Review

Bilal Akliai

June 2024

General idea : We choose a reference molecule v_q. The idea is to find the k molecules that are closest to it. For this, we will calculate the Tanimoto coefficient between v_q and each molecule. We will store our coefficients in a list. Finally, we will sort our list in descending order and choose the k molecules for which the Tanimoto coefficient with v_q is the highest.

Representation of compounds First of all, we have to represent molecules. For that, we can use fingerprints (sequence of bits, with 1s if a molecule meets a certain property and 0s otherwise) or graphs :

- In the case of fingerprints, we have to choose an efficient measure. The first research paper compared 8 of them, and concluded that the Tanimoto (percentage of bits in common) is the best. They also said that Chemaxon Chemical Fingerprint (an hashed fingerprint) is a good fingerprint (better than ECFP4).
- However, in the case of graphs we can be more efficient. We can use extended reduced graphs and graph editing distance. GED calculates a cost distance between two graphs, i.e., the minimum number of changes needed to transform one graph into another. Each modification can be one of six operations: insertion, deletion, and substitution for nodes and edges of the graph.

HSNW structure : Finally, we need to store our molecules in the right way to be effective in our research. The idea is to build an HSNW index using the Tanimoto coefficient as the measure. HSNW then organizes the vectors in a hierarchical graph structure that allows for fast nearest neighbor searches. We can thus efficiently find the k compounds that most resemble our reference compound. We will use the HSNW index to search for the compounds most similar to the query compound in terms of the Tanimoto coefficient. Once the similar compounds are found, we will sort the compounds according to their Tanimoto coefficient in descending order and return the top k most similar compounds. Thanks to this structure, we can find an element in $O(\log(N))$, and add an element in $O(N \log(N))$ where N is the number of elements in the graph. It is more efficient that the NSW structure which is polylogarithmic. The difference between these two structures is that with NSW, we only mix the

elements before inserting them, after which each inserted element is connected to its k nearest neighbors. However, to build an HSNW graph, we also use a hierarchy of layers. So, for each element to be inserted, we draw a random number that corresponds to the maximum layer in which we can insert the element. Insert the element at the determined maximum layer, connecting it to the nearest neighbors already present at that layer. Move down the layers, connecting the element to its nearest neighbors at each layer. Of course, we use the Tanimoto coefficient to determine the nearest neighbors for the new element at each layer. In order to find an element, we begin the search from the highest layer where elements are connected. Then, we use a greedy search algorithm to move through the layers: at each layer, we move to the closest neighbor of the current element, and we continue this process until reaching the lowest layer. So, changing levels is like changing scales: we zoom in as we go.

Next steps + questions:

- Think to how to compute fingerprints. For example, with a molecule in RdKit, we want to compute it Chemaxon Chemical Fingerprint, or choose another fingerprint...
- Implement a first version of HSNW using Tanimoto coefficients. HSNW is implemented in C++ and Python and support for exemple l2 distance. So, we can try to adapt the existing code to support Tanimoto coefficient distance (code from zero, or try to re-use their code ?).
- Implement a second version using GED distance : how to define costs ? how to compute the graph edit distance ? (maybe we can use the Networkx library) how to optimize it ? (maybe we can use the algorithm called Bipartite).