Review 5

Bilal Akliai

11 Jul 2024

1 NN-Descnent Algorithm

I tried to fix the problem of the NN-Descnent Algorithm. I think my algorithm is now correct. It works very well for a small number of coumpounds: it finds the k-nearest neighbors. However, when there is more compounds it becomes less efficient. Moreover, in terms of time it is not much better than what we have currently.

It takes around 40s for 30k compunds. However, the precision is sometimes good (84%, Figure 1) and sometimes very bad (0% for example, Figure 2).

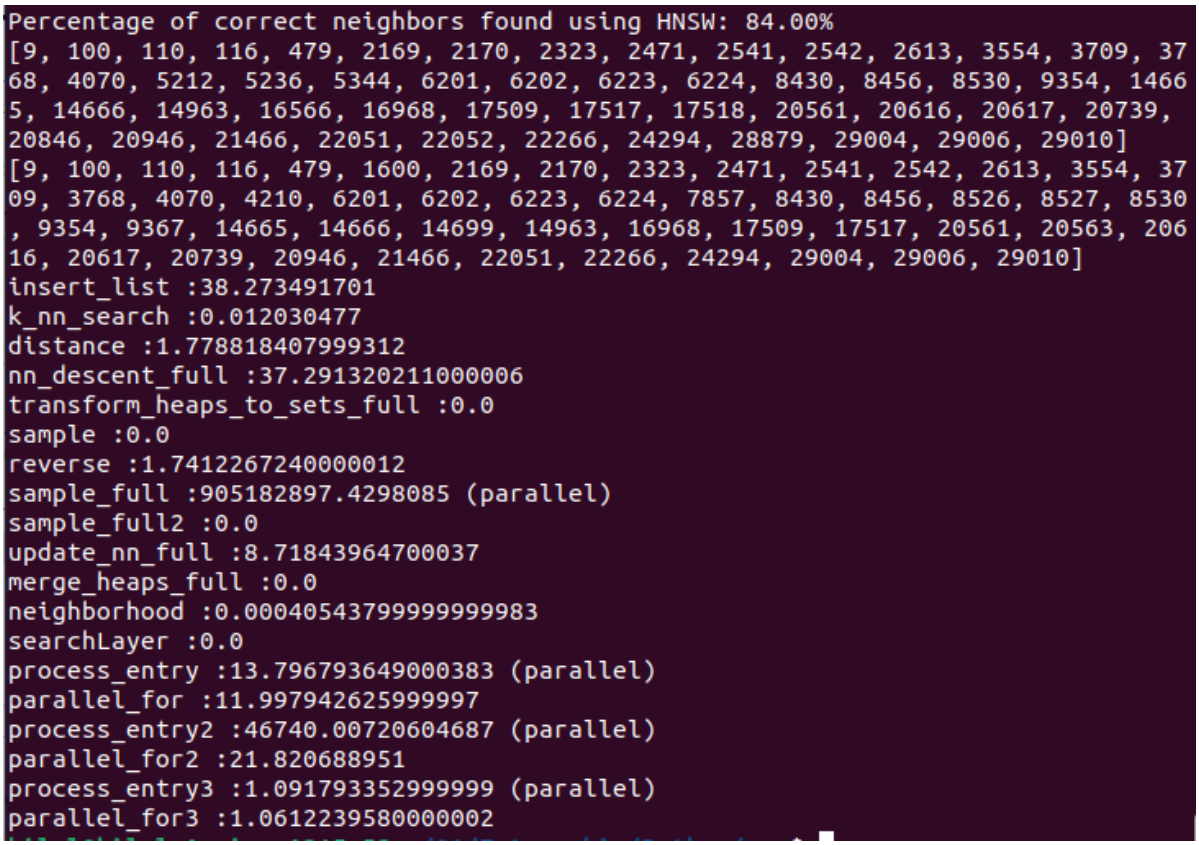

Figure 1: 50-nearest neighbors using NN-Descnent Algorithm for 30k compounds

So, for the insertion I began to recode what we had in Python (without the NN-Descnent Algorithm).

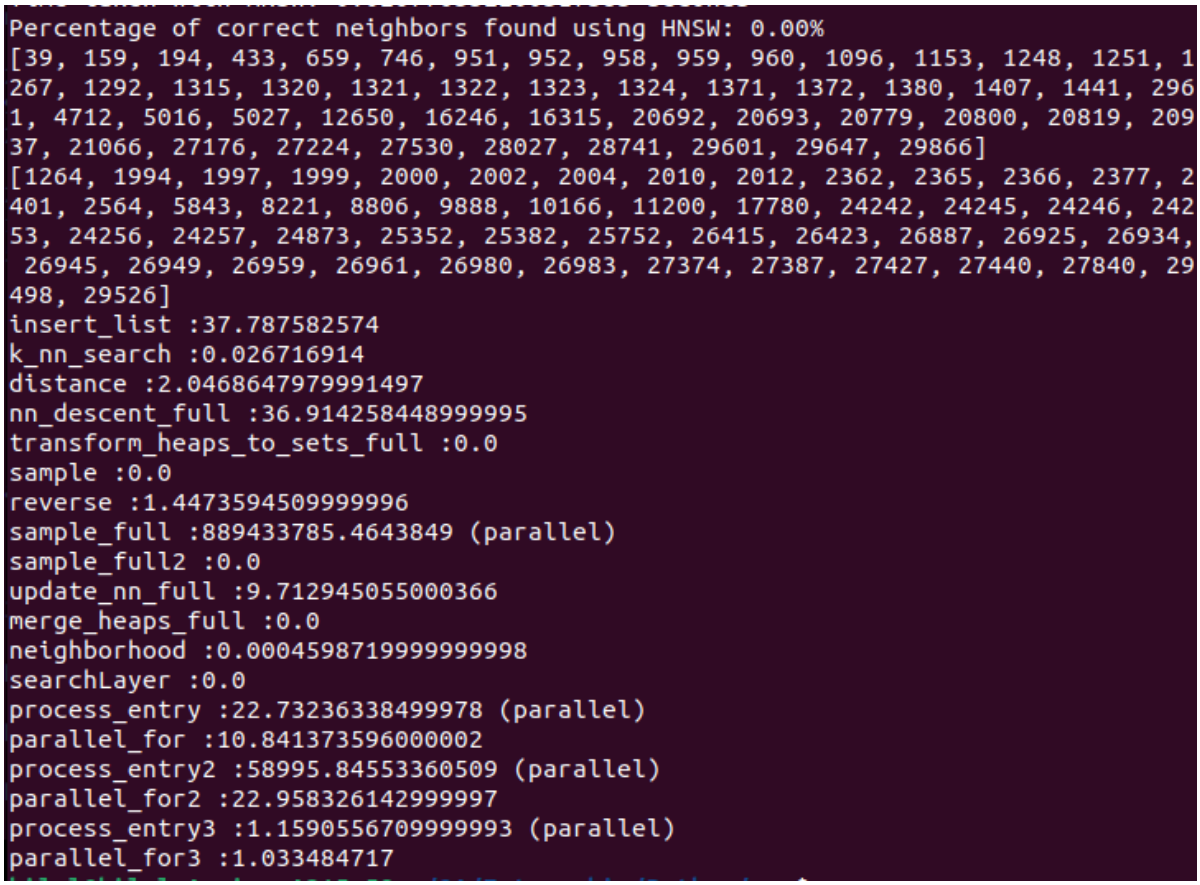

Figure 2: 50-nearest neighbors using NN-Descnent Algorithm for 30k compounds (not the same as for the Figure 1)

2 Partial parallelizing

I managed to improve the insertion time by partially parallelizing the algorithm. My idea is not to insert elements one by one, but in groups. For example, let's say I've already inserted 30k elements. I'm going to insert 100 in parallel: I'll search for the k nearest neighbors of each of the 100 molecules (with search_layer) in parallel. The problem is that for the 30100th element, I'm going to search for its nearest neighbors among the first 30k (because the elements from 30001 to 30099 are not yet inserted as it is in parallel) and not among the first 30099. So I'll also select closest molecules to j for i¡j from my group of 100 (and I can also do it in parallel !).

This improves time quite a bit. For example, for 300k molecules, we go from an insertion time of 430s (about 7 minutes, Figure 3) to 333s (22% better, about 5 minutes, Figure 4).

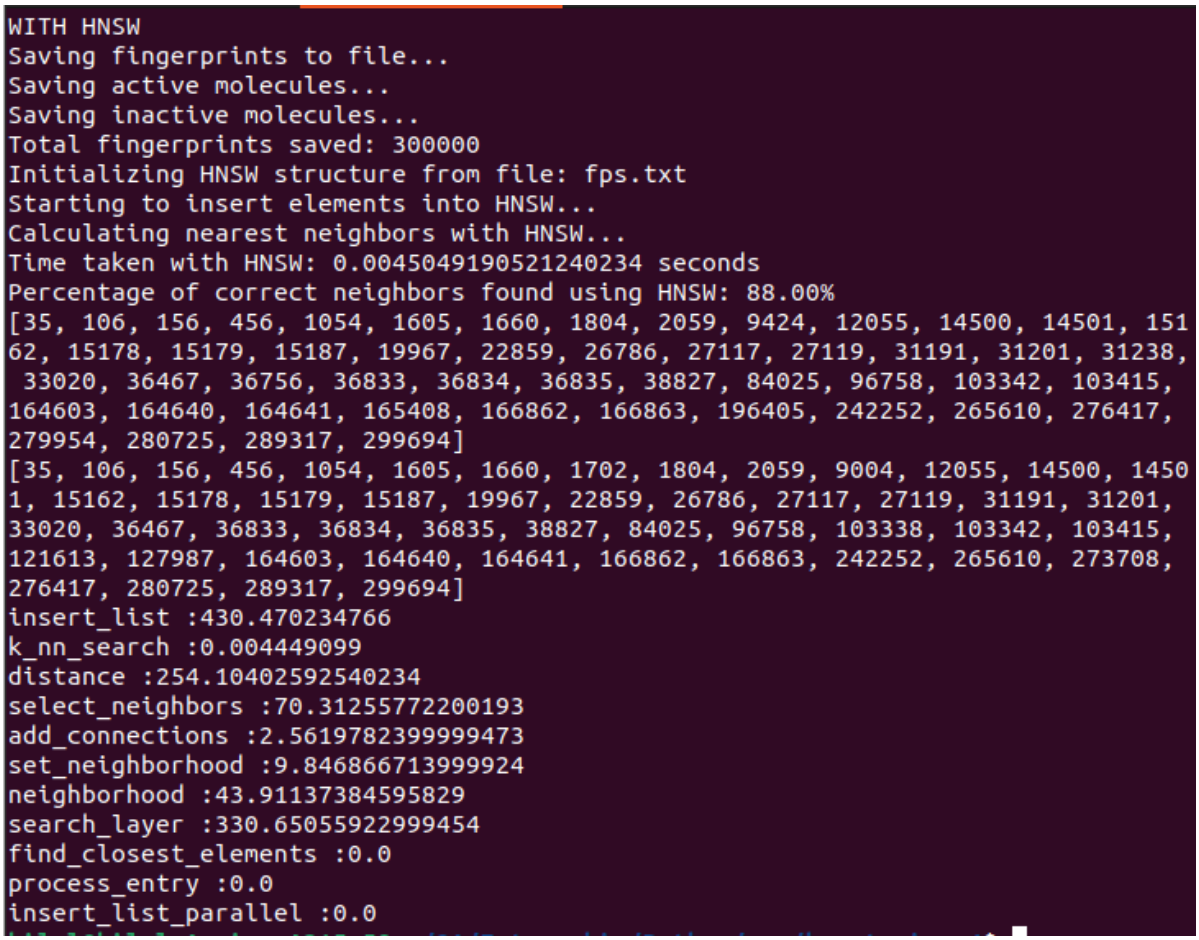

Figure 3: 50-nearest neighbors for 300k compounds with M=15, Mmax=2*M, efConstruction=150 and WITHOUT PARALLELISATION

3 Memory

I read and used the paper that you suggested. Its reduces memory space by a factor of 2 (Figure 5): we work now with 128 'char' for each fingerprint instead of 2048 bits. Moreover, it don't affect a lot the precision. However, it increases the time because bit-by-bit operations are more efficient, and in our new distance there's the presence of conditioning (if..) which increases calculation time.

```
Calculating nearest neighbors with HNSW...
Time taken with HNSW: 0.005311250686645508 seconds
Percentage of correct neighbors found using HNSW: 86.00%
[35, 106, 156, 456, 1054, 1605, 1660, 1804, 2059, 9424, 12055, 14500, 14501, 151
62, 15178, 15179, 15187, 19967, 22859, 26786, 27117, 27119, 31191, 31201, 31238,
 33020, 36467, 36756, 36833, 36834, 36835, 38827, 84025, 96758, 103342, 103415,
164603, 164640, 164641, 165408, 166862, 166863, 196405, 242252, 265610, 276417,
279954, 280725, 289317, 299694]
[35, 106, 156, 456, 1054, 1605, 1660, 1804, 2059, 9004, 12055, 14501, 15162, 151
78, 15179, 15187, 15474, 17666, 22859, 26786, 27117, 27119, 31191, 31201, 31238,
 33020, 36467, 36833, 36834, 36835, 38827, 84025, 103338, 103342, 103415, 104577
, 127987, 164603, 164640, 165408, 165768, 166862, 166863, 196405, 242252, 265610
, 276417, 279954, 280725, 299694]
insert_list :333.583453617
k_nn_search :0.005245678
distance :4103.485027739492
select_neighbors :366.0712510119928
add_connections :3.916807528000031
set_neighborhood :0.0
neighborhood :512.3832887785495
search_layer :6535.390513604887
find_closest_elements :422.59810915500475
process_entry :7829.7803113240325
insert_list_parallel :333.444327325
```

Figure 4: 50-nearest neighbors for 300k compounds with M=15, Mmax=2*M, efConstruction=150 and WITH PARALLELISATION

```
bilal@bilal-Aspire-A315-58:~/2A/Internship/Python/cpp/boost_mix_v4$ g++ -o test
test.cpp
bilal@bilal-Aspire-A315-58:~/2A/Internship/Python/cpp/boost_mix_v4$ ./test
Tanimoto similarity: 0.9322033898
Elapsed time for Tanimoto: 0.0000019920 seconds
Reduced fingerprints distance: 0.9375000000
Elapsed time for reduced fingerprints: 0.0000040850 seconds
bilal@bilal-Aspire-A315-58:~/2A/Internship/Python/cpp/boost_mix_v4$
```

Figure 5: Comparison of a calculation with the tanimoto distance on 2048bits, and its approximation on 128 'char' (=1024 bits)

4 For more molecules

I tried to reduce one of hnsw's parameters (number of nearest neighbors searched in search_layer). This makes the algorithm more efficient: 1M inserted in about 20 minutes (Figure 6). However, it reduces precision. I think I'll have to adapt this factor to the density (search for more neighbors only when necessary, in order to keep a good insertion time and a good precision).

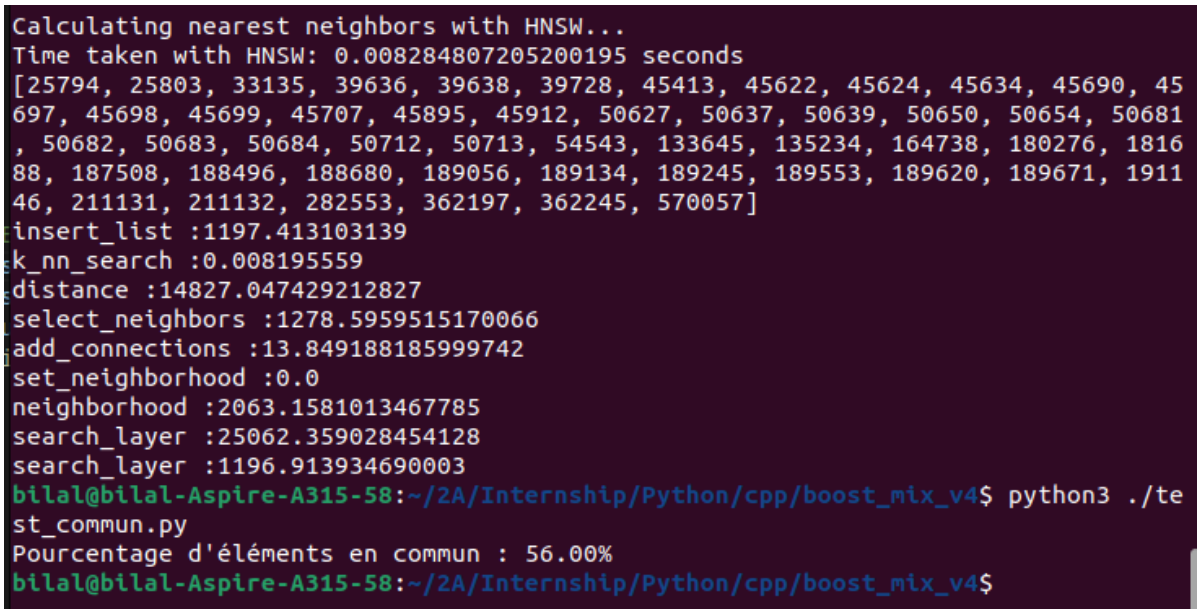

Figure 6: 50-nearest neighbors for 1M compounds with M=15, Mmax=2*M, efConstruction=150 and with parallelisation

5 Objectives

I think I have to improve the time taken by the distance (because it can improve a lot the insertion and the search time). I have also to optimize the search_layer (because it can improve the search time which is about 8ms. Moreover a small improvement of that time can also improve a lot the insertion time).