Stockholms universitet

Neighborhood Correlation in the age of megagenomics: scaling down to scale up

Yrin Eldfjell

Masteruppsats i datalogi
Master Thesis in Computer Science

Neighborhood Correlation in the age of megagenomics: scaling down to scale up

Yrin Eldfjell

December 2, 2023

Abstract

The computational problem of protein homology inference is simultaneously facing the challenges of handling dramatically increasing amounts of sequenced species and maintaining – or preferably improving – both sensitivity and specificity. *Neighborhood Correlation (NC)* – a key method in improving both sensitivity and specificity when inferring homology between multidomain proteins – can in its current form not scale up to meet these needs, much due to its computational cost increasing not only directly with the size of the query set (Q) but also indirectly with the size of Q, as the method relies on comparing Q with itself.

This project therefore explores the possibility of using a much smaller set of reference sequences that Q is compared to. The results show that this is – according to the available metrics – a feasible strategy with comparable classification performance which is expected to lead to a significant speed-up. To further help bridge the gap between the increasing needs and the capacity of NC, an algorithm for computing Neighborhood Correlation with improved time complexity over existing algorithms is presented. This improved algorithm – PNC – should be easy to implement as a distributed computation.

Sammanfattning

Beräkningsproblemet att härleda evolutionärt släktskap för proteiner står inför dubbla utmaningar: att hantera en dramatisk ökning av antalet sekvenserade arter och att samtidigt bibehålla – eller helst förbättra – sensitivitet och specificitet. *Neighborhood Correlation (NC)* – en central metod för att förbättra både sensitivitet och specificitet vid inferens av släktskap mellan multidomänproteiner – kan i sin nuvarande form inte skalas upp för att möta dessa behov, mycket beroende på att dess beräkningskostnad inte bara ökar direkt med storleken på frågemängden (Q) utan även indirekt med storleken på Q, eftersom att Q jämförs med sig själv.

I detta projekt utforskas därför möjligheten att använda en mycket mindre mängd referenssekvenser som Q jämförs med. Resultaten visar att detta – enligt tillgänliga mått – är en realistisk strategi med jämförbar förmåga att klassificera homologer och som väntas leda till betydligt snabbare beräkningar. För att ytterligare minska gapet mellan de ökande behoven och NC:s förmåga, presenteras här även en algoritm för beräkna Neighborhood Correlation med förbättrad tidskomplexitet jämfört med befintliga algoritmer. Denna förbättrade algoritm – PNC – bör även vara enkel att implementera som en distribuerad beräkning.

Acknowledgments

I would like to thank my thesis advisor Lars Arvestad for always taking his time to bounce ideas and answer questions, my family for supporting me, everyone who has built our civilization to the point where work like this is possible[1] and finally all those whose curiosity, creativity and compassion make life a worthwhile endeavor.

[1] I'm sorry I couldn't include everyone in the bibliography.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Homology inference – the identification of genes sharing a common ancestry – is of profound importance in biology. Applications include gene annotation, function prediction and of course phylogenetic inference (Song et al. 2008).

It is predicted that there are on the order of 10 million eukaryote species in total (Mora et al. 2011). Identifying homologs in 10 million species with an assumed 10000 proteins each would with $O(N^2)$ scaling require 10^{22} computational steps, thus posing a significant challenge even for exascale computers. This is not only a theoretical challenge, however – already by the end of this decade, the Earth BioGenome Project aims to have sequenced about 1.5 M genomes representative of all eukaryotic species (Lewin et al. 2018) – highlighting the urgent need for homology inference methods that can handle such vast amounts of sequences.

Neighborhood Correlation Parallel to the problem of achieving computational efficiency is the issue of correctly identifying multidomain proteins. They are important and abundant – about 40% of proteins in metazoan (animal) species have more than one domain (Tordai et al. 2005). Multidomain proteins can arise from *gene duplication* or *speciation*, but also from of the *insertion* of an unrelated domain (see *Glossary* for definitions). This poses a major problem for the biologist trying to determine homology as genomes – unlike well-maintained software – do not come with a version control system. In order to tell apart:

1. homologous genes with a domain insertion, and
2. unrelated genes who just happen to share a domain with the other

various heuristics are used (such as alignment length). Song et al. (2008) question these heuristics and proposes their own way of identifying homologs. For each pair of candidate sequences x and y, don't score the similarity between x and y (like your typical BLAST-based homology search), but instead calculate the *sample Pearson correlation coefficient (PCC)* of their respective homology to a reference database R, as determined by a BLAST-search. This similarity to the reference database sequences is encoded in vectors of alignment scores (bit-scores) $(X, Y) = (\text{align}(x \times R), \text{align}(y \times R))$, with $R = \{\text{reference sequences}\}$. That is, X and Y describe the protein sequence similarity to the entries of R for x and y, respectively, and $\text{PCC}(X, Y)$ describes the linear correlation in their respective relations to R. They call the PCC of (X, Y) the *Neighborhood Correlation score (nc-score)* for (x, y).

Song et al. (2008) have compared candidate pairs of both types – (1) and (2) as described above – and claim that the nc-score measure is able to tell apart the true

homologs from the ones who just happen to have had the same domain inserted. They validate this claim using a manually curated test-set (also used in this project) consisting of 1577 proteins from human and mouse, each with the correct family specified.

Their results look promising and their software implementation NC_STANDALONE runs quickly on small (few species) data-sets. But will it scale? We will explore this question in the *Theory* section below.

Purpose The main goal of this project is to determine if it is possible to use a significantly smaller reference database (preferably of fixed size) when using Neighborhood Correlation, thereby potentially greatly expanding the method's applicability.

Specifically this means that we will no longer use the set of query sequences (Q) as reference database R (i.e. applying NC to the results of BLAST-searching $Q \times Q$) but rather need to construct a smaller R (subsampled from Q or taken from some other source entirely).

In the pursuit of this goal, the following subgoals became apparent:

1. Find a way to validate the quality of a set of nc-scores – in particular the *sensitivity* and *specificity*.

2. Find a suitable source of reference sequences and determine how to best create a database from them.

3. Determine the optimal parameters for the method. This will give a downsized database the best chance of working while ensuring best-case performance from the standard approach to serve as a gold standard. This includes figuring out settings for the aligner and how to process the reference sequences before alignment.

4. Verify that the reimplementation of the NC method – SNC – required for this project (as the existing implementation NC_STANDALONE requires $R = Q$) produces similar results.

5. Learn as much as possible about what makes a good reference database in general. Address questions like:

   a) Will including similar – redundant – sequences in the database make the classification better, and if so how many are needed before we see diminishing returns? Or should redundancy be avoided not only for speed – but also for quality?

   b) Is it okay to include intact multidomain proteins or should they be fragmented into their domain parts first? How important is this?

2 Theory

Here we will consider the computational requirements of the two key softwares needed to identify homologs using Neighborhood Correlation: the aligner DIAMOND and three different implementations of NC.

For parameter estimation from experimantal results, this section refers to reference databases (`HSA+MMU` and `frag_all_aln`) explained in table 2.

2.1 Neighborhood Correlation

Variable	Definition
Q	The set of all query sequences
$x, y \in Q$	A pair of query sequences to compute nc-score for
$q_i \in Q$	Alternative, indexed notation for query sequences
n	Number of sequences in Q
$R = (r_1, ..., r_s)$	Reference sequences r_i in the database R
$X = (x_1, ..., x_s)$	Log(alignment scores of x against each $r_i \in R$)
$Y = (y_1, ..., y_s)$	Log(alignment scores of y against each $r_i \in R$)
s	Number of sequences in R
k	Number of alignments (hits) from queries to R
m	Number of hits (alignments) per query
$m_{avg} = k/n$	Average number of hits (alignments) per query

Table 1: Variables used for time complexity analysis

The variables used in this section are explained in table 1.

The *nc-score* of x and y is defined as the *sample Pearson correlation coefficient* (PCC) for the pair (X, Y) of alignment scores (bit-scores):

$$\text{nc-score}(x, y) = \text{PCC}(X, Y) = \frac{\sum_{i=1}^{s}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{s}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{s}(y_i - \bar{y})^2}} \quad (2.1)$$

$$= \frac{\Sigma_{i=1}^{s} x_i y_i - s\bar{x}\bar{y}}{\sqrt{\Sigma_{i=1}^{s} x_i^2 - s\bar{x}^2}\sqrt{\Sigma_{i=1}^{s} y_i^2 - s\bar{y}^2}} \quad (2.2)$$

The rewrite[1] (2.2) is the computationally most efficient way of computing the PCC

[1] By using $\Sigma_i^n x_i a = n\bar{x}a$ for all three square sums.

– *at least using this formula* – as the complexity-determining step is the cross-product ($x_i y_i$) sum and omitting the calculation of non-zero cross-product terms will alter the result, so an algorithm with a lower order of steps won't work.

A note on estimating m : In this thesis we will assume $m = m_{avg}$ in order to express m in terms of k and n. In reality m depends on both Q and R. Determining the nature of this relationship is critical to giving accurate estimates of time complexity for the method. Seeing as we are mainly concerned with comparing the relative performance of different Neighborhood Correlation methods however, it's not critical to know the exact m.

When it comes to implementing this method into an algorithm, one would like to avoid computing – or having to consider whether to compute – terms where the cross-product $x_i y_i$ is zero (naively comparing all $x_i y_i$ terms for all pairs of queries would take $O\left(n^2 s\right)$ steps). This turns out to be hard to do. With a straight-forward translation of (2.1) or (2.2) into an algorithm, one would first have to identify the pairs for which nc-scores should be computed. NC_STANDALONE and SNC (see algorithm 1 and 2) do this differently but still arrive at having to compute the PCC for each of the identified pairs (x, y). Using the database hits of x as a template $[O(m)]$, the algorithm would then have to determine whether y has corresponding hits to the same target, which would make the cross-product $x_i y_i$ non-zero and worth computing. This takes at least $O(m)$ computations.

How many candidate (i.e. with shared database hits) pairs (x, y) are there? While upper-bounded by n^2 there is obviously much fewer in a normal situation. On average, for each of the s targets, there are $\frac{k}{s}$ queries aligning to it. This means there are $O\left((\frac{k}{s})^2\right)$ pairs for this target. Assuming the hits to the reference database are distributed uniformly over the targets (which they are not – we will get back to that), there would be $O\left(s(\frac{k}{s})^2\right)$ pairs in total.

In total, an algorithm that works this way would, for each of these $O\left(s(\frac{k}{s})^2\right)$ pairs, have to compute the PCC with $O(m)$ steps, thus having a time complexity of $O\left(s(\frac{k}{s})^2 m\right)$. Assuming $m = m_{avg}$ this is equal to $O\left(\frac{k^3}{ns}\right)$. SNC actually achieves this.

We have now found a way to avoid computing the cross-product $x_i y_i$ when both factors are zero, thus going from $O\left(n^2 s\right)$ to $O\left(\frac{k^3}{ns}\right)$. Is there a way to avoid the computation for pairs with only one factor being zero? Turns out there is. The idea is that the cross-product only needs to be computed when there is input data for it, i.e. a pair of alignment scores for a pair (x, y) aligning to the same r_i. So do exactly that! Going through the alignment scores – grouped by target sequence [so $O(s)$] – such an algorithm would compute, for the set of queries $q(r_i)$ aligning to the current (i:th) target r_i: $\{x_i y_i \mid (x, y) \in (q(r_i) \times q(r_i))\}$. These partial (only calculated for the current i) cross-products [of which there are $O\left((\frac{k}{s})^2\right)$] would then have to be stored by incrementally summing them to some kind of key-value data structure, e.g. a hashmap. This step can be done in $O(1)$ time. Once this procedure has been repeated for all target sequences, the algorithm has to go through all the $O\left(s(\frac{k}{s})^2\right)$ cross-products and finish the computation of the sample Pearson correlation coefficient. This step can be done in $O(1)$ time per cross-product.

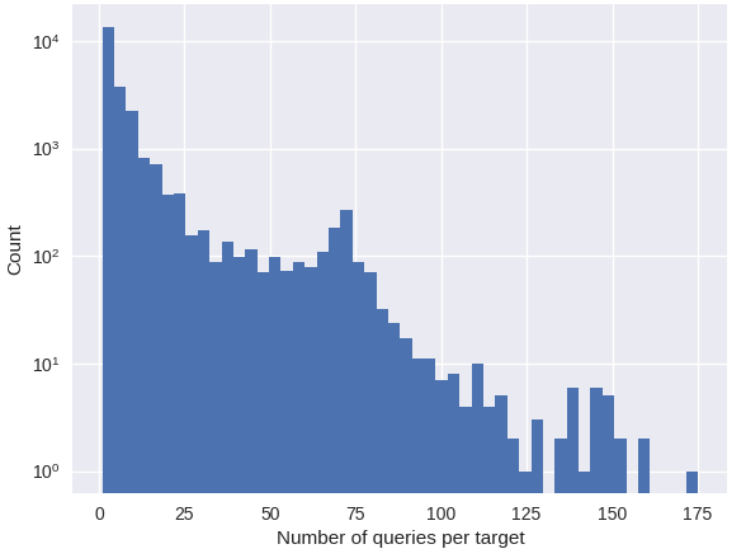

Figure 1: Number of queries per target for $Q =$ `HSA+MMU`, $R =$ `frag_all_aln`. Log scale y-axis. Average $m = k/n = 9.4$ (out of the queries that aligned to any target – so targets with zero hits are not included in the plot).

In conclusion, we have put together an algorithm that avoids the redundant $O(m)$ extra steps per pair of queries, thus bringing us down from $O\left(n^2 s\right)$ through $O\left(\frac{k^3}{ns}\right)$ to $O\left(\frac{k^2}{s}\right)$ steps to compute NC.

2.1.1 A remark on the skewness of m^2

In general $m \neq m_{avg}$. How skewed is the distribution of the hits? A quick test using the $Q = R =$ `HSA+MMU` data-set showed that the actual average m^2 was 4.6 times higher than the theoretical $m^2 = (k/n)^2$. (When instead using `frag_all_aln`, the ratio was 4.1.) See figure 1 for an example of the distribution of queries over the reference sequences.

As mentioned above we will be ignoring this skewness going forward. The main thing to keep in mind is that an $O(m^3)$ algorithm will be even worse affected by the skewness than an $O(m^2)$ one.

There are ways of mitigating the detrimental effects of too many queries matching a single reference sequence, among them are fragmentation – which is described in detail in this thesis – and filtering out some of the most frequently matched reference sequences.

2.1.2 PNC

The idea leading up to the $O\left(\frac{k^2}{s}\right)$ estimation above is fully developed in the PNC algorithm (see alg. 3) which also has a working pure Python implementation, which should – and appear to – give effectively identical results to SNC. I wrote it very late in the project, so it has not been used for the analysis. A preliminary test showed it ran about ten times faster than SNC on the $Q = R =$ `HSA+MMU` data-set.

2.1.3 SNC

SNC was developed by project advisor Lars Arvestad specifically for this project and thus it supports using scaled-down reference databases ($R \neq Q$) (Arvestad 2023). See algorithm 2. Internally it uses sparse matrices.

2.1.4 NC_standalone

NC_STANDALONE – written by Jacob Joseph, see Song et al. (2008) – is the original implementation of NC, designed to be used on all-versus-all alignment experiments where the query set Q equals the reference database R. See algorithm 1. As this project is all about *not using* $R = Q$, this implementation has only been used as a reference for what level of performance is achievable.

2.1.4.1 Minimum score – only with NC_standalone

NC_STANDALONE uses a default minimum bit-score for pairs of sequences with no significant hit. This default value is based on the size of the database and number of query sequences (Song et al. 2008, eqn. 2) and works out to about bit-score 28 for their human+mouse test case. The current implementation of SNC does not use any default score, so it's left at 0. I have not studied the consequences of this specific difference, however a comparison between NC_STANDALONE and SNC can be found in section 4.3.

2.2 A remark on O(k) – estimating scaling of NC for large n

If one is interested in predicting the run time of NC based on number of queries n and fixed size of database s, we can note that $m = k/n \Leftrightarrow k = nm$. So the optimal NC time complexity of $O\left(\frac{k^2}{s}\right)$ can be written as $O\left(\frac{n^2m^2}{s}\right)$. How big is m? Using an example of a database from the project with 35.7 k sequences (`frag_all_aln`) with sufficient redundancy to perform (almost) on par with a whole-species reference database, 37.6 k queries yielded 227 k alignments to $Q =$ `HSA+MMU`. Thus, m for this database was ~ 6.3. It's not unreasonable – though certainly not proven in this thesis – that an m of about 10–20 for a fixed-size downscaled database could be enough for most purposes. This would correspond to a database size s of about 100000. Inserting these numbers we have: $\frac{300n^2}{10^5} = 0.003n^2$. This is in contrast to the original NC: $O(nm^3)$, where m was just under 100 for $Q = R =$ `HSA+MMU`. Assuming worst-case scaling as more genomes are

added[2], we expect that $\sim 0.25\%$ of a genome matches, i.e. $m = 0.0025n$. This gives us the following estimate for the original NC: $n(0.0025n)^3 \approx 1.6 \times 10^{-8} n^4$. At $n = 1\,\mathrm{M}$ sequences the difference between PNC and NC would thus be close to seven orders of magnitude (and even greater if accounting for the skewness in m).

2.3 P for Parallel – a note on the parallelization of PNC

The P in PNC stands for parallel as it has two features that makes it suitable for parallelization:

1. Input is grouped by target sequence, so a reference database could be partitioned into smaller sets with each instance of PNC working on just that set.

2. The memory demanding items (the cross-products) can easily be partitioned as well, which means they could be stored on different nodes in-memory on a compute cluster, or even stored on disk if needed. Once complete, each such partition can (together with some auxiliary data of $O(n)$) itself produce a complete part of the final output, thus requiring only a trivial merge step to go from compute node outputs to the complete program output. (Or just let the output sit in memory on the nodes and query it directly.)

2.4 DIAMOND

DIAMOND is a protein sequence *aligner* similar to BLAST but focused on achieving maximum speed. For the version used in this thesis (v2.0.7), the authors of DIAMOND (Buchfink, Reuter, and Drost 2021) claim a sensitivity fully comparable to BLAST while accomplishing an 80-fold speed-up. Buchfink et al. give the numbers for a performance evaluation they did: $n = 281\,\mathrm{M}$ and $s = 39\,\mathrm{M}$. This took 18 hours to run on 20800 cores at highest sensitivity. In other words, assuming an $R \sim 3$ orders of magnitude smaller and a computer with correspondingly fewer cores, even an absurdly large query set would only take about a day to align (assuming proportional scaling). It's clear that at no scale will aligning the sequences be the rate-limiting step for computing NC (assuming DIAMOND or a similar tool is sensitive enough).

[2] I.e. same m regardless how distant a genome is.

Algorithm 1 Neighborhood Correlation v2.1

Key idea: Consider the graph G with the aligned sequences as vertices V and their alignment bitscore (if any) as edges E. Consider a pair of sequences (a, b). If they are to have *any* shared hits, i.e. a pair of alignments (a, s) and (b, s) – a requirement to calculate a non-zero nc-score – their distance in G can be at most 2. This algorithm traverses G up to depth 2 for each $v \in V$, thus finding all nc-scorable candidates.

Input: *alignments*: a list of alignment pairs with bit-scores $(q, r, bitscore)$.

Output: nc-scores for sequence pairs (q, r, nc_score).

Require:

For the input alignments, the reference database has to equal the set of query sequences (i.e. R=Q).

▷ Note: *villagers* is just a terse way of saying *neighbor's neighbors*.

for $query \in \{q \mid (q, r, -) \in alignments\}$ **do** ▷ $O(n)$
  $neighbors \leftarrow \{r \mid (q, r, -) \in alignments, q = query\}$ ▷ $O(k/n)$
  $villagers = \{vill \mid (q, vill, -) \in alignments, q \in neighbors, vill \neq q\}$ ▷ $O(\frac{k}{n})^2$
  for $other \in (neighbors \cup villagers)$ **do**
    $score \leftarrow$ pearson_corr_coeff$(alignments, \{query, other\})$ ▷ $O(m) = O(\frac{k}{n})$
    print $query, other, score$
  end for ▷ Time complexity for NC (see main text): $O(nm^3) = O(\frac{k^3}{n^2})$
end for

Algorithm 2 snc

Key idea: This algorithm is built upon the observation that a pair of sequences with zero shared hits to the reference database can't have a non-zero nc-score. It utilizes this by grouping the alignments by reference-db target.

Input: *alignments*: a list of alignment pairs with bit-scores $(q, r, bitscore)$.

Output: nc-scores for query sequence pairs (x, y, nc_score).

$neighbors = \{\}$
for $target \in \{r \mid (q, r, -) \in alignments\}$ **do** ▷ $O(s)$
  $queries \leftarrow \{q \mid (q, r, -) \in alignments, r = target\}$
  $neighbors \leftarrow neighbors \cup \{queries \times queries\}$ ▷ $O((\frac{k}{s})^2)$
end for
$adjacency\_matrix \leftarrow$ make_sparse_row_matrix$(alignments)$
for $\{x, y\} \in neighbors$ **do**
  $nc\_score \leftarrow$ pearson_corr_coeff$(adjacency_matrix, \{x, y\})$ ▷ $O(m)$
  print x, y, nc_score
end for ▷ Time complexity for SNC (see main text): $O(s(\frac{k}{s})^2 m) = O(\frac{k^3}{ns})$

Algorithm 3 PNC

Key idea: In SNC and NC_STANDALONE calculating the cross-term $\Sigma_i^n(x_i y_i)$ is the inner-most loop, done for every candidate pair. The idea here is to turn this loop inside-out and accumulate this cross-sum bit by bit over the entire set of alignments – in PNC it's the sequence pairs (x, y) that become the inner-most loop. *This algorithm achieves optimal time complexity for this problem. It's not possible to calculate NC with a lower order of steps as the cross-term has to be calculated for all pairs of queries for each target sequence they have in common.*

Input: *alignments*: a list of alignment pairs with bit-scores $(q, r, bitscore)$.
Output: nc-scores for query sequence pairs (x, q, nc_score).

q_sums = array()
q_sums_square = array()
$cross\_terms$ = hashmap()
n = size($\{q \in alignments\}$)
for $target \in \{r \mid r \in alignments\}$ **do** ▷ $O(s)$
    $queries \leftarrow \{(q, score) \mid (q, r, score) \in alignments, r = target\}$
    for $(q, score) \in queries$ **do**
        $q\_sums[q] \leftarrow q\_sums[q] + score$
        $q\_sums\_square[q] \leftarrow q\_sums\_square[q] + score^2$
    end for
    for $\{(q_1, score_x), (q_2, score_y)\} \in \{queries \times queries\}$ **do** ▷ $O((\frac{k}{s})^2)$
        $cross\_terms[(q_1, q_2)] \leftarrow cross\_terms[(q_1, q_2)] + score_x * score_y$
    end for
end for
for $(x, y) \in$ keys($cross\_terms$) **do**
    $\Sigma xy = cross\_terms[(x, y)]$
    $\Sigma x = q\_sums[x]$
    $\Sigma y = q\_sums[y]$
    $\Sigma x^2 = q\_sums\_square[x]$
    $\Sigma y^2 = q\_sums\_square[y]$
    $\bar{x} = \Sigma_x / n$
    $\bar{y} = \Sigma_y / n$
    $nc\_score = \dfrac{\Sigma xy - n\bar{x}\bar{y}}{\sqrt{\Sigma x^2 - n\bar{x}^2}\sqrt{\Sigma y^2 - n\bar{y}^2}}$
    print x, y, nc_score
end for

▷ Time complexity for PNC (see main text): $O(s(\frac{k}{s})^2) = O(\frac{k^2}{s})$

3 Method

Homolog classification pipeline A major part of this project has consisted of varying the settings of a mostly consistent classification pipeline (see section 3.4)

Synthetic multidomain proteins In order to test the hypothesis that multidomain proteins in the reference database cause false positives – and therefore should be fragmented before inclusion – I produced artificial multidomain proteins. It makes more sense to describe this process after the main pipeline has been explained. See section 4.2.2.

3.1 Data

These are the main sources of sequence data used in the project:

Name	Source	Count	Description
Pfam-A (ver. 36.0)	Mistry et al. 2021	Usable domains: 7956	Major database of protein families and domains. Contains domains and a listing of sequences aligning to it.
Pfam-A-RP15	Chen et al. 2011	Usable domains: 7677	Downscaled, less redundant version of the full Pfam-A database. *I only used it for clustering.*
HSA	UniProtKB 2023a	20413	Human ref. proteome from UniProtKB.
MMU	UniProtKB 2023b	17179	Mouse ref. proteome from UniProtKB.
UniProt	Consortium 2023	About 125000 used	The Pfam domain alignments refer to these UniProt sequences. Most sequences were used for randomization experiments. The main experiments used 7957 seqs.
Curated proteins	Song et al. 2008	1577	Proteins from HSA and MMU manually annotated with correct protein family membership. From the original Neighborhood Correlation paper. Number of homologous pairs: 426 k.

The curated protein sequences were used for validation (calling TP, FP and FN).

For queries, the combined HSA ∪ MMU set of sequences (37.6 k) was used consistently throughout the project since that was what the curated set of proteins covered.

The following sections will explain how the reference databases were created.

The actual identifiers in the code are long and complex, so in this document I've typically referred to them only using the most important part – this is what this table describes.

My identifier	**Source**	**Seq count**	**Description**
`hsa+mmu no_frag`	HSA+MMU	37584	All human and mouse sequences, full length.
`pfam longest extract_only_primary_aln`	Pfam-A (domains), UniProt (aln. seqs)	7956	For each Pfam-A domain, take the sequence having the longest alignment to this domain. Then extract only the aligned domain.
`pfam longest frag_primary_aln`		27278	Same as above but also keep the non-extracted parts and fragment into length=300 parts.
`pfam longest extract_all_aln`		16281	Use the longest aligned seq as above, but extract all non-overlapping domains in all of Pfam-A (not just the current domain.)
`pfam longest frag_all_aln`		35735	Same as above but also keep the non-extracted parts and fragment into length=300 parts.
`pfam longest discard_all_aln`		19455	The complement to `extract_all_aln`. Consists mostly of "non-domain" seqs, fragmented into length=300.

`pfam longest frag_raw_seq:50`
⋮
`frag_raw_seq:400`
		103773
		⋮
		16522
			Take the longest aligned sequence as above, but ignore all domain information and fragment the sequence into length=50 parts.
			⋮
			Fragment length 400.

`pfam longest no_frag`		7957	Take the longest aligned sequence as above. Keep the full sequence.
random:1.no_frag		7868	Unlike above, for each Pfam-A domain just take $1-5$ random sequences aligning to it.
random:3.no_frag		23441	Keep the full sequence.
random:5.no_frag		38842	
synthetic multidomains (only primary domains)		398–7956	*Described in section 4.2.2*
synthetic multidomain (all domains)		815–16281	*Described in section 4.2.2*

Table 2: Reference databases created in the project

These databases are all based on the data described in 3.1. See section 3.3.3 for the processing of an example sequence. See section 4.2for comparison of results from using these databases.

3.2 Software

See Theory (chapter 2) for descriptions of the algorithms.

In order to run Neighborhood Correlation, two key softwares are needed: an *aligner* to produce the $(seq_1, seq_2, bit-score)$ pairs used as input for NC, and a suitable implementation of NC.

3.2.1 Choice of aligner

Given the availability of modern high-performance aligners such as DIAMOND (v.2.0.7) (Buchfink, Reuter, and Drost 2021) and MMseqs2, there was no reason to use BLASTP for this project. A direct comparison of DIAMOND and MMseqs2 showed DIAMOND to be both more sensitive and faster than MMseqs2 (Buchfink, Reuter, and Drost 2021, fig. 1).[1]

Another factor was that I wasn't sure how easy it would be to implement MMseqs2 – an apparently complex tool – in a workflow that's originally designed for using BLAST (or a direct replacement) specifically. Altogether, I decided to use DIAMOND, which has worked well.

3.2.2 Neighborhood Correlation implementation

Only snc was available at the start of this project and able to work with $R \neq Q$. See the Theory section for details.

3.2.3 Additional software used

- SciPy (Virtanen et al. 2020)
- matplotlib (Hunter 2007)
- Seqtk (Li 2023)

3.3 Making the reference database *R*

See table 2 for a complete list of all reference databases created in this project. Below follows a description of how they were generated.

3.3.1 The control

The query set $Q = \text{HSA} \cup \text{MMU}$ was used as a reference (control) for the other databases as I expected that using $R = Q$ would perform better than any other database I could put together. It was also necessary to use this database to compare snc with NC_standalone, as NC_standalone requires $R = Q$.

[1] It should be noted that this paper was written by the authors of DIAMOND and didn't use the currently (2023) latest version of MMseqs2 (if that would make any difference).

3.3.2 Pfam-A domains as an index of what to include

Dannie Durand had previously suggested using Pfam to select or sample sequences to thesis advisor Lars Arvestad (personal communication, Feb 23, 2022). Pfam-A (version 36.0) contains close to 8000 domain entries, and I tried using them. In the version of the database I accessed, only the aligned segments are available.[2] So I used the UniProt API to download one or more of the aligned sequences for each domain entry.

The question then became how to select what alignment to use out of the frequently tens of thousands of alignments available for each entry. I tried three main approaches:

1. Longest alignment. Each alignment comes with an accession and the alignment range, e.g. "`Q6FNP0_CANGA/235-469`". This method simply selects the alignment with the longest range. Or tries to select, I should say. I found that frequently, UniProt entries being referenced by Pfam-A were marked as "inactive" or otherwise permanently unavailable – so I made the program keep trying to get the next best sequence until it could get one.

2. Random. One or more aligned sequences selected randomly. I did this for up to 5 sequences per entry (which were then subsampled to provide sets of 3 and 1 random sequences per entry, respectively). I performed several trials to get stable average values. See the results section.

3. Clustering. Early on in the project I figured there would be a need to select multiple sequences per Pfam-domain, and that it would be preferential to select a set of sequences as diverse as possible. I used the SciPy implementation of the UPGMA linkage method to create clusters of the alignments for each Pfam domain (technically using Pfam-A-RP15 and further subsampling to get a manageable number of sequences (≤ 2000) per domain for the clustering). I would then select one sequence per *cluster*.

Out of these, "longest" was the option I used for most of the experiments. I abandoned the clustering method once I got results showing that multiple sequences per domain weren't really helpful, as the clustering process was slow and cumbersome.

3.3.3 Processing of the selected sequences

Consider the following example protein sequence (numerals represent aligned Pfam domains, dashes+letters the sequences between them):

`EXAMPLE01 --a--1111122222222--b--3333--c--4444444444`

And let's say that the current Pfam-A domain entry being processed is:

`EXAMPLE01/6-10`

Then the entries generated for the reference database will be the following assuming a fragment length setting of 5 unless specified (please reference table 2):

[2] I first tried extracting only the multiple sequence alignments and building a reference database of those, but that didn't really work at all.

Identifier	Comma-separated list of entries to be added to the ref. db.
extract_only_primary_aln	11111
frag_primary_aln	11111,--a--,22222,222--,b--33,33---,--c--,44444,44444
extract_all_aln	11111,22222222,3333,4444444444
frag_all_aln	--a--,11111,22222222,--b--,3333,-----,c--,4444444444
discard_all_aln	--a--,--b--,-----,c--
frag_raw_seq:10	--a--11111,22222222--,b--3333---,--c--44444,44444
no_frag	--a--1111122222222--b--3333-----c--4444444444

Note that the extracted domains are not further fragmented. All fragments (including domains) shorter than 15 amino acids are discarded for the `frag_all_aln, discard_all_aln, frag_primary_aln` and `frag_raw_seq` database types, i.e. the ones having undergone fixed-length fragmentation and are thus likely to get "sequence stubs".[3]

Implementation-wise this was done by putting the Pfam-A domain-alignment coordinates into a 2,8 GB SQLite database with a table consisting of (accession, start_pos, end_pos) – meaning that for any sequence (EXAMPLE01 in our example above) I could immediately look up what parts of the sequence has domains aligned to it. In the "all" variants above, overlapping domains are ignored (so if e.g. the `11111` domain overlaps into the full `22222222` domain alignment (not visible here), the latter would not be extracted unless it has another, shorter, alignment that doesn't overlap with the `11111` domain).

An option would have been to extract all overlapping alignments, but I was concerned about database size.

3.4 Neighborhood Correlation pipeline

Pipeline key steps:

1. Take query sequences Q and a reference database R and align them using DIAMOND.
2. Use SNC or NC_STANDALONE to produce nc-scores for candidate homologs.
3. Validate the nc-scores by calling true/false positives and false negatives using a curated set of homologous proteins with a known family assignment.

For an overview, see algorithm 4. In addition, much was done manually.

3.4.1 Validation using curated proteins

Song et al. (2008) has published their set of curated proteins annotated with the correct family name. By simply looking up the family name for each of the sequences in an identified homolog pair, one can – if both sequences are in the annotated list – tell whether this is a true positive or false positive. This can be done for a discrete range of

[3] It would have been smarter to merge them with the previous fragment, but I didn't think of that.

nc-score thresholds, by calling FP and TP on the set of homolog pairs that passed each particular threshold.

Identified "trivial self-homologs" i.e. pairs (x, x) that are returned by NC are ignored by the validation script and also excluded from the "All" counts in the tables. Pairs are counted correctly (only once).

Algorithm 4 Main NC pipeline

Purpose: Test (mainly) SNC under diverse conditions and validate the results using a curated test set with known family classifications.

Input:

R: a list of protein sequences to construct the reference database R out of.
Q: a list of protein sequences to find homologs in.
$bitscore, transform$: Bit-score threshold and bit-score transform.
$curated$: a mapping of accesion to family e.g. [("PROT0001", "kinase"), ...], used for validation.

Output:

Significant nc-scores for the query pairs: (q_1, q_2, nc_score).
Validation results containing counts of true positives (TP), false positives (FP) and false negatives (FN), as well as a total count (ALL) of the number of identified homologous pairs.

Require:

Depending on the precise settings, the pipeline may have restrictions such as $R = Q$ (when using NC_standalone) or may e.g. only be able to handle small databases when testing very low bit-score thresholds.

This is a coneptual description of the typical case.

1: diamond makedb -d R.
2: diamond blastp –ultra-sensitive –max-target-seqs 1000 -d R –min-score $bitscore$ -o alignments
3: transform-bitscore log10 alignments > alignments.log10
4: snc (nc-score report threshold 0.05) alignments.log10 > scores.snc
5: validate-scores scores.log10.snc –annotated-proteins $curated$ > validation-results

4 Results

These results will demonstrate how important it was to systematically tweak the pipeline to give optimal performance. It turned out to be difficult to predict *which* settings would be most important and learning this proved more valuable than finding the exact optimal *value* for everything.

4.1 Optimizing the alignment stage

The settings used for the DIAMOND aligner turned out to be of critical importance. In retrospect this should have been expected – the normal use case (and hence choice of default settings) is *not* to return massive amounts of vaguely related sequences.

4.1.1 DIAMOND sensitivity

As can be seen in table 3, using the "fast" sensitivity instead of the highest "ultra-sensitive" mode cuts the TPR % roughly in half at comparable FDR % in multiple comparisons using different reference databases and NC implementations. Therefore, DIAMOND is simply unusable for NC unless a high sensitivity[1] is used (then it performs very well on the other hand).[2] Using the default option, DIAMOND returns only about $20\,\% - 25\,\%$ as many hits and it's apparently not enough.

[1] I only tested the highest, it's not limiting step anyway in terms of computational complexity.
[2] It should be noted that I never compared DIAMOND with BLAST.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
(a) all-vs-all using NC_standalone														
(HSA∪MMU)=R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align(ultra-sensitive)}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{NC\_standalone}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	86.90	86.32	85.93	85.76	85.38	84.88	84.17	83.40	82.25	80.52	77.90	74.87	70.29	63.22
FNR (%)	13.10	13.68	14.07	14.24	14.62	15.12	15.83	16.60	17.75	19.48	22.10	25.13	29.71	36.78
FDR (%)	**0.12**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	451	89	26	8	8	2	0	0	0	0	0	0	0	0
All	2.8 M	2.2 M	2.0 M	1.8 M	1.6 M	1.5 M	1.4 M	1.4 M	1.3 M	1.2 M	1.2 M	1.1 M	1.0 M	0.9 M
(HSA∪MMU)=R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align(fast)}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{NC\_standalone}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	44.61	33.85	26.14	20.67	16.67	13.79	11.53	9.96	8.64	7.55	6.52	5.64	4.72	3.64
FNR (%)	55.39	66.15	73.86	79.33	83.33	86.21	88.47	90.04	91.36	92.45	93.48	94.36	95.28	96.36
FDR (%)	**0.05**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	91	0	0	0	0	0	0	0	0	0	0	0	0	0
All	0.9 M	0.8 M	0.7 M	0.6 M	0.5 M	0.5 M	0.4 M	0.4 M	0.4 M	0.3 M	0.3 M	0.3 M	0.3 M	0.2 M
(b) all-vs-all using snc														
(HSA∪MMU)=R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align(ultra-sensitive)}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	88.28	87.79	87.00	86.49	86.30	85.94	85.75	85.62	85.29	84.90	84.43	83.45	81.98	79.80
FNR (%)	11.72	12.21	13.00	13.51	13.70	14.06	14.25	14.38	14.71	15.10	15.57	16.55	18.02	20.20
FDR (%)	**0.53**	**0.23**	**0.14**	**0.07**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1986	877	509	256	87	27	5	0	0	0	0	0	0	0
All	4.6 M	3.2 M	2.7 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M
(HSA∪MMU)=R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align(fast)}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	51.63	40.22	31.38	24.38	19.05	15.22	12.24	10.14	8.53	7.09	5.88	4.87	3.82	2.88
FNR (%)	48.37	59.78	68.62	75.62	80.95	84.78	87.76	89.86	91.47	92.91	94.12	95.13	96.18	97.12
FDR (%)	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	28	0	0	0	0	0	0	0	0	0	0	0	0	0
All	1.2 M	1.0 M	0.8 M	0.7 M	0.6 M	0.5 M	0.5 M	0.4 M	0.4 M	0.4 M	0.3 M	0.3 M	0.3 M	0.2 M
(c) frag-all-aln using snc														
{σ | σ w. longest aln. to $\delta \in$ Pfam-A domains} $\xrightarrow[frag.\ remains\ (l=300)]{extract\ all\ dom.\ in\ \sigma}$ R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align(u.-s.)}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ cur.]{val.}$ TP FN FP														
TPR (%)	86.83	86.71	86.21	85.79	85.64	85.28	85.08	84.63	84.10	83.37	82.49	81.27	79.17	75.76
FNR (%)	13.17	13.29	13.79	14.21	14.36	14.72	14.92	15.37	15.90	16.63	17.51	18.73	20.83	24.24
FDR (%)	**0.14**	**0.03**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	518	127	35	4	4	4	4	4	4	4	4	4	4	4
All	2.2 M	2.1 M	1.9 M	1.8 M	1.7 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.1 M	1.0 M	0.9 M
{σ | σ w. longest aln. to $\delta \in$ Pfam-A domains} $\xrightarrow[frag.\ remains\ (l=300)]{extract\ all\ dom.\ in\ \sigma}$ R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align(fast)}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ cur.]{val.}$ TP FN FP														
TPR (%)	46.90	40.44	34.13	28.99	24.31	20.22	16.64	13.61	11.08	8.89	7.07	5.43	4.11	3.11
FNR (%)	53.10	59.56	65.87	71.01	75.69	79.78	83.36	86.39	88.92	91.11	92.93	94.57	95.89	96.89
FDR (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	3	2	1	0	0	0	0	0	0	0	0	0	0	0
All	0.4 M	0.4 M	0.3 M	0.3 M	0.3 M	0.2 M	0.2 M	0.2 M	0.2 M	0.2 M	0.1 M	0.1 M	0.1 M	0.1 M
nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70

Table 3: Comparison of DIAMOND sensitivity modes

Three sets of comparisons of `--ultra-sensitve` vs. `--fast`: (a) R=(HSA+MMU) using NC_standalone. (b) R=(HSA+MMU) using snc. (c) R={A set of UniProt sequences fragmented by aligned domains (see sec. 4.1.1)}. The redundant, very similar outcomes are included to underscore that the dramatic difference is consistent, not caused by a particular NC implementation and not apparently dependant on the evolutionary distance between the query set and database R (self-species vs. sampled from all species). `--max-target-seqs 1000` was used for this comparison. **Colors:** The green and gray background is just to make it easier to compare the same metrics across experiments. The yellow background highlights (some of) the changes across experiments. FDR % $\geq$ 0.01 has red color.

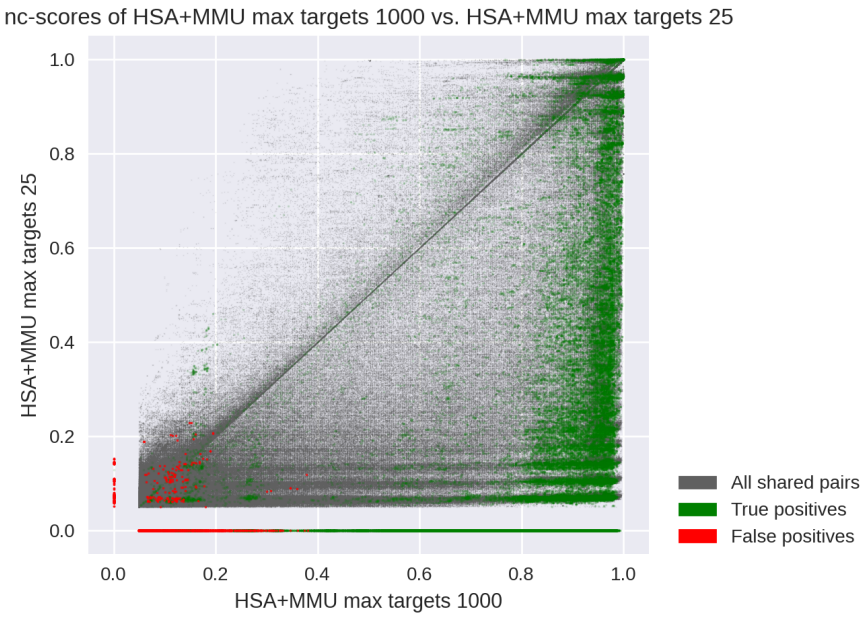

Figure 2: Comparison of different `-max-target-seqs` (`HSA+MMU`)
Comparison of effects of changing number of alignments returned per reference sequence (at BS=30, log_{10}). $R =$ `HSA+MMU`. In these scatter plots, the green dots (TP) are bigger and the red dots (FP) biggest. Note the large amount of TP that are misclassified (have low nc-score) when limiting the number of target sequences to the default setting. Missing scores set to 0.

4.1.2 DIAMOND max-target-seqs

As can be seen from figure 2 (true positives in green in the bottom right quadrant of both the top and middle figures) and 3 (similarly for the TP and also the nc-score distribution of TP in the histogram), there is dramatic loss of true positives returned by NC when limiting the number of target matches to the default of 25. One possibility is that for common multidomain proteins, the 25 hits are easily "consumed" by one of the domains, leaving no matches left to the other domains, thus NC can't "know" they exist. The setting of max-target-seqs 1000 has been used for all alignments in this project except for this very comparison.

Note that similar settings are available for other aligners, e.g. BLAST.

4.1.3 Other DIAMOND options

No other aligner options were changed. I didn't really look into it.

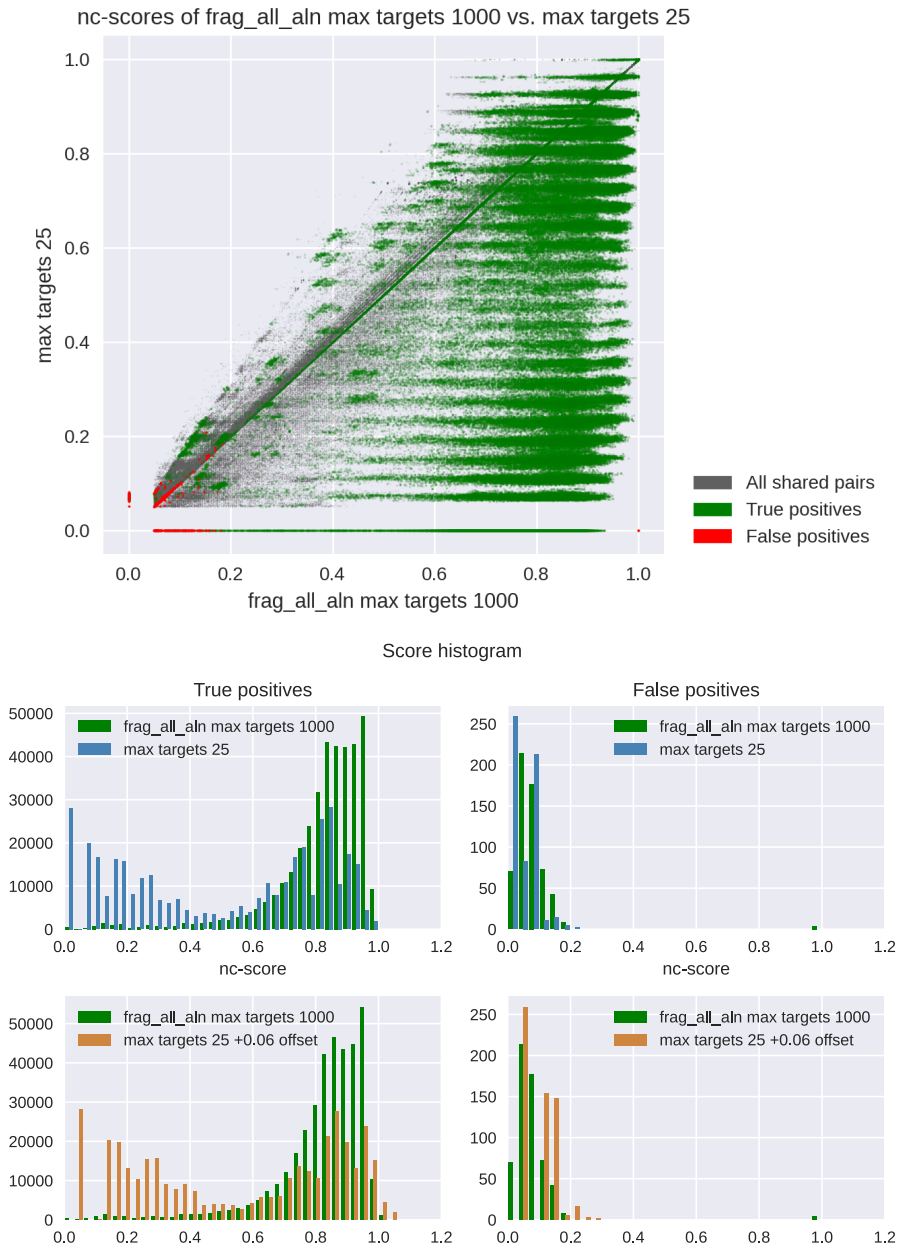

Figure 3: Comparison of different `-max-target-seqs` (`frag_all_aln`)
Comparison of effects of changing number of alignments returned per reference sequence (at BS=30, log_{10}). Top (scatter) and bottom (histogram): $R =$ `frag_all_aln`. In the scatter plots, the green dots (TP) are bigger and the red dots (FP) biggest. Scatterplots are intended to be mostly qualitative while the histogram provides a quantative view of the same data. Note the large amount of TP that are misclassified (have low nc-score) when limiting the number of target sequences to the default setting. The offset in histogram is to make the nc-score average the same. Missing scores set to 0.

4.1.4 Bit-score threshold

I compared the nc-scores resulting from running DIAMOND at bit-score thresholds of 5, 10, 15, 20, 22, 24, 26, 28, 30, 32, 35, 40 and 50 on the `pfam longest no_frag` database. As can be seen from table 4, going below bit-score 30 doesn't improve TPR % at e.g. FDR = 0.00 % but comes at the cost of having more alignments to deal with (the run time of SNC was considerable longer). Going above 32 one sees a drop-off in TPR at the same FDR, but a plus is having less data to deal with.

Interestingly, compare bit-score 50 and bit-score 32 at FDR = 0.02 %. TPR only goes down from 85.5 % to 83.2 %, but the total number of pairs is more than cut in half. It's not clear how to interpret this. Is there a selection bias with Pfam-A so that NC will perform better with annotated proteins (such as those in the validation set) compared to all proteins (including potentially many unknown)? Or is much of the excess of 1.1 M identified pairs false positives? One way to answer that question is to look at the corresponding results when using $R =$ `HSA` $\cup$ `MMU`. (Results not shown.) The exact same trend is there when using SNC on this data-set, so it doesn't seem to be caused by an annotation bias. Further investigation needed.

The optimal bit-score threshold out of these is 32, but the difference compared to 30 is small. In general, the optimal bit-score threshold may depend on multiple factors, but its relation to which BLOSUM scoring matrix is used warrants investigation.

Perhaps the key take-away is that when working with low-FDR classifications there exists an optimal value for the bit-score threshold – it's not a case of "more is better if you can afford it". [3]

[3] It should be pointed out that with NC_STANDALONE there is actually little difference between bit-scores of $10 - 30$. But at least the lower scores are not apparently better.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 5]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	87.51	86.91	86.17	85.79	84.88	83.97	83.30	82.54	81.64	80.81	79.36	76.78	72.92	65.92
FNR (%)	12.49	13.09	13.83	14.21	15.12	16.03	16.70	17.46	18.36	19.19	20.64	23.22	27.08	34.08
FDR (%)	**6.46**	**1.21**	**0.36**	**0.13**	**0.06**	**0.03**	**0.01**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00
FP	25723	4527	1318	481	224	105	48	21	10	2	1	0	0	0
All	24.0 M	7.4 M	3.7 M	2.5 M	2.0 M	1.6 M	1.4 M	1.2 M	1.1 M	1.0 M	0.8 M	0.7 M	0.6 M	0.5 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 20]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	87.50	86.93	86.20	85.86	84.98	84.07	83.39	82.66	81.76	80.95	79.57	77.00	73.29	66.58
FNR (%)	12.50	13.07	13.80	14.14	15.02	15.93	16.61	17.34	18.24	19.05	20.43	23.00	26.71	33.42
FDR (%)	**6.17**	**1.25**	**0.37**	**0.14**	**0.06**	**0.03**	**0.01**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00
FP	24511	4703	1371	508	227	110	51	22	13	3	2	0	0	0
All	22.3 M	7.2 M	3.7 M	2.6 M	2.0 M	1.7 M	1.4 M	1.3 M	1.1 M	1.0 M	0.9 M	0.8 M	0.7 M	0.5 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 26]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	87.91	87.04	86.61	86.25	85.82	85.47	85.18	84.71	84.20	83.47	82.78	81.79	79.65	76.58
FNR (%)	12.09	12.96	13.39	13.75	14.18	14.53	14.82	15.29	15.80	16.53	17.22	18.21	20.35	23.42
FDR (%)	**3.76**	**0.85**	**0.41**	**0.23**	**0.11**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00
FP	14631	3160	1532	851	417	179	55	25	14	7	6	6	6	6
All	7.7 M	5.1 M	3.6 M	2.9 M	2.5 M	2.2 M	1.9 M	1.7 M	1.6 M	1.4 M	1.3 M	1.1 M	1.0 M	0.9 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 30]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	86.99	86.61	86.35	86.11	85.86	85.52	85.35	84.96	84.51	83.77	83.25	82.53	81.10	78.63
FNR (%)	13.01	13.39	13.65	13.89	14.14	14.48	14.65	15.04	15.49	16.23	16.75	17.47	18.90	21.37
FDR (%)	**1.61**	**0.44**	**0.18**	**0.10**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	6059	1633	667	359	185	67	32	12	2	0	0	0	0	0
All	4.3 M	3.4 M	2.8 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.5 M	1.4 M	1.3 M	1.2 M	1.0 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 32]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	86.72	86.64	86.31	85.70	85.52	85.42	85.16	84.71	84.23	83.56	83.06	82.25	80.83	78.32
FNR (%)	13.28	13.36	13.69	14.30	14.48	14.58	14.84	15.29	15.77	16.44	16.94	17.75	19.17	21.68
FDR (%)	**1.30**	**0.30**	**0.13**	**0.06**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	4877	1115	465	207	83	33	14	2	0	0	0	0	0	0
All	3.6 M	2.9 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.1 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 35]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	86.25	86.09	85.53	85.46	85.39	85.20	84.71	84.29	83.63	83.08	82.59	81.30	79.56	77.33
FNR (%)	13.75	13.91	14.47	14.54	14.61	14.80	15.29	15.71	16.37	16.92	17.41	18.70	20.44	22.67
FDR (%)	**0.93**	**0.25**	**0.11**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	3433	927	387	186	75	23	11	0	0	0	0	0	0	0
All	3.0 M	2.5 M	2.2 M	2.0 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.1 M	1.0 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 40]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	85.79	85.75	85.08	84.64	84.26	83.78	83.48	82.89	82.28	81.30	79.93	78.31	76.75	73.58
FNR (%)	14.21	14.25	14.92	15.36	15.74	16.22	16.52	17.11	17.72	18.70	20.07	21.69	23.25	26.42
FDR (%)	**0.17**	**0.09**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	608	337	180	72	18	6	2	0	0	0	0	0	0	0
All	2.2 M	1.9 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.3 M	1.2 M	1.1 M	1.1 M	1.0 M	1.0 M	0.9 M
	$\{\sigma \mid \sigma$ w. longest aln. to $\delta \in$ Pfam-A domains$\} \rightarrow$ R $\times$ (HSA $\cup$ MMU) $\xrightarrow[BS \geq 50]{align}$ $\xrightarrow[log10]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP													
TPR (%)	83.86	83.59	83.18	82.26	81.20	80.14	78.93	77.71	76.46	74.79	72.42	69.22	65.74	61.03
FNR (%)	16.14	16.41	16.82	17.74	18.80	19.86	21.07	22.29	23.54	25.21	27.58	30.78	34.26	38.97
FDR (%)	**0.10**	**0.06**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	347	203	86	35	16	0	0	0	0	0	0	0	0	0
All	1.3 M	1.1 M	1.0 M	1.0 M	1.0 M	0.9 M	0.9 M	0.9 M	0.8 M	0.8 M	0.8 M	0.7 M	0.7 M	0.6 M
nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70

Table 4: Comparison of DIAMOND bit-score thresholds

Database `frag_all_aln`. Log10 transformed.

4.1.5 Bit-score transformation

The need for the log_{10}-preprocessing was determined experimentally by Song et al. (2008). They reason that since the Pearson correlation coefficient captures *linear* relationships, the logarithm is needed to compress the bit-score range into a more linear scale.

Table 5 shows the results of using different bit-score transformations on bit-scores from aligning $Q = R =$ `HSA` $\cup$ `MMU`. I didn't plan to include the results of using the highly fragmented `frag_all_aln` reference database, but there are important differences – see table 6. The identity transformation again stands out the most – but it still has twice the TPR at nc-score ≥ 0.5 compared to the results in table 5.

4.1.5.1 Key observations

- The identity transformation is inferior. This is consistent with what Song et al. (2008) found.
- The binary/unweighted transformation – of great interest as it can be stored 16 times more efficiently than the 16-bit floating point numbers likely to be the optimal storage of log-transformed bit-scores – showed great promise, particularly on the fragmented database. However, it did perform worse than log_{10} (regardless of reference database). This is generally consistent with what Song et al. (2008) found even though they seem more skeptical. My results indicate that this transform is really only worth using if hardware limitations require it. If used, additional tests should be performed to validate that it performs well on the specific reference database used.
- ln and $\log_{10}$ are practically indistinguishable.

4.1.5.2 Decision

Use $\log_{10}$. This is consistent with what Song et al. (2008) did.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70

(a) logs, binary and identity transformations.

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. log10* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	88.28	87.79	87.00	86.49	86.30	85.94	85.75	85.62	85.29	84.90	84.43	83.45	81.98	79.80
FDR (%)	**0.53**	**0.23**	**0.14**	**0.07**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1986	877	509	256	87	27	5	0	0	0	0	0	0	0
All	4.6 M	3.2 M	2.7 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. binary* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	88.35	87.88	87.33	86.52	86.38	85.98	85.78	85.64	85.32	84.88	84.26	83.43	81.95	79.99
FDR (%)	**0.78**	**0.26**	**0.15**	**0.10**	**0.06**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	2963	960	557	375	215	40	7	0	0	0	0	0	0	0
All	5.1 M	3.5 M	2.9 M	2.6 M	2.4 M	2.2 M	2.0 M	1.9 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.1 M

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. log_e* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	88.28	87.79	87.00	86.49	86.30	85.94	85.75	85.62	85.29	84.90	84.43	83.45	81.98	79.80
FDR (%)	**0.53**	**0.23**	**0.14**	**0.07**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1986	877	509	256	87	27	5	0	0	0	0	0	0	0
All	4.6 M	3.2 M	2.7 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. identity* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	85.83	84.35	82.38	79.76	75.90	70.52	63.56	55.63	46.92	38.41	30.81	23.77	16.93	11.34
FDR (%)	**0.03**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	96	14	0	0	0	0	0	0	0	0	0	0	0	0
All	2.3 M	1.9 M	1.6 M	1.4 M	1.3 M	1.1 M	1.0 M	0.9 M	0.8 M	0.7 M	0.7 M	0.6 M	0.5 M	0.4 M

(b) root transformations.

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. square root* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	87.87	86.82	86.43	86.31	85.94	85.70	85.35	84.96	84.44	83.71	82.59	80.99	78.41	74.25
FDR (%)	**0.32**	**0.13**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1206	482	170	78	22	5	0	0	0	0	0	0	0	0
All	3.7 M	2.8 M	2.4 M	2.2 M	2.0 M	1.9 M	1.8 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.2 M	1.1 M

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. cube root* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	88.14	87.63	86.63	86.42	86.19	85.90	85.71	85.47	85.15	84.73	84.08	82.92	81.30	78.84
FDR (%)	**0.43**	**0.21**	**0.10**	**0.04**	**0.02**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1623	780	380	144	65	18	6	0	0	0	0	0	0	0
All	4.2 M	3.1 M	2.6 M	2.4 M	2.2 M	2.0 M	1.9 M	1.8 M	1.7 M	1.5 M	1.4 M	1.3 M	1.3 M	1.2 M

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. 4th root* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	88.25	87.73	86.85	86.46	86.26	85.93	85.74	85.58	85.25	84.86	84.35	83.29	81.85	79.54
FDR (%)	**0.50**	**0.23**	**0.13**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1872	851	497	178	80	23	5	0	0	0	0	0	0	0
All	4.5 M	3.2 M	2.7 M	2.4 M	2.2 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M

(HSA ∪ MMU)=R×(HSA ∪ MMU) —align, BS≥30→ *transf. 5th root* —snc→ All —validate, w. curated→ TP FN FP

TPR (%)	88.29	87.80	86.98	86.49	86.30	85.94	85.75	85.62	85.29	84.89	84.42	83.43	81.93	79.77
FDR (%)	**0.54**	**0.24**	**0.14**	**0.07**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	2034	884	516	258	90	26	5	0	0	0	0	0	0	0
All	4.6 M	3.3 M	2.8 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70

Table 5: Comparison of different bit-score transformations
Using R=Q=(HSA ∪ MMU). (a) logarithmic, binary and identity transformations. The default transformation in this project – log_{10} – is shown on top. (b) roots 2-5. See section 4.1.5.

(a) log_{10} transformation.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
{σ | σ w. longest aln. to $\delta \in$ Pfam-A domains} $\xrightarrow[frag.\ remains\ (l=300)]{extract\ all\ dom.\ in\ \sigma}$ R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align}$ $\xrightarrow[log_{10}]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	86.83	86.71	86.21	85.79	85.64	85.28	85.08	84.63	84.10	83.37	82.49	81.27	79.17	75.76
FNR (%)	13.17	13.29	13.79	14.21	14.36	14.72	14.92	15.37	15.90	16.63	17.51	18.73	20.83	24.24
FDR (%)	**0.14**	**0.03**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	518	127	35	4	4	4	4	4	4	4	4	4	4	4
All	2.2 M	2.1 M	1.9 M	1.8 M	1.7 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.1 M	1.0 M	0.9 M

(b) binary transformation.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
{σ | σ w. longest aln. to $\delta \in$ Pfam-A domains} $\xrightarrow[frag.\ remains\ (l=300)]{extract\ all\ dom.\ in\ \sigma}$ R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align}$ $\xrightarrow[binary]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	86.85	86.77	86.18	85.78	85.62	85.28	84.99	84.53	83.93	83.19	82.16	80.68	78.24	73.75
FNR (%)	13.15	13.23	13.82	14.22	14.38	14.72	15.01	15.47	16.07	16.81	17.84	19.32	21.76	26.25
FDR (%)	**0.19**	**0.07**	**0.02**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	695	245	60	14	4	4	4	4	4	4	4	4	4	4
All	2.2 M	2.1 M	2.0 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.0 M	1.0 M	0.9 M

(b) identity transformation.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
{σ | σ w. longest aln. to $\delta \in$ Pfam-A domains} $\xrightarrow[frag.\ remains\ (l=300)]{extract\ all\ dom.\ in\ \sigma}$ R×(HSA∪MMU) $\xrightarrow[BS\geq 30]{align}$ $\xrightarrow[identity]{transf.}$ $\xrightarrow{snc}$ All $\xrightarrow[w.\ curated]{validate}$ TP FN FP														
TPR (%)	86.65	86.49	86.09	85.65	85.35	84.94	84.39	83.59	82.25	80.19	77.09	72.75	66.74	59.26
FNR (%)	13.35	13.51	13.91	14.35	14.65	15.06	15.61	16.41	17.75	19.81	22.91	27.25	33.26	40.74
FDR (%)	**0.04**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	142	45	17	8	4	4	4	4	4	4	4	4	4	4
All	2.1 M	2.0 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.3 M	1.2 M	1.1 M	1.0 M	0.8 M

Table 6: Additional comparisons of different bit-score transformations
Using R=`frag_all_aln`. (a) log_{10} (b) binary. (c) identity. See section 4.1.5.
The excluded transforms performed very similarly to the logs.

4.2 Selecting the best reference database *R*

All of the methods for generating reference databases explored in this section will have the set of Pfam-A domains as a starting point, using different methods to select and process sequences for each domain.

4.2.1 Fragmentation of reference sequences

All of the databases in figure 5 and all but the first database (shown for reference) in table 7 are based on the the same set of 7957 UniProt sequences, generated by going through all domains in Pfam-A and selecting the accession with the longest alignment for that domain.

First it's important to point out that the distribution of the nc-score depends on the database, which means one can't compare TPR and FDR directly between fragmentation methods in the figure[4] – that's what the table is for (see also *Discussion* for more about comparing nc-scores generated with different databases).

4.2.1.1 Key observations

Instead what we can get from figure 5 is general features. Observations from the figure and table:

- The `extract_only_primary_aln` database (dotted violet) stands out as performing very poorly in terms of TPR. Since it has the extracted full alignment for each of the 7957 Pfam-A domains, I thought it would perform better.
- Interestingly, the next smallest database – `extract_all_aln` (dotted green) – is only about twice as big and achieves an 85 % TPR – only about 1 percentage point less than the best I've come up with (or equally NC_standalone) at a 0.00 % FDR. I don't understand why it performs so much better than its half-sized companion database – it's based on the same 8 k Pfam domains, just with more homologs. Some of the difference could be due to sequences in the alignment gaps that get included as well.
- It's important to note that while `extract_all_aln` generates (within 1-1.5 percentage points) a similar TPR (at the same FDR $< 0.00\,\%$) to `frag_all_aln` (dashed green), there is a large difference in the total number of identified homologous pairs – SNC finds $\sim 50 - 70\,\%$ more pairs with the latter.
- So we see that databases with very comparable TPR at the same FDR can vary a factor of 2 when it comes to total number of identified homologous pairs. How many of these are false positives? We have no way of knowing for sure and this is a limitation with this analysis. But in terms of evaluating the database-scale-down method, we can compare the numbers with the complete database case (first

[4] Ideally these plots should be aligned so the midpoint of the x-axis always represents the nc-score giving same FDR, for example 0.1%. The x-axis would then instead of nc-score show an nc-score *offset*.

entry in table 7), and then there is no question that the databases containing the complete sequences – fragmented or not – are better.[5]

- Very surprisingly, the "non domain" fragments database (dotted gray) performed on-par with or better than the primary-domain-only database. Notably it achieves a 67 % TPR at a 0.01 % FDR. Keep in mind that in this database we have gone through the effort of *filtering out* all non-overlapping Pfam-A domain alignments to each sequence. It should be *mostly clean of* annotated domains[6], yet it performs as well as a database containing *only* annotated domains.[7] While the filtering process could be incomplete, this suggests that annotated domains only contain part of the information needed for homology inference. Is the rest mostly unannotated domains or other sequence features?

4.2.1.2 Decision

Apart from the method that extracts only one domain per UniProt sequence, all of these methods perform well – with $\sim 84-86$ % TPR with so few FP that they can be counted on one hand. See figure 4 for a scatter-plot and histogram of the comparing the nc-score distributions of `HSA+MMU` (our "gold standard" query-species reference) and `pfam frag_all_aln` (see table 2). Note that there are many horizontal bands but hardly any vertical. I'm interpreting this as NC being able to identify more fine-grained differences in homology using the larger and evolutionarily closer `HSA+MMU` data-set. There are some bands with very low scores for `HSA+MMU` that are in the $0.45-0.70$ range for `pfam frag_all_aln`. At the same time, the latter database gives scores for all FP's safely below 0.2 so it's hard to know for sure what to make of this. We do unfortunately see a band of TP at the bottom that `HSA+MMU` caught but `frag_all_aln` is missing. I think the reasonable interpretation is that `frag_all_aln` is missing some pairs, but achieves better separation on the pairs it finds – which is what we would expect to find if the method (fragmentation) is better but the database (not species-specific) is worse.

I've chosen `frag_all_aln` as the default database for the project as it's the best non-species-specific one.

[5] It should be mentioned that ideally one would like to compare the actual called pairs – not just the total count. I.e. is it that one method just finds more pairs, or do they find *different* pairs?

[6] So one would think – but this should be verified.

[7] I added the "discarded"-database to my analysis mostly for fun, not knowing if it would work at all.

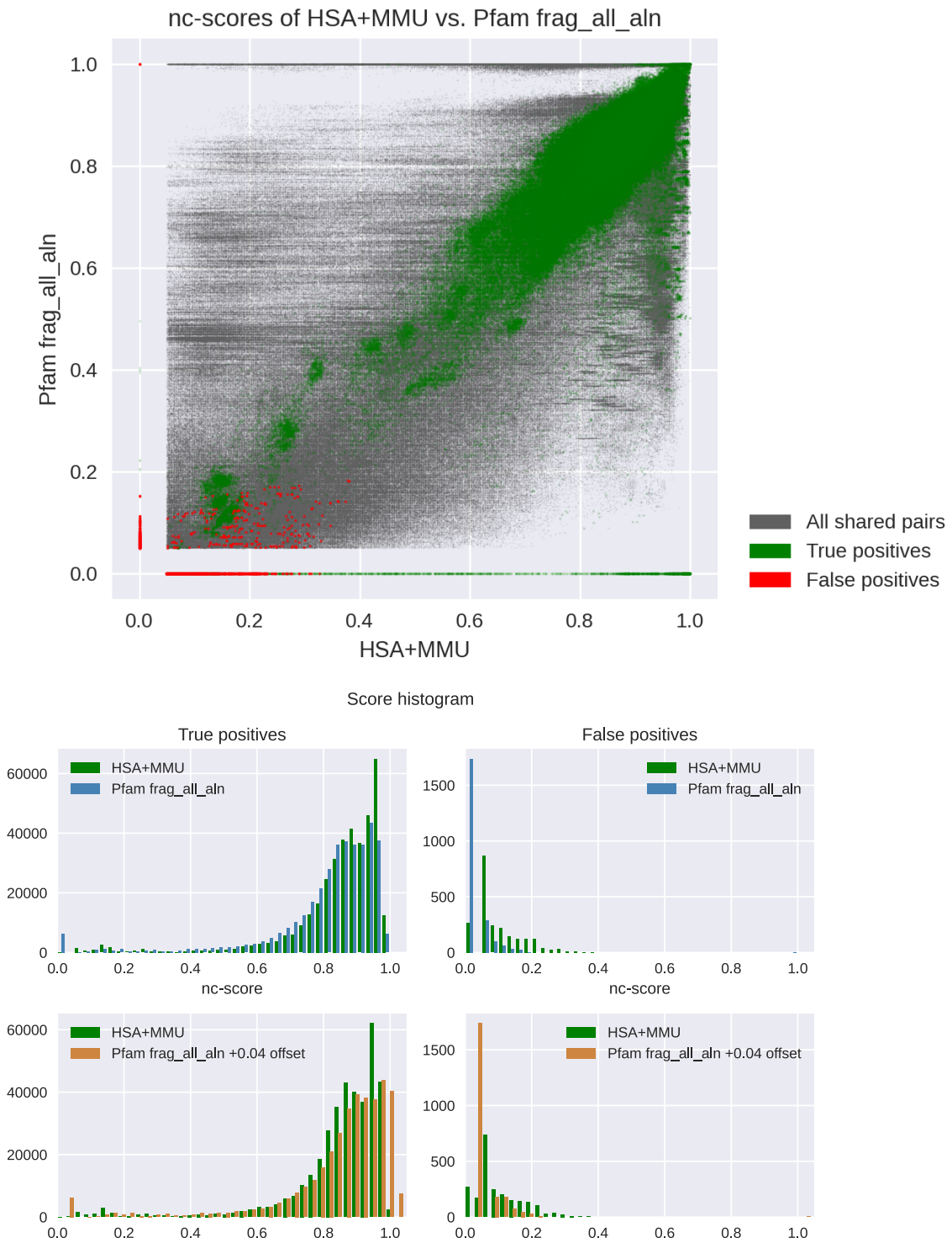

Figure 4: Comparison of reference databases: `HSA+MMU` vs `frag_all_aln`
Top figure (scatter-plot): N.B.: Green dots (TP) are bigger and red dots (FP) biggest. Bottom figure: histogram of same data (the lower half has the scores generated by the `frag_all_aln` database offset by +0.04 (the average score difference) to make the datasets more comparable. Scores of FP and TP set to 0 if missing. See section 4.2.1.2.

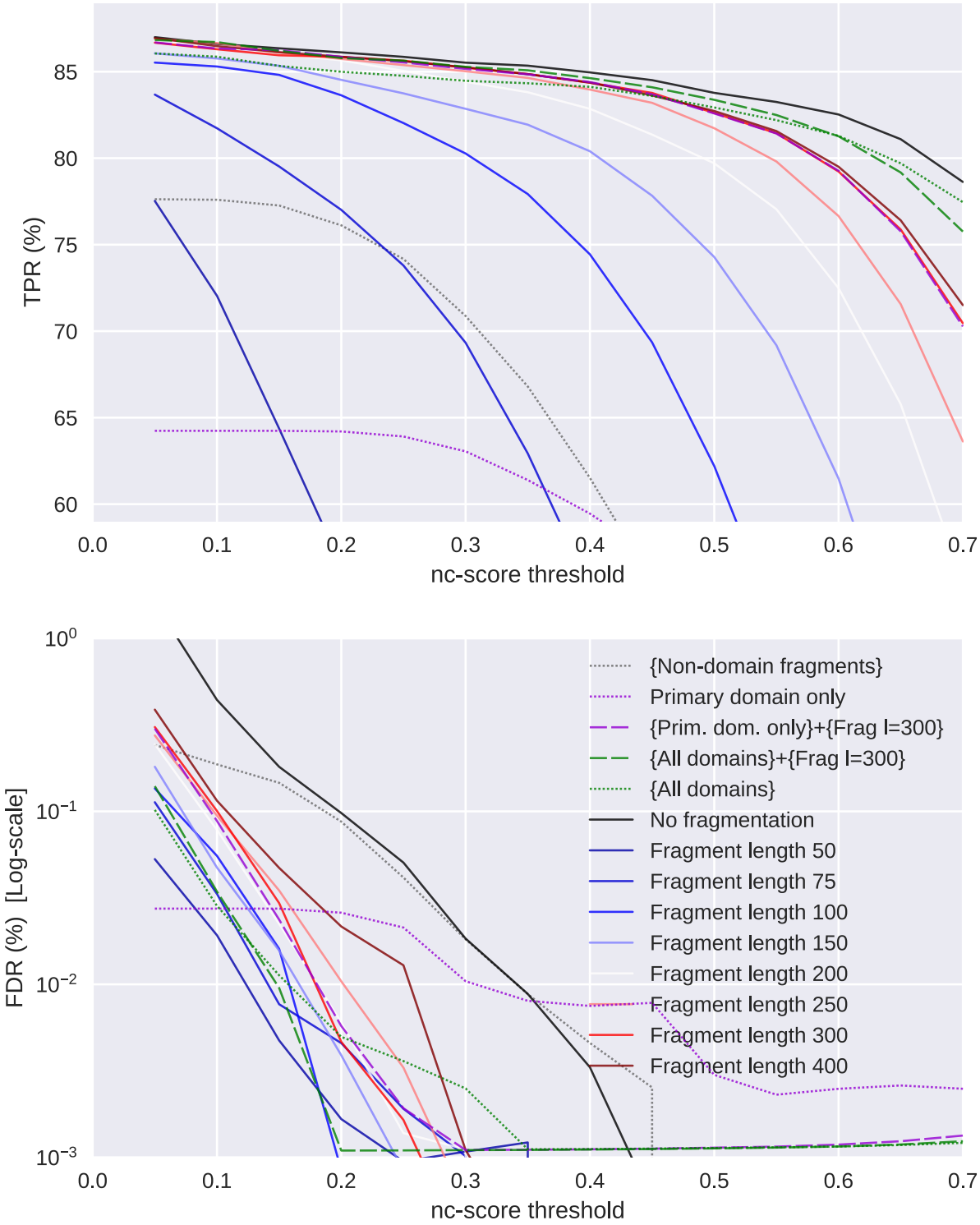

Figure 5: Comparison of different fragmentation methods
The figure shows the TPR % (upper plot) and FDR % (lower plot, log scale) of different reference db sequence fragmentation methods. Default settings (BS $\geq$ 30, log_{10}) and Q =HSA$\cup$MMU was used. See section 4.2.1. See table 2 for databases.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
hsa+mmu no_frag														
(HSA∪MMU) =R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	88.28	87.79	87.00	86.49	86.30	85.94	85.75	85.62	85.29	84.90	84.43	83.45	81.98	79.80
FNR (%)	11.72	12.21	13.00	13.51	13.70	14.06	14.25	14.38	14.71	15.10	15.57	16.55	18.02	20.20
FDR (%)	**0.53**	**0.23**	**0.14**	**0.07**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1986	877	509	256	87	27	5	0	0	0	0	0	0	0
All	4.6 M	3.2 M	2.7 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M
extract_only_primary_aln														
{σ | σ w. longest aln. to δ ∈ Pfam-A domains} —extract prim. dom. in σ→ R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	64.24	64.24	64.24	64.20	63.91	63.05	61.39	59.45	57.05	54.83	50.99	47.14	45.11	37.70
FNR (%)	35.76	35.76	35.76	35.80	36.09	36.95	38.61	40.55	42.95	45.17	49.01	52.86	54.89	62.30
FDR (%)	**0.03**	**0.03**	**0.03**	**0.03**	**0.02**	**0.01**	**0.01**	**0.01**	**0.01**	0.00	0.00	0.00	0.00	0.00
FP	75	75	75	71	58	28	21	19	19	7	5	5	5	4
All	0.7 M	0.7 M	0.7 M	0.7 M	0.7 M	0.7 M	0.7 M	0.6 M	0.6 M	0.6 M	0.5 M	0.5 M	0.5 M	0.4 M
extract_all_aln														
{σ | σ w. longest aln. to δ ∈ Pfam-A domains} —extract all domains in σ→ R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	86.05	85.87	85.33	84.99	84.76	84.47	84.33	84.12	83.60	82.93	82.20	81.29	79.70	77.46
FNR (%)	13.95	14.13	14.67	15.01	15.24	15.53	15.67	15.88	16.40	17.07	17.80	18.71	20.30	22.54
FDR (%)	**0.10**	**0.03**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	374	104	41	18	13	9	4	4	4	4	4	4	4	4
All	1.4 M	1.3 M	1.2 M	1.2 M	1.1 M	1.0 M	1.0 M	0.9 M	0.9 M	0.9 M	0.8 M	0.8 M	0.7 M	0.7 M
frag_all_aln														
{σ | σ w. longest aln. to δ ∈ Pfam-A domains} —extract all dom. in σ, frag. remains (l=300)→ R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	86.83	86.71	86.21	85.79	85.64	85.28	85.08	84.63	84.10	83.37	82.49	81.27	79.17	75.76
FNR (%)	13.17	13.29	13.79	14.21	14.36	14.72	14.92	15.37	15.90	16.63	17.51	18.73	20.83	24.24
FDR (%)	**0.14**	**0.03**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	518	127	35	4	4	4	4	4	4	4	4	4	4	4
All	2.2 M	2.1 M	1.9 M	1.8 M	1.7 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M	1.1 M	1.0 M	0.9 M
frag_raw_seq:300														
{σ | σ w. longest aln. to δ ∈ Pfam-A domains} —fragment σ (l=300)→ R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	86.68	86.30	85.95	85.85	85.63	85.23	84.86	84.36	83.77	82.61	81.44	79.23	75.88	70.46
FNR (%)	13.32	13.70	14.05	14.15	14.37	14.77	15.14	15.64	16.23	17.39	18.56	20.77	24.12	29.54
FDR (%)	**0.31**	**0.10**	**0.03**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1138	370	108	17	6	1	1	0	0	0	0	0	0	0
All	2.6 M	2.4 M	2.2 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.1 M	1.0 M	0.9 M
no_frag														
{σ | σ w. longest aln. to δ ∈ Pfam-A domains} → R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	86.99	86.61	86.35	86.11	85.86	85.52	85.35	84.96	84.51	83.77	83.25	82.53	81.10	78.63
FNR (%)	13.01	13.39	13.65	13.89	14.14	14.48	14.65	15.04	15.49	16.23	16.75	17.47	18.90	21.37
FDR (%)	**1.61**	**0.44**	**0.18**	**0.10**	**0.05**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	6059	1633	667	359	185	67	32	12	2	0	0	0	0	0
All	4.3 M	3.4 M	2.8 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.5 M	1.4 M	1.3 M	1.2 M	1.0 M
random														
{σ | 1 random σ with align to δ ∈ Pfam-A domains} →R×(HSA∪MMU) —align BS≥30→ —transf. log10→ —snc→ All —validate w. curated→ TP FN FP														
TPR (%)	87.71	87.20	86.85	86.45	85.96	85.70	85.41	85.22	84.89	84.37	83.62	82.92	81.92	79.90
FNR (%)	12.29	12.80	13.15	13.55	14.04	14.30	14.59	14.78	15.11	15.63	16.38	17.08	18.08	20.10
FDR (%)	**1.08**	**0.36**	**0.20**	**0.15**	**0.10**	**0.07**	**0.05**	**0.03**	**0.01**	0.00	0.00	0.00	0.00	0.00
FP	4071	1347	744	559	373	254	196	126	51	1	0	0	0	0
All	4.8 M	3.6 M	3.0 M	2.6 M	2.4 M	2.2 M	2.0 M	1.8 M	1.7 M	1.5 M	1.4 M	1.3 M	1.2 M	1.0 M
nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70

Table 7: Comparison of different fragmentation methods

Comparing four different ways of fragmenting, possibly subsampling the exact same sequences. No fragmentation shown second to last. A completely different set of sequences shown at the top (HSA+MMU complete database). At the very bottom: randomly selecting one aligned sequence per Pfam-A domain. See section 4.2.1.

4.2.2 To fragment or not to fragment? Lessons from artificial multidomain proteins

Using the `extract_only_primary_aln` set of domain-only sequences ($\sim 8\,\mathrm{k}$) extracted from UniProt seqs based on alignments to Pfam-A domains, I generated artificial multidomain sequences a by randomly concatenating k ($k \in \{1, 2, 3, 4, 5, 7, 10, 15, 20\}$) of these domains together (with each domain belonging to exactly one "artificial" sequence):

$$M_k = \{a = (d_1, ..., d_k) \mid d_{1 \leq i \leq k} \in \texttt{extract\_only\_primary\_aln}\}$$

The size of M_k is $\mathrm{len}(source)/k$ meaning the smallest database (20 domains per seq) is only 398 sequences long.

Eight trials were ran to get stable averages. As we see on figure 6, the validation results indicate that there is no "safe level" of domain multiplicity, and especially the FDR is badly affected – which one might expect. To understand why this might be expected, let's look at an example. Consider the following two database sequences:

$$r_1 = (d_1 d_2), r_2 = (d_3 d_4)$$

and this pair of query sequences:

$$q_1 = (d_1 d_3)$$
$$q_2 = (d_2 d_4)$$

Because r_1 and r_2 are multidomain sequences, q_1 and q_2 will get an optimal nc-score! If the reference sequences had been properly fragmented into single domains $r_i = \{d_i\}$, then (q_1, q_2) would have gotten a zero nc-score (assuming no other shared hits).

However, since this database is so small it might not be all that representative. For this reason I ran the same analysis using the similar but larger `extract_all_aln` database ($\sim 16\,\mathrm{k}$ seqs) instead, let's call these databases N_k (see figure 7). At low nc-scores the FDR is much worse. At the same time TPR is up to twice as high for the higher nc-score thresholds. It seems to me that there wasn't enough information in the M_k's and that the N_k's are more representative of the "real" situation. It seems the NC method is very capable of filtering out the false identifications that multidomain proteins cause – but at the cost of a significant drop in TPR. Looking at the case of 3 domains/seq, we seem to be loosing about $1.5 - 3$ percentage-points in TPR at the same FDR compared to 1 domain/seq. It's also important to point out that there is a lot of variation between the random trials. Getting the "wrong" domains (common ones – I guess) merged can apparently have big effect.

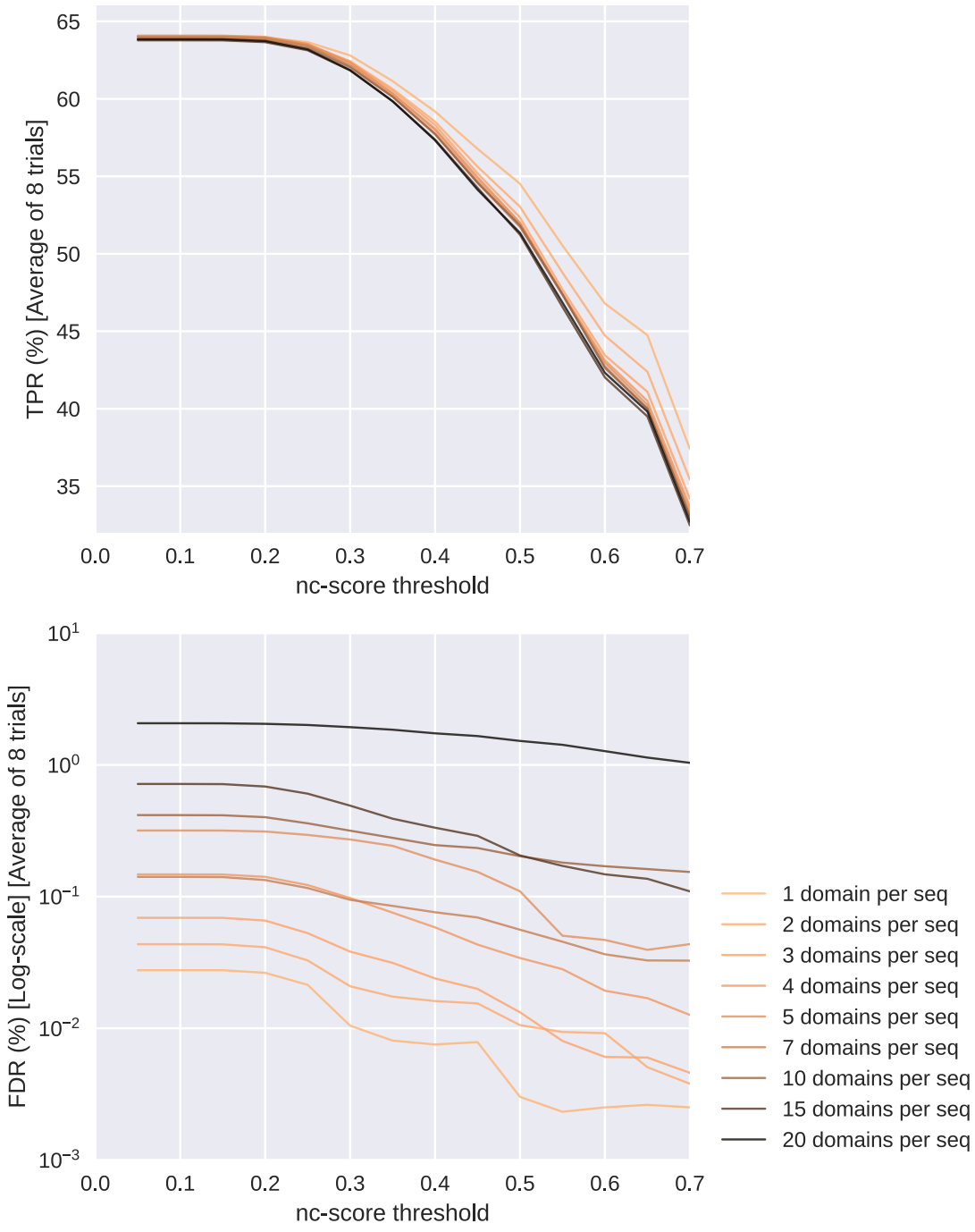

Figure 6: Artificial multidomain seqs as ref. db (**based on** `extract-only-primary-aln`) The figure shows the average TPR % (upper plot) and FDR % (lower plot) of eight randomized trials for each domain count. Default settings (BS $\geq$ 30, log_{10}) and Q =HSA$\cup$MMU was used. See section 4.2.2.

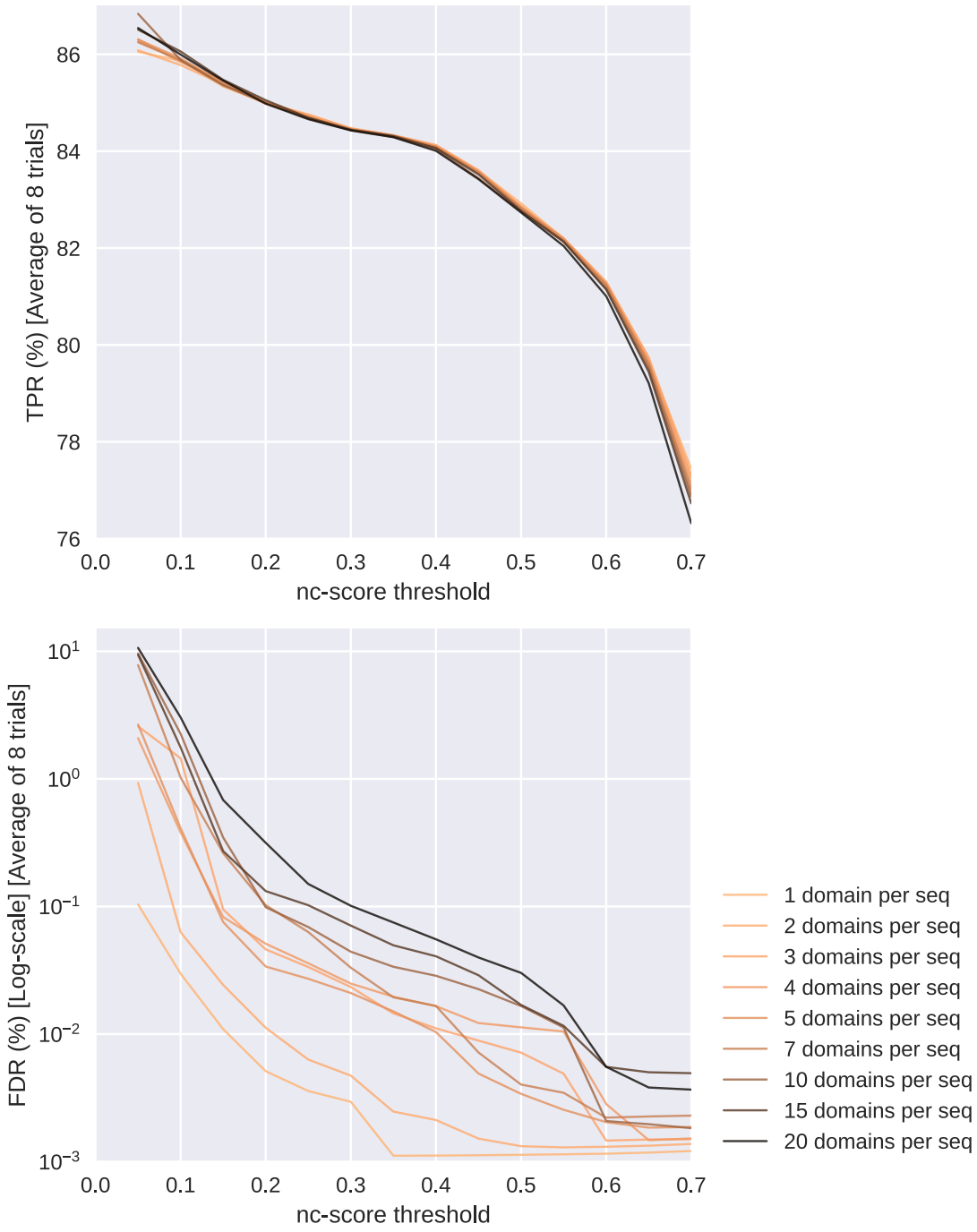

Figure 7: Artificial multidomain seqs as ref. database (**based on** `extract-all-aln`) The figure shows the average TPR % (upper plot) and FDR % (lower plot) of eight randomized trials for each domain count. Default settings (BS $\geq$ 30, log_{10}) and Q =HSA$\cup$MMU was used. See section 4.2.2.

4.2.3 Sequence with longest alignment vs. random sequence(s)

Selecting the UniProt sequence that has the longest alignment to a given Pfam domain entry seems reasonable. One way of testing this assumption is to validate the longest-alignment-sequence against simply selecting a random sequence, which is what we will now do. Compare the last two entries in table 7. At low FP counts, the randomly selected database is very slightly inferior.

Including additional random sequences (3 or 5 in total) per domain is of little use as can be seen in figure 8: the TPR is barely affected until very high nc-scores, the FDR is still comparatively high – and all this comes at a massive cost in terms of database size.

4.3 Consistency of snc vs. NC_standalone

The distribution of scores of true positives and false negatives can be seen in figure 9 (histogram) and in figure 10 (scatter-plot at bit-score threshold 30). My best guess is that the rather big difference in average nc-score is caused by the default score that NC_standalone adds to unaligned pairs (see section 2.1.4.1).[8]

As can be seen in table 8, once the nc-score offset has been accounted for, the implementations perform very similarly. For example at bit-score 30, SNC has a TPR of 85.94 % (at FP=27), while NC_STANDALONE has a TPR of 85.93 % (at FP=26).

One thing to note is that it is very comforting to see that while the distribution of true positives sadly has a pretty long tail into "false negative territory", the distribution of true negatives comfortably tight, further increasing the confidence in the Neighborhood Correlation method.

In conclusion, the implementations show similar classification performance.

[8] I have not tested this hypothesis as it didn't seem to affect the end results much.

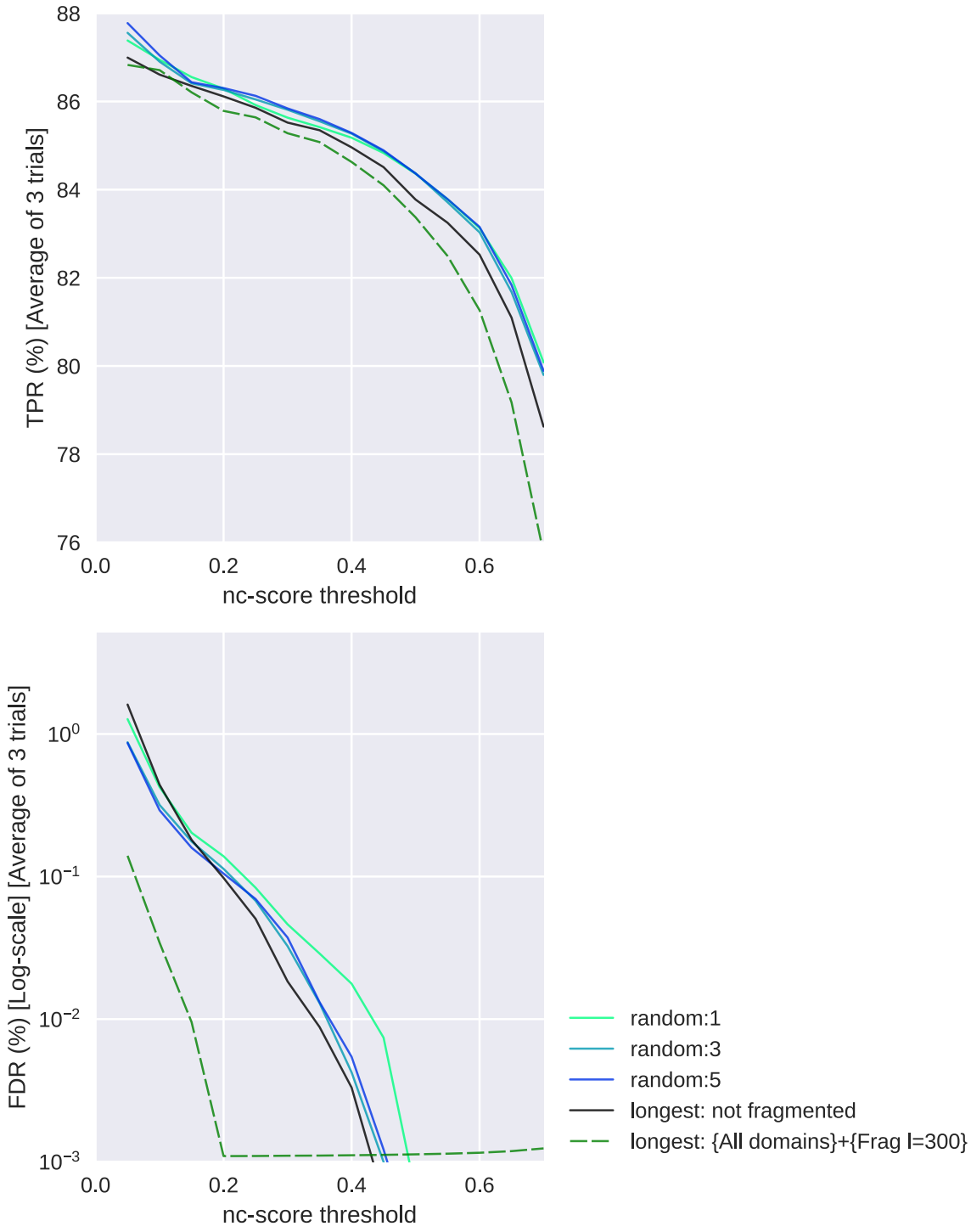

Figure 8: Comparison of number of random sequences per Pfam domain
The figure shows the TPR % (upper plot) and FDR % (lower plot, log scale) of different numbers of random sequences selected per Pfam domain. Default settings (BS ≥ 30, log_{10}) and Q =HSA$\cup$MMU was used. Two databases using the longest domain alignment included for reference.

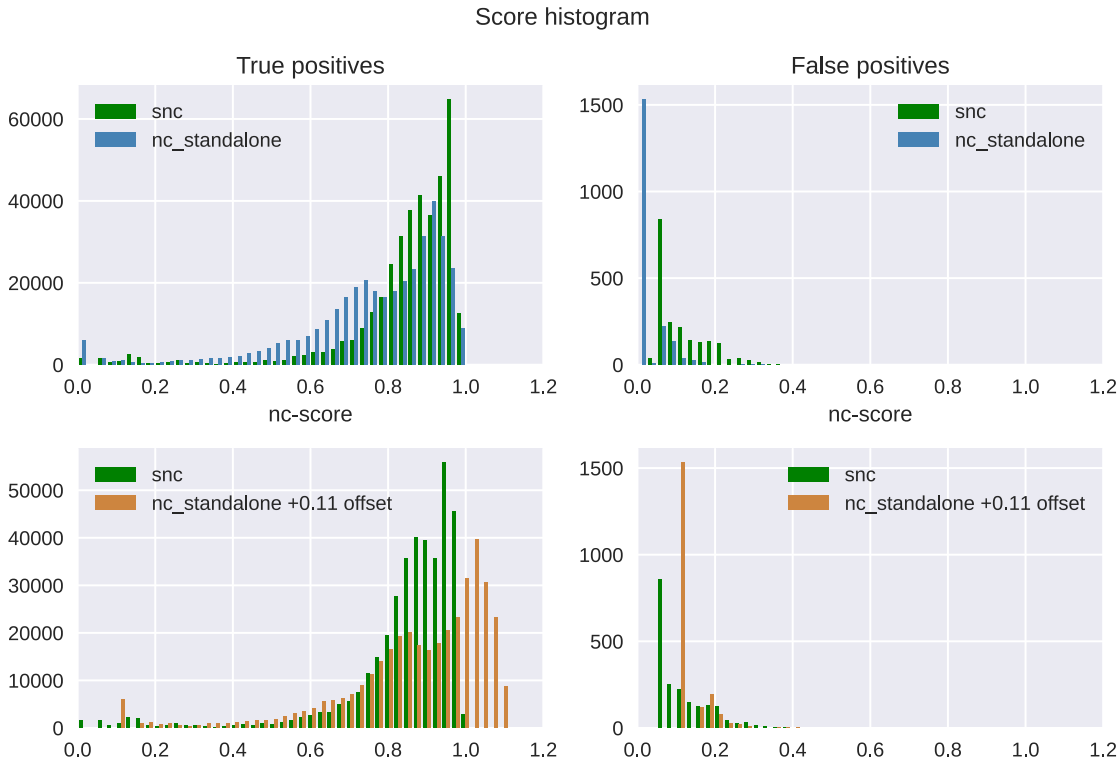

Figure 9: SNC vs. NC_STANDALONE: histograms of nc-scores compared TP (left column) and FP (right column) of identified homologous pairs as determined by validation against the curated annotated proteins. Top row: raw nc-scores compared. Bottom row: nc-scores with the scores of NC_STANDALONE offset by +0.11 (the average score difference). Score set to 0 if missing. Underlying data: $Q = R =$ `HSA+MMU`, bit-score threshold 30, $\log_{10}$ transformed.

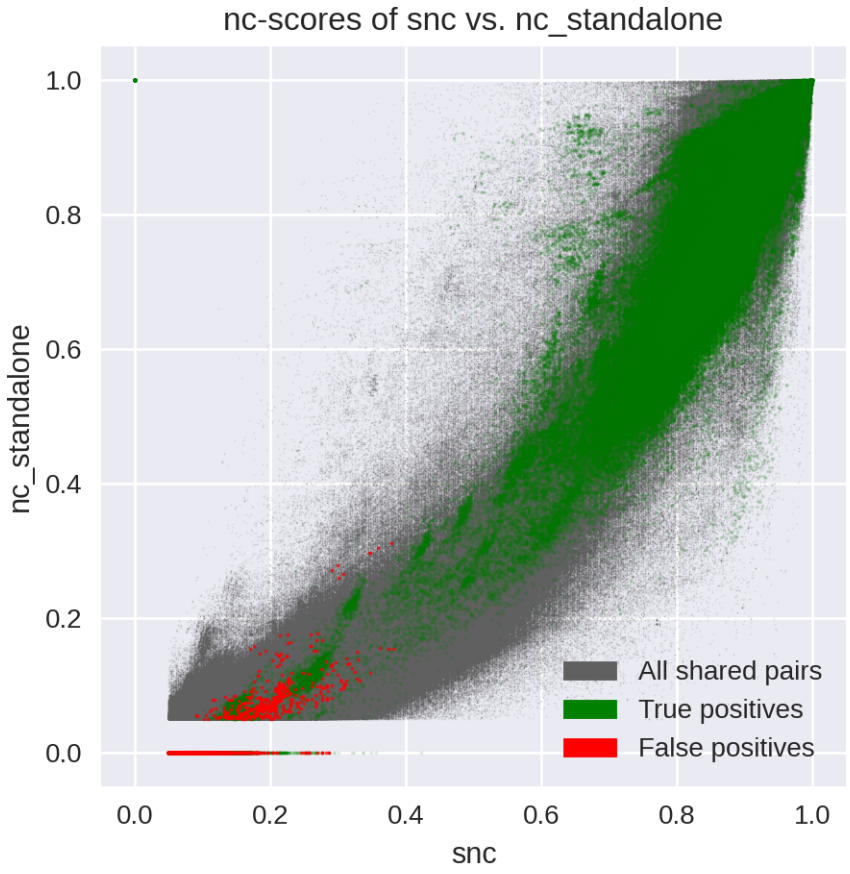

Figure 10: SNC vs. NC_STANDALONE: nc-scores compared
Equal output from both programs would correspond to the line x=y. Green dots (TP) are bigger and drawn on top of the gray dots (all homologs found by both programs). The red dots (FP) are made to stand out even more. Scores of FP and TP set to 0 if missing. See table 8 for a numeric summary. See section 4.3. Underlying data: $Q = R =$ `HSA+MMU`, bit-score threshold 30, $\log_{10}$ transformed.

nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70
(a) bit-score = 20														
(HSA ∪ MMU)=R × (HSA ∪ MMU)	align, BS≥20 → transf., log10 → NC_standalone → All → validate, w. curated → TP FN FP													
TPR (%)	86.46	86.05	85.85	85.58	85.10	84.53	83.92	83.15	81.94	80.12	77.47	74.36	69.57	62.35
FNR (%)	13.54	13.95	14.15	14.42	14.90	15.47	16.08	16.85	18.06	19.88	22.53	25.64	30.43	37.65
FDR (%)	**0.10**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	372	64	19	8	6	0	0	0	0	0	0	0	0	0
All	2.7 M	2.1 M	1.9 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.3 M	1.2 M	1.1 M	1.1 M	1.0 M	0.9 M
(b) bit-score = 30														
(HSA ∪ MMU)=R × (HSA ∪ MMU)	align, BS≥30 → transf., log10 → snc → All → validate, w. curated → TP FN FP													
TPR (%)	88.28	87.79	87.00	86.49	86.30	85.94	85.75	85.62	85.29	84.90	84.43	83.45	81.98	79.80
FNR (%)	11.72	12.21	13.00	13.51	13.70	14.06	14.25	14.38	14.71	15.10	15.57	16.55	18.02	20.20
FDR (%)	**0.53**	**0.23**	**0.14**	**0.07**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	1986	877	509	256	87	27	5	0	0	0	0	0	0	0
All	4.6 M	3.2 M	2.7 M	2.5 M	2.3 M	2.1 M	1.9 M	1.8 M	1.7 M	1.6 M	1.5 M	1.4 M	1.3 M	1.2 M
(HSA ∪ MMU)=R × (HSA ∪ MMU)	align, BS≥30 → transf., log10 → NC_standalone → All → validate, w. curated → TP FN FP													
TPR (%)	86.90	86.32	85.93	85.76	85.38	84.88	84.17	83.40	82.25	80.52	77.90	74.87	70.29	63.22
FNR (%)	13.10	13.68	14.07	14.24	14.62	15.12	15.83	16.60	17.75	19.48	22.10	25.13	29.71	36.78
FDR (%)	**0.12**	**0.02**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	451	89	26	8	8	2	0	0	0	0	0	0	0	0
All	2.8 M	2.2 M	2.0 M	1.8 M	1.6 M	1.5 M	1.4 M	1.4 M	1.3 M	1.2 M	1.2 M	1.1 M	1.0 M	0.9 M
(c) bit-score = 40														
(HSA ∪ MMU)=R × (HSA ∪ MMU)	align, BS≥40 → transf., log10 → snc → All → validate, w. curated → TP FN FP													
TPR (%)	86.44	86.00	85.73	85.38	84.93	84.26	83.97	83.66	82.94	81.80	80.75	79.24	77.60	75.06
FNR (%)	13.56	14.00	14.27	14.62	15.07	15.74	16.03	16.34	17.06	18.20	19.25	20.76	22.40	24.94
FDR (%)	**0.16**	**0.08**	**0.05**	**0.03**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	584	290	194	103	5	0	0	0	0	0	0	0	0	0
All	3.0 M	2.2 M	1.9 M	1.7 M	1.6 M	1.5 M	1.5 M	1.4 M	1.4 M	1.3 M	1.3 M	1.2 M	1.1 M	1.1 M
(HSA ∪ MMU)=R × (HSA ∪ MMU)	align, BS≥40 → transf., log10 → NC_standalone → All → validate, w. curated → TP FN FP													
TPR (%)	85.97	85.63	85.07	84.45	83.73	83.19	82.41	81.23	79.64	77.96	75.92	72.82	68.06	60.57
FNR (%)	14.03	14.37	14.93	15.55	16.27	16.81	17.59	18.77	20.36	22.04	24.08	27.18	31.94	39.43
FDR (%)	**0.06**	**0.01**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FP	222	31	8	8	0	0	0	0	0	0	0	0	0	0
All	2.2 M	1.8 M	1.6 M	1.5 M	1.4 M	1.4 M	1.3 M	1.3 M	1.2 M	1.2 M	1.1 M	1.1 M	1.0 M	0.9 M
nc-score threshold	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70

Table 8: Comparison of SNC vs. NC_STANDALONE

Comparison of NC implementations at three different bit-score thresholds: (a) Bit-score = 20 [data for SNC is unavailable (out of memory)]. (b) Bit-score = 30. (c) Bit-score = 40.

5 Discussion

5.1 The case for fragmentation

One of the hardest pipeline parameters to determine was fragmentation (if any). There is a trade-off between implementation complexity, speed and optimal sensitivity and specificity. However in the general case, we have seen that *fragmentation* outperforms *no fragmentation* even using a simplistic homogeneous algorithm. It only makes sense then to use fragmentation, but to try to make the fragments map to biologically meaningful subsequences, such as aligned domains. Perhaps there are better ways of doing this than using Pfam domains.

Going back to the artificial multidomain experiment in section 4.2.2, I think the conclusion is that multidomain entries have the potential of generating many false positives, but that the Neighborhood Correlation algorithm is very effective in removing these. However, we observe no downside to fragmenting the sequences in terms of TPR and FDR – in fact, the TPR is a noticeably higher (fig. 7).

It's worth noting that classification performance isn't the only factor in deciding to fragment: it also affects the speed of the program. We saw in the Theory section that NC has (optimally) time complexity $O\left(\frac{n^2m^2}{s}\right)$. What is the effect of fragmentation? Let's assume we have a database of size s =100000 entries, $m = 20$ and cut each sequence in half. s is now 200000, and k will *likely* be *mostly* constant (even though it will increase a bit, since DIAMOND by default only reports one HSP per alignment), thus m will be cut in half as well. This would mean: $O\left(\frac{n^2m^2}{s}\right) \xrightarrow[(pieces=2)]{fragment} O\left(\frac{n^2m^2}{8s}\right)$, i.e. an almost order-of-magnitude improvement in run-time from just fragmenting once.

In conclusion, while clearly there is no point in fragmenting too much – the proteins won't align properly anymore – we now have the following arguments in favor of extensive fragmentation:

1. Experimental findings of improved performance on actual data (see table 7).
2. Evidence of improved performance on simulated data (see section 4.2.2).
3. An argument of reason:
   We are trying to create protein "fingerprints" that look similar for homologs. Matching features (domains mostly) rather than entire proteins makes sense in this context. Another way of saying this is that the "likeness" we find by aligning to a whole protein we would have found with a fragment anyway, but the whole protein comes with unwanted "likeness" included. To put it succinctly: whole – especially

large – proteins in the database *will contaminate the queries* – making it look like they have similarity to domains they don't contain.

4. An efficiency argument as described above, clearly showing how inefficient it is to compute all the cross-product terms for a large group of partially unrelated proteins that happens to match an overly-generic target sequence.

5. Possibly: the potential for increased discoverability for the identified homologs, i.e. if the database is (somewhat) curated and annotated, just looking at what database sequences are involved will immediately give some clues about the functionality of a novel homologous cluster – in contrast to the case where they "happened to match a bunch of random sequences".

In conclusion, some form of fragmentation is necessary for scalable Neighborhood Correlation.

5.2 Comparing nc-scores generated with different R

An important point can be learned from the results in section 4.2: an nc-score without some kind reference is *close to meaningless!* [1] A "high" score with one database can be "low" with another. See figure 9 for a nice illustration of this effect. However, the good news is that – as discussed in section 4.3 – normalizing the nc-scores (so that they have equal averages) seems to work well. [2]

5.3 Limitations of my study

See also *Recommendations for future studies* below and *Things I didn't have time to try but might be worth looking into* in Appendix.

Key limitations

1. Lack of extensive validation data for `HSA+MMU` and lack of validation data for more species, thus making it hard to make general conclusions on the performance of the method.

2. Small data-sets for testing – it would have been nice to illustrate the theoretical discussion on scalability with actual data.

3. Only one main method for generating reference databases (i.e. using Pfam-A domains). It would have been nice to test some methods using a selection of sequences from UniRef50 – that's however a project all in its own right.

[1] The difference in TPR at the same FDR between fragment lengths 100 and 300 is, for example, only a couple of percentage points.

[2] Of course this requires a suitable training set so that pairs with "known" scores can be used as a standard.

5.4 Recommendations for future studies

See also the list of ideas in Appendix.

5.4.1 More extensive validation

All of the experiments in this project have been judged chiefly on calling true and false positives using the small set of curated proteins provided by Song et al. 2008, covering only 4.2% (by number of proteins) of the current reference proteomes of human and mouse. This is problematic, especially as these two proteomes already are a minimally small data-set (two species) for this type of analysis. A better way of calling false positives in particular – on a larger data-set – is needed to confirm the quality of the homolog identifications generated by the methods explored in this project.

5.4.2 General idea: create the reference database as two-step process:

1. Generate candidate sequence fragments

2. Select a subset of these that are non-redundant – for most query sequences at the bit-score threshold that will be used when aligning to them. One way to to this could be to use a vast set of diverse sequences (e.g. UniRef100 (UniProt 2023)) and filter out the candidate sequences having mostly the same alignments – as well as filter out the ones with too few alignments. This could be costly, but wouldn't have to be repeated often.
   (The candidate sequences could be clustered, and their respective alignments to UniRef100 stored as bit-vectors (1=aligned). Pairwise comparisons within these clusters could then be done quite efficiently. As of Sept 2023 UniRef100 contains 365 M sequence clusters. This means that each alignment bit-vector would only use 46 MB of memory and could be handled using *Hamming weight* (bit summation) and *bitwise and* – vectorized of course.)

Note that this idea somewhat goes against the concept of keeping R as small as possible.

5.4.3 Verify the sequence diversity the selected "pfam longest" sequences

I recommend that one tabulate the species of the selected "longest pfam alignment" sequences (see table 2) to see how many come from human or mouse. I've kind of assumed that they are more or less randomly distributed across the entirety of UniProt, but it could be that the HMM:s used by Pfam have a bias for reference species such as human or mouse. This would undermine the validity of the comparisons made in this project under the assumption that the `HSA+MMU` test data is as "random" or evolutionary distant as anything that would be thrown at this database.

5.4.4 Try default scores

Replicate the way NC_STANDALONE assigns default scores in SNC / PNC and compare the results.

5.4.5 Try a different scoring matrix

A different alignment scoring matrix than BLOSUM62 might be optimal for the application of finding distant homologs. There are papers discussing the pros and cons of different scoring matrices. Experiment with this, particularly in combination with different bit-score thresholds.

5.5 A note on *Neighborhood Correlation* being able to find more distant homologs than direct alignments

As is noted in Pearson (2013), a bit-score of 50 is needed to achieve a significant hit (E-value < 0.001) in a database with 7 M entries (assuming typical protein sizes). Pearson recommends a bit-score threshold of 50 as a rule-of-thumb when using direct alignments to identify homology. But as we have seen in section 4.1.4, a bit-score threshold of 50 leaves out many homologous pairs that would have been found at a threshold of 32. At a comparable FDR of 0.00 % (rounded to two decimal points), TPR drops from 85.16 % to 81.20 % for the curated test set when lowering the bit-score threshold from 50 to 32.

Neighborhood Correlation can therefore be seen not only as a method for improving the *specificity* of a homology search (by removing erroneous multidomain matches) but also for potentially increasing the *sensitivity* by allowing a higher sensitivity in the underlying alignments.

6 Conclusion

This thesis has shown that a down-scaled, species-agnostic reference database can produce classification results that are on-par with using the full query-set as reference, according to the available metrics.

We have explored the inherent time complexity of the Neighborhood Correlation method and drawn important conclusions about scalability and reference database design.

The benefits of fragmenting the reference genome have been investigated in multiple ways and a case has been made for fragmentation to be seen as a best-practice. A number of parameters have been tweaked and optimal or near-optimal values have been provided. The method of using Pfam-A as an index of diverse sequences covering a large pool of domains has been explored and found viable.

This thesis presents an algorithm for computing Neighborhood Correlation achieving an improved time complexity compared to existing NC algorithms. During the project a fully working implementation of this algorithm was developed.

Specific key lessons:

1. The optimal nc-score threshold depends on the reference database and nc-scores needs a reference scale to be meaningful (without a reference, one basically has to guess if particular score is "good" or not).
2. There is such a thing as an optimal bit-score threshold, at least with the databases, settings and choice of default score (0) used in this project.
3. Log_{10} as bit-score transform is as good as any.
4. It's super-important to set the correct aligner settings, particularly to use a high sensitivity and a high `--max-target-seqs`.
5. Species-specific databases are better, but a global, species-agnostic database *could probably be almost as good* – but this has to be confirmed.
6. Fragmentation in some form is – as discussed in the *Results* section – needed for optimal performance of NC (both in terms of classification and running the software). Focus on fragmenting into biologically meaningful units (e.g. domains). This seems like an obvious machine learning problem.
7. Just throwing more "random" sequences at NC isn't likely the way forward – it doesn't scale well and gives poor marginal improvements. Rather some kind of "intelligent" choices about both sequence selection and the processing of the sequences should be made. That is not to say that the reference databases should necessarily be very small – that's a price-versus-performance trade-off in the end.

Appendix

Pitfalls

Repeated FASTA accessions

As long as `--max-target-seqs` isn't reached, DIAMOND doesn't seem to care about duplicated accessions – a quick test showed an equal number of alignments. The output of SNC will however differ, thus creating a subtle pitfall which causes incorrectly named sequences to yield unintended results. This was close to corrupting the results of my *synthetic multidomain sequence* analysis, had I not used the base case of domain count of 1 as a control and compared it with my previous results.

Number of alignments reported

The DIAMOND option `--max-target-seqs` is discussed elsewhere in the thesis, but it has to be mentioned here. Had I not stumbled upon this option more-or-less by accident I would have missed it and gotten misleading results for the entire project. It is easy to assume that the default values of a tool are good enough (and often better than any guess a novel user might make), but the lesson in this is that this heuristic often breaks down catastrophically when using a tool in a way that differs from the typical use-case. A user of an aligner typically doesn't want to sort through a thousand matches – hence the lower default threshold – but in this project we (or rather SNC) do.

It's clear I'm not the first one to get bitten by settings like this, see Shah et al. (2019) and Gonzalez-Pech, Stephens, and Chan (2018) – here in the context of BLAST specifically.

SciPy fcluster sometimes returns too many clusters

The function `scipy.cluster.hierarchy.fcluster` takes a linkage matrix (and some additional parameters) and returns a "flat" clustering (which is what I wanted in order to subsample sequences). I used the `maxclust` criterion and specified my requested number of clusters. This is supposed to be a maximum, but I found that the function sometimes returned more than this. The documentation says the function internally tries to find a threshold `r`. I guess this could fail – e.g. if the nodes are too close in the linkage matrix.

The clustering was abandoned, and the effect of this problem was rather small as I recall it.

UniProt API error messages

As mentioned earlier, UniProt entries referenced by Pfam-A were frequently marked as "inactive". When a sequence is missing some formats may not be available, not work as intended or not give the full error message. It was not always immediately clear if an error message is permanent (linked to the entry) or temporary. I only got disconnected from the UniProt API once, and that was after an update to my code that fetched sequences quicker. I found no issues when fetching 2-3 entries per second. A more explicit spec for what error messages are possible would be nice.

On a side-note, there is a large difference in size between the different formats. I found that JSON was many times larger than FASTA.

Also, I built my own cache-solution, which saved me many extra downloads when I had to re-run various scripts.

Things I didn't have time to try but might be worth looking into

Note that this list is a bit out of order.

- Investigate DIAMOND settings for repeat masking, possibly consider adding a separate step for this.
- Attempt to use different reference databases and give the alignments from them different weights – e.g. something like `frag_all_aln` could be given a high weight, while something like a database of clustered UniRef50 sequences could be added in with a lower weight.
- Attempt to give different sequences in the database different weight – one might think that a "real" domain should be "worth" more than some repeat-element. The line is blurry of course. One measure that could be tried is simply multiply by degree of uniqueness – rarer reference sequences will thus have a higher score.
- Verify that the "discarded fragments" database really is mostly clean of Pfam-A domains and try to characterize the sequences in it.
- Find a real-world use-case to demonstrate the method (i.e. scaled-down database), thus motivating further study.
- Use HMMs instead of a sequence aligner to identify domains or motifs within query sequences. (Possible drawback: slow.)
- Try to find a more expansive domain/motif annotation than Pfam, keeping in mind that the database itself does not need to be accurate or "correct", only provide a good quality "ink" with which to "fingerprint" each query protein.

- Alternatively, go beyond using domain databases and try to find a good set of kmers from UniProt (a good candidate would occur across species but not be *too* common). A potential downside (apart from likely being slower) is that it becomes harder to interpret the results – what kind of similarity is it that this method finds? Can we be sure it's actually homology?

- Experiment with heuristics for customizing fragment length (e.g. fragment at certain words – this can even be turned into a machine learning problem).

- Attempt to "clean" the database of domains and other sequence elements that are very frequently occurring. Can the classification process be sped up without loosing accuracy? Can accuracy even be improved by reducing "noise" from such elements? (We have already seen what could be an indication of the latter by the marginally improved classification performance when going from bit-score-threshold 10 to 30.) It should be noted that by using `--max-target-seqs 1000` (as opposed to even higher) we're effectively already doing this to an extent.
  A good start for this is to plot the distribution of the number of aligned sequences per domain.

- Attempt to use clustering on the fragmented database to identify repeated sequences. I can see by just scrolling through the file that there are some.

- Investigate whether sequence features other than domains are important for inference of homology. Start by using additional Pfam entry types: coiled-coil, disordered and motif.

- Larger scale testing. May require snc to be re-implemented. Requires additional validation data for a comprehensive quality analysis.

- Additional validation methods, possibly including manual validation of a random sample of identified homologous pairs.

- Test if returning multiple HSPs per alignment makes any difference.

- Determine if it helps increasing Diamond's `--max-target-seqs` beyond 1000 – or alternatively if it safely can be reduced. For the `HSA+MMU` all-vs-all experiment, 112 queries maxed out the cap of 1000 target sequences. For `frag_all_aln`, no queries maxed out.

- Do something more clever with overlapping domain alignments when fragmenting database sequences. Currently the code just takes the next non-overlapping alignment, but it could consider alignment length, try to achieve optimal coverage, allow a certain amount of overlapping alignments and/or prioritize based on the uniqueness of a given alignment (i.e. how many other sequences this domain aligns to).

- *Outside the scope of using a scaled-down database, but related:*
  Try applying the Pfam-domain-alignment-based fragmentation method to the "classic" use-case of NC, i.e. when the reference database equals the set of query sequences. Could fragmentation improve the classification accuracy?

Glossary

Accession, accession number A unique identifier for a *sequence* in a sequence database, such as *UniProt*.

Aligner A software that produces *alignments*.

Alignment, sequence alignment Given a pair of *sequences* $A = (a_i, ..., a_n), B = (b_1, ..., b_m)$, an alignment can be described as a list of pointers $P = (p_1, ..., p_m)$ satisfying $i < j \Rightarrow p_i \leq p_j$ with each p_i specifying the "best" index of where to place b_i in A. If several sequential p's point to the same a_k only the last p is meaningful (which would cause several letters in B to be "lost", i.e. not aligned, we call this a "gap"). Similarly, a "gap " in A can be caused simply by p_{i+1} being two or more positions ahead of p_i. To denote that the tail part of B isn't aligning to A a special position $p = n + 1$ is used.

What the meaning of "best index" is depends on the particular algorithm, scoring system and other settings.

Intuitively, a sequence alignment is a way of putting two sequences together with the "similar" parts together, much like a translator who is trying to figure out the meaning of two texts – of presumed similar origin but written in two different obscure languages – might start by lining up certain similar-looking words, numbers or names together. Just like that translator might mistakenly put two similar-looking but entirely different words together, an alignment algorithm can accidentally identify sequence similarity that's not biologically meaningful – it's just there by random chance.

The "quality" of an alignment is measured using *bit-score* and *E-value*.

(*Swedish*: linjering.)

Alignment pair A pair of *sequences* that have been identified to be similar by an *aligner*, having produced an *alignment* that describes how they relate to each other.

Alignment score The *bit-score* of an *alignment* between a pair of *sequences*.

Amino acid, aa. The building blocks of proteins, the individual characters of a protein *sequence*. Fur the purposes of homology identification, there are 20 "standard" amino acids worth considering.

Bit-score, BS A measure of alignment quality that corresponds to the log_2 of the database size expected to produce such an alignment by chance. Importantly for our use, it is effectively a function of only the two aligned sequences – allowing for consistent

comparisons across data-sets and predictable results when using a fixed threshold. A useful discussion regarding the determination of significant bit-scores in the context of homology inference can be found in Pearson (2013).

BLAST (software package) BLAST is a set of sequence *aligners* (for different types of sequences and use-cases) using heuristics to produce pairwise alignments at acceptable *sensitivity* faster than more accurate methods. See Theory section. (Altschul et al. 1997)

BLASTP BLASTP the specific variant of *BLAST* that works on protein sequences.

BLOSUM62 In bioinformatics, a BLOSUM matrix is a 20-by-20 symmetric scoring matrix used for calculating distances between aligned protein *sequences*, where each entry s_{ij} contains the cost (positive or negative depending on similarity) of substituting *amino acid* i to j. BLOSUM62 is a commonly used scoring matrix and default for *DIAMOND*.

Deletion When performing *multiple sequence alignment*, some sequences may lack a part of the sequence that other sequences have, thus causing an alignment mismatch. The sequence(s) lacking this bit of sequence are said to have a *deletion* at this point. See also *insertion*. Insertions and deletions are collectively referred to as *indels*.

Domain, protein domain A domain is a functional subunit of a protein and often performs a specialized function or provide a specific structure. The same domain often occur (possibly with small or major modifications) in many proteins, and one protein may have one or several domains. Loosely perhaps one can think of a domain like a Unix tool – does one specific job well – and proteins like shell command "one-liners". In this analogy, programmers typically fork the software repository of the tool every time they write a new shell-command, in order to tweak it just right for the job (or change it significantly).

DIAMOND Sequence aligner. See *Theory* section.

E-value, expectation value Reported by *aligners* such as BLAST or DIAMOND, the E-value for an *alignment* is the expected number of times such an alignment would occur by chance, given the database size and alignment quality (Fassler and Cooper 2011). So an E-value of 1 would mean that, in expectation, one alignment occurring just by random chance would be reported by the aligner when making a query with a quality (*bit-score*) threshold at this level.

FASTA Human-readable file format for storing a list of (*accession*, *sequence*) entries (optionally also a text string of arbitrary metadata per entry). Invented in 1983 (Pearson 2023).

False discovery rate, FDR For a binary classification, FDR $= \frac{FP}{FP+TP}$ is the proportion of false positives among the reported positives.

False negative, FN False negative in a binary classification.

False negative rate, FNR For a binary classification, FNR $= \frac{FN}{\text{all actually positive}} = \frac{FN}{TP+FN} = 1-\text{TPR}$.

False positive, FP False positive in a binary classification.

Hamming weight, bit count, bit summation Defined for a binary number x, the Hamming weight of x is the number of bits set to 1 in x. Can be computed efficiently in modern computers with just one instruction to count an entire register.

Hidden Markov model, HMM A statistical model used (among other things) to model biological *sequences,* capable of modeling *insertions* and *deletions.*

Homology Similarity between biological *sequences* (DNA, RNA or protein) caused by a shared evolutionary ancestry.

HSA Species code for human. In this project referring to the data-set of human proteins, see 3.1.

High-scoring Segment Pair, HSP The BLAST algorithm uses HSPs – a top-scoring gap-less local alignment between the *query* and a *reference* sequence. If it can, BLAST tries to combine multiple HSPs into a longer – possibly global – alignment between the pair of sequences. (Fassler and Cooper 2011)

Insertion (of domains) The term insertion can refer to two different concepts: the insertion of an entire domain (sequence fragment) into another protein (sequence) – likely causing major changes in the function of the protein. See the introduction for more on this.

Insertion (multiple sequence alignment terminology) The term can also refer to the presence of a sequence fragment (1 letter or longer) in one of the *aligned* sequences that is missing in (some of) the other. This makes the sequence no longer align perfectly to other sequences. It's essential to be able to model or handle this when performing *multiple sequence alignment.* See also the companion term *deletion.*

k-mer, kmer *(Primarily in bioinformatics):* A k-mer is a substring of fixed length k. A 7-mer is thus a substring of length 7. A frequency table of k-mers occurring in a sequence can be used to characterize that sequence. K-mers can also be used as signatures, e.g. in a lookup table in an aligner.

MMU Species code for mouse. In this project referring to the data-set of mouse proteins, see 3.1.

Multidomain protein Multidomain proteins – in contrast to single domain proteins – are simply those containing more than one *domain*. The individual domains may be related to other domains (in the same species or other species) in different ways – see *insertion, orthology* and *paralogy* – thus making it difficult to determine the evolutionary relationship between a pair of multidomain proteins.

Multiple sequence alignment, MSA The process of *aligning* more than two biological *sequences* together, often for the purpose of finding homologous genes. The most naive algorithm would be to produce pairwise *alignments* between all pairs of sequences in the input set. Results can be reported in e.g. Stockholm format.

NC-score See the Introduction.

NC_standalone See *Neighborhood Correlation [software].*

Neighborhood Correlation, NC [method] See the Introduction and Theory sections.

Neighborhood Correlation, NC [software] Implementation of Neighborhood Correlation by Song et al. (2008). See algorithm 1 and chapter 2.

Orthology *Homology* caused by *speciation* – i.e. the "same gene" but in another species.

Paralogy *Homology* caused by *gene duplication* (i.e. the gene being duplicated and now existing in two copies in the genome) *at some point in the evolutionary history* of the species (including possibly *today*). This means paralogous relationships can exist between genes both *within* and *across* species.

Pfam Major protein and family domain database. Domains are defined using *HMM*:s. (Mistry et al. 2021)

Pfam-A The main set of entries of Pfam. There is also a lower quality but more complete Pfam-B database, which has not been used in this project.

Pfam-A domain Domain entry in *Pfam-A*. Each entry contains list of sequences aligning to it, which have been used in this project to identify sequences for inclusion in reference databases.

Proteome In the context of this thesis, the set of protein *sequences* for a given species. In general the proteome of an individual cell in a multicellular organism such as you – the reader (assuming you're a person) – may differ from other cells in the same organism (i.e. a skin cell and an immune cell have different proteomes), or differ between different time points.

PNC Implementation of *Neighborhood Correlation* by thesis author Yrin Eldfjell. See algorithm 3 and Theory section.

Reference database, reference, R The database of sequences that an *aligner* attempts to match *query* sequences to. In this thesis *"R"* specifically refers to the reference database used by the Neighborhood Correlation pipeline.

References sequence One entry in a *reference database*.

RP15 Less redundant version of Pfam-A. (Chen et al. 2011)

SciPy Open-source Python library for scientific computing.

Sensitivity Same as *true positive rate*.

Sequence Bioinformatics / biology term for what in computer science is known as a *String*.

snc Implementation of *Neighborhood Correlation* by thesis advisor Lars Arvestad. Uses sparse matrices internally and supports using reference databases $R \neq Q$. See algorithm 2 and Theory section.

Speciation The evolutionary process causing one species – for whatever reason – to become two or more distinct species. Two species being separated by a *speciation event* simply means they are now different species.

Specificity Same as *true negative rate*.

Stockholm format Text-based file format for *multiple sequence alignments*.

TN True negative in a binary classification.

TP True positive in a binary classification.

True positive rate, TPR For a binary classification, $\text{TPR} = \frac{TP}{\text{all actually positive}} = \frac{TP}{TP+FN} = 1-\text{FNR}$.

UPGMA UPGMA is a hierarchical clustering method. Its distance measure for a pair of clusters (A, B) is the mean distance between an element of A and an element of B.

Query set, queries, Q The complete set of *query sequences* used during a single run of an *aligner*. In this thesis *"Q"* specifically refers to the input sequences of the Neighborhood Correlation pipeline – the set of sequences we want to find *homologous pairs* within.

Query sequence The sequence being *aligned* to a *reference database* by an *aligner*. If they are similar, the aligner will report a similarity score such as *bit-score*.

Target, target sequence In this thesis: same as *reference sequence*.

UniProt A central repository combining multiple protein databases, containing on the order of hundreds of millions of sequences (Consortium 2023).

UniProt sequence A single entry – specified by a UniProt *accession* – in UniProt.

Bibliography

Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaeffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (Sept. 1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* 25.17, pp. 3389–3402. ISSN: 0305-1048. URL: https://doi.org/10.1093/nar/25.17.3389.

Arvestad, Lars (Oct. 24, 2023). *snc*. URL: https://github.com/arvestad/snc.

Buchfink, Benjamin, Klaus Reuter, and Hajk-Georg Drost (Apr. 2021). Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature Methods* 18.4, pp. 366–368. ISSN: 1548-7105. URL: https://doi.org/10.1038/s41592-021-01101-x.

Chen, Chuming, Darren A. Natale, Robert D. Finn, Hongzhan Huang, Jian Zhang, Cathy H. Wu, and Raja Mazumder (Apr. 2011). Representative Proteomes: A Stable, Scalable and Unbiased Proteome Set for Sequence Analysis and Functional Annotation. *PLOS ONE* 6.4, pp. 1–9. DOI: 10.1371/journal.pone.0018910. URL: https://doi.org/10.1371/journal.pone.0018910.

Consortium, The UniProt (Jan. 2023). UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Res* 51.D1, pp. D523–D531. ISSN: 0305-1048. URL: https://doi.org/10.1093/nar/gkac1052.

Fassler, Jan and Peter Cooper (2011). *BLAST Glossary*. Bethesda (MD): National Center for Biotechnology Information (US). URL: https://www.ncbi.nlm.nih.gov/books/NBK62051/.

Gonzalez-Pech, Raul A., Timothy G. Stephens, and Cheong Xin Chan (Aug. 2018). Commonly misunderstood parameters of NCBI BLAST and important considerations for users. *Bioinformatics* 35.15, pp. 2697–2698. ISSN: 1367-4803. URL: https://doi.org/10.1093/bioinformatics/bty1018.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Lewin, Harris A. et al. (Apr. 2018). Earth BioGenome Project: Sequencing life for the future of life. *Proceedings of the National Academy of Sciences* 115.17, pp. 4325–4333. DOI: 10.1073/pnas.1720115115. URL: https://doi.org/10.1073/pnas.1720115115.

Li, Heng (2023). *Seqtk*. URL: https://github.com/lh3/seqtk.

Mistry, Jaina et al. (2021). Pfam: The protein families database in 2021. *Nucleic acids research* 49 (D1), pp. D412–D419. DOI: 10.1093/nar/gkaa913.

Mora, Camilo, Derek P. Tittensor, Sina Adl, Alastair G. B. Simpson, and Boris Worm (2011). How many species are there on Earth and in the ocean? *PLoS biology* 9 (8), e1001127. DOI: 10.1371/journal.pbio.1001127.

Pearson, William R. (2013). An introduction to sequence similarity ("homology") searching. *Current protocols in bioinformatics* Chapter 3, pp. 3.1.1–3.1.8. DOI: 10.1002/0471250953.bi0301s42.

— (Nov. 14, 2023). *Was FASTA ever popular?* URL: https://www.biostars.org/p/214943/#215417.

Shah, Nidhi, Michael G. Nute, Tandy Warnow, and Mihai Pop (May 2019). Misunderstood parameter of NCBI BLAST impacts the correctness of bioinformatics workflows. *Bioinformatics* 35.9, pp. 1613–1614. ISSN: 1367-4803. URL: https://doi.org/10.1093/bioinformatics/bty833.

Song, Nan, Jacob M. Joseph, George B. Davis, and Dannie Durand (May 2008). Sequence Similarity Network Reveals Common Ancestry of Multidomain Proteins. *PLOS Computational Biology* 4.5, e1000063. DOI: 10.1371/journal.pcbi.1000063. URL: https://doi.org/10.1371/journal.pcbi.1000063.

Tordai, Hedvig, Alinda Nagy, Krisztina Farkas, Laszlo Banyai, and Laszlo Patthy (Oct. 2005). Modules, multidomain proteins and organismic complexity. *The FEBS Journal* 272.19, pp. 5064–5078. ISSN: 1742-464X. URL: https://doi.org/10.1111/j.1742-4658.2005.04917.x.

UniProt (Nov. 13, 2023). *UniRef.* URL: https://www.uniprot.org/help/uniref.

UniProtKB (Oct. 19, 2023a). *Human reference proteome.* URL: https://www.uniprot.org/uniprotkb?query=reviewed%3Atrue+AND+proteome%3Aup000005640.

— (Oct. 19, 2023b). *Mouse reference proteome.* URL: https://www.uniprot.org/uniprotkb?query=reviewed%3Atrue+AND+proteome%3AUP000000589.

Virtanen, Pauli et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Masteruppsats 2023:1
Datalogi
November 2023

www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm