ACM SIGMOD Programming Contest 2023

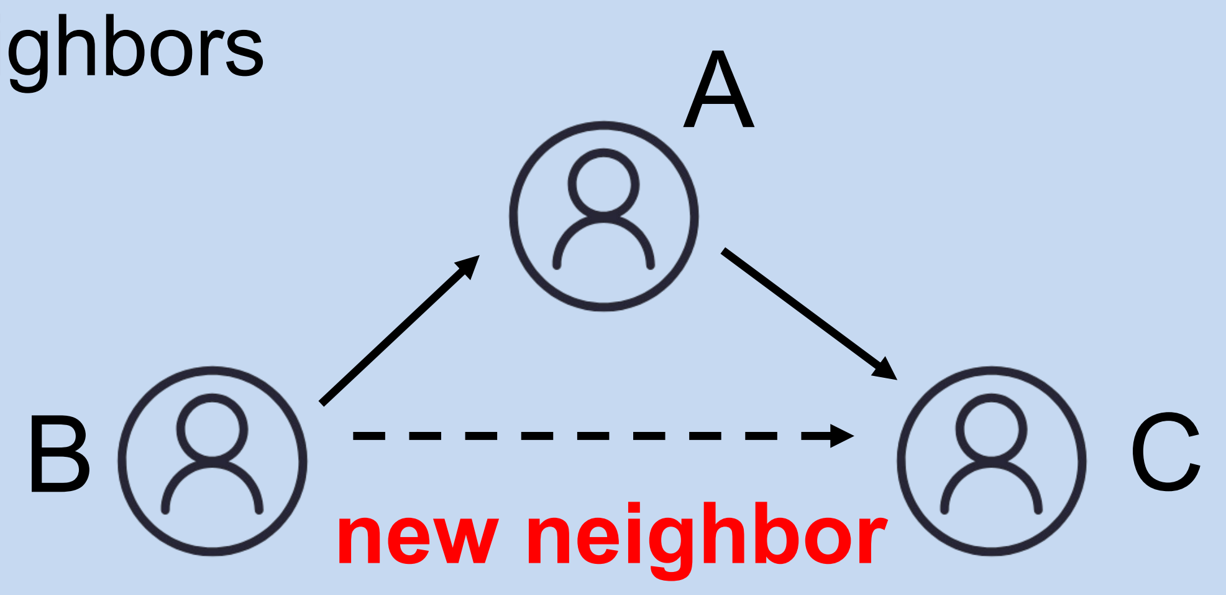
Team HelloWorld
DB Group, Dept. of Computer Science and Technology, Tsinghua University
Jiayi Wang Advisor: Guoliang Li
jiayi-wa20@mails.tsinghua.edu.cn

0. Task Overview

- **Problem definition:**
  - Build an approximate K-NN Graph for a set of vectors.
  - For n d-dimensional vectors (nodes), find the approximate k nearest neighbors of each of them using Euclidean Distance in a limited time.
  - n=10
  - d=100
  - k=100
  - 30 minutes
- **Measurement:**
  - $Recall = \frac{number\ of\ true\ top\ 100\ nearest\ neighbors}{100}$
- **Testing Environment:**
  - Azure Standard F32s_v2 with 32 CPU x 2.7 GHz, 64 GB Main Memory, and 32 GB Disk Storage.

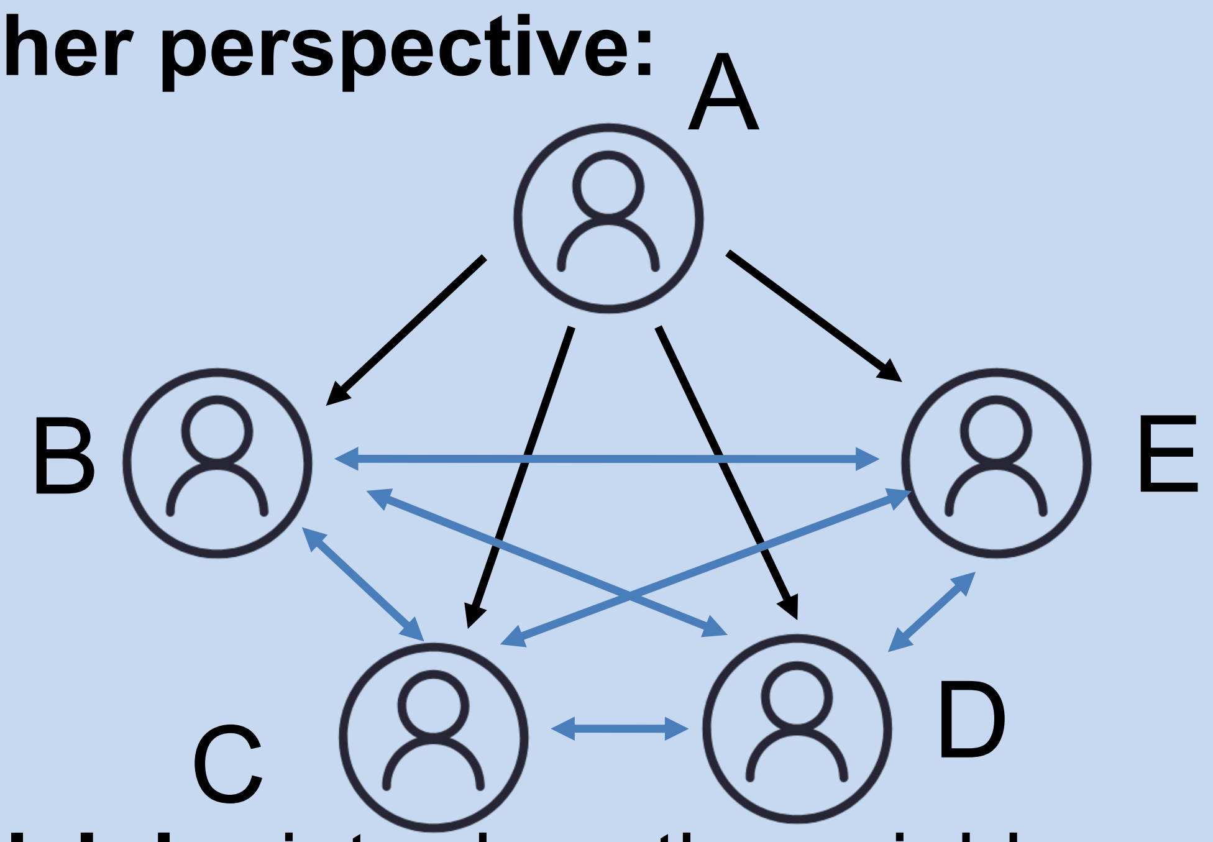
1. Basic Algorithm: NN-Descent

□ **Main idea:** Neighbors' neighbors are likely to be neighbors



- With this idea, we can optimize the current NN-graph by **exploring the neighbors' neighbors** of each node
- The graph will be optimized iteratively

□ **Another perspective:**



- **Local Join:** introduce the neighbors of each node to "get to know" each other
- **Advantage**: better locality, thus higher efficiency

2. Bottleneck in NN-Descent

Local Join

To introduce the neighbors of each node to "get to know" each other

- Need to compute distances between each pair of neighbors
- Need to get the neighbor's lock to update
- Need to maintain a list of neighbors for each node

3. Accelerate Distance Computation

- **For Euclidean Distance:**
  - $(\mathbf{X}-\mathbf{Y})^2 = |\mathbf{X}|^2 - 2\mathbf{XY} + |\mathbf{Y}|^2$
  - $|\mathbf{X}|^2$ of each vector can be precomputed
  - **XY** can be converted to matrix multiplication computations
  - Both can be accelerated by vectorization using Intel MKL

4. Efficient Use of Locks

- **Naïve way:**

```
for u in neighbors:
    for v in neighbors:
        Dist(u,v)
        Get_lock_and_update(u)
        Get_lock_and_update(v)
```

  - Frequent lock acquisition and release
  - Insufficient localization and cache utilization

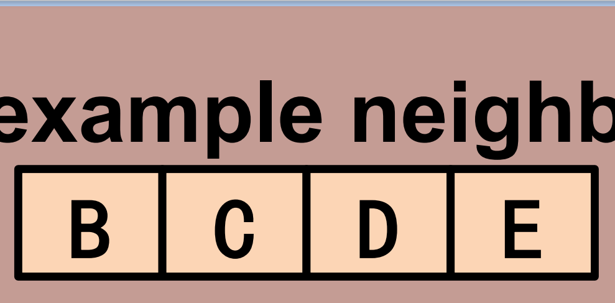
- **Optimized way:**

```
Compute_all_dist(neighbors) // Vectorization
for u in neighbors:
        Get_lock(u)
        Update_all_neighbors(u)
```

  - Less lock acquisition and release
  - Better localization

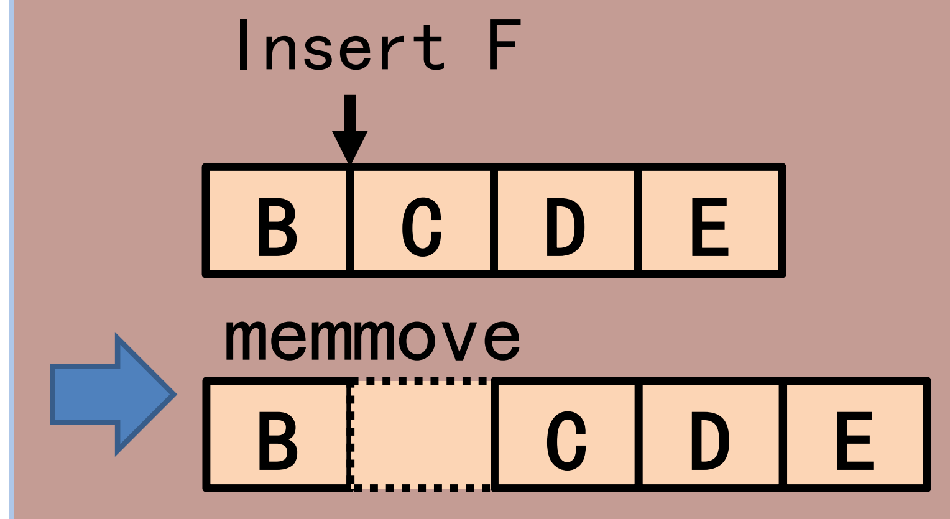
5. Efficient Update of Neighbors List

- **An example neighbors list:**



  - For each neighbor:

```
struct{
    uint32_t  id;
    float      dis;
    bool       flag;
}
12 bytes in total
```



Brings 3***12** = 36 bytes of memmove

- **Compress the information in *flag***:
  - n= 10^7 uses only 24 bits of the uint32_t
  - Use any of the remaining 8 bits to record the flag information
  - Reduce memove by 33%

```
struct{
    uint32_t  id;
    float      dis;
}
8 bytes in total
```

6. Results

Average Recall: 0.987
Runtime: 1854s

Reference:
[1] Dong W, Moses C, Li K. Efficient k-nearest neighbor graph construction for generic similarity measures[C]//Proceedings of the 20th international conference on World wide web. 2011: 577-586.