CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search

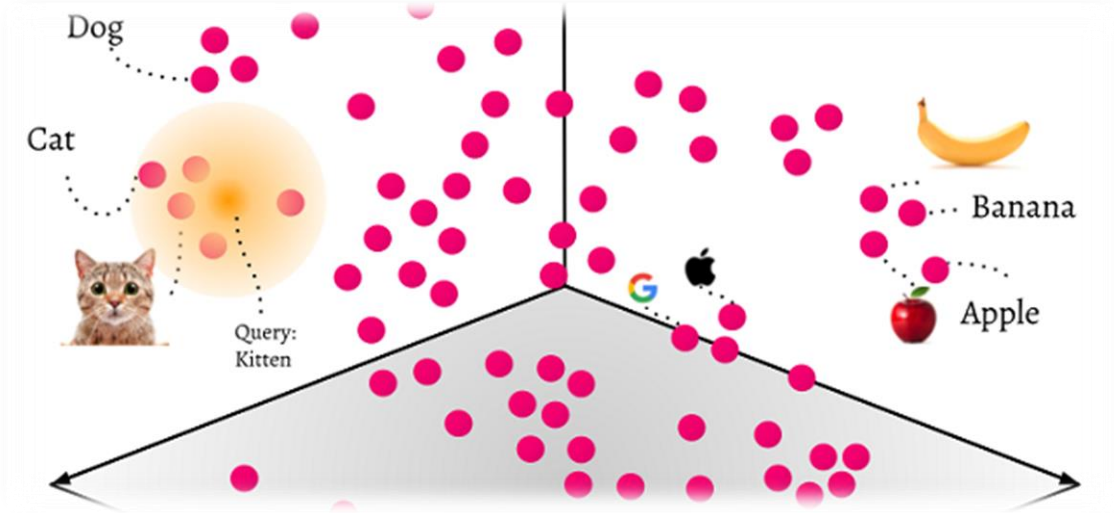
Ming Yang, Yuzheng Cai, Weiguo Zheng
School of Data Science, Fudan University, China
{ yangm24, yuzhengcai21 } @m.fudan.edu.cn zhengweiguo@fudan.edu.cn

Reporter: Ming Yang
Date: 2024/11/02

What is Approximate Nearest Neighbor Search (ANNS)?

- **Target:** to find the vectors which are similar to the query vectors as fast as possible.

- **Application:**
  - IR
  - Recommendation
  - RAG

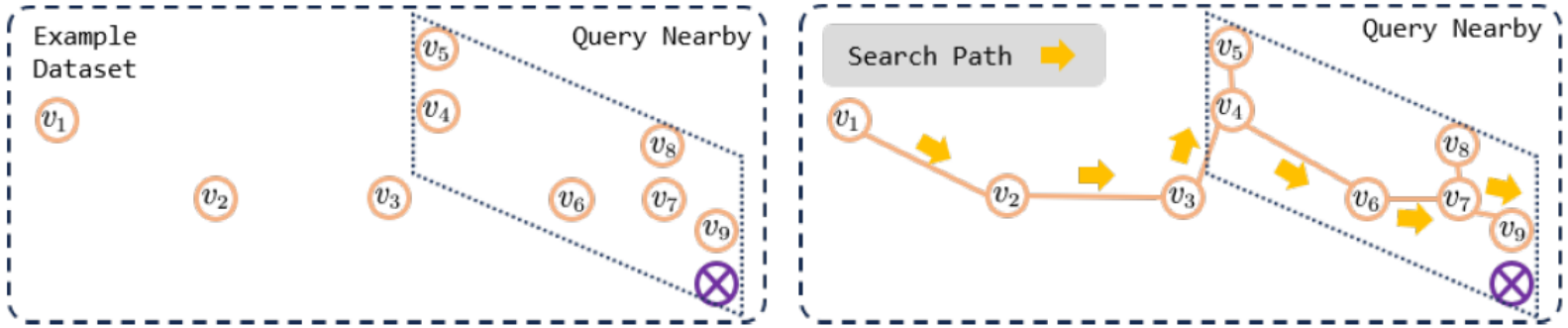


Milvus
Pinecone
PostgreSQL
GreptimeDB™
chroma
OCEANBASE

Motivation

- ANNS is increasingly used in AI scenarios, so the system has higher and higher requirements for the end-to-end query latency and precision.

- The Graph methods have good results in terms of accuracy and speed, and is increasingly being used in various systems.



(a) An example dataset of vectors (b) A Relative Neighborhood Graph (RNG) [39].

Figure 1: An example dataset of vectors and its proximity graph.

- **Goal:** to present a schema which can speedup mainstream graph-based approaches.

Intuition

- A "good" vector search is essentially to approach the query point from a point in the vector space "as fast as possible" and make the query result "as accurate as possible".
- **[How to be as fast as possible?]** We hope to take long steps to quickly approach the query when we are far away from the query.
- **[How to be as accurate as possible?]** We hope to take short steps to accurately search points near the query when we are close to the query.
- Build a good proximity graph must have both long and short edges.
  - conflict!
  - graph degree cannot be too high or too low (a dual requirement for speed/accuracy).

Our Approaches

#1 Random Partitioning

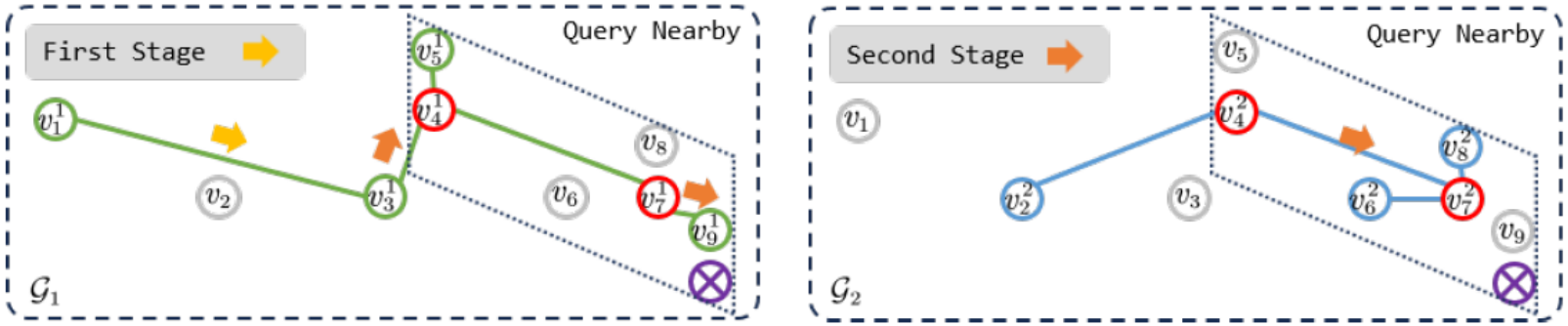


Figure 2: An example of *CSPG* index, where the proximity graphs are built using relative neighborhood graph $\mathcal{G}_1$ and $\mathcal{G}_2$ (with very similar degree to Figure 1b.

- **[Route Vectors]** Route vectors are vectors shared by all partitions. When a node is in RV, it will appear in all partitions; otherwise, it will only appear in a certain partition.

- **[Random partition]** Sample route vector (RV) from dataset D the and randomly assign the vectors in D/RV to partitions. This keeps different partitions with the same data distribution as the original dataset. Then we build sparse proximity graphs (SPGs) for each partition, which are faster than that built for the whole dataset.

Our Approaches

#2 Two-stage searching

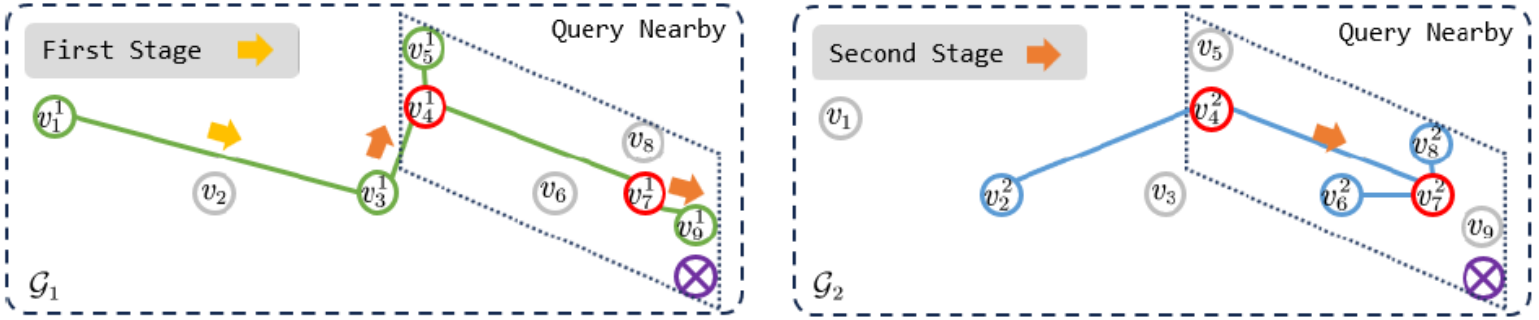


Figure 2: An example of *CSPG* index, where the proximity graphs are built using relative neighborhood graph $\mathcal{G}_1$ and $\mathcal{G}_2$ (with very similar degree to Figure 1b).

- The First Stage: exploring single partition for fast approaching
- The Second Stage: cross-partition expansion for precise search

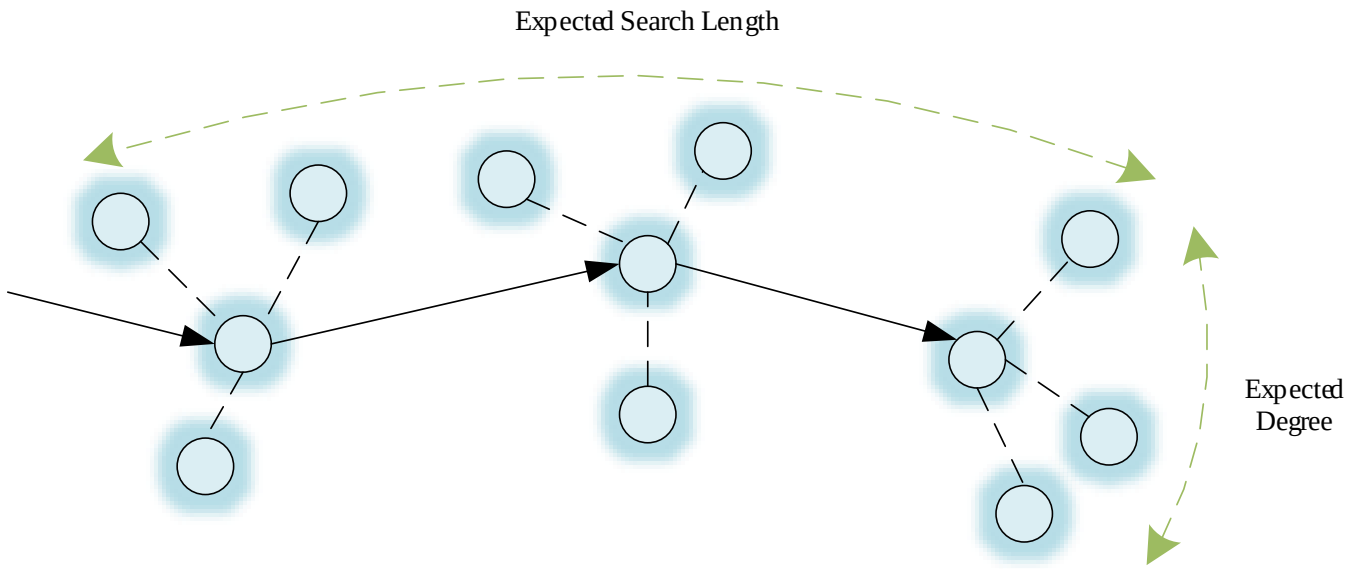
Theoretical Analysis

#1 expected number of distance computations

- Expected number of distance computation

$$\mathbb{E}[C] = \sigma \mathbb{E}\left[|p \rightsquigarrow q|\right]$$

- So to estimate the expected number of distance computation, we use the expected search length.



Theoretical Analysis
#2 Monotonic Search Network (MSNET)

- There is never backtracking in a MSNET.

- the search length expectation of an MSNET is [1]

$$\mathbb{E}^M = \mathcal{O}\left(n^{\frac{1}{d}} log\ n^{\frac{1}{d}} / \Delta r\right)$$

As n decreases, Δr increases.

[1] Cong Fu, Chao Xiang, ChangxuWang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. arXiv preprint arXiv:1707.00143, 2017.

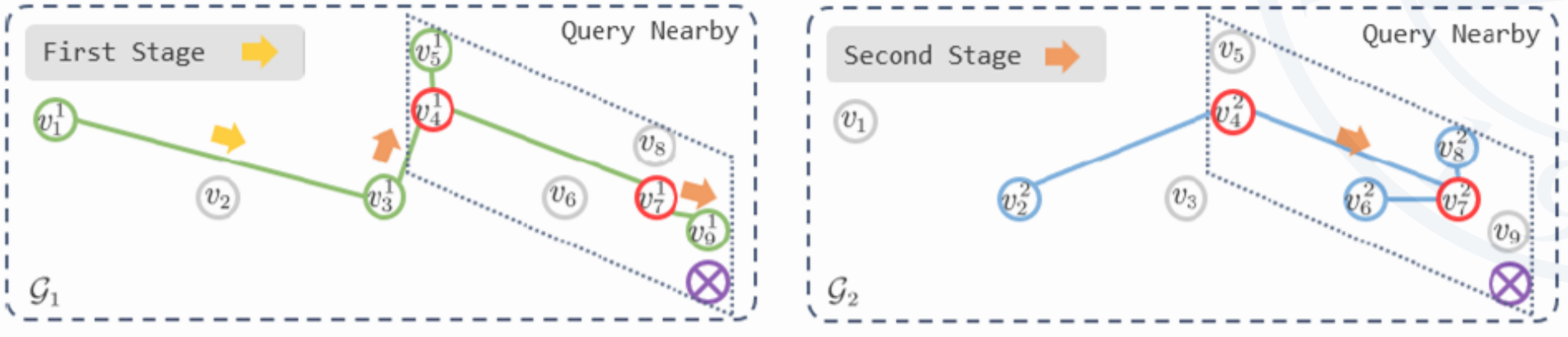
Theoretical Analysis

#3 the expected search length of CSPG

Theorem 1. *Given a start vector $s \in RV$ and a query vector $q \in RV$, by performing greedy beam search from s on each MSNET $\mathcal{G}_i$ independently, we can obtain m monotonic paths $s^i \rightsquigarrow q^i$, where $s^i, q^i \in \mathcal{G}_i$. It holds that $\mathbb{E}^{\mathcal{G}_i}[|s^i \rightsquigarrow q^i|] = \mathbb{E}^{\mathcal{G}_j}[|s^j \rightsquigarrow q^j|]$ for $1 \leq i, j \leq m$.*

Theorem 2. *Denote $\mathbb{E}^{CSPG}[|p \rightsquigarrow q|]$ as the expected length of search sequence in CSPG. Denote $\mathbb{E}^{\mathcal{G}_i}[|p \rightsquigarrow q|]$ as the expected sequence length when searching only on the graph $\mathcal{G}_i$. It holds that*

$$\mathbb{E}^{CSPG}[|p \rightsquigarrow q|] = \mathbb{E}^{\mathcal{G}_i}[|p \rightsquigarrow q|].$$



Theoretical Analysis

#4 Approximate Monotonic Search Network (AMSNET)

Definition 7 (Approximate Monotonic Search Network, shorted as AMSNET). *Given a dataset $\mathcal{D}$ of n vectors, a proximity graph $\mathcal{G}$ built on $\mathcal{D}$ is called an approximate monotonic search network iff for every vector $u \in \mathcal{D}$, its neighbor set $\mathcal{N}(u)$ satisfies that $|\mathcal{N}(u)| \leq \sigma$ while maximizing $\rho(u)$.*

- we say that vector u conquers q iff $\exists v \in N(u)$, $\delta(v, q) < \delta(u, q)$, denoted by $u \succ q$.
- The probability of a vector u conquer any query q on given graph is

$$\rho(u) = \frac{\sum_{q \in \mathcal{G}} \mathbb{I}(u \succ q)}{n}$$

- AMSNET allows backtrackings, but minimizing their probability.

Theorem 3. *For datasets with the same distribution, as the number of vectors n decreases, $\rho(u)$ is monotonically non-decreasing.*

- This property indicates that as n decreases, the probability (expected value) of the search backtracking decreases.

Theoretical Analysis

#5 speedup

Almost every proximity graph in practice is an AMSNET.

$$\hat{\mathbb{E}}^{CSPG}\left[|p \rightsquigarrow q|\right] = \mathcal{O}\left(w\left(\lambda n + \frac{n(1-\lambda)}{m}\right)^{\frac{1}{d}} \log\left(\lambda n + \frac{n(1-\lambda)}{m}\right)^{\frac{1}{d}} / \Delta r\right)$$

$$Speedup = \frac{\sigma \times \hat{\mathbb{E}}^{PG}[|p \rightsquigarrow q|]}{\sigma \times \hat{\mathbb{E}}^{CSPG}[|p \rightsquigarrow q|]} = \left(\frac{w^{PG}}{w^{CSPG}} \times \frac{\Delta r^{CSPG}}{\Delta r^{PG}}\right) \times \frac{n^{\frac{1}{d}} \log n^{\frac{1}{d}}}{\left(\lambda n + \frac{n(1-\lambda)}{m}\right)^{\frac{1}{d}} \log\left(\lambda n + \frac{n(1-\lambda)}{m}\right)^{\frac{1}{d}}}$$

≥ 1

- W is a detour factor, which can be computed as

$$w = \frac{\text{length of search sequence}}{\text{length of search sequence} - \text{number of distance backtracking}}$$

Experiment
#1 setup

- We use the most commonly used public datasets come from Ann-benchmarks [2], like SIFT, DEEP, GIST, etc.

- Three well-known graph-based ANNS algorithms HNSW, Vamana, and HCNNG are selected as baselines, which achieved competitive performance on previous studies.

[2] Martin Aumuller, Erik Bernhardsson, and Alexander Faithfull. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Information Systems, 87:101374, 2020.

Experiment

#2 overall query performance

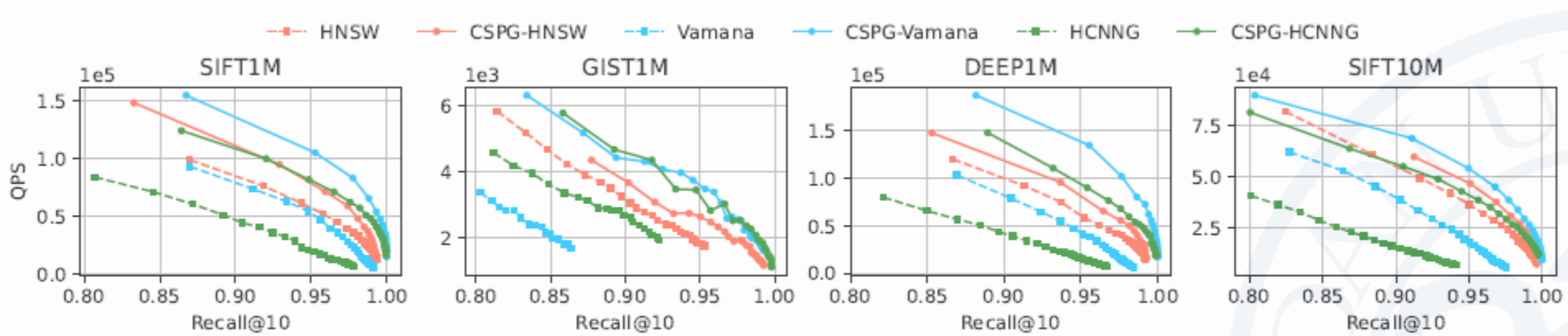


Figure 3: QPS v.s. recall curves for comparing query performance

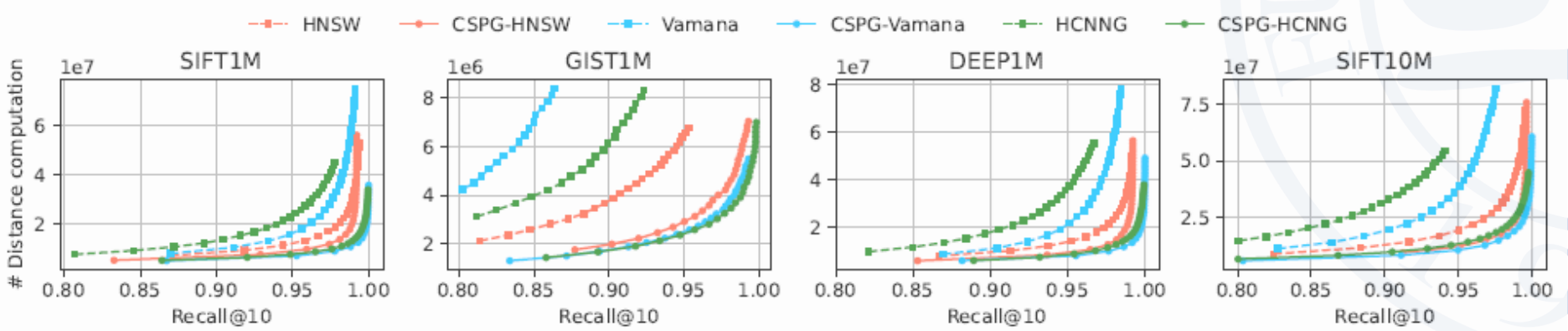


Figure 10: Number of distance computation v.s. recall curves for comparing query performance

Experiment

#2 construction performance

Table 1: Comparison for index construction cost, in which DS is the data size (MB), IS is the graph index size (MB), and IT is the index construction time (s)

index		SIFT1M			GIST1M			DEEP1M			SIFT10M		
		DS	IS	IT	DS	IS	IT	DS	IS	IT	DS	IS	IT
PG	HNSW	488	253	33	3,662	254	237	366	251	28	1,221	2,596	416
	Vamana		126	97		120	388		128	90		1296	1,122
	HCNNG		44	85		53	390		54	79		633	742
CSPG	HNSW		389	50		380	382		387	48		3,894	627
	Vamana		195	145		186	608		192	136		1,938	1,627
	HCNNG		77	131		81	465		86	119		934	1,128

Experiment

#3 impact of factors and parameters

- Varying the dataset size
- Varying the number of partitions
- Varying the sampling ratio
- Varying the candidate set sizes
- Statistics for detour factor

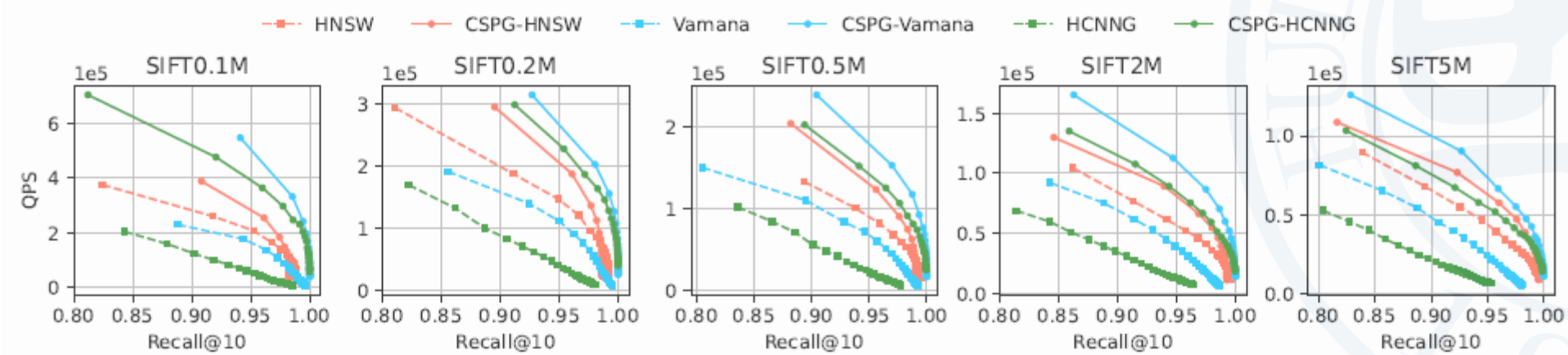


Figure 4: Query performance when varying the dataset size n

Experiment

#3 impact of factors and parameters

- Varying the dataset size
- Varying the number of partitions
- Varying the sampling ratio
- Varying the candidate set sizes
- Statistics for detour factor

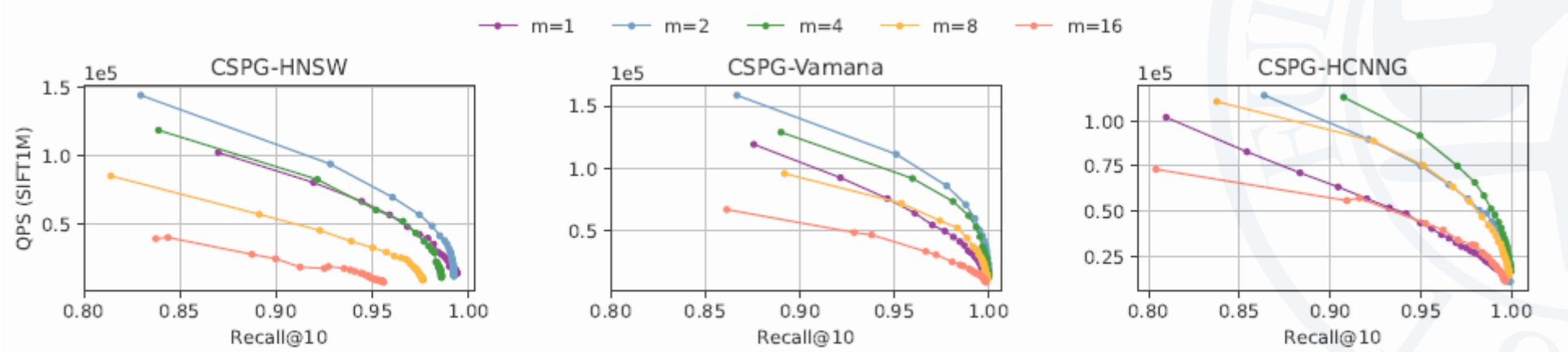


Figure 5: Query performance when varying the number of partitions m

Experiment

#3 impact of factors and parameters

- Varying the dataset size
- Varying the number of partitions
- Varying the sampling ratio
- Varying the candidate set sizes
- Statistics for detour factor

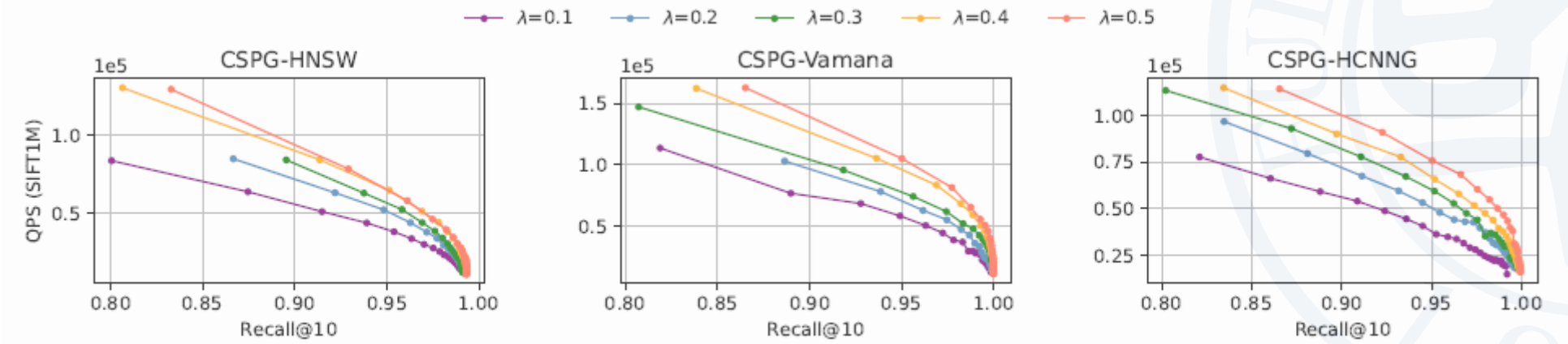


Figure 6: Query performance when varying the sampling ratio λ

Experiment

#3 impact of factors and parameters

- Varying the dataset size
- Varying the number of partitions
- Varying the sampling ratio
- Varying the candidate set sizes
- Statistics for detour factor

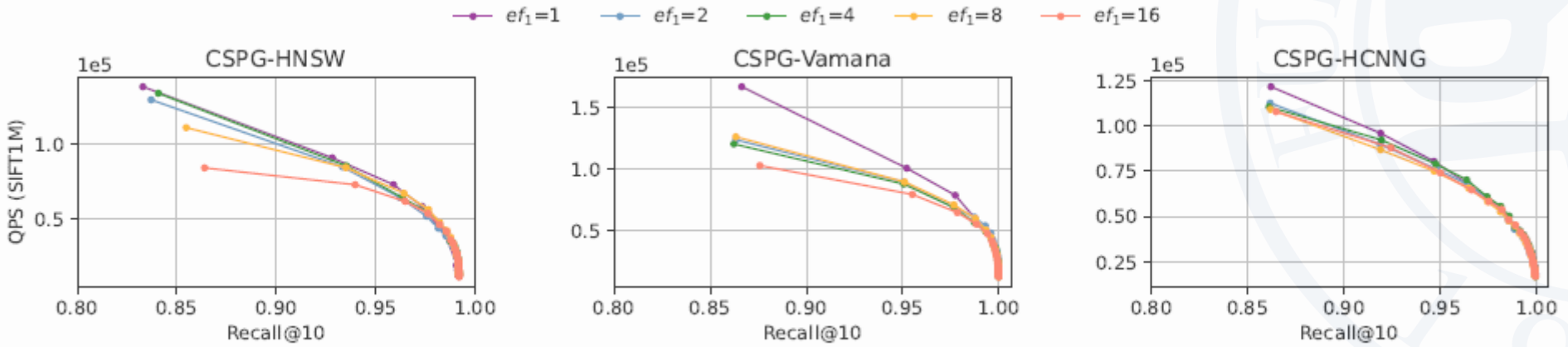


Figure 7: Query performance when varying the candidate set size ef_1 in the first stage

Experiment

#3 impact of factors and parameters

- Varying the dataset size
- Varying the number of partitions
- Varying the sampling ratio
- Varying the candidate set sizes
- Statistics for detour factor

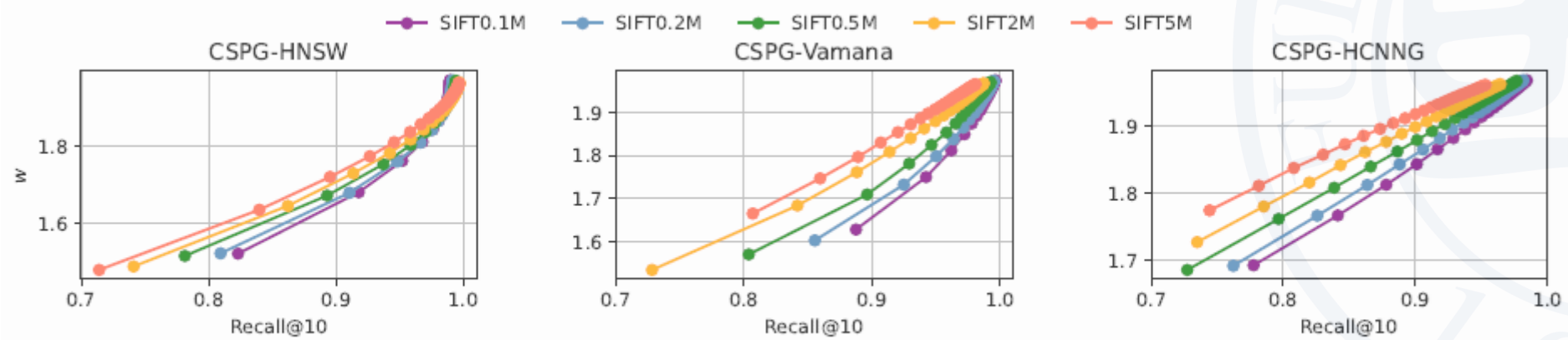
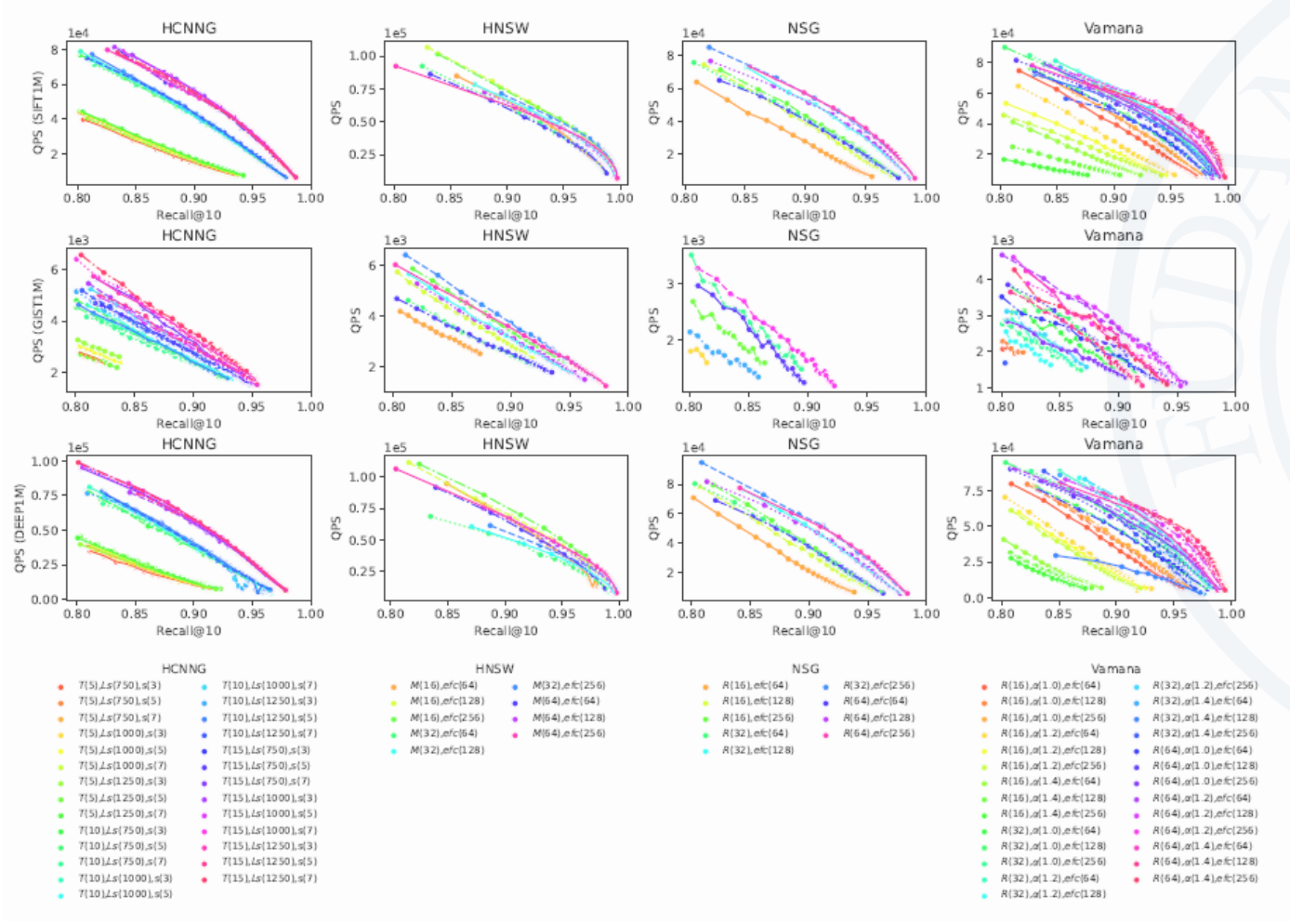


Figure 8: Detour factor when varying the dataset size n

Experiment

#4 praetor frontiers and ann-benchmarks

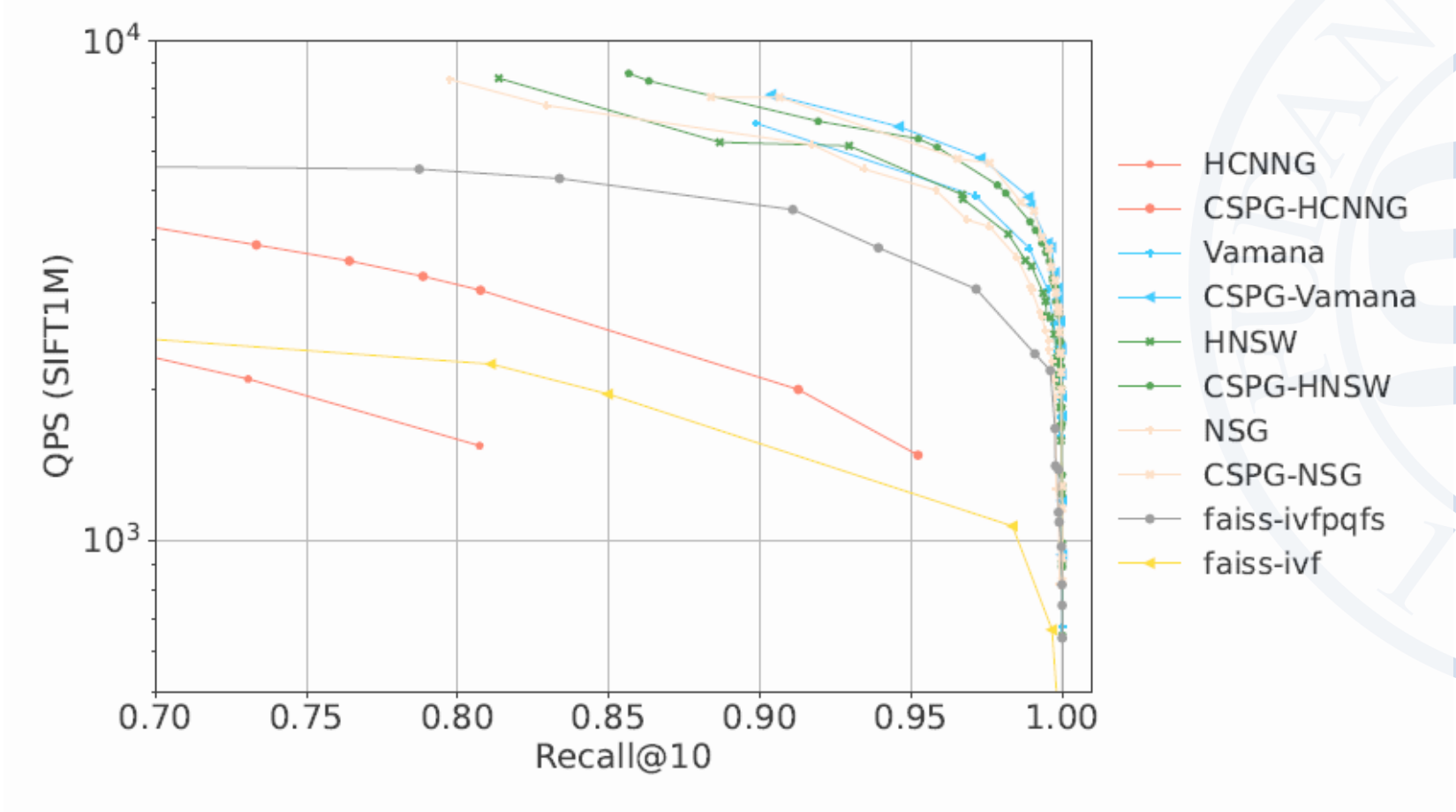
- To show the pareto frontier of our schema, we also conduct hyperparameter grid search and evaluation with ANN-Benchmarks



Experiment

#4 praetor frontiers and ann-benchmarks

- To show the pareto frontier of our algorithm. We also conduct hyperparameter grid search and evaluation with ANN-Benchmarks



Thank you!