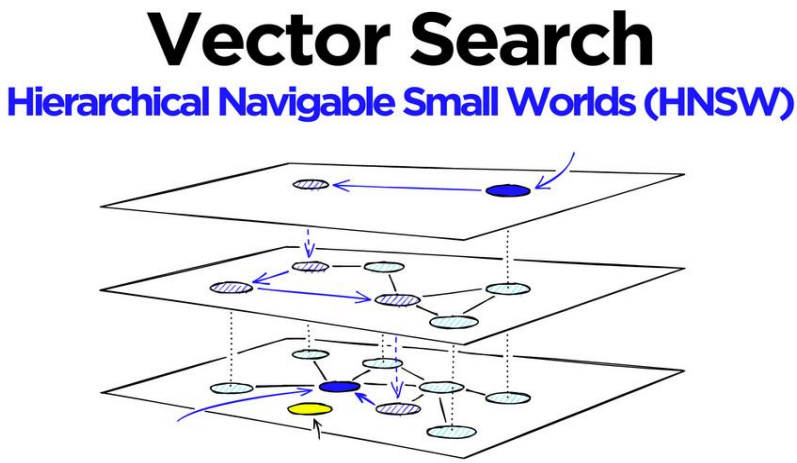
Cairo University
Faculty of Engineering

ADB Project Initial Design

Team 17

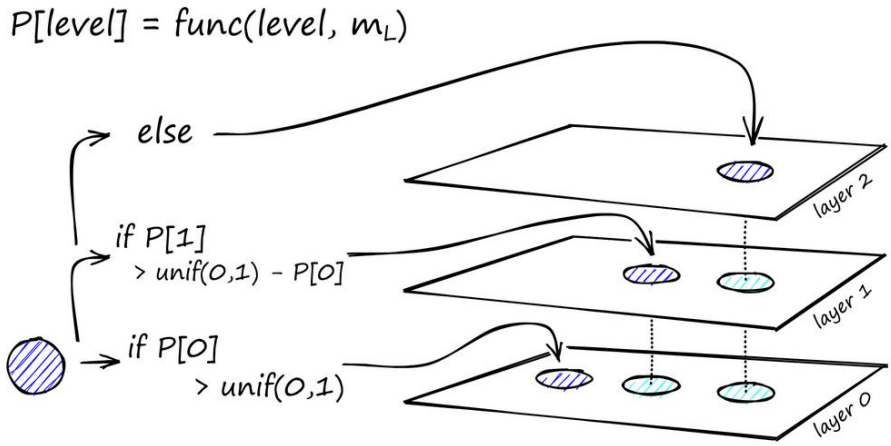
Name	Email
Youssef Khaled Al-Sayed	yousef.elsayed01@eng-st.cu.edu.eg
Moaz Mohamed Hassan	Moaz.Bayoumi00@eng-st.cu.edu.eg
Mohamed Nabil Abdelfattah	mohamed.nabi.5596@gmail.com
Ahmed Hosny Abdelrazik	ahmed.alghany01@eng-st.cu.edu.eg

Algorithm



We are applying an algorithm for vectors indexing called HSNW (Hierarchical Navigable Small Worlds).

First: Creating the index:



The index is a multi-layer graph, that is, upper layers contain nodes less than lower layers, bottom layer (0) has all the nodes in it (one graph).

A node is a record of (id, vector, list of node pointers).

In each layer, we place some nodes, based on a probability calculated for each layer, and comparing it with random value, when inserting a node into the index.

The probability of a vector insertion at a given layer is given by a probability function normalized by the 'level multiplier' m_L, when m_L is increased, we insert more nodes in upper layers, and when it is decreased, we push more nodes to be in lower layers.

Example:
Let P[layer 0] = 0.4, P[layer 1] = 0.3
When inserting a node into the index: compare P[layer 0] with random number between 0, 1, if the random number is 0.2, then insert the node into layer 0, and finish, else if the random number is 0.5, then it will be inserted in an upper layer.
At the upper layer 1: compare P[layer 1] with random number – P[layer 0], if the random number is 0.6, then P[layer 1] > 0.6 – 0.4 = 0.2, so, insert it in layer 1, and in all layers below current layer and finish.

For every node inserted in upper layer we insert it in all below layers.

Connection Between Nodes:

We connect between nodes based on cosine similarity.
Steps of connection:

1. for each layer:
2. for each node:
3. Calculate the distance between the node and other nodes in the layer and get the nearest M (maximum number of edges) nodes and connect between them

M will be hyper parameter that will be constant in each layer except for layer 0

Searching for a node:

During the search, we enter the top layer from the entry node. We get the cosine similarity between the search node and the entry node and its neighbors, greedily moving to the nearest node until we find a local minimum. at this point, we shift to the current node in the lower layer and begin searching again. We repeat this process until finding the local minimum of our bottom layer — *layer 0*.

In this explanation we assumed that we search for the nearest one node, we need to consider searching for K nodes as we when move to a lower layer move with multiple nodes (N) that allow us to move to the next layer with M (maximum number of edges) * N and we finish with the K in layer 0

Hyper parameters and trade-offs:

1. M_L (level multiplier): we control with it the number of nodes in each layer increasing it will make the search faster but makes overlap between layers and this will affect the storage
2. M_{max} and M_{max0}: the maximum number of edges of each node in layers and layer 0

Data structures:

1. Node class (id, vector, list of node ids)
   When loading the layer file into memory we have a dict <id, Node>

Note: we store the index in the disk and no need for storing the original data in separate file