M-ACORN: ACORN 기반의 메타데이터 통합 구축을 통한 Hybrid Query 최적화 알고리즘

양현서[O], 김휘군, 권세인, 이지은, 박상현†
연세대학교 컴퓨터과학과
{hseo0608, jinhuijun, seinkwon97, jieun199624, sanghyun}@yonsei.ac.kr

M-ACORN: A Metadata-Aware Extension of the ACORN Based Index for Optimized Hybrid Query Processing

Hyunseo Yang[O], Huijun Jin, Sein Kwon, Jieun Lee, Sanghyun Park†
Dept. of Computer Science, Yonsei University

요 약

대규모 비정형 데이터 검색을 위한 근사 최근접 이웃 탐색(ANNS)과 구조화된 metadata 제약조건을 함께 질의하는 것을 Hybrid Query라고 한다. ACORN 알고리즘은 검색 단계에서 필터를 만족하는 서브 그래프를 탐색하여 Hybrid Query를 효율적으로 처리하지만, 그래프 구축 시 metadata를 고려하지 않아 추가적인 검색 최적화의 기회를 놓치고 있다. 이에 본 연구에서는 그래프 구축 단계에서 metadata의 불일치에 따른 패널티를 적용하여, 동일한 metadata를 가진 벡터 데이터들이 자연스럽게 군집화되도록 하는 M-ACORN 프레임워크를 제안한다. M-ACORN을 검증하기 위해 기존 모델과의 비교 실험을 진행하였으며, 검색 정확도와 쿼리 응답 속도에서 우수한 성능을 보임을 확인하였다.

1. 서 론

스마트 시티에서 텍스트, 이미지, 비디오 등 비정형 데이터의 최근접 이웃 탐색(Nearest Neighbor Search, NNS) 기술은 대규모 정보 시스템에서 핵심적인 역할을 수행한다[1]. 하지만 NNS로 k개의 정확한 해를 구하는 것은 데이터 규모 증가에 따라 계산 복잡도가 급격히 상승한다[2]. 이를 위해 여러 근사 최근접 이웃 탐색(Approximate Nearest Neighbor Search, ANNS) 알고리즘이 제안되었고, 정확도와 효율성 사이의 최적 균형을 도모하는 기법들이 연구되고 있다[3].

실제 ANNS 시스템은 벡터 유사도 검색뿐 아니라 벡터와 연관된 구조화된 정보인 metadata를 함께 고려하는 경우가 빈번하다. 이에 metadata를 통합한 복합적인 검색 요구사항을 질의하는 Hybrid Query 기법의 중요성이 최근 대두되고 있다[4].

Hybrid Query는 사용자가 요구하는 필수적인 검색 조건을 만족시키면서 벡터 탐색 공간을 좁힐 수 있다. 이를 위한 알고리즘을 filtered-ANNS라 하며, ANNS의 정확성을 효율적으로 향상시킨다[5]. filtered-ANNS 방식에는 pre-filtering, post-filtering, specialized indices가 있다. pre-filtering은 vector similarity search를 먼저 수행하고, 검색 결과에서 metadata를 적용하며, post-filtering은 조건에 따라 탐색 범위를 좁히고, 필터링된 데이터셋에서 벡터 유사도 검색을 수행한다. 그런데 pre-filtering, post-filtering은 벡터 유사도 검색 과정과 filtering 과정을 별개의 과정으로 처리하여 만족스러운 쿼리 지연시간과 recall을 얻기 어렵다. 그리하여 최근 연구들은 Hybrid Query를 위한 index를 구축하는 동향으로 이어지고 있다[6].

* 이 논문은 2025년도 정부(과기정통부)의 재원으로 한국연구재단의 지원(No. RS-2023-00229822)과 국토교통부의 스마트시티 혁신인재육성사업으로 지원을 받아 수행된 연구임.

† 교신 저자: sanghyun@yonsei.ac.kr

그 중 ANN Constraint-Optimized Retrieval Network (ACORN)[7]은 Hierarchical Navigable Small Worlds (HNSW)[8]을 기반으로 하며, 필터 조건을 만족하는 부분 그래프를 탐색하여 Hybrid Query를 처리한다. HNSW는 널리 쓰이는 그래프 기반 ANNS 알고리즘으로, 다층 그래프 구조의 상위층에서 하위층으로 내려가며 가장 가까운 이웃을 찾는 방식으로 대규모 고차원 데이터에서 유사한 벡터를 빠르게 찾는다.

그러나 ACORN은 그래프 구축 과정에서 HNSW와 동일하게 벡터 거리만을 기준으로 그래프를 생성하며, 구축 과정에서는 제약조건인 metadata를 고려하지 않는다. 이는 같은 metadata를 가진 노드들이 떨어져 서로 연결될 가능성이 낮아질 수 있어, 구축 단계에서의 metadata를 활용한 검색 최적화 기회를 놓친다.

따라서 본 논문에서는 그래프 구축 단계에서부터 metadata에 따라 penalty를 적용하여 계산된 거리로 그래프를 구축하는 M-ACORN 프레임워크를 제안한다. 이는 검색 시점에 metadata를 만족하는 주변 노드들을 더 빠르게 찾아 Hybrid Query를 효율적으로 처리하도록 한다. 본 논문의 기여는 다음과 같다:

- Metadata를 고려한 그래프를 구축하여 Hybrid Query를 효율적으로 처리하는 M-ACORN 프레임워크를 제안한다.
- M-ACORN의 정량적 평가를 위해 기존 모델과의 비교 실험을 진행하여 M-ACORN의 우수성을 입증한다.

2. 본론

M-ACORN 구조는 그림 1과 같다. 동일한 metadata를 가진 노드를 그림에서 같은 색으로 나타내었다. M-ACORN은 데이터셋에서 샘플을 추출하여 벡터 평균 거리를 구하고, penalty factor를 곱하여 penalty를 계산한다. 그 후 그래프 구축 과정에서 노드의 이웃을 결정할 때, metadata가 다른 노드들은 penalty를 부여하여 metadata가 서로 다른 노드들끼리 이웃이 될 확률을 낮추고, metadata가 같은 노드들끼리 군집화되도록 한다.

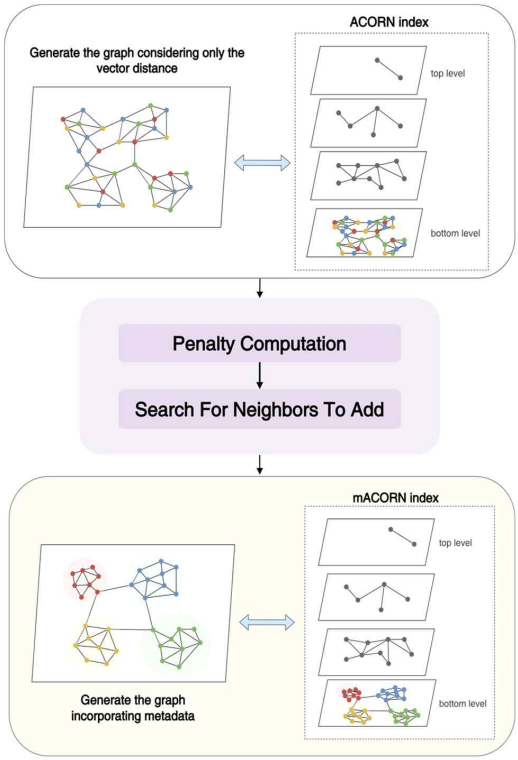

그림 1. M-ACORN 프레임워크

2.1 Penalty Computation

M-ACORN은 먼저 식 (1)과 같이 m개의 dim차원 벡터로 구성된 데이터셋 **X**에서 n개의 샘플을 무작위로 추출하여 식 (2)와 같이 샘플 **S**를 구성한다.

$$X = \{x_1, x_2, \ldots, x_m\}, x_i \in \mathbb{R}^{\dim} \quad (1)$$

$$S \sim Uniform(X, n) \quad (2)$$

그 후 샘플의 모든 가능한 조합에 대해 식 (3)과 같이 유클리디안 거리식(L2)을 적용하여 벡터 거리 집합 **D**를 구하고, 이를 식 (4)와 같이 샘플 개수로 나눠 데이터셋 샘플의 벡터 거리 평균 d를 구한다. 그리고 벡터 거리 평균 d를 penalty factor(pf)가 1인 기본 penalty(Π)로 설정한다. 이는 데이터의 벡터 거리 스케일을 고려하여 penalty를 설정하기 위함이다.

$$D = \left\{ \| s_i - s_j \|^2 \mid 1 \le i < j \le n, s_i, s_j \in S \right\} \quad (3)$$

$$d = \frac{\sum_{\dim \in D} \dim}{|D|} \quad (4)$$

하지만 샘플 데이터셋의 벡터 거리 평균이 penalty로써 최적값이 아닐 가능성을 고려하여 penalty factor(pf)를 통해 penalty (Π)를 동적으로 조절할 수 있도록 하였다.

$$\Pi = d \times pf \quad (5)$$

2.2 Search For Neighbors To Add

섹션 2.1에서 계산된 최종 penalty(Π)는 M-ACORN index 구축 과정에서 searchNeighborsToAdd 함수에 적용된다.

HNSW 구축 과정에서, 각 노드는 지수적으로 감소하는 확률 분포를 따라 무작위로 최대 계층을 할당받아, Navigable Small World(NSW)와 유사한 다계층 구조를 형성한다. 새로 삽입된 노드는 최상위 계층의 랜덤 진입점(ep)에서 시작하여 각 계층별로 Greedy Search를 수행하며 가장 가까운 이웃으로 연결된다.

ACORN은 이 HNSW 구조를 확장하여, 두 단계 이웃 탐색 과정으로 필터 조건을 고려한 연결성을 유지한다. 구체적으로, searchNeighborsToAdd 함수는 새 노드가 삽입될 때 해당 노드의 이웃을 결정한다. 이때 벡터 거리 기반으로 후보 이웃을 수집하고, 필터로 그래프가 단절되지 않도록 더 많은 이웃을 수집하여 그래프를 조밀하게 형성한다.

이때 ACORN은 그래프 구축 시 벡터 거리만을 사용하고, 검색 단계에서 metadata를 만족하는 서브 그래프를 탐색하여 쿼리와 가까우면서 filter를 만족하는 노드들을 찾는다. 그러나 본 논문에서는 index 구축 과정에서 metadata를 고려하여 penalty를 부여한 거리를 계산하고, 이를 반영하여 그래프가 구축되도록 하였다. 이 과정을 searchNeighborsToAdd (alg.1)로 나타내었다.

alg.1에서는 초기화 단계에서 후보 queue C 와 결과 queue R을 생성하고 진입점 ep 와 거리 d_ep를 C 와 R 에 추가하여 탐색의 시작점을 설정한다(line 1-3). 탐색루프(line 4-9)에서는 후보 큐 C 가 비어 있지 않은 동안, 가장 먼 거리 후보의 거리보다 멀거나 M 보다 큰 경우 탐색을 종료한다(line 6). 이를 만족하지 않는 경우, 현재 이웃 후보 c 를 노드 v 로 설정한다.

그리고 v 의 이웃 u 를 레벨 l 에서 가져와 q 와 u 사이의 거리 d_u 를 거리 함수 dis로 계산한다(line 11). 이때, 만약 u 와 v 의 metadata가 다르다면, d_u 에 페널티 π 를 더한다(line 12-13). 이는 동일한 metadata를 가진 노드를 우선적으로 이웃으로 선택하도록 유도한다. 이후 R 크기가 efc 보다 작거나 R의 가장 먼 이웃의 거리($R.Top.d$)가 d_u 보다 크다는 조건을 만족하면, u 를 이웃으로 추가할 수 있으므로 R 과 C 에 d_u, u를 추가한다. 만약 R 의 크기가 candidate result의 최대 크기인 efc 를 초과하면, 가장 먼 노드를 제거한다(line 18). 최종적으로

Algorithm 1. searchNeighborsToAdd

Input: current element q, entry point ep, distance from q to eq d_ep, current layer l, HNSW graph H, compute distance dis, desired number of neighbors M, distance penalty π, maximum number of candidates in results efc
Output: updated priority queue with nearest neighbors R

```
1  C ← new priority queue   // candidates queue
2  C.PUSH(d_ep, ep)   // add entry point to candidates
3  R.PUSH(d_ep, ep)   // add entry point to results
4  while C not empty
5      c ← C.TOP   // get farthest candidate
6      if (c.d > R.TOP.d) or R.SIZE ≥ M
7          break
8      v ← c.id    // get current node ID
9      C.POP
10 for each u in H.neighbors(v) at layer l
11     d_u ← dis(v, u)      // compute distance to neighbor
12     if metadata[u] ≠ metadata[v]
13         d_u ← d_u + π      // apply penalty
14     if R.SIZE < efc or R.TOP.d > d_u
15         R.PUSH(d_u, u)
16         C.PUSH(d_u, u)
17         if R.SIZE > efc
18             R.POP   // remove farthest node
19 return R   // return nearest neighbors
```

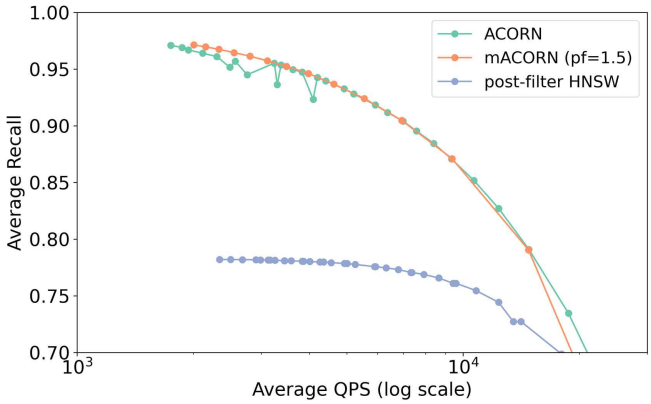

그림 2. SIFT1M 데이터셋에 대한 평균 Recall@10과 QPS

q 에 대한 최근접 이웃을 포함하는 결과 queue인 R 를 반환한다(line 19).

3. 실험 및 결과

3.1 실험 환경

본 논문에서는 M-ACORN의 성능을 평가하기 위해 SIFT1M 데이터셋을 대상으로 recall과 QPS(Query per second)를 측정하였다[9]. 본 논문은 ACORN과 동일하게 SIFT1M 데이터셋에 metadata로 1부터 12까지의 정수를 무작위로 할당하여 Hybrid Query로 처리될 수 있도록 하였다. Baseline은 ACORN, post-filtering, pre-filtering으로 설정하였다. post-filtering 방식은 HNSW로 벡터 유사도 검색을 수행하고, filtering을 수행하며, pre-filtering 방식은 filtering 과정을 먼저 거치고, brute-force 방식으로 벡터 유사도 검색을 수행한다.

파라미터는 efc 값은 40, M은 16으로 설정하였다. efs는 10부터 400까지 10씩 증가시키면서 recall과 QPS를 측정하였다.

3.2 실험 결과

M값이 16일 때, penalty factor를 1.5로 설정하였다. 실험 결과는 그림 2과 같다. M-ACORN을 ACORN과 비교하는 경우, recall이 95% 수준에서 M-ACORN의 qps는 3480 정도로 ACORN의 qps인 2486.57보다 40% 더 높은 결과를 보여주었으며, recall이 97% 수준에서는 M-ACORN의 qps는 2150으로 ACORN의 qps인 1869.1보다 15% 높은 결과를 보여주었다. post-filtering 방식과 비교하는 경우, post-filtering 방식의 최대 리콜은 약 78%이므로 post-filtering 방식이 도달 가능한 리콜 범위 내에서 비교하면, post-filtering 방식의 qps가 2340고, M-ACORN은 qps가 14747로, 5~6배 이상 높은 qps를 달성하였다. pre-filtering 방식과 비교하는 경우, pre-filtering 방식은 brute-froce 방식으로 검색을 수행하기 때문에 recall은 항상 1이지만, qps가 16.91로 매우 낮아 실용적으로 쓰이기 어려운 한계를 가진다.

그림 3은 최적의 penalty factor를 찾기 위해 여러 가지 penalty factor에 대한 평균 recall과 qps 값을 구하여 그래프로 나타낸 것이다. penalty factor는 1.5배일 때 정확도 면에서 가장 우수했으며, 1.75배일 때 qps 면에서 가장 우수하였다.

4. 결론

본 논문에서는 index 구축 단계에서부터 metadata를 고려하여 그래프를 구축하여 Hybrid Query 처리를 최적화하는 M-ACORN 프레임워크를 제안하였다. 그리고 ACORN, Post-filter HNSW, Pre-filter와 같은 기존 방법론들과 recall과 qps를 비교하여

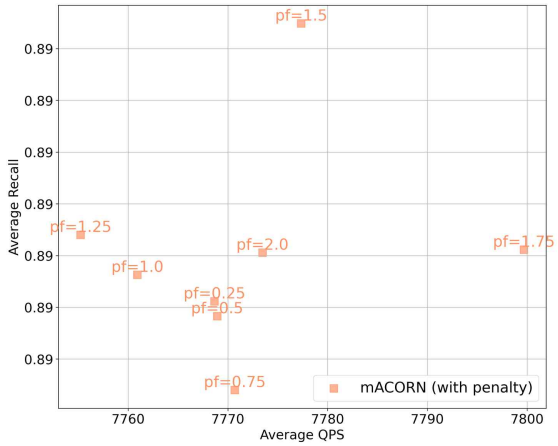

그림 3. 여러 penalty factor에 대한 평균 Recall과 QPS

M-ACORN이 Hybrid Query를 더 효율적으로 처리함을 보였다. 추가적으로 최적의 penalty factor를 찾기 위하여 여러 penalty factor를 적용하여 평균 recall과 qps 값을 비교하여 제안된 프레임워크의 우수성을 입증하였다.

참고문헌

[1] Glasmeier, Amy, and Susan Christopherson. "Thinking about smart cities." Cambridge Journal of Regions, Economy and Society 8.1 (2015): 3-12.

[2] Abbasifard, Mohammad Reza, Bijan Ghahremani, and Hassan Naderi. "A survey on nearest neighbor search methods." International Journal of Computer Applications 95.25 (2014).

[3] Li, Wen, et al. "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement." IEEE Transactions on Knowledge and Data Engineering 32.8 (2019): 1475-1488.

[4] Gollapudi, Siddharth, et al. "Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters." Proceedings of the ACM Web Conference 2023. 2023.

[5] Wang, Mengzhao, et al. "Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints." arXiv preprint arXiv:2203.13601 (2022).

[6] Wu, Wei, et al. "HQANN: Efficient and robust similarity search for hybrid queries with structured and unstructured constraints." Proceedings of the 31st ACM International Conference on Information & Knowledge Management. 2022.

[7] Patel, Liana, et al. "Acorn: Performant and predicate-agnostic search over vector embeddings and structured data." Proceedings of the ACM on Management of Data 2.3 (2024): 1-27.

[8] Malkov, Yu A., and Dmitry A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." IEEE transactions on pattern analysis and machine intelligence 42.4 (2018): 824-836.

[9] http://corpus-texmex.irisa.fr