

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).

```

Percentage of correct neighbors found using HNSW: 84.00%
[9, 100, 110, 116, 479, 2169, 2170, 2323, 2471, 2541, 2542, 2613, 3554, 3709, 37
68, 4070, 5212, 5236, 5344, 6201, 6202, 6223, 6224, 8430, 8456, 8530, 9354, 1466
5, 14666, 14963, 16566, 16968, 17509, 17517, 17518, 20561, 20616, 20617, 20739,
20846, 20946, 21466, 22051, 22052, 22266, 24294, 28879, 29004, 29006, 29010]
[9, 100, 110, 116, 479, 1600, 2169, 2170, 2323, 2471, 2541, 2542, 2613, 3554, 37
09, 3768, 4070, 4210, 6201, 6202, 6223, 6224, 7857, 8430, 8456, 8526, 8527, 8530
, 9354, 9367, 14665, 14666, 14699, 14963, 16968, 17509, 17517, 20561, 20563, 206
16, 20617, 20739, 20946, 21466, 22051, 22266, 24294, 29004, 29006, 29010]
insert_list :38.273491701
k_nn_search :0.012030477
distance :1.778818407999312
nn_descent_full :37.291320211000006
transform_heaps_to_sets_full :0.0
sample :0.0
reverse :1.7412267240000012
sample_full :905182897.4298085 (parallel)
sample_full2 :0.0
update_nn_full :8.71843964700037
merge_heaps_full :0.0
neighborhood :0.00040543799999999983
searchLayer :0.0
process_entry :13.796793649000383 (parallel)
parallel_for :11.997942625999997
process_entry2 :46740.00720604687 (parallel)
parallel_for2 :21.820688951
process_entry3 :1.0917933529999999 (parallel)
parallel_for3 :1.0612239580000002

```

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).

```

Percentage of correct neighbors found using HNSW: 0.00%
[39, 159, 194, 433, 659, 746, 951, 952, 958, 959, 960, 1096, 1153, 1248, 1251, 1
267, 1292, 1315, 1320, 1321, 1322, 1323, 1324, 1371, 1372, 1380, 1407, 1441, 296
1, 4712, 5016, 5027, 12650, 16246, 16315, 20692, 20693, 20779, 20800, 20819, 209
37, 21066, 27176, 27224, 27530, 28027, 28741, 29601, 29647, 29866]
[1264, 1994, 1997, 1999, 2000, 2002, 2004, 2010, 2012, 2362, 2365, 2366, 2377, 2
401, 2564, 5843, 8221, 8806, 9888, 10166, 11200, 17780, 24242, 24245, 24246, 242
53, 24256, 24257, 24873, 25352, 25382, 25752, 26415, 26423, 26887, 26925, 26934,
26945, 26949, 26959, 26961, 26980, 26983, 27374, 27387, 27427, 27440, 27840, 29
498, 29526]
insert_list :37.787582574
k_nn_search :0.026716914
distance :2.0468647979991497
nn_descent_full :36.914258448999995
transform_heaps_to_sets_full :0.0
sample :0.0
reverse :1.4473594509999996
sample_full :889433785.4643849 (parallel)
sample_full2 :0.0
update_nn_full :9.712945055000366
merge_heaps_full :0.0
neighborhood :0.0004598719999999998
searchLayer :0.0
process_entry :22.732363384999978 (parallel)
parallel_for :10.841373596000002
process_entry2 :58995.84553360509 (parallel)
parallel_for2 :22.958326142999997
process_entry3 :1.1590556709999993 (parallel)
parallel_for3 :1.033484717

```

Figure 2: 50-nearest neighbors using NN-Descnet 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 ij 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).

```

WITH HNSW
Saving fingerprints to file...
Saving active molecules...
Saving inactive molecules...
Total fingerprints saved: 300000
Initializing HNSW structure from file: fps.txt
Starting to insert elements into HNSW...
Calculating nearest neighbors with HNSW...
Time taken with HNSW: 0.0045049190521240234 seconds
Percentage of correct neighbors found using HNSW: 88.00%
[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, 1702, 1804, 2059, 9004, 12055, 14500, 1450
1, 15162, 15178, 15179, 15187, 19967, 22859, 26786, 27117, 27119, 31191, 31201,
33020, 36467, 36833, 36834, 36835, 38827, 84025, 96758, 103338, 103342, 103415,
121613, 127987, 164603, 164640, 164641, 166862, 166863, 242252, 265610, 273708,
276417, 280725, 289317, 299694]
insert_list :430.470234766
k_nn_search :0.004449099
distance :254.10402592540234
select_neighbors :70.31255772200193
add_connections :2.5619782399999473
set_neighborhood :9.846866713999924
neighborhood :43.91137384595829
search_layer :330.65055922999454
find_closest_elements :0.0
process_entry :0.0
insert_list_parallel :0.0

```

Figure 3: 50-nearest neighbors for 300k compounds with $M=15$, $M_{max}=2*M$, $efConstruction=150$ and WITHOUT PARALLELISATION

3 Memory

I read and used the paper that you suggested. It 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 doesn'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 $M=15$, $M_{max}=2*M$, $efConstruction=150$ 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).

```

Calculating nearest neighbors with HNSW...
Time taken with HNSW: 0.008284807205200195 seconds
[25794, 25803, 33135, 39636, 39638, 39728, 45413, 45622, 45624, 45634, 45690, 45
697, 45698, 45699, 45707, 45895, 45912, 50627, 50637, 50639, 50650, 50654, 50681
, 50682, 50683, 50684, 50712, 50713, 54543, 133645, 135234, 164738, 180276, 1816
88, 187508, 188496, 188680, 189056, 189134, 189245, 189553, 189620, 189671, 1911
46, 211131, 211132, 282553, 362197, 362245, 570057]
insert_list :1197.413103139
k_nn_search :0.008195559
distance :14827.047429212827
select_neighbors :1278.5959515170066
add_connections :13.849188185999742
set_neighborhood :0.0
neighborhood :2063.1581013467785
search_layer :25062.359028454128
search_layer :1196.913934690003
bilal@bilal-Aspire-A315-58:~/2A/Internship/Python/cpp/boost_mix_v4$ python3 ./te
st_commun.py
Pourcentage d'éléments en commun : 56.00%
bilal@bilal-Aspire-A315-58:~/2A/Internship/Python/cpp/boost_mix_v4$

```

Figure 6: 50-nearest neighbors for 1M compounds with $M=15$, $M_{\max}=2*M$, $efConstruction=150$ 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).