Michał Nowotka[1], Anne Hersey[1]

[1] European Molecular Biology Laboratory - European Bioinformatics Institute
Email: mnowotka@ebi.ac.uk

EMBL-EBI

# Locality Sensitive Hashing for compound similarity search.

## Big Data Challenge

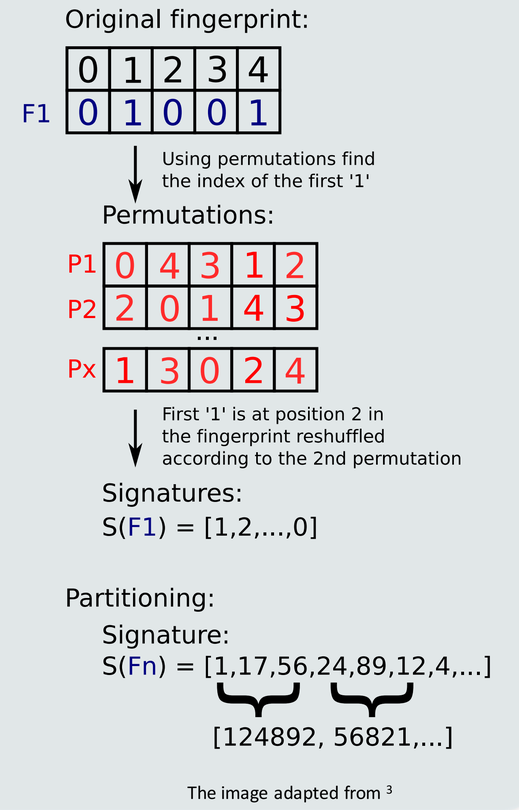
Along with the development of computer mass storage and advancements of high-throughput and virtual screening, databases storing chemical content are growing at a rapid pace. Entering the Big Data territory can cause performance issues when executing essential cheminformatics methods, such as searching a database for similar compounds.

The Locality Sensitive Hashing (LSH) is a popular technique used in audio fingerprinting. The idea behind the LSH is to reduce the dimensionality of data by hashing input items in order to maximize the probability of hash collisions for similar documents. The applicability and feasibility of this method for compound similarity searching has not yet been fully investigated. There are currently two research papers[1,2], directly describing the application of LSH to chemical similarity but none of them aims at providing a working solution or a prototype.

The LSH approach seems to be particularly interesting when used in combination with NoSQL key-value stores. These offer performance that is better than traditional RDMBS and usually don't allow any extensions (cartridges) to be executed on the server side, but instead offer a rich repertoire of ad hoc queries through which the LSH technique can be easily implemented.

## LSH technique

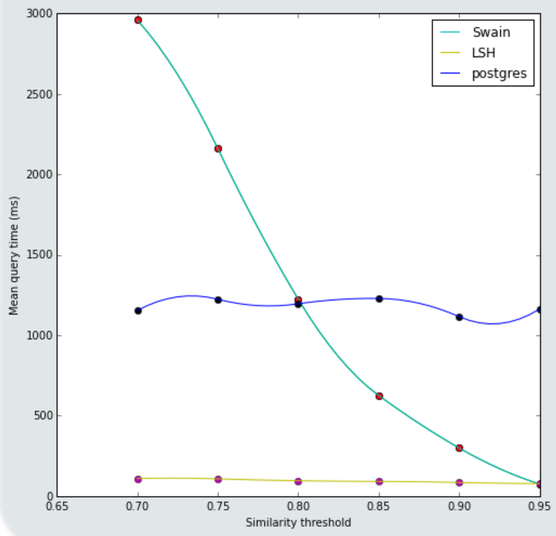
- Generate X random permutations of the length of a fingerprint.
- For each fingerprint: find the position of the first '1' bit in each permutation.
- Divide each signature into Y chunks. Using a left shift binary operation convert each chunk into a single number.
- Create Y different dictionaries with keys corresponding to numbers in chunks and values corresponding to compound ids.
- For the query compound use the method to generate chunked signature. For each chunk check the respective dictionary for the compounds at this key. Get the distinct set of all compounds found by this process for all Y chunks. On this set, compute Tanimoto distance and return compounds with a similarity score higher than the given threshold.




The image adapted from [3]

## Time complexity

After implementing the LSH algorithm in Python and calling it from an IPython notebook, a series of benchmarks has been performed (measuring a mean time of 5 repetitions of searching a sample of 1k random compounds against 1455713 compounds from ChEMBL 20 for each discrete threshold value). In the same way the RDKit postgreSQL cartridge performance has been measured. The performance was compared with the RDKit cartridge and with the state of art MongoDB based solution described and improved by various authors (Swain et al.) [4,5,6,7]. The LSH software was expecting SMILES strings as an input in contrast to the Swain's script, which was using RDKit binary molecule objects. This is why the script was modified and rerun on the same hardware.




**Implementation details:**

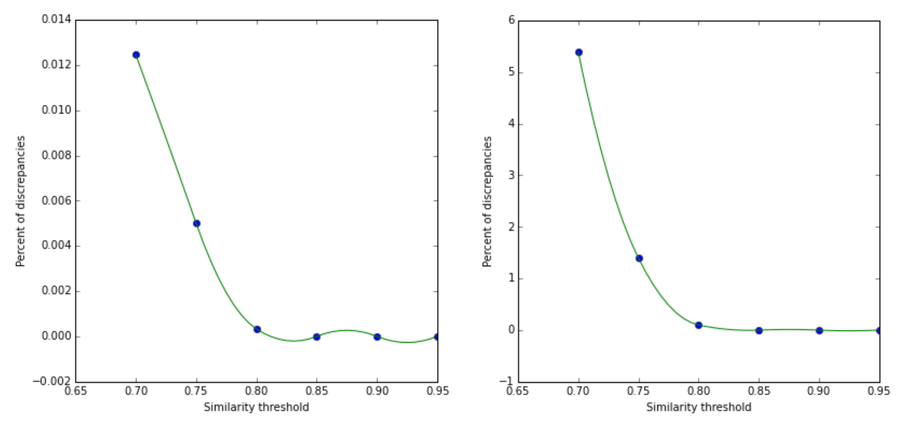
The benchmarks used Morgan 2048-bit fingerprints. The code was executed using a Docker container with Ubuntu 14.04 as host OS.

**Hardware:**

The container was running on a Gigabyte BRIX machine with an Intel Core i7-3537U 2GHz CPU, 16GB RAM and 256GB SSD.

## Accuracy

The same sample of 1k compounds was used to compare results obtained by using LSH-hash and standard folded fingerprints. For each pair of results, the size of symmetric set difference between results in the pair has been counted and divided by the size of sum (union) of both result sets. The same procedure was repeated for different threshold values. As an alternative error measure, the number of compounds, for which any discrepancy in results has been observed was counted (regardless of its size) and divided by the number of compounds in sample (1k).




As can be noticed, for thresholds greater or equal to 0.8 the results are almost the same. For threshold 0.7, less than 6% of query compounds have different results. The result of LSH similarity is always a subset of the original method.

## Further improvements

There are a number of improvements that can be explored further:

- Generating permutations for LSH: currently, the standard numpy random function generating uniformly distributed numbers was used. It would be interesting to check if using different probability distributions can give better results. A paper by Baluja et al.[8] suggests this strategy can lead to significant improvements.
- Investigating how performance/accuracy changes in a function of number of permutations (X) and buckets (Y) used.
- Checking different types of fingerprints, fingerprint lengths and the fact that they are folded/unfolded.
- Experimenting with different database engines including relational ones.

## Code

The code implemented as an IPython notebook can be downloaded from the GitHub repository using the following URL: https://github.com/chembl/mychembl/blob/master/ipython_notebooks/XX_mongoDB_similarity_search.ipynb. The live version of the Ipython notebook will be available in future releases of myChEMBL.



After completing all those preliminary steps we can finally define our new similarity_search_lsh function:

```
In [13]: permutations = [p['permutation'] for p in db.permutations.find()]

         def similarity_search_lsh(smiles, threshold=0.8):
             """Perform a similarity search using aggregation framework."""
             if not smiles:
                 return
             mol = Chem.MolFromSmiles(smiles)
             if not mol:
                 return
             fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)
             qfp = list(fp.GetOnBits())
             qfp_bits = [int(n) for n in fp]
             min_hash = []
             for perm in permutations:
                 for idx, i in enumerate(perm):
                     if qfp_bits[i]:
                         min_hash.append(idx)
                         break
             hash_groups = hash_to_buckets(min_hash)
             qn = len(qfp)                       # Number of bits in query fingerprint
             qmin = int(ceil(qn * threshold))    # Minimum number of bits in results fingerprints
             qmax = int(qn / threshold)          # Maximum number of bits in results fingerprints
             ncommon = qn - qmin + 1             # Number of fingerprint bits in which at least 1 must
         be in common
             if db.mfp_counts:
                 reqbits = [count['_id'] for count in db.mfp_counts.find({'_id': {'$in': qfp}}).sort('count
         ', 1).limit(ncommon)]
             else:
                 reqbits = qfp[:ncommon]

             nested_res = [ list(i)[0]['molecules'] for i in
                     [db['hash_' + str(i)].find({'_id':h},{'molecules':1}) for i,h in enumerate(hash_groups
         )]]

             hashed_ids = [ObjectId(x) for x in {set([str(item) for sublist in nested_res for item in subli
         st])}]
             aggregate = [
                 {'$match': {'_id':{'$in': hashed_ids}, 'mfp.count': {'$gte': qmin, '$lte': qmax}, 'mfp.bit
         s': {'$in': reqbits}}},
```

## References and Acknowledgements

1) Dutta, D., Guha, R., Jurs, P.C., Chen, T., Scalable partitioning and exploration of chemical spaces using geometric hashing. J Chem Inf Model. DOI: 10.1021/ci050403o (2006).
2) Nasr, R., Vernica R., Li, C., Baldi, P. Speeding Up Chemical Searches Using the Inverted Index:The Convergence of Chemoinformatics and Text Search Methods. J. Chem. Inf. Model. DOI: 10.1021/ci200552r (2012)
3) Ciumac, S., Duplicate songs detector via audio fingerprinting, URL: http://www.codeproject.com/Articles/206507/Duplicates-detector-via-audio-fingerprinting#worksinspiration (retrieved 28 June 2016)
4) Swain, M., Chemical similarity search in MongoDB, URL: http://blog.matt-swain.com/post/87093745652/chemical-similarity-search-in-mongodb (retrieved 28 June 2016)
5) Suvee, D., The joy of algorithms and NoSQL: a MongoDB example, URL: http://datablend.be/?p=254 (retrieved 28 June 2016)
6) Guha, R., Fingerprint Similarity Searches in MongoDB, URL: http://blog.rguha.net/?p=1261 (retrieved 28 June 2016)
7) Swamidass, S.J., Baldi, P. Bounds and Algorithms for Fast Exact Searches of Chemical Fingerprints in Linear and Sublinear Time. J. Chem. Inf. Model. DOI:10.1021/ci600358f (2007).
8) Baluja, S., et al., "Permutation Grouping: Intelligent Hash Function Design for Audio & Image Retrieval," Proc. ICASSP, DOI: 10.1109/ICASSP.2008.4518065 (2008).



wellcome trust Strategic Award

EMBL-EBI

EMBL-EBI
Wellcome Trust Genome Campus
Hinxton, Cambridgeshire, CB10 1SD, UK

Tel. +44 (0) 1223 494 444
comms@ebi.ac.uk
www.ebi.ac.uk