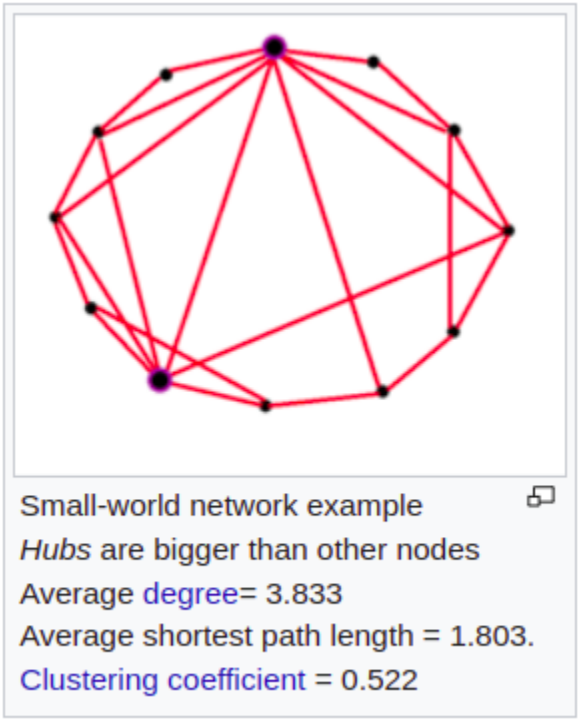
Approximate Nearest Neighbor Retrieval

Task: given a recommendation vector(provided by recommendation algorithm) fetch M-elements closest to it from the database(containing videos) ensuring at the same time they are not too similar(same content with resolution difference etc.)

1st approach is based on **Hierarchical Navigable Small World graphs**

Terminologies:

proximity graph : is simply a graph in which two vertices are connected by an edge if and only if the vertices satisfy particular geometric requirements.
Delaunay triangulation : for a given set **P** of discrete points in a general position is a triangulation DT(**P**) such that no point in **P** is inside the circumcircle of any triangle in DT(**P**).
small-world graph : is a type of mathematical graph in which most nodes are not neighbors of one another, but the neighbors of any given node are likely to be neighbors of each other and most nodes can be reached from every other node by a small number of hops or steps.Specifically, a small-world network is defined to be a network where the typical distance *L* between two randomly chosen nodes (the number of steps required) grows proportionally to the logarithm of the number of nodes *N* in the network.



Small-world network example
Hubs are bigger than other nodes
Average degree= 3.833
Average shortest path length = 1.803.
Clustering coefficient = 0.522

Construction of NSW graph:

The NSW graph is constructed via consecutive insertion of elements in random order by bidirectionally connecting them to the M closest neighbors from the previously inserted elements. The M closest neighbors are found using the structure's search procedure (a variant of a greedy search from multiple random enter nodes).Links to the closest neighbors of the elements inserted in the beginning of the construction later become bridges between the network hubs that keep the overall graph connectivity and allow the logarithmic scaling of the number of hops during greedy routing.

Key Idea:

The idea of Hierarchical NSW algorithm is to separate the links according to their length scale into different layers and then search in a multilayer graph. In this case we can evaluate only a needed fixed portion of the connections for each element independently of the networks size, thus allowing a logarithmic scalability. the search starts from the upper layer which has only the longest links. The algorithm greedily traverses through the elements from the upper layer until a local minimum is reached. After that, the search switches to the lower layer(which has shorter links), restarts from the element which was the local minimum in the previous layer and the process repeats. The maximum number of connections per element in all layers can be made constant, thus allowing a logarithmic complexity scaling of routing in a navigable small world network.

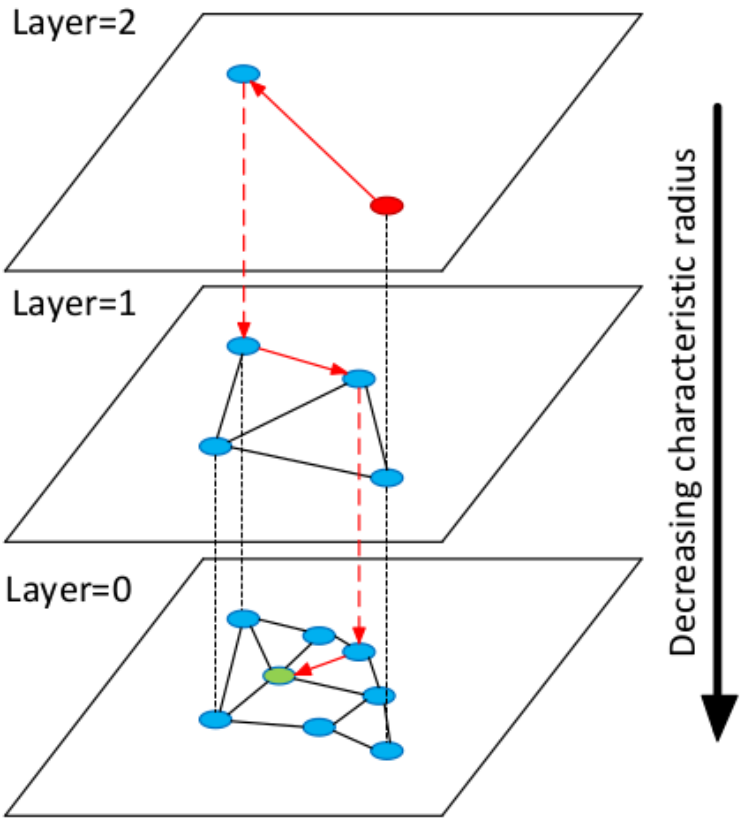


Fig. 1. Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

Algorithms:

Algorithm 1

INSERT($hnsw$, q, M, M_{max}, $efConstruction$, m_L)

Input: multilayer graph $hnsw$, new element q, number of established connections M, maximum number of connections for each element per layer M_{max}, size of the dynamic candidate list $efConstruction$, normalization factor for level generation m_L

Output: update $hnsw$ inserting element q

1 $W \leftarrow \emptyset$ // list for the currently found nearest elements
2 $ep \leftarrow$ get enter point for $hnsw$
3 $L \leftarrow$ level of ep // top layer for $hnsw$
4 $l \leftarrow \lfloor -\ln(unif(0..1)) \cdot m_L \rfloor$ // new element's level
5 **for** $l_c \leftarrow L \ldots l+1$
6 $W \leftarrow$ SEARCH-LAYER(q, ep, ef=1, l_c)
7 $ep \leftarrow$ get the nearest element from W to q
8 **for** $l_c \leftarrow \min(L, l) \ldots 0$
9 $W \leftarrow$ SEARCH-LAYER(q, ep, $efConstruction$, l_c)
10 $neighbors \leftarrow$ SELECT-NEIGHBORS(q, W, M, l_c) // alg. 3 or alg. 4
11 add bidirectionall connectionts from $neighbors$ to q at layer l_c
12 **for** each $e \in neighbors$ // shrink connections if needed
13 $eConn \leftarrow neighbourhood(e)$ at layer l_c
14 **if** $|eConn| > M_{max}$ // shrink connections of e
// if $l_c = 0$ then $M_{max} = M_{max0}$
15 $eNewConn \leftarrow$ SELECT-NEIGHBORS(e, $eConn$, M_{max}, l_c)
// alg. 3 or alg. 4
16 set $neighbourhood(e)$ at layer l_c to $eNewConn$
17 $ep \leftarrow W$
18 **if** $l > L$
19 set enter point for $hnsw$ to q

Algorithm 2

SEARCH-LAYER(q, ep, ef, l_c)

Input: query element q, enter points ep, number of nearest to q elements to return ef, layer number l_c

Output: ef closest neighbors to q

```
1  v ← ep      // set of visited elements
2  C ← ep      // set of candidates
3  W ← ep      // dynamic list of found nearest neighbors
4  while |C| > 0
5    c ← extract nearest element from C to q
6    f ← get furthest element from W to q
7    if distance(c, q) > distance(f, q)
8      break   // all elements in W are evaluated
9    for each e ∈ neighbourhood(c) at layer l_c   // update C and W
10     if e ∉ v
11       v ← v ⋃ e
12       f ← get furthest element from W to q
13       if distance(e, q) < distance(f, q) or |W| < ef
14         C ← C ⋃ e
15         W ← W ⋃ e
16         if |W| > ef
17           remove furthest element from W to q
18 return W
```

Algorithm 3

SELECT-NEIGHBORS-SIMPLE(q, C, M)

Input: base element q, candidate elements C, number of neighbors to return M

Output: M nearest elements to q

return M nearest elements from C to q

Algorithm 4

SELECT-NEIGHBORS-HEURISTIC(*q*, *C*, *M*, l_c, *extendCandidates*, *keepPrunedConnections*)

Input: base element *q*, candidate elements *C*, number of neighbors to return *M*, layer number l_c, flag indicating whether or not to extend candidate list *extendCandidates*, flag indicating whether or not to add discarded elements *keepPrunedConnections*

Output: *M* elements selected by the heuristic

```
1  R ← ∅
2  W ← C   // working queue for the candidates
3  if extendCandidates   // extend candidates by their neighbors
4   for each e ∈ C
5     for each e_adj ∈ neighbourhood(e) at layer l_c
6       if e_adj ∉ W
7         W ← W ⋃ e_adj
8  W_d ← ∅   // queue for the discarded candidates
9  while |W| > 0 and |R| < M
10   e ← extract nearest element from W to q
11   if e is closer to q compared to any element from R
12     R ← R ⋃ e
13   else
14     W_d ← W_d ⋃ e
15  if keepPrunedConnections  // add some of the discarded
                              // connections from W_d
16   while |W_d| > 0 and |R| < M
17     R ← R ⋃ extract nearest element from W_d to q
18 return R
```

Algorithm 5

K-NN-SEARCH(*hnsw, q, K, ef*)

Input: multilayer graph *hnsw*, query element *q*, number of nearest neighbors to return *K,* size of the dynamic candidate list *ef*

Output: *K* nearest elements to *q*

1 $W \leftarrow \emptyset$ // set for the current nearest elements

2 $ep \leftarrow$ get enter point for *hnsw*

3 $L \leftarrow$ level of *ep* // top layer for *hnsw*

4 **for** $l_c \leftarrow L \ldots 1$

5 $W \leftarrow$ SEARCH-LAYER(q, ep, ef=1, l_c)

6 $ep \leftarrow$ get nearest element from W to q

7 $W \leftarrow$ SEARCH-LAYER(q, ep, ef, l_c =0)

8 **return** K nearest elements from W to q