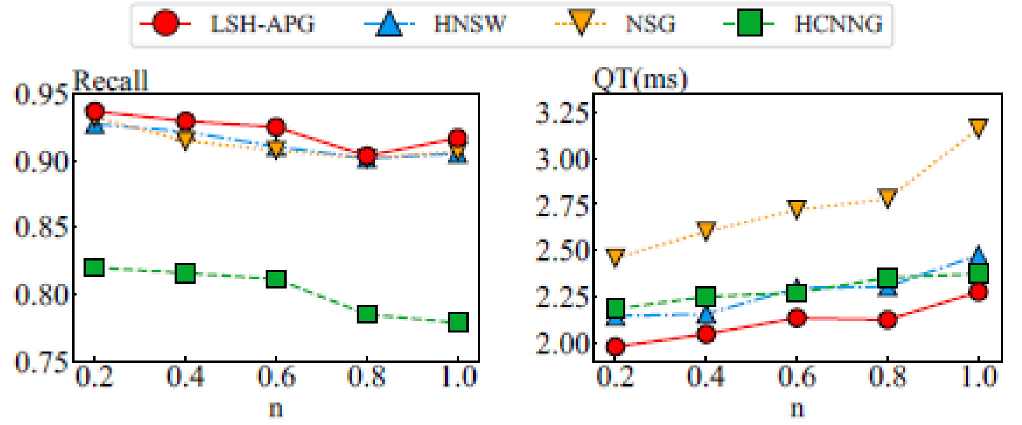


Figure 9: Performance on SIFT100M when Varying n

8. Conclusion

- 그래프 기반 방법의 높은 구축 비용 문제를 해결하기 위해, 효율적이고 정확한 LSH 기반 쿼리 전략을 설계
  - APG에 데이터를 순차적으로 삽입하는 방식을 구현

- 고품질 entry point selection 기술과 LSH 기반 pruning condition을 개발 -> 탐색 과정에서 확인해야 할 점의 수를 줄임

- 데이터셋 크기(Cardinality)가 쿼리 비용에 미치는 영향을 줄임
  - 쿼리 처리 시간과 인덱싱 시간을 동시에 감소시킬 수 있음을 이론적으로 증명

- 대규모 데이터셋 처리에 적합함 실험을 통해 확인

- 데이터셋이 진화함에 따라 LSH-APG를 저비용으로 점진적으로 유지 관리할 수 있음을 증명함

감사합니다