Review 3

Bilal Akliai

20 June 2024

What I've changed in my algo

After plotting a few curves to see which algorithm was the most expensive, I quickly noticed that I needed to optimize search_layer (which finds the k nearest neighbors): it is useful for the search phase, as well as for the insertion phase. I therefore tried to optimize this algorithm as much as possible, and opted for the use of a heap structure for the list of candidates and the result list: this saves an enormous amount of time, as we have directly sorted lists. So, as we often search for the maximum and minimum of these lists, we have them directly and don't have to recalculate them (which would be very costly as we'd have to recalculate all the distances from the q element to be inserted: for example, c = min(C, key=lambda e: self.distance(e, q)) become c = C[0]).

Results

We get pretty good results for the search phase: for example, for 1 million molecules (50-NN) it takes about 6ms with HNSW and nearly 2s without. However, the problem we're quickly faced with is that to be able to carry out more substantial examples, we need to optimize the memory (for example, for more than 4 or 5 million molecules, my PC crashes as soon as the files are read (even before the insertion starts)), and we absolutely must reduce the insertion time, otherwise we won't really be able to process many molecules. For example, here with M=10, ef=150 we're looking for the 50 closest molecules out of a million, and while we get a good result (98% correct), it takes far too long (almost 2 hours).

But why is the insertion too long ? Let's analyze per algorithm !

The insertion phase is very long, so we must at least try to reduce its time, and even optimize it if we can. We can see that the most time-consuming algorithms for this insertion phase are insert, which uses search_layer, which in turn uses distance. So, either reduce the number of calls to search_layer, or reduce the number of calls to 'distance'.

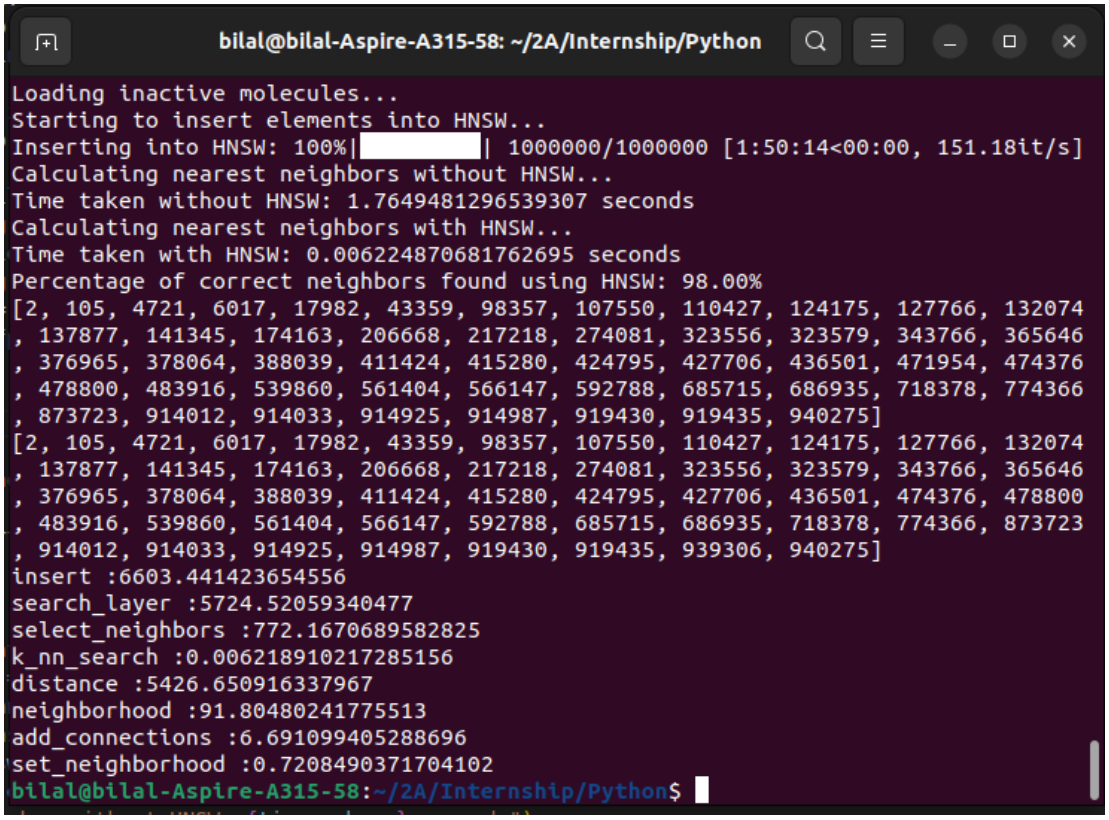

Figure 1: Percentage of correct neighbors + time taken with/without HNSW, 1M compounds, 50-NN, M=10, ef=150

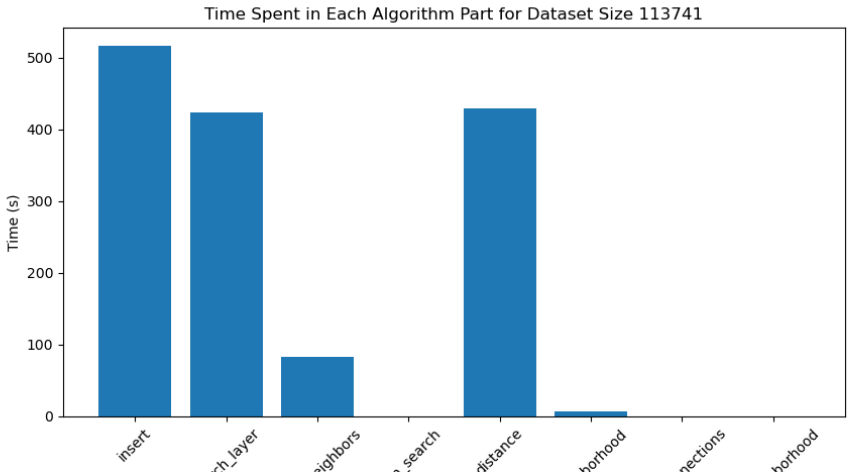

Figure 2: Time taken per algorithm during the insertion phase, 110k compounds, 50-NN, M=10, ef=150

What I tried I tried to reduce the calls to the distance function in the search_layer algorithm by trying to use triangular inequality, but it didn't work. I also tried to use an inequality I read about in a research paper (An Intersection Inequality Sharper than the Tanimoto Triangle Inequality for Efficiently Searching Large Databases) but unfortunately it's not usable because it's the wrong way. I think a good approach would be to try not to reduce calls to "distance" but to approximate it (so that we only have to call it when necessary).

I've also thought of storing Tanimoto distances already calculated in a dictionary, but that's too memory-intensive and not really useful.

The parameters (ef and M) have a major impact on the algorithm, both in terms of time and the performance of correct results. We therefore need to play with these parameters (I've tried to choose parameters that reconcile the two). I think, as suggested in the paper, that we should try to adapt these parameters in the algorithm (e.g. small ef and M values when neighbors are far away, and large values when they are close).