



Technische Hochschule Bingen
Fachbereich 2 – Technik, Informatik und Wirtschaft
Angewandte Bioinformatik (B. Sc.)

Proteinerkennung auf Basis physikalischer Eigenschaften mittels Fourier-Transformation

Bachelorarbeit
abgegeben am: 26.08.2024
von: Franz-Eric Sill

Betreuer: Prof. Dr. Asis Hallab

Zusammenfassung

...

Abstract

...

Abbildungsverzeichnis

1	Spektralanalyse eines Intervalls/Fensters der STFT	2
2	Bisherige Ergebnisse von prot-fin	3
3	Raincloud Plot der Trainings-Protein Sequenzlängen	5
4	Beispielsequenz als numerische Repräsentation	7
5	Constellation-Map und Hashing	10
6	Schematischer Aufbau eines Hashes	10
7	Ermittlung des S1-Score	14
8	Häufigkeit gewählter Frequenzen über alle Trainings-Proteine vor Exp. 1 .	20
9	Häufigkeit gewählter Frequenzen über alle Trainings-Proteine nach Exp. 1	21
10	Single-Protein-Matching Exp. 1	22
11	Single-Protein-Matching Exp. 2	23
12	Family-Matching Exp. 2	24
13	Single-Protein-Matching Exp. 3	25
14	Single-Protein-Matching Exp. 4, Ansatz 1: Wahl nur mit Grenzwerten . .	26
15	Family-Matching Exp. 4, Ansatz 1: Wahl nur mit Grenzwerten	27
16	Single-Protein-Matching Exp. 4, Ansatz 2: Wahl der Ausreißer der lokalen Maxima/Minima	28
17	Family-Matching Exp. 4, Ansatz 2: Wahl der Ausreißer der lokalen Maxi- ma/Minima	29
18	Single-Protein-Matching Exp. 4, Ansatz 3: Wahl nach Grenzwertabweichung	30
19	Family-Matching Experiment (Exp.) 4 (Seite 18), Ansatz 3: Wahl nach Grenzwertabweichung	31

Tabellenverzeichnis

1	Kidera-Faktoren	7
2	Experiment-Parameter	19

Abkürzungsverzeichnis

ASCII	“American Standard Code for Information Interchange”	16
CM	Constellation-Map	9
CSV	Comma Separated Values	12
Exp.	Experiment	III
FG	Fenstergröße	19
JSI	Jaccard Similarity Index	14
KF	Kidera-Faktor	1
MB	Megabyte	
STFT	Short Time Fourier Transformation	2
TP	Trainings-Protein	11
TZ	Target-Zone	17

Algorithmenverzeichnis

1	Übersetzen einer Aminosäuresequenz in einen numerischen Vektor	6
2	Sammeln von Strukturdaten	8
3	Hashing	9
4	Erstellung der Datenbank	11
5	Treffer-Bewertung beim Single-Protein-Matching	13

Inhaltsverzeichnis

Abstract	II
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Abkürzungsverzeichnis	V
Algorithmenverzeichnis	VI
1 Einleitung	1
2 Material	5
2.1 Trainings-Proteine	5
2.2 UniRef90	5
3 Methoden	6
3.1 Grundalgorithmus	6
Übersetzen einer Aminosäuresequenz in einen numerischen Vektor	6
Sammeln von Strukturdaten	8
Hashing	9
Erstellung der Datenbank	11
Single-Protein-Matching	12
Family-Matching	15
3.2 Experiment 1: UniRef90 Sampling	16
3.3 Experiment 2: Filter Hashes	17
3.4 Experiment 3: Target-Zone	17
3.5 Experiment 4: Selection-Method	18
3.6 Durchführung	18
4 Ergebnisse	20
4.1 UniRef90 Sampling	20
4.2 Filter Hashes	23
4.3 Target-Zone	24
4.4 Selection-Method	25
4.4.1 Ansatz 1: Wahl der Frequenzen nur über Grenzwerte	25
4.4.2 Ansatz 2: Wahl der Frequenzen über Ausreißer in den Extrempunkten	27
4.4.3 Ansatz 3: Wahl nach Grenzwertabweichung	29
5 Diskussion	32
6 Anhang	37
Literatur	38

1 Einleitung

Die Methoden des Next Generation Sequencing ermöglichen sehr effizient das schnelle und kostengünstige Entschlüsseln von ganzen Genomen und folglich ihren Proteomen. Um aus Sequenzen des Proteoms die Funktionen der Proteine abzuleiten, werden in der Bioinformatik seit jeher mittels paarweisen Alignments deren Sequenzen verglichen, mit dem Hintergrund, dass ähnliche Sequenzen auf eine ähnliche Funktion der verglichenen Proteine hindeuten. In der Regel wird dafür BLAST (Basic Local Alignment Search Tool) verwendet, welches mit einer Datenbank sehr zuverlässig in Sequenzen Homologien identifiziert, welche auf Verwandtschaft und in Proteinsequenzen auf funktionelle Ähnlichkeit hindeuten. Allerdings besteht bei zunehmender evolutionärer Distanz eine wachsende Schwierigkeit, solche Homologien von zufälliger Ähnlichkeit zu unterscheiden, da der potentielle Vorfahr zeitlich so weit entfernt ist, dass er ebenso ein Vorfahr eines komplett anderen Proteins sein könnte. So kann es dann sein, dass BLAST mit einer kleinen Datenbank für eine Suchsequenz Homologien findet, die bei einer größeren Datenbank schon nicht mehr als signifikant gewertet werden [vgl. Pea13].

Schon 1981 hat Doolittle sich mit diesem Problem beschäftigt und verschiedene Ansätze angesprochen, welche die Entscheidung über Signifikanz verbessern würden. So könnten zum Beispiel, und werden auch, Multiple Sequenzalignments verwendet werden, also Alignments, bei denen noch zusätzliche Sequenzen herangezogen werden, um Sicherheit über gefundene Homologien in stark konservierten Regionen zu geben. Das würde die Weite evolutionärer Distanz erhöhen, um Verwandtschaft vorherzusagen. Ebenso erwähnt sie die dreidimensionale Struktur der Proteine, die auch weitere Möglichkeiten der Analyse erlauben würde [Doo81]. Allerdings war die Vorhersage dieser Struktur damals sehr unzuverlässig und selbst heute in 2024 mit Verfahren der Künstlicher Intelligenz noch sehr neu und in weiterer Entwicklung, aber mit hohem Zukunftspotential [Jum+21][Var+23].

In dieser Bachelorarbeit soll zwar nicht die vorhergesagte dreidimensionale Struktur der Proteine zur Erkennung funktionaler Ähnlichkeit Verwendung finden, dafür aber ein Algorithmus entwickelt werden, der sich dennoch von den üblichen Sequenzalignments unterscheidet und Information über die physikalischen Eigenschaften der Proteine einbezieht. Der Algorithmus dazu wird in dem Projekt prot-fin entwickelt.

Eine Grundlage hierfür bildet die Arbeit von Kidera u. a., welcher in seiner Forschungsgruppe mittels statistischer Faktorenanalyse 188 physikalische Eigenschaften der 20 natürlich vorkommenden Aminosäuren auf lediglich 10 sogenannte Kidera-Faktoren (KFs) reduziert hat, die zusammen all diese Eigenschaften am besten erklären [vgl. Kid+85]. So ist beispielsweise die Hydrophobizität ein ebensolcher Faktor, da diese mit vielen anderen Eigenschaften stark in Korrelation steht. Vorteil der Reduktion auf 10 KF ist, dass in den Analysen der physikalischen Eigenschaften nicht alle 188 verschiedenen Eigenschaften separat betrachtet werden müssen, sondern dies durch die Korrelation schon indirekt erfolgt, was die Effizienz der Auswertung immens erhöht.

Der Einfluss eines jeden KF in einer Aminosäure lässt sich numerisch darstellen, so dass eine Aminosäuresequenz in 10 Vektoren übersetzt werden kann, welche dadurch ein

statistisch auswertbares Abbild der physikalischen Struktur des Proteins erzeugen.

Der Algorithmus für die Analyse dieser Struktur ist von SHAZAM inspiriert, einer Anwendung, die Musiktitel anhand kürzester Tonaufnahmen identifiziert, selbst wenn diese Störgeräusche aufweisen. Basis hierfür stellt die schnelle Short Time Fourier Transformation (STFT) dar, welche die digitale Tonspur intervall-/fensterweise spektral auf periodisch auftretende Signale analysiert, wodurch auch die Störgeräusche eine geringe Relevanz haben [vgl. Wan03].

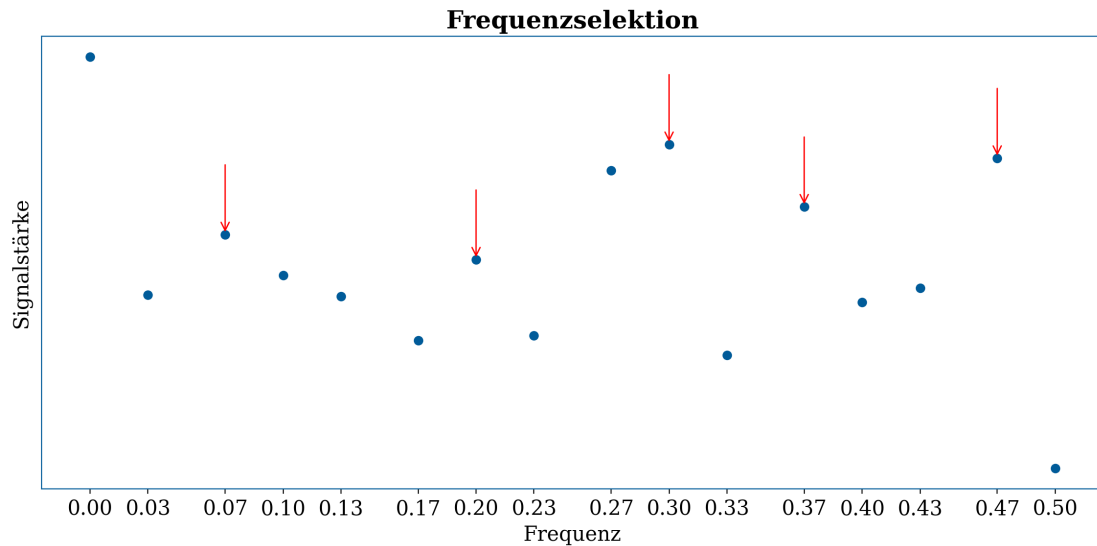


Abbildung 1: Spektralanalyse eines Fensters der STFT: Auf der x-Achse sind die Frequenzen und auf der y-Achse deren Signalstärke dargestellt. Die Pfeile markieren die lokalen Maxima als signifikant, sodass deren Frequenzen in die Datenbank einfließen. Das Reziproke einer Frequenz bedeutet, jedes wievielte Element sich wiederholt.

In Abbildung 1 (Seite 2) wird dieser Sachverhalt für ein Fenster der STFT dargestellt, also das transformierte Intervall eines numerischen Vektors. Es werden die Signalstärken für alle möglichen Frequenzen ermittelt, wobei das Reziproke einer Frequenz hier entspricht, jedes wievielte Element betrachtet wird. Bei Frequenz 0.5 wäre es also jedes $\frac{1}{0.5} = 2$ te Element, hier offenbar sehr schwach ausgeprägt.

Damit die Musikererkennung funktioniert, wird vorher eine Datenbank erstellt, welche die Periodizitäten, also die auffälligen Frequenzen der Eingabesongs, mit denen sich Elemente der Aufnahme wiederholen, mittels Hashing effizient auffindbar abspeichert, sodass der Abgleich mit einer Tonaufnahme sehr schnell und korrekt abläuft [vgl. Wan03]. Zudem werden hierfür aus den STFT-Ergebnissen auch nicht alle Frequenzen verwendet, sondern nur möglichst signifikante davon. Bisher wurde das in prot-fin mittels der lokalen Maxima der Amplituden erreicht.

Für die Anwendung auf Proteine werden statt des musikalischen Spektrums die nume-

rischen Vektoren der Aminosäuresequenzen verwendet. Nun gibt es in prot-fin zwei verschiedene Anwendungsansätze:

1. **Single-Protein-Matching:** Als Eingabe erfolgt eine einzelne Aminosäuresequenz, für die das best passende Protein gesucht wird. Je mehr Übereinstimmung herrscht, desto funktionsähnlicher sollte es sein.
2. **Family-Matching:** Als Eingabe erfolgt eine Proteinfamilie. Die Periodizitäten, in denen sich alle Mitglieder dieser Familie ähneln, die also spezifisch für die Familie sind, werden verwendet, um Proteine zu finden, die auch in die Familie passen.

Um den Algorithmus, der von einer SHAZAM-Implementierung in Python ausgeht [Str21], für beide Ansätze auf die vergleichsweise kurzen Sequenzen von wenigen 100 Elementen abzustimmen (ein solcher Vektor für eine Sekunde Musik hätte etwa 40.000 Elemente), wurde in vorangegangenen Experimenten versucht, die Fenstergröße und die Überlappung zwischen diesen Fenstern bei der STFT zu optimieren, beziehungsweise auch die Anzahl gewählter Frequenzen, deren Amplituden auf Periodizität hindeuten. Hierbei zeigte sich bisher allerdings eine schlechte Performanz hinsichtlich Speicher- und Laufzeitkomplexität. Die Ergebnisse dazu sind in der Abbildung 2 (Seite 3) dargestellt.

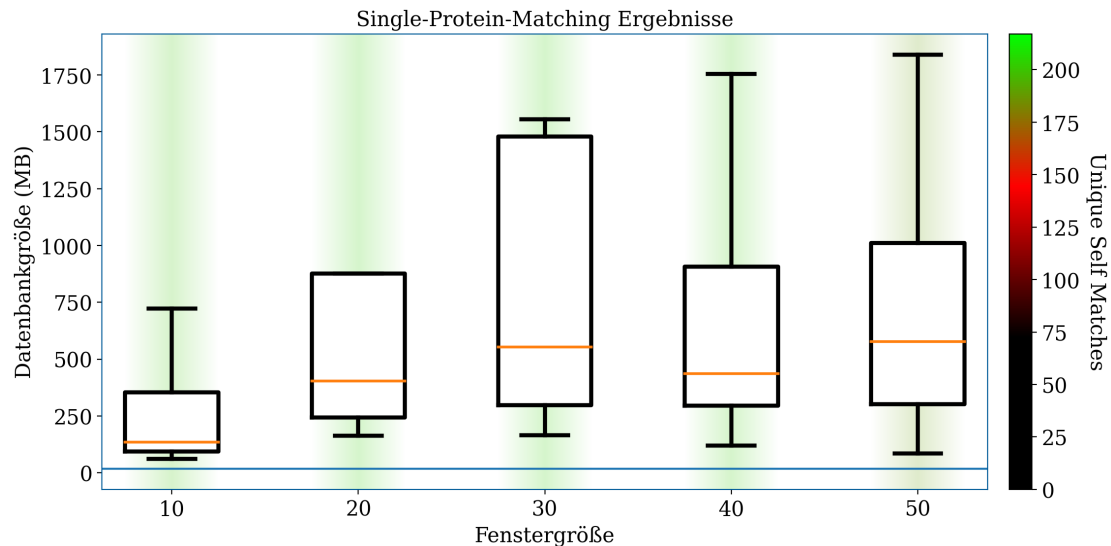


Abbildung 2: Bisherige Ergebnisse von prot-fin: Überblick der Ergebnisse für die Hydrophobizität mit Verwendung verschiedener Parameter, die die STFT betreffen (siehe Unterabschnitt 3.6 (Seite 18)). Die Farbe stellt dar, wie viele Eingabeproteine eindeutig identifiziert wurden, die blaue Linie die Größe der Eingabedaten. Die Boxen bilden die Verteilung der Datenbankgrößen über die verschiedenen Parameterkonfigurationen ab und sind entlang der x-Achse nach der Größe eines STFT-Fensters gruppiert. Der Family-Matching Ansatz hat keine Vorergebnisse, da er erst in dieser Arbeit entwickelt wurde. Die Originaldaten für diesen Plot sind in den Dateien `previous/*.csv` im Anhang (Seite 37).

Die Wahl der Frequenzen der STFT-Fenster und deren Abspeicherung müssen folglich noch verbessert werden, weshalb dazu in dieser Arbeit mehrere Experimente angegangen wurden.

1. **UniRef90-Sampling:** In vergangenen Experimenten hat sich gezeigt, dass die Selektion der Frequenzen mittels lokaler Maxima (Abbildung 1 (Seite 2)) suboptimal ist, weil es gelegentlich dazu kam, dass bei den kurzen Sequenzen in einem Fenster keine Maxima vorkamen. Es wurde auch versucht, für eine höhere Spezifität die Amplituden mit einzubeziehen, statt nur der Frequenzen selbst, was aber die Datenbankgröße stark erhöhte. Das UniRef90 Experiment soll beides vereinigen. Es sollen möglichst wenige aber sehr spezifische Frequenzen in einem Fenster ausgewählt werden, damit die Datenbank schlank bleibt. In diesem Experiment werden solche Fenster aus der UniRef90 Datenbank gesampelt. Sie enthält etwa 180 Mio. Aminosäuresequenzen, sodass aus jeder ein zufälliges Fenster gewählt wird. Die je Frequenz seltensten Amplituden sollen nun als Schwellwert ein Wahlkriterium für eine Frequenz sein, was zu möglicherweise weniger selektierten Frequenzen mit dennoch guter Signifikanz führt.
2. **Filter Hashes:** Ein weiterer Ansatz die Datenbankgröße zu verringern ist das Entfernen von Hashes, die sehr häufig auftreten, also folglich wenig Informationsgehalt [Sha48] für die Identifikation eines Proteins haben. In diesem Experiment wird daher geprüft, wie streng das erfolgen darf, um nicht zu viel Daten zu verlieren, sodass das Matching dadurch beeinträchtigt würde.
3. **Target-Zone:** Die effiziente Abspeicherung mittels Hashing basiert darauf, die Frequenzen eines Fensters mit den Frequenzen der Nachfolgefenster zu kombinieren. Die Target-Zone bezeichnet dabei die Anzahl betrachteter Nachfolger und ist aktuell unbeschränkt. Folglich wird hier eine Größe ermittelt, die einen guten Kompromiss zwischen der Anzahl an Kombinationen und der Genauigkeit des Matchings bildet.
4. **Selection-Method:** Das letzte durchgeführte Experiment dieser Arbeit dient der Ermittlung einer Methode für die Frequenzselektion, die vom UniRef90-Sampling profitiert, aber die Auswahl zusätzlich reduziert. Dafür wird vor Betrachtung der Schwellwerte eine Vorselektion durchgeführt, einmal anhand der lokalen Maxima der Amplituden, dann anhand der stärksten Abweichungen der Amplituden von ihren Schwellwerten und natürlich ohne Vorselektion als Nullprobe, die nur das Sampling einbezieht.

Als Trainingsdaten wird eine Referenz-Datenbank von Pflanzen-Proteinen verwendet, die in Gruppen derselben Funktion eingeordnet sind. Diese Zuordnung wurde manuell von Experten durchgeführt in sogenannte MapMan-Bins, was heißt:
Derselbe Bin \rightarrow dieselbe Funktion [Sch+19][LOH+14].

2 Material

2.1 Trainings-Proteine

Für die Entwicklung von prot-fin wurden 41669 MapMan-Referenz-Pflanzenproteine verwendet (`protein.fa` im Anhang (Seite 37)), die dem Trainieren von Bin-spezifischen Hidden Markov Models von MapMan4 dienen [Sch+19]. MapMan ist eine für Pflanzen entwickelte Methode, um deren Proteinsequenzen in diesen Bins nach Funktion zu klassifizieren, siehe `mapman.tsv` im Anhang (Seite 37). Dies erfolgt in einer Baumstruktur, wobei die Wurzelknoten sowie die Knoten je Ebene eines Baums von 1 aufsteigend nummeriert sind. Die Wurzelknoten sind grobe Beschreibungen der Funktion, wie zum Beispiel der Photosynthese. Je mehr der Baum in die Tiefe geht, desto präziser wird die Beschreibung, bis an den Blättern die Proteine mit ihrer Annotation stehen [Sch+19][LOH+14]. Ein Bin setzt sich aus der Konkatenation seiner Knoten mittels Punkt als Separator zusammen. So ist in `mapman.tsv` im Anhang (Seite 37) zum Beispiel der Bin 1.1.1, wobei die linke Ziffer für die Wurzel Photosynthese steht, die mittlere Ziffer für die Photophosphorylation und die dritte für das betreffende Photosystem 2. Aufgrund dieser Klassifizierung nach Funktion werden die Bins der Proteine in prot-fin als Familien betrachtet und gehen daher in die Bewertung der Identifikation von Funktionsgleichheit ein.

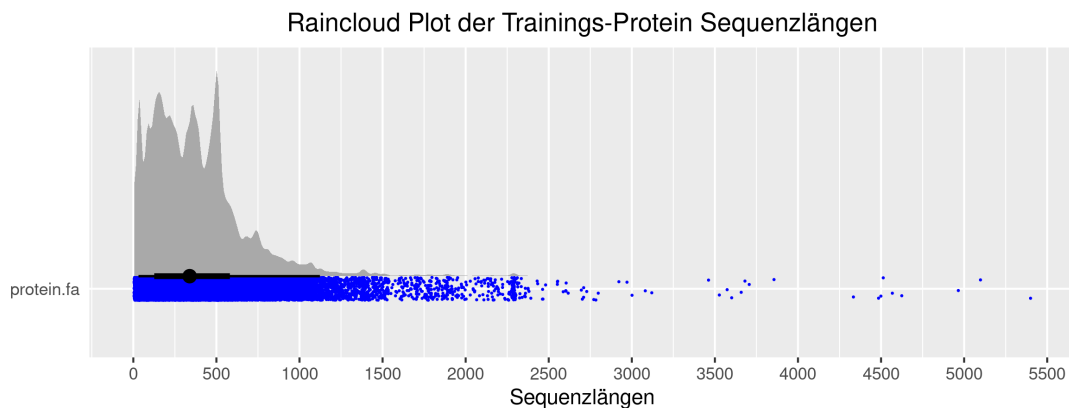


Abbildung 3: Die graue “Wolke” des Raincloud Plots stellt die Dichteverteilung der Sequenzlängen über die 41669 MapMan-Referenz-Proteine (Trainings-Proteine) dar, die blauen Punkte darunter, welche Werte tatsächlich angenommen wurden, und dazwischen ist ein Boxplot, der Aufschluss über den Median (schwarzer Kreis) und die anderen Quartile bietet. Die mediale Länge liegt bei etwa 300 Aminosäuren.

2.2 UniRef90

In der UniRef90-Datenbank sind Sequenzen aus der UniProtKB [Con20] mit 90% Sequenzübereinstimmung in sogenannten UniRef-Einträgen gruppiert, welche verschiedens-

te Informationen über die jeweilige Gruppe bieten. Zudem ist jedem Eintrag die Proteinsequenz zugeordnet, die die Gruppe am besten repräsentiert [Suz+14]. UniRef-Datenbanken werden regelmäßig erneuert. Für die Experimente in prot-fin mit der UniRef90-Datenbank wird der Release 2024_02 (27.03.2024) verwendet:

https://ftp.uniprot.org/pub/databases/uniprot/previous_releases/release-2024_02/

Die entpackte FASTA-Datei ist etwa 82 Gigabyte groß und enthält 187.136.236 Proteinsequenzen. Sie befindet sich zum Zeitpunkt der Abgabe dieser Arbeit auf dem Compute Cluster der Hochschule an folgendem Pfad:

`/media/BioNAS/UniProtKB/20240408/uniref90.fasta`

3 Methoden

Jeglicher Code, ob für die konkrete Implementierung der Algorithmen oder der durchgeführten Experimente, wurde versioniert und ist auf GitHub verfügbar. Die genauen Verweise, was wo hinterlegt ist und wie reproduziert werden kann, sind in `README.txt` im Anhang (Seite 37) zu finden.

3.1 Grundalgorithmus

Algorithmus 1 Übersetzen einer Aminosäuresequenz in einen numerischen Vektor

Eingabe sind eine Aminosäuresequenz und die Nummer eines Kidera-Faktors, beginnend mit 0. Dann wird die Sequenz zeichenweise in einen Vektor übersetzt, wobei erst der Wert des KF abgerufen wird und dieser dann auf einen positiven Wert normalisiert wird.

Vorbedingung $sequence \in \{A-Z, '\Psi', '\Omega', '\Phi', '\zeta', '\Pi', '+', '-'\}^*$

$0 \leq kf \leq 9$

Nachbedingung $|aa_vector| = |sequence|$ $\triangleright aa = \text{Aminosäure}$

$v \geq 0 \forall v \in aa_vector$

1: $aa_vector \leftarrow \text{array}()$

2: **for each** aa **in** $sequence$ **do**

3: $kf_value \leftarrow \text{get_kf_value}(aa, kf)$ $\triangleright kf = \text{Kidera Faktor}$

4: $min_kf_value \leftarrow \text{get_min_kf_value}()$

5: $aa_vector.append(kf_value + \text{abs}(min_kf_value))$

6: **end for**

Voraussetzung für den Algorithmus ist ein numerischer Vektor, so wie es die digitale Tonspur bei SHAZAM darstellt. Um dies im Kontext von Proteinen zu erreichen, wird in prot-fin auf sogenannte Kidera-Faktoren zurückgegriffen. Diese Faktoren stammen aus einem Forschungsprojekt von Kidera u. a., welches 1985 publiziert wurde [Kid+85]. Inhalt des Projekts war die statistische Faktorenanalyse von 188 physikalischen Eigenschaften der 20 natürlichen Aminosäuren zur Ermittlung von 10 dieser Eigenschaften, durch die

die anderen aufgrund hoher Korrelation erklärt werden können. Dadurch, dass die wenigen KF all die Eigenschaften repräsentieren, können diese effizient gleichzeitig analysiert werden. In Tabelle 1 (Seite 7) sind die KF dargestellt.

Tabelle 1: Kidera-Faktoren

Jeder Kidera-Faktor hat eine eigene Zeile mit seiner beschreibenden Bezeichnung in der linken Spalte, danach folgen die Werte je Aminosäure. Die vollständige Tabelle ist in `Amino_Acid_Kidera_Factors.csv` im Anhang (Seite 37) zu finden und stammt aus dem R-Paket “Peptides” [ORT15].

Beschreibung	A	C	D	E	F	G	...
Helix/bend preference	-1.56	0.12	0.58	-1.45	-0.21	1.46	...
Side-chain size	-1.67	-0.89	-0.22	0.19	0.98	-1.96	...
Extended structure preference	-0.97	0.45	-1.58	-1.61	-0.36	-0.23	...
Hydrophobicity	-0.27	-1.05	0.81	1.17	-1.43	-0.16	...
Double-bend preference	-0.93	-0.71	-0.92	-1.31	0.22	0.1	...
Partial specific volume	-0.78	2.41	0.15	0.4	-0.81	-0.11	...
Flat extended preference	-0.2	1.52	-1.52	0.04	0.67	1.32	...
Occurrence in alpha region	-0.08	-0.69	0.47	0.38	1.1	2.36	...
pK-C	0.21	1.13	0.76	-0.35	1.71	-1.66	...
Surrounding hydrophobicity	-0.48	1.1	0.7	-0.12	-0.44	0.46	...

Folglich kann eine Aminosäuresequenz pro KF in einen numerischen Vektor übersetzt werden, wobei ein höherer absoluter Wert für mehr Relevanz des KF steht.

Um für die Fourier Transformation nur positive Werte zu verwenden, wird der Vektor anschließend dahingehend normalisiert, dass das absolute Minimum aller Werte aus Tabelle 1 (Seite 7) aufaddiert wird. Das absolute Minimum ist 2.33, sodass die Übersetzung der Beispielsequenz **EVKEFDGQGCF** für die Hydrophobizität folgendermaßen geschehe:

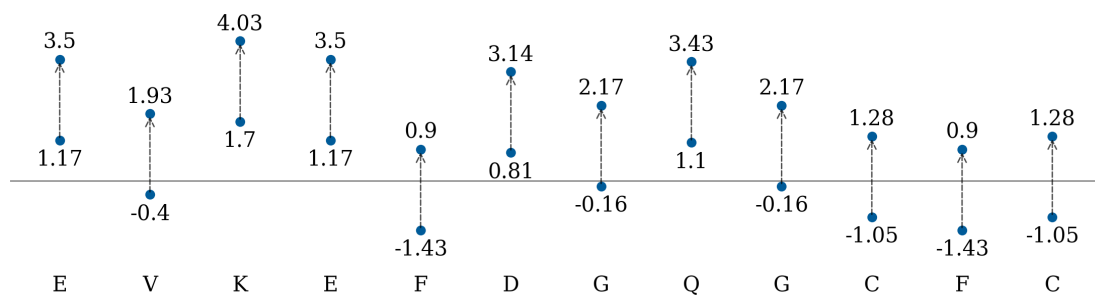


Abbildung 4: Beispielsequenz als numerische Repräsentation: Je Aminosäure auf der x-Achse entspricht der untere Punkt ihrem Wert für “Hydrophobicity” aus Tabelle 1 (Seite 7). Der Pfeil symbolisiert die Normalisierung dieses Wertes durch das Inkrement von 2.33. Die horizontale Linie ist die Nulllinie.

Algorithmus 2 Sammeln von Strukturdaten

Eingabe ist der Vektor aus Algorithmus 1 (Seite 6). Dieser wird mittels STFT fensterweise transformiert. Anschließend werden je Fenster die Frequenzen ausgewählt, deren Amplituden ein lokales Maximum bilden, und zur Constellation-Map hinzugefügt.

Vorbedingung *aa_vector* aus Algorithmus 1 (Seite 6) $\triangleright aa = \textit{Aminosäure}$

Nachbedingung *constellation_map* is an Array of Arrays of Floats

```
1: constellation_map  $\leftarrow$  array()
2: stft_result  $\leftarrow$  stft_transform(aa_vector)
3: for each window in stft_result do
4:   selected_frequencies  $\leftarrow$  select_maxima(window)
5:   constellation_map.append(selected_frequencies)
6: end for
```

Der erhaltene Vektor aus Algorithmus 1 (Seite 6) wird strukturell analysiert. Das dafür genutzte Vorgehen basiert auf der STFT, welche den Vektor fensterweise auf periodische Signale untersucht, wie z.B. dem wiederholten Auftreten von hydrophoben Aminosäuren im gleichen Abstand. Für die STFT in prot-fin Version 0.4 wird die Methode **stft** aus dem Untermodul **signal** des Python-Moduls **scipy** [Vir+20] Version 1.11.4 verwendet. Je Fenster werden die Frequenzen der auffälligsten Signale ausgewählt, wie in Abbildung 1 (Seite 2) dargestellt mittels der lokalen Maxima, sodass über alle Intervalle eine sogenannte Constellation-Map entsteht, welche in Abbildung 5 (Seite 10) im linken Teil dargestellt ist. Diese Map wird dabei intern als Array repräsentiert, wobei jedes Element hierbei eine Liste darstellt, die ihrem Index entsprechend (x-Achse) die gewählten Frequenzen des jeweiligen Fensters beinhaltet (y-Achse).

Algorithmus 3 Hashing

Eingabe ist eine Constellation-Map aus Algorithmus 2 (Seite 8), die ID des betreffenden Proteins und die Nummer des Kidera-Faktors, beginnend bei 0. Je Fenster wird jede Frequenz A paarweise mit den Frequenzen aus allen Nachfolgefenstern kombiniert. Die Frequenz A, die jeweilige Nachfolgefrequenz, der Abstand zur Nachfolgefrequenz und die Nummer des Kidera-Faktors bilden den Hash. Der Abstand darf 12 Bit, also 2^{12} nicht überschreiten, siehe Abbildung 6 (Seite 10). Je Hash wird die Position des Fensters von Frequenz A in der Constellation-Map gespeichert und zusätzlich die Protein-ID, um die Verbindung von Hash und Protein zu erhalten.

Vorbedingung *constellation_map* aus Algorithmus 2 (Seite 8)

protein_id is a String

$0 \leq kf \leq 10$

Nachbedingung *hashes* is a HashMap of: $Int \rightarrow Int, String$

```
1: hashes  $\leftarrow$  hashmap()
2: window_idx  $\leftarrow$  0
3: repeat
4:   selected_frequencies  $\leftarrow$  constellation_map.pop(0)
5:   for each frequency in selected_frequencies do
6:     successor_count  $\leftarrow$  min( $2^{12}$ , length(constellation_map)) - 1
7:     for successor_idx  $\leftarrow$  0 to successor_count do
8:       succ_frequencies  $\leftarrow$  constellation_map[successor_idx]
9:       for each succ_frequency in succ_frequencies do
10:        hash  $\leftarrow$  create_hash(frequency, succ_frequency, successor_idx, kf)
11:        hashes[hash]  $\leftarrow$  (window_idx, protein_id)
12:      end for
13:    end for
14:  end for
15:  window_idx  $\leftarrow$  window_idx + 1
16: until constellation_map is empty
```

Die erhaltene Constellation-Map (CM) wird elementweise gehashed, um einen effizienten Vergleich mit anderen CMs zu ermöglichen. Um das zu erzielen, wird jede ausgewählte Frequenz eines Fensters mit jeder weiteren Frequenz der Nachfolgefenster gepaart. Bildlich gesprochen werden also Kanten gebildet, wodurch die CM zu einem Graphen wird. Jede dieser Kanten bildet einen Hash, also einer Kombination aus den beiden Frequenzen/Kantenenden und der Kantenlänge (hier *successor_idx*, beginnend mit 0). In einer HashMap (für prot-fin Version 0.4 in Python ein Dictionary) wird sich folgend für den Hash die Position der Kante in der CM gemerkt, sowie die ID des Proteins, für die diese CM erstellt wurde. Sollte ein Hash dabei mehrfach vorkommen, verbleibt lediglich seine letzte Position. Dieses Verfahren wird im rechten Teil von Abbildung 5 (Seite 10) repräsentativ dargestellt, wobei rote Linien die ignorierten Kanten abbilden.

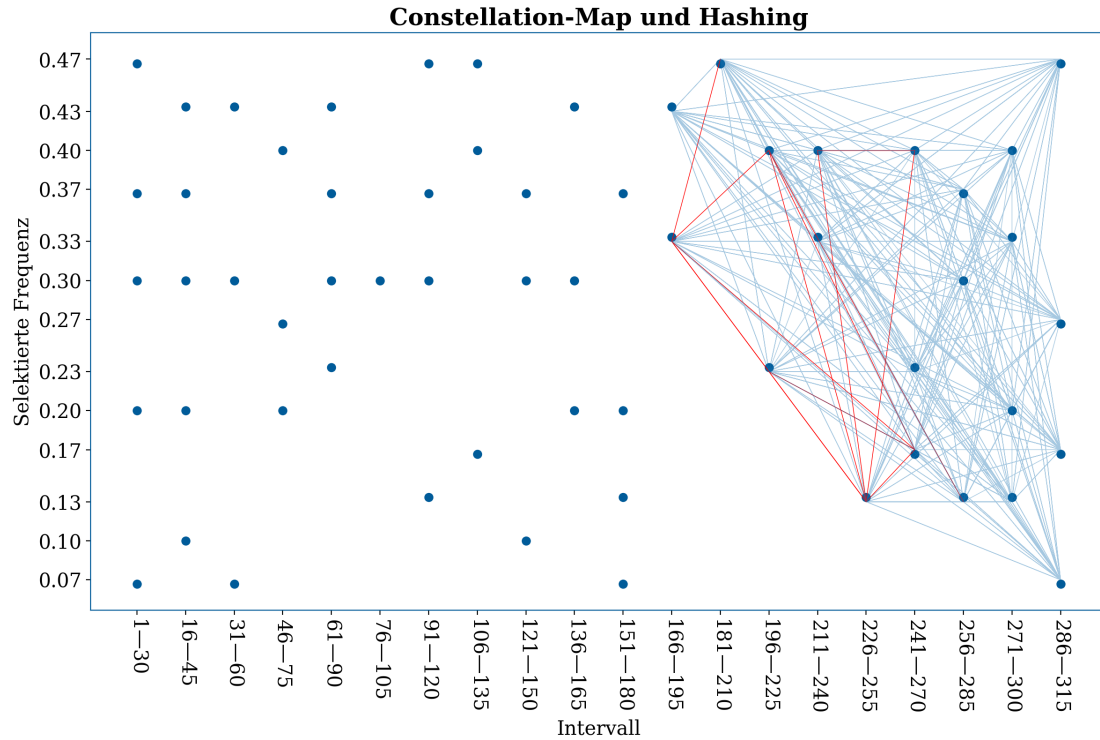


Abbildung 5: Die Punkte bilden die Constellation-Map. Auf der y-Achse ist die Frequenz und auf der x-Achse das Intervall des STFT-Fensters. Die zur Übersicht nur rechts eingezeichneten Kanten repräsentieren die Hash-Bildung aus dieser Constellation-Map, wobei rote Kanten ignorierte Hashes darstellen, weil sie in einem Nachfolgefenster erneut auftauchen und nur das letzte Vorkommen gespeichert wird.

Die Hashfunktion ist für garantierte Individualität bijektiv gestaltet, indem seine Bestandteile nach folgendem Schema auf Bit-Ebene hintereinandergereiht werden:

6 Bit	4 Bit	12 Bit	5 Bit	5 Bit
<i>unbelegt</i>	Kidera-Faktor	Kantenlänge/Fensterabstand	Frequenz Nachfolger	Frequenz Ursprung

Abbildung 6: Schematischer Aufbau eines Hashes: 4 Bit werden für die Nummer eines Kidera-Faktors belegt. Die Fenstergrößen belaufen sich auf unter 64, sodass mit 5 Bit alle möglichen Frequenzen abgedeckt werden können. Da die Anzahl Frequenzen immer gleich ist, können diese aufsteigend durchnummeriert werden, sodass die x-Achse in Abbildung 1 (Seite 2) der Folge von 0 bis 15 entspräche. 12 Bit enthalten die "Kantenlänge" eines Hashes. Die restlichen 6 Bits zu einem 32-Bit Integer können in der weiteren Entwicklung des Algorithmus beliebig belegt werden.

Algorithmus 4 Erstellung der Datenbank

Eingabe ist eine Datei im FASTA-Format. Aus jeder darin befindlichen Sequenz wird nach Algorithmus 1 (Seite 6) ein numerischer Vektor erstellt, dieser dann mittels Algorithmus 2 (Seite 8) in eine Constellation-Map übersetzt und diese anschließend mit dem Algorithmus 3 (Seite 9) entsprechend gehashed. Je Hash wird das Tupel aus Position, an der der Hash in der Constellation-Map auftauchte, und Protein-ID in eine Liste in der Datenbank eingefügt, auf die der Hash verweist.

Vorbedingung *fasta* is a FASTA-formatted file

Nachbedingung *database* is a HashMap of: $Int \rightarrow Array\ of\ (Int, String)$

```
1: database  $\leftarrow$  hashmap()
2: for each protein_id, sequence in fasta do
3:   for kf  $\leftarrow$  0 to 10 do                                      $\triangleright kf = \text{Kidera Faktor}$ 
4:     aa_vector  $\leftarrow$  get_aa_vector(sequence, kf)                  $\triangleright aa = \text{Aminosäure}$ 
5:     constellation_map  $\leftarrow$  get_constellation_map(aa_vector)
6:     hashes  $\leftarrow$  get_hashes(constellation_map, protein_id)
7:     for each hash in hashes do
8:       if hash not in database then
9:         database[hash]  $\leftarrow$  array()
10:      end if
11:      database[hash].append(hashes[hash])
12:    end for
13:  end for
14: end for
15: save_to_file(database)
```

Die beschriebenen Algorithmen 1 bis 3 beschreiben den Weg von einer Aminosäuresequenz in Textform zu den Hashes, die die strukturelle Information des Proteins entsprechend der spektralen Zerlegung mittels STFT repräsentieren sollen. Übrig bleibt nur der Schritt, der die Hashes einer Menge von mehreren Proteinen in einer Datenbank vereinigt, sodass im Anschluss die Identifizierung von Eingabesequenzen erfolgen kann. Dafür werden je Trainings-Protein (TP) für alle Kidera-Faktoren die Hashes gebildet und in die Datenbank geschrieben, welche eine HashMap ist. Im Gegensatz zu der resultierenden HashMap in Algorithmus 3 (Seite 9) verweisen die Hashes in der Datenbank allerdings nicht auf eine Position des Hashes für ein Protein, sondern auf eine Liste von solchen. Das heißt, dass für die Datenbank ein neuer Hash mit einem leeren Array initialisiert wird, in das darauf all diese Position-Protein-ID-Tupel eingefügt werden.

Version 0.4 von `prot-fin` ist in Python implementiert. Von daher wird für die Persistierung (`save_to_file`) der Datenbank, einem `Dictionary`, zur Einfachheit das `pickle`-Modul verwendet, welches mit der genutzten Python-Version 3.10.12 mitgeliefert wird und Python-Objekte effizient in Dateien ablegen kann.

Wurde die Datenbank erstellt, ist sie für die Identifizierung funktionsähnlicher Proteine anhand einer Eingabe verwendbar. Hierfür gibt es zwei Ansätze:

a) **Single-Protein-Matching:** Eingabe ist eine FASTA-Datei, also eine Menge an Suchsequenzen. Ausgabe je Sequenz ist eine Liste von Treffern, sortiert nach Übereinstimmung der Hashes der CMs von Treffer und Suchsequenz. Je höher der Rang eines Treffers, desto funktionsähnlicher sollte das entsprechende Protein sein. Treffer mit dem besten Score erhalten Rang 1, mit dem zweitbesten Score Rang 2, ... und der Treffer mit dem schlechtesten Score den letzten Rang. Die Ausgabe in prot-fin Version 0.4 sind Comma Separated Values (CSV), also eine durch Kommata separierte Tabelle, mit folgenden Spalteninhalten:

1. Rank \rightarrow Rang
2. Match_Protein_ID \rightarrow Protein-ID des Treffers
3. JSI \rightarrow Jaccard Similarity Index (siehe Algorithmus 5 (Seite 13))
4. Score \rightarrow Score (Übereinstimmung der Constellation-Map)
5. Input_Protein_ID \rightarrow Protein-ID der Suchsequenz
6. Input_Sequence_Length \rightarrow Sequenzlänge der Suchsequenz
7. Input_Found_Hashes \rightarrow Anzahl Hashes der Suchsequenz

Algorithmus 5 Treffer-Bewertung beim Single-Protein-Matching

Eingabe sind die Hashes für die Suchsequenz nach Algorithmus 3 (Seite 9) und die trainierte Datenbank aus Algorithmus 4 (Seite 11). Für jeden Hash der Suchsequenz S wird für jedes Protein P in der Datenbank mit diesem Hash der Abstand/Offset seiner Position in der Constellation-Map (CM) von der Position von P in dessen CM gebildet. Für P wird gezählt, wie oft ein jeder Offset vorkommt. Der Score von P bildet sich durch die Anzahl des häufigsten Offsets, welche gleichbedeutend mit der Anzahl in Position übereinstimmender Hashes in beiden CMs ist, und dem nachfolgend in Gleichung 1 (Seite 14) beschriebenen Jaccard Similarity Index.

Vorbedingung *hashes* aus Algorithmus 3 (Seite 9)

database aus Algorithmus 4 (Seite 11)

Nachbedingung *match_scores* is a HashMap of: *String* \rightarrow *Float*

```
1: matches_per_tp  $\leftarrow$  hashmap()  $\triangleright tp = TrainingsProtein$ 
2: for each hash, position in hashes do
3:   if hash in database then
4:     for each tp_position, protein_id in database[hash] do
5:       if protein_id not in matches_per_tp then
6:         matches_per_tp[protein_id]  $\leftarrow$  hashmap()
7:       end if
8:       offset  $\leftarrow tp\_position - position$ 
9:       if offset not in matches_per_tp[protein_id] then
10:        matches_per_tp[protein_id][offset]  $\leftarrow$  0
11:      end if
12:      offset_count  $\leftarrow$  matches_per_tp[protein_id][offset]
13:      matches_per_tp[protein_id][offset]  $\leftarrow offset\_count + 1$ 
14:    end for
15:  end if
16: end for
17: match_scores  $\leftarrow$  hashmap()
18: for each match_protein, offsets in matches_per_tp do
19:   score  $\leftarrow$  get_most_common_offset(offsets)
20:   match_protein_hashes  $\leftarrow$  database.get_hashes(match_protein)
21:   jsi  $\leftarrow$  get_jsi(hashes, match_protein_hashes)
22:   match_scores[match_protein]  $\leftarrow score \cdot jsi$ 
23: end for
```

Um den Score zu bestimmen, also die Ähnlichkeit der CM der Eingabe mit denen der TP, werden pro Eingabe-Hash die Differenzen zwischen dessen Position mit den Positionen der trainierten Hashes gebildet und global pro Protein gezählt. Diese Differenzen repräsentieren den Abstand/Offset des Hashes/Kante in der Eingabe-CM zur Kante der jeweiligen TP-CM, also wie weit die Eingabe-CM verschoben wäre, sollte es sich bei dem TP um das Original handeln. Auf diese Weise sammeln sich

pro TP mehrere solcher potentiellen Abstände, wobei der Abstand, der am häufigsten aufgetreten ist, offensichtlich die meiste Übereinstimmung in den Kanten zeigt. Diese Tatsache qualifiziert diese Maximalanzahl als geeigneten Score (S1) für ein Match.

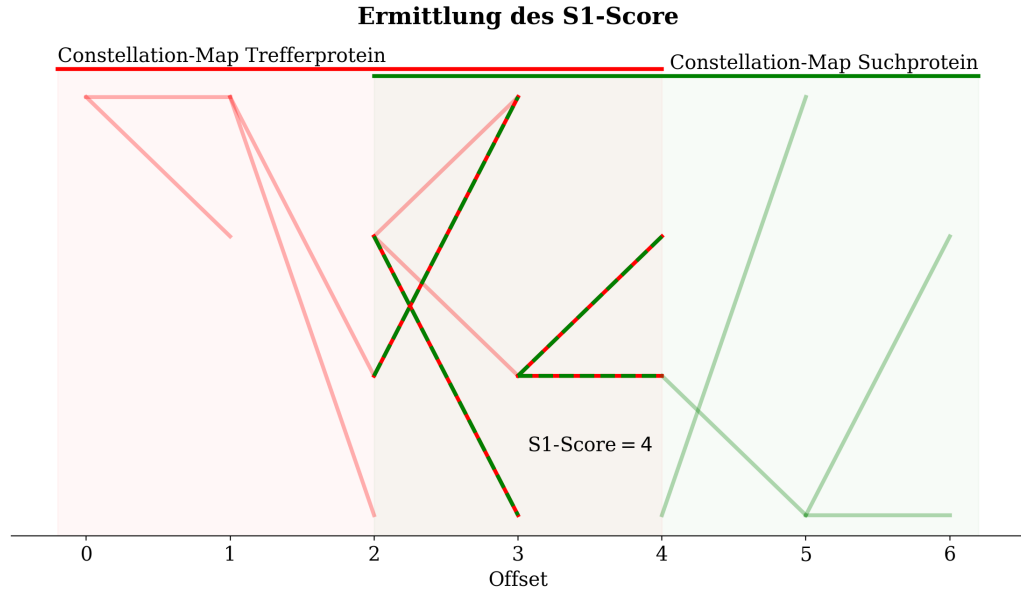


Abbildung 7: Die Constellation-Maps werden aneinander verschoben. Die maximale Überschneidung bei Offset 2 ist der Score. Hier stimmen 4 Hashes in Position überein, sie liegen genau übereinander und sind somit gestrichelt dargestellt.

Da große Proteine mit sehr langen Aminosäuresequenzen kürzere Sequenzen kleinerer funktionsungleicher Proteine enthalten können, reicht der ermittelte Score alleine nicht aus, da in diesem Fall sehr viele Kanten der Eingabe-CM übereinstimmen würden, sodass trotz Mis-Match der nahezu maximale Score erreicht werden würde. Bezogen auf das Beispiel zu S1 in Abbildung 7 (Seite 14), wäre dort der grüne Bereich vollständig vom roten Bereich eingeschlossen mit Überschneidungen in beinahe allen Kanten.

Um das zu umgehen, wird der Jaccard Similarity Index (JSI) verwendet, einem Maß, das die Übereinstimmung zweier Mengen A und B wie folgt bewertet:

$$JSI(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

Dieser Index nimmt einen Wert von 0 an, wenn beide Mengen disjunkt sind, und nähert sich der 1 je größer die Schnittmenge ist. Im Fall des Vergleichs zweier Constellation-Maps, also zwei Hash-Mengen, wird hier bewertet, wie viele Kanten sich die beiden Maps positionsunabhängig teilen. Durch diese Unabhängigkeit reicht der JSI alleine nicht als Score aus, sodass nur in Kombination/Multiplikation mit dem S1 ein robuster Score entsteht, da beide zusammen ihre Schwächen aufheben. Der JSI in Abbildung 7

(Seite 14) beträgt $\frac{4}{14} \approx 0.286$, da die Schnittmenge beider Hashmengen hier gleichzeitig den S1-Score bilden und die restlichen Hashes disjunkt zueinander sind. Der S1 wäre nur noch 3, wenn eine der markierten Kanten an einer anderen Position wäre, wobei der JSI davon unberührt bliebe.

- b) **Family-Matching:** Eingabe ist eine CSV-Tabelle mit der Zuordnung von Protein-ID und -familie (mittels Familien-ID). Die für das Matching verwendeten Hashes sind hier diejenigen, die sich alle Mitglieder einer Suchfamilie teilen. Die Idee dahinter ist, dass diese Hashes spezifisch für diese Familie, beziehungsweise deren Funktion ist. Als Bewertungsmaß wird dabei die Anzahl Hashes des Treffers verwendet, die mit den Familienhashes übereinstimmen, anstatt des S1 und JSI aus dem Single-Protein-Matching. Der Vorteil dieses Ansatzes ist, dass durch die Reduktion der betrachteten Hashes die Laufzeit im Vergleich zum Single-Protein-Matching verringert wird. Denn für jeden Hash muss in der Datenbank durch alle Proteine iteriert werden, die ebenfalls diesen Hash haben. Je weniger Hashes also betrachtet werden, desto weniger Iterationen gibt es. Da die geteilten Hashes einer Familie in der Anzahl nur kleiner oder gleich der Anzahl Hashes des Familienmitglieds mit den wenigsten Hashes sein können, ist dieser Ansatz somit mindestens so schnell wie für das Single-Protein-Matching mit diesem kleinsten Mitglied.

Für die Ausgabe werden diese Treffer zusammengefasst. Dazu gibt es zwei Kriterien, den F-Score und die Schärfe (Sharpness), die nach folgender Berechnung ermittelt werden:

$$\begin{aligned}
 t &= \text{max_score}(TrP) \\
 \text{Sharpness} &= \begin{cases} \frac{t - \text{max_score}(FP)}{t}, & \text{if } t > 0 \\ -1, & \text{sonst} \end{cases} \\
 \text{Precision} &= \frac{|TrP|}{|TrP| + |FP|} \\
 \text{Recall} &= \frac{|TrP|}{|TrP| - \text{member_count}} \\
 F_Score &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
 \end{aligned} \tag{2}$$

TrP ist hierbei die Menge der Treffer, die tatsächlich in der Familie vorkommen (true positives), wobei FP diejenigen sind, die das nicht tun (false positives) und einen besseren Score als der letzte korrekte Treffer haben. “member_count” ist die Anzahl an Mitgliedern der Familie.

Die Schärfe stellt das Verhältnis der besten Scores von FP und TrP dar, also wie weit der beste korrekte Treffer im Score von dem besten falschen Treffer entfernt ist.

Die Präzision gibt an, wie viele der Treffer korrekt gewesen sind, während der Recall zeigt, wie viele der Familienmitglieder gefunden wurden. Der F-Score bringt diese beiden Werte zusammen.

Die Ausgabe von prot-fin Version 0.4 als CSV-Tabelle beinhaltet aktuell diese Zusam-

menfassung zu Entwicklungszwecken. Sollte prot-fin ausgereift sein, wird die Ausgabe vermutlich die Treffer je Suchfamilie enthalten. In den Spalten wird aktuell folgendes dokumentiert:

1. Family_ID → ID der Suchfamilie
2. F_Score
3. Precision
4. Recall
5. Sharpness
6. Member_Count → Anzahl der Mitglieder der Suchfamilie
7. Match_Count → Anzahl gefundener Treffer
8. Hash_Intersec_Size → Anzahl der Hashes, die sich die Mitglieder teilen

3.2 Experiment 1: UniRef90 Sampling

Ein Bestandteil der Strukturanalyse in Algorithmus 2 (Seite 8) ist die Selektion signifikanter Frequenzen zur Erstellung der CM. Es wäre zwar möglich, alle Frequenzen auszuwählen und dafür die Signalstärke in den Hash einfließen zu lassen, jedoch führe diese Vorgehensweise zu wesentlich mehr Hashes und einer folglich sehr großen Datenbank, was wiederum das Scoring/Matching verlangsamt. Ein Anspruch an prot-fin ist, dass die Datenbankgröße die Eingabegröße nicht wesentlich übersteigt, wobei es sich bei der Eingabe um eine einfache FASTA-Datei handelt.

Wenn eine Datenbank nicht viel größer als die Eingabe sein soll, dürfen die Daten einer Sequenz nicht den Speicherbedarf der Textrepräsentation ihrer Aminosäuren überschreiten. Diese Zeichen sind Teil des “American Standard Code for Information Interchange” (ASCII) und benötigen daher nur 1 Byte Speicher, was bedeutet, dass bei einer Sequenz der TP mit einer medialen Länge von etwa 300 Aminosäuren nur ebensoviele Bytes verwendet werden dürften. Hierzu eine Rechnung unter der Annahme, es gäbe nur eine Frequenz pro Fenster:

$$\begin{aligned}
 \text{fenster_größe} &= 30, \text{ überlappung} = 15, \text{ sequenz_länge} = 300 \\
 \text{anzahl_fenster} &= \frac{\text{sequenz_länge}}{\text{fenster_größe} - \text{überlappung}} = 20 \\
 \text{target_zone} &= 8, \text{ hash_größe} = 32 \text{ Bit} = 4 \text{ Byte} \\
 \text{speicher_pro_kf} &= \text{target_zone} \cdot \text{anzahl_fenster} \cdot \text{hash_größe} = 640 \text{ Byte} \\
 \text{anzahl_kf} &= 10 \\
 \text{datenbank_speicher} &= \text{speicher_pro_kf} \cdot \text{anzahl_kf} = 6400 \text{ Byte}
 \end{aligned} \tag{3}$$

Selbst bei nur einer Frequenz pro Fenster wäre die Datenbank nach dieser Abschätzung doppelt so groß wie die Eingabe. Diesem Problem soll durch ein Sampling-Experiment abgeholfen werden. Darin wurden aus etwa 180 Millionen Sequenzen der UniRef90 Datenbank je ein zufälliges Fenster fester Größe für die STFT ausgewählt, transformiert

und die Signalstärken je Frequenz gemerkt. Um daraus eine Selektionsmethode abzuleiten, wurden die Grenzquantile einer Frequenz ermittelt, um signifikant seltene Signalstärken zu ermitteln. Folglich war es möglich, für die CM nur diejenigen Frequenzen zu behalten, welche in den Randbereichen der Signalstärken liegen, sodass nicht nur Signale infrage kommen, die für eine besonders starke Ausprägung eines Kidera-Faktors sprechen, sondern auch für den Fall der umgekehrten Ausprägung, wie z.B. Hydrophilie statt Hydrophobie.

Algorithmus 2 (Seite 8) wurde daher anschließend insofern angepasst, dass bei der Frequenz-Selektion in Zeile 4 (Seite 8) die gewählten Frequenzen so ermittelt werden, dass je Grenzwert die Ausreißer ausgewählt werden, also einmal für den oberen Wert und dann für den unteren Wert, und aus diesen Ausreißermengen die Maxima selektiert werden. Zudem wird einem Hash je Frequenz noch mit einem Bit die Information hinzugefügt, ob die Amplitude besonders hoch oder niedrig ist, also aus dem oberen oder unteren Randbereich kommt.

Die zu klärende Frage war, welche Grenzquantile verwendet werden müssen, um eine möglichst gute Frequenzauswahl zu erzielen, für ein gutes Matching mit kleineren Datenbanken.

3.3 Experiment 2: Filter Hashes

Die Experimente zu vorigen Versionen von prot-fin haben zu sehr großen Datenbanken und entsprechend langsamem Matching geführt (siehe Abbildung 2 (Seite 3)). Mit dem Ziel, die Datenbank auf Eingabegröße zu reduzieren, sollte darin je Protein nur das Notwendigste gespeichert werden. In Exp. 1 (Seite 16) wurde dafür die Frequenzwahl angegangen, indem die Grenzwerte mit der Wahl der Maxima kombiniert wurden. Alternativ soll hier die Größe durch einen Filter ermöglicht werden, wobei die Grenzwerte als alleiniges Selektionskriterium dienen. Zum Filtern wurden dazu nach der Datenbankerstellung nur die Einträge der nach Quantil seltensten Hashes behalten. Die entfernten Hashes an sich wurden zudem in einer Blacklist gespeichert, um sie vor dem Matching auch aus den Hashes der Suchsequenzen zu entfernen, da sie ansonsten den JSI verfälschen würden, weil entfernte Hashes nicht in der Schnittmenge auftauchen könnten.

Auch in diesem Experiment musste herausgefunden werden, welches Quantil sich am besten eignet. Je kleiner es ist, desto kleiner wird auch die Datenbank, aber umso ungenauer auch das Matching. Es galt einen guten Kompromiss zu finden.

3.4 Experiment 3: Target-Zone

Die Target-Zone (TZ) beschreibt in Abbildung 5 (Seite 10) die maximal mögliche Kantenlänge eines Hashes, also die Anzahl Nachfolgefenster, die für das Hashing herangezogen werden. Je größer die TZ, desto näher kommt die Constellation-Map einem vollständigen Graphen, was die Datenbankgröße entscheidend beeinflusst. Da die TZ mit 12 Bit (Abbildung 6 (Seite 10)), also 4096, für Aminosäuresequenzen der TP nahezu unbeschränkt

ist (siehe Abbildung 3 (Seite 5)), wurde in diesem Experiment geprüft, wie wenige Bits ausreichen, um noch ein gutes Matching zu gewährleisten. Umgesetzt wurde dies in Zeile 6 von Algorithmus 3, wo über die Nachfolger iteriert wurde. Um die TZ einzubeziehen, wurden die hard-coded 2^{12} mit einer zusätzlichen Variable zu $2^{\text{targetZoneBits}}$ erweitert. Als Selektionsmethode wurde der Ansatz aus Exp. 1 (Seite 16) verwendet.

3.5 Experiment 4: Selection-Method

Dieses letzte Experiment knüpft an Exp. 1 (Seite 16) an. Dort war die Intention, die ermittelten Grenzwerte mit einer anschließenden Maxima-Wahl zu kombinieren. Alternativ wurden nun folgende Ansätze versucht:

1. Es werden alle Frequenzen ausgewählt, deren Amplituden den oberen/unteren Grenzwert aus Exp. 1 (Seite 16) über-/unterschreiten. Es bilden die Grenzwerte allein die Selektionsbedingung.
2. Es werden erst Frequenzen über Maxima ausgewählt und anschließend aus diesen nur diejenigen behalten, die den oberen Grenzwert überschreiten. Gleiches wird für Minima mit dem unteren Grenzwert wiederholt.
3. Die Amplituden der Frequenzen werden in ihren Abstand zum oberen Grenzwert umgerechnet. Für jede Frequenz wurde in Exp. 1 (Seite 16) zudem die Standardabweichung der gesampelten Amplituden ermittelt. Die Abstände werden durch die Division dadurch normalisiert. In diesen normalisierten Abständen werden die Frequenzen nach lokalen Maxima ausgewählt und nur die behalten, die den oberen Grenzwert überschreiten. Das ganze Verfahren wird für den unteren Grenzwert ebenfalls mit lokalen Maxima wiederholt.

Diese drei Methoden wurden zusätzlich um einen Parameter k erweitert, welcher bestimmt, dass die ersten k Frequenzen im Nachhinein aus der Auswahl entfernt werden (entsprechend ihrer Reihenfolge wie im Beispiel in Abbildung 1 (Seite 2), also $k = 2$ entfernt Frequenzen 0 und 1). Das hatte den Hintergrund, dass in vorigen Experimenten die niedrigen Frequenzen öfter ausgewählt wurden.

Die vorigen Ergebnisse hatten zudem veranlasst, das Signifikanzniveau der gesampelten Amplituden von 5% zu verringern, sodass zusätzlich die Grenzwerte für ein α von 0.1%, 0.01% und 0.001% ermittelt wurden.

Um die Ansätze effizient zu testen, wurde die Datenbankerstellung für dieses Experiment in Algorithmus 4 (Seite 11) so angepasst, dass sie abgebrochen wird, sobald der Speicherbedarf das 6-fache der Eingabedaten übersteigt.

3.6 Durchführung

Die Durchführung der Experimente läuft nach einem immergleichen Schema ab. Zuerst werden mit den MapMan-Referenz-Proteinen aus Unterabschnitt 2.1 (Seite 5) die Datenbanken erstellt und anschließend mit einer Teilmenge dieser Proteine das Single-Protein-

Matching durchgeführt. Diese Teilmenge wird über die MapMan-Bins ausgewählt. Um die Suchproteine möglichst funktional divers zu halten, werden je Wurzelknoten der Bins sieben zufällige Sequenzen ausgewählt. Für das Family-Matching werden alle die MapMan-Bins mit ihren Proteinen einbezogen, die durch mehr als einem Protein vertreten werden. Im Abschnitt **Experiment-Code** in **README.txt** im Anhang (Seite 37) wird auf die Verfügbarkeit des Codes der Experimente über ihre spezifischen **README**-Dateien verwiesen. Für die Implementierung der Auswahl der beschriebenen Teilmenge wird darin der Subcommand **select-samples** von Skript **evaluation.py** in Funktion und Nutzung beschrieben.

Neben den experimentspezifischen Parametern, wie Quantilen oder der Größe der Target-Zone, werden zusätzlich jedes Mal Parameter getestet, die die STFT und somit alle Experimente betreffen, da sich die Werte dazu aufgrund der zu digitaler Musik vergleichsweise kurzen Sequenzen nicht direkt von SHAZAM ableiten lassen. Hierbei geht es um die Fenstergröße (FG), die Länge der Überlappung zwischen benachbarten Fenstern und n_peaks , der Anzahl Frequenzen, die von den ausgewählten signifikanten Frequenzen verwendet werden. Sind also $n + x$ Frequenzen in der Auswahl, werden die x am wenigsten signifikanten Frequenzen entfernt, im Gegensatz zu k aus Exp. 4 (Seite 18), welches unabhängig der Anzahl immer dieselben Frequenzen entfernt. Alle Parameter mit ihren getesteten Werten sind in Tabelle 2 (Seite 19) aufgelistet:

Tabelle 2: Experiment-Parameter

Bei dem Parameter Quantil handelt es sich um das Quantil der Hashes, das beim Filtern der Datenbank behalten wird (siehe Exp. 2 (Seite 17)). Die Signifikanz bezeichnet das Signifikanzniveau α , welches in Exp. 1 (Seite 16) verwendet wird, um aus den Fenster-Samples die Grenzwerte für auffällige Amplituden einer Frequenz zu bestimmen. Für das Matching wird bei Exp. 1 (Seite 16) lediglich $\alpha = 5\%$ betrachtet.

Parameter	Werte					Experimente
Fenstergröße (FG)	10	20	30	40	50	1, 3
		20	30	40	50	2
			30	40	50	4
n_peaks	alle	3	5			1, 3, 4
		3	5			2
Overlap	0	25%	50%	75%	FG - 1	1, 2, 3
		25%	50%	75%		4
Target-Zone	8	16	32	64		3
	8					4
Quantil	0.1	0.2	...	0.9	1	2
Signifikanz	5%	10%	1%	0.1%		1
			1%	0.1%		4
	5%					2, 3

4 Ergebnisse

Im Folgenden werden die Ergebnisse der vier ab Seite 16 beschriebenen Experimente dargestellt. Der Hauptfokus liegt dabei auf der Datenbankgröße und Erkennungsperformanz. Ein Abschnitt zur Reproduzierbarkeit ist in `README.txt` im Anhang (Seite 37) vermerkt.

4.1 UniRef90 Sampling

Beim UniRef90 Sampling wurden die Amplituden-Grenzwerte für vier verschiedene Signifikanzniveaus ermittelt, für 5%, 0.1%, 0.01% und 0.001%, um die dadurch die Frequenzen mit den als Ausreißer definierten Amplituden als signifikant zu identifizieren. In Abbildung 8 (Seite 20) ist die Auswahlhäufigkeit der Frequenzen in der bisherigen Implementierung von prot-fin dargestellt, bevor die ermittelten Grenzwerte einbezogen wurden.

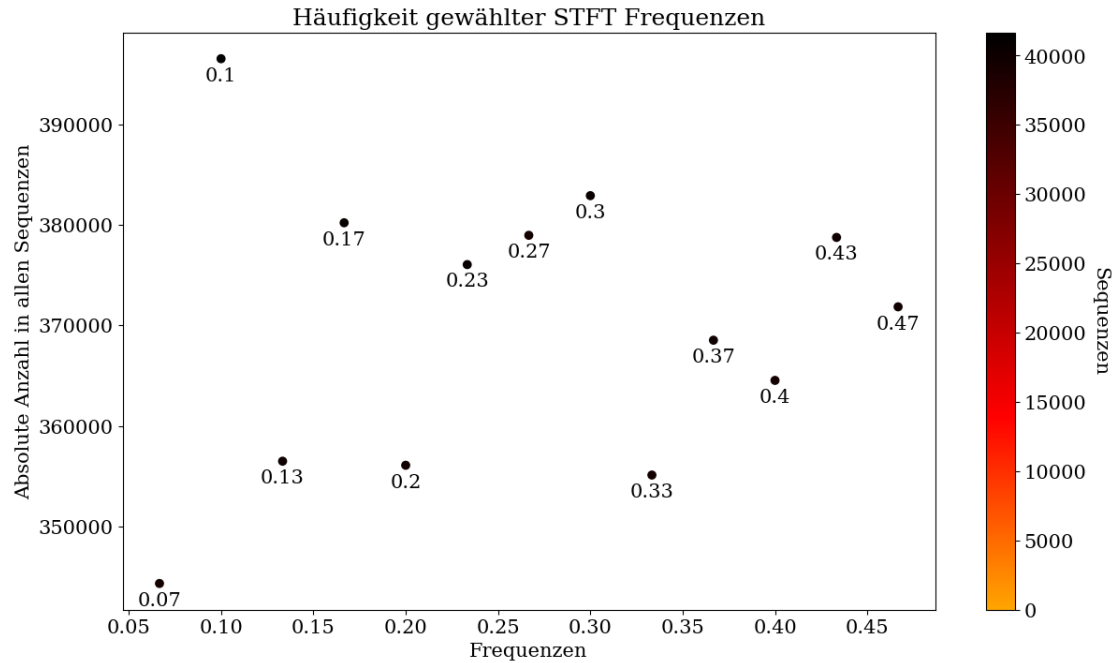


Abbildung 8: Häufigkeit gewählter Frequenzen über alle TPs vor Exp. 1: Die Häufigkeiten, wie oft eine Frequenz in einer Sequenz gewählt wird, werden über alle TPs summiert (y-Achse). Die Bezeichnungen an den Punkten repräsentieren die genaue Frequenz, wie auf der x-Achse dargestellt. Das Reziproke einer Frequenz bedeutet die Länge der Periode. Die Farbe der Punkte gibt an, in wie vielen Sequenzen die jeweilige Frequenz vorkam. Die Werte wurden exemplarisch für Fenstergröße 30 mit Überlappung 15 und $n_peaks = alle$ (siehe Unterabschnitt 3.6 (Seite 18)) durchgeführt, allerdings nur für den KF Hydrophobizität.

Die Häufigkeiten der Frequenzen lagen bisher in prot-fin im Mittel bei ungefähr 360000, was bei circa 40000 Proteinen bedeutet, dass in einer Aminosäuresequenz jede Frequenz etwa 9-mal vorkommt. Frequenz 0.1, also eine Periode von jeder 10. Aminosäure (das Reziproke der Frequenz ist die Länge der Periode), scheint zudem besonders häufig gewählt zu werden.

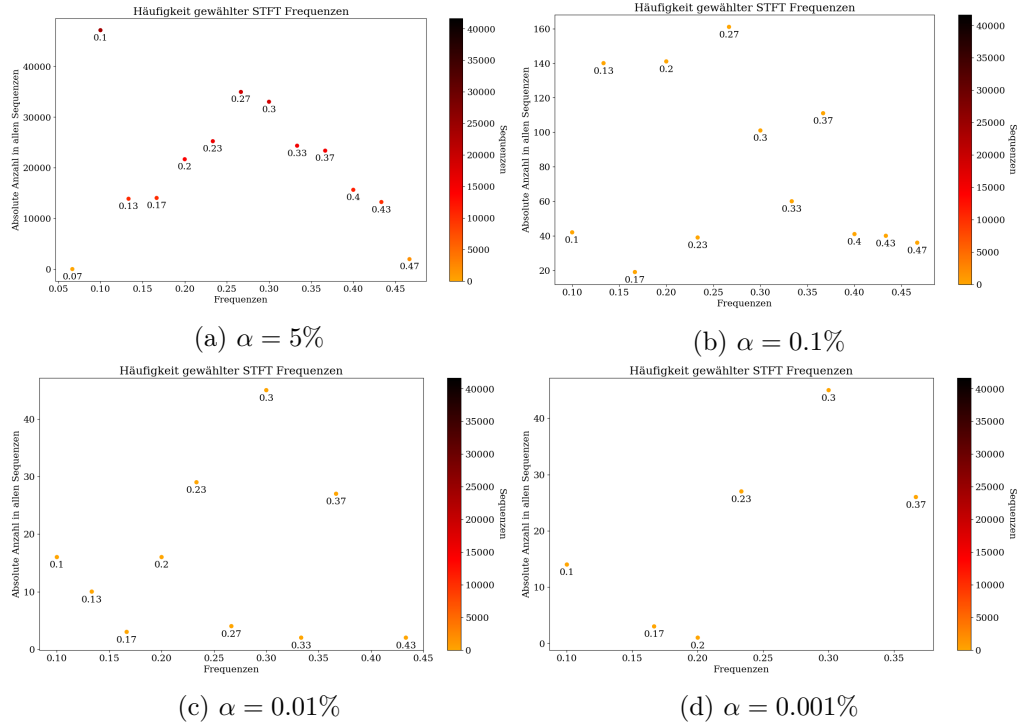


Abbildung 9: Häufigkeit gewählter Frequenzen über alle TPs nach Exp. 1: Die Häufigkeiten, wie oft eine Frequenz in einer Sequenz gewählt wird, werden über alle TPs summiert (y-Achse). Die Bezeichnungen an den Punkten repräsentieren die genaue Frequenz, wie auf der x-Achse dargestellt. Das Reziproke einer Frequenz bedeutet die Länge der Periode. Die Farbe der Punkte gibt an, in wie vielen Sequenzen die jeweilige Frequenz vorkam. Die Werte wurden exemplarisch für Fenstergröße 30 mit Überlappung 15 und $n_peaks = alle$ (siehe Unterabschnitt 3.6 (Seite 18)) durchgeführt. Die Signifikanzniveaus bedeuten, dass α und $1 - \alpha$ als Grenzwert für die Selektion verwendet wurden.

Abbildung 9 (Seite 21) zeigt, wie häufig die einzelnen Frequenzen nach Einbezug der Grenzwerte ausgewählt wurden. Wie Abbildung 8 (Seite 20) ist die Selektion nur exemplarisch für FG 30 mit Überlappung von 15 und $n_peaks = alle$, hier allerdings mit Beachtung aller KFs. Es wurden nicht mehr Parameter dargestellt, weil primäres Ziel die Ermittlung der Grenzwerte je Signifikanzniveau war, Exp. 3 (Seite 17) wurde mit derselben Selektionsmethode für alle Parameter mit $\alpha = 5\%$ durchgeführt. Die Ergebnisse

dazu sind ab Seite 24 zu finden. Für Exp. 4 (Seite 18) wurden alle Signifikanzniveaus verwendet, allerdings mit anderen Selektionsansätzen.

Die Häufigkeiten werden mit schrumpfendem α deutlich kleiner. So kommt eine Frequenz mit 5% Grenze im Mittel nur in etwa jeder zweiten Sequenz vor, und ab 0.1% noch deutlich seltener. Die maximalen Häufigkeiten liegen bei unter 200, wobei auch die Anzahl verschiedener Frequenzen sinkt. So sind es in Abbildung 9d (Seite 21) nur sechs verschiedene, mit einer maximalen Auswahlrate von etwa 40 Mal. Im Vergleich zu Abbildung 8 (Seite 20) mit nur einem betrachteten KF gab es eine sehr starke Reduktion in der Auswahl.

Ursprünglich war Exp. 1 (Seite 16) so konzipiert, dass nur ein 5%-alpha ermittelt wird. Da die Datenbanken aber weiterhin zu groß waren, wie in Abbildung 10 (Seite 22), Unterabschnitt 4.2 (Seite 23) und Unterabschnitt 4.3 (Seite 24) zu sehen, wurde es um die strengeren Signifikanzniveaus erweitert. Die Identifizierung ist ab FG 20 sehr gut und hat eine nahezu 100%-ige Schärfe. FG 10 ist zwar unter der Eingabegröße der Datenbank, scheitert aber bei der Identifikation (Unique Self Matches) und wurde daher nicht für alle Experimente betrachtet.

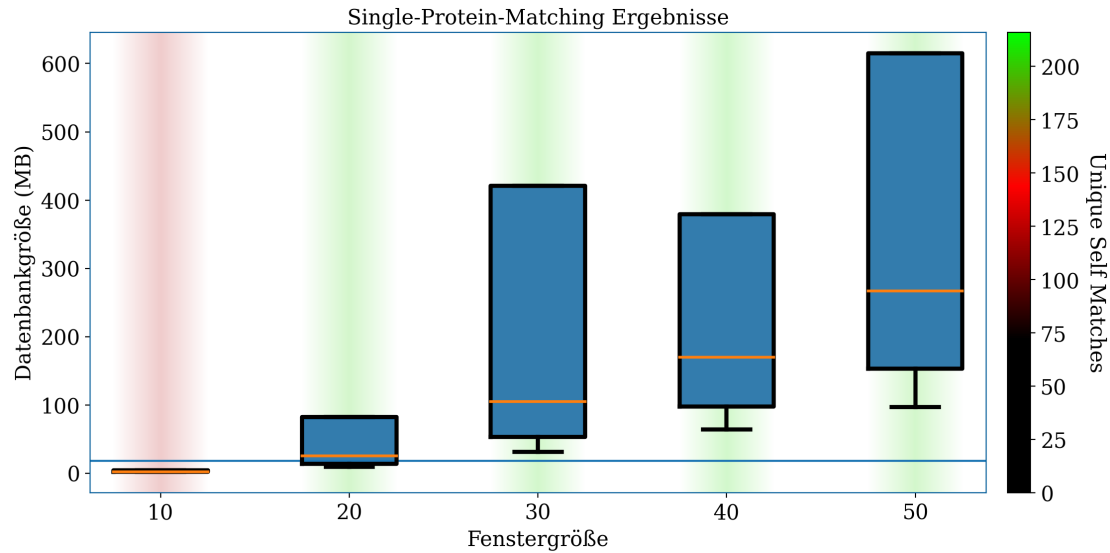


Abbildung 10: Single-Protein-Matching-Ergebnisse für $\alpha = 5\%$. Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Je mehr die Box also prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten. Die genauen Werte dieses Plots sind in den Dateien `UniRef90 Sampling/*` im Anhang (Seite 37) zu finden.

4.2 Filter Hashes

In Exp. 2 war die Intention, in vielen Proteinen auftretende Hashes aus der Datenbank herauszufiltern, da Hashes, die in nahezu allen Proteinen vorkommen, bei der Identifikation eines Proteins wenig Bedeutung haben. Bei der Durchführung der Experimente wurden die seltenen Hashes nach Quantil behalten. Getestet werden sollten alle Quantile in 10%-Schritten von 10% bis 100%, Letzteres als Nullprobe ohne Filtern. Aufgrund eines Eingabefehlers wurde der geplante Parameter $Quantil = 0.8$ ausgelassen (prot-fin wurde “.8.” übergeben statt “.8” und konnte den Wert nicht als `float` parsen).

Es wurden beide Matching-Ansätze durchgeführt (siehe ab Seite 12). Die Ergebnisse für das Single-Protein-Matching sind in Abbildung 11 (Seite 23) dargestellt.

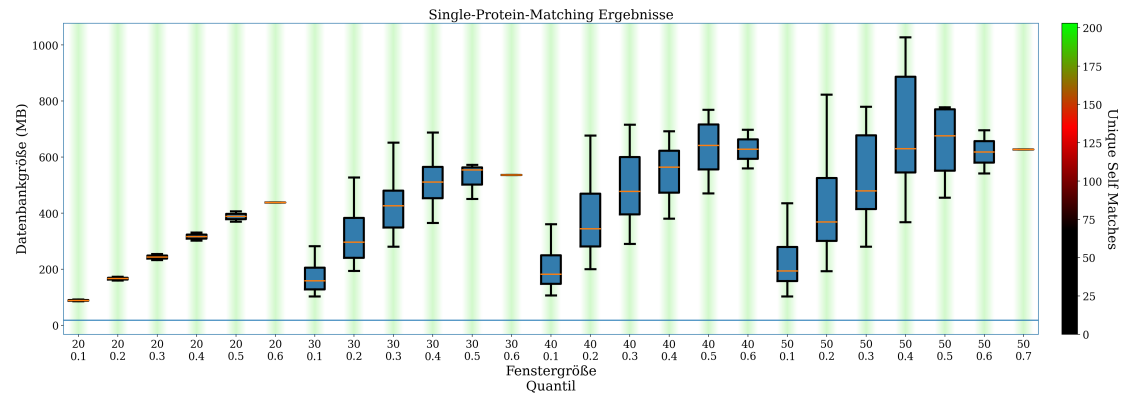


Abbildung 11: Single-Protein-Matching-Ergebnisse: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab, wobei das Experiment aufgrund zu hoher Laufzeit für die oberen Filterquantile vorzeitig abgebrochen wurde. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten. Die Originaldaten für diesen Plot sind in den Dateien **Filter Hashes/*** im Anhang (Seite 37) zu finden.

Aufgrund zu langer Laufzeit wurde das Single-Protein-Matching vorzeitig abgebrochen, sodass die Daten der pro FG jeweils letzten Quantile nicht vollständig sind. Abbildung 11 (Seite 23) zeigt für die Suchproteine für alle getesteten Parameter eine hohe Identifikationsrate mit hoher Schärfe. Mit sinkendem Quantil schrumpft auch die Datenbankgröße annähernd linear. Dennoch ist keine davon unter dem Limit der Eingabe.

Der Family-Matching-Ansatz in Abbildung 12 (Seite 24) hat eine geringe Schärfe von

unter 0.5, wobei sich für die getesteten Parameter zeigt, dass kleinere Quantile einen höheren F1-Score haben, der aber deutlich unter 1 liegt. Zwischen dem 70%- und 100%-Quantil deutet sich ein leichter Anstieg des F1-Scores an.

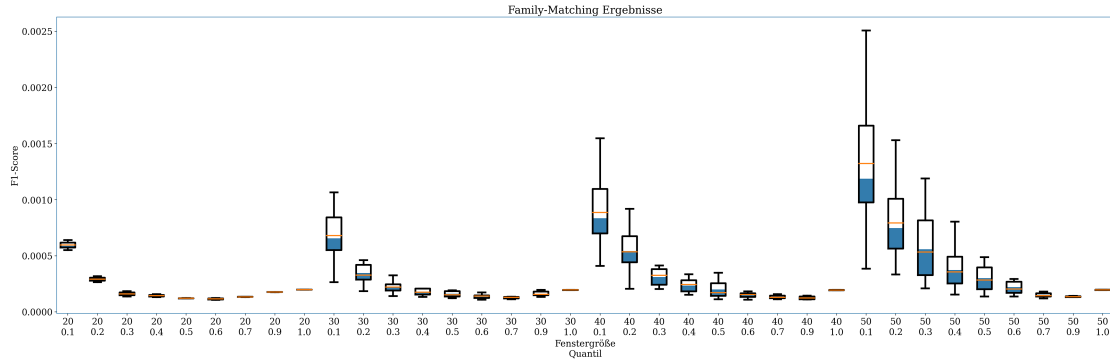


Abbildung 12: Family-Matching-Ergebnisse: Die Boxen bilden die mittleren F1-Scores (siehe Gleichung 2 (Seite 15)) für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die exakten Werte der Ergebnisse, die abgebildet sind, sind in `Filter Hashes/summary_match_family.csv` im Anhang (Seite 37) abgelegt.

4.3 Target-Zone

In Exp. 3 wurde der Selektionsansatz aus Exp. 1 mit $\alpha = 5\%$ verwendet, was somit eine Erweiterung der Ergebnisse aus Unterabschnitt 4.1 (Seite 20) darstellt. Wie dort wird hier lediglich der Single-Protein-Matching-Ansatz durchgeführt, allerdings mit dem Unterschied, dass die in Exp. 3 auf Seite 17 beschriebene Target-Zone eingeschränkt wird. Getestet werden die Werte 8, 16, 32, 64, da sich diese Werte auf eine ganze Zahl an Bits aufteilen lassen (3, 4, 5, 6), um in den Hash (Abbildung 6 (Seite 10)) einzufließen.

Für das Testen verschiedener TZs ist in Abbildung 13 (Seite 25) eine Tendenz zu exponentiellem Wachstum der Datenbankgröße bei ansteigender TZ erkennbar. Je kleiner die Fenstergröße und je kleiner die TZ, desto kleiner wird die Datenbank. Wie beim Sampling in Abbildung 10 (Seite 22) scheitert die Identifikation bei FG 10 hinsichtlich der Unique Self Matches, im Gegensatz zu den anderen Größen, welche auch eine hohe Schärfe haben.

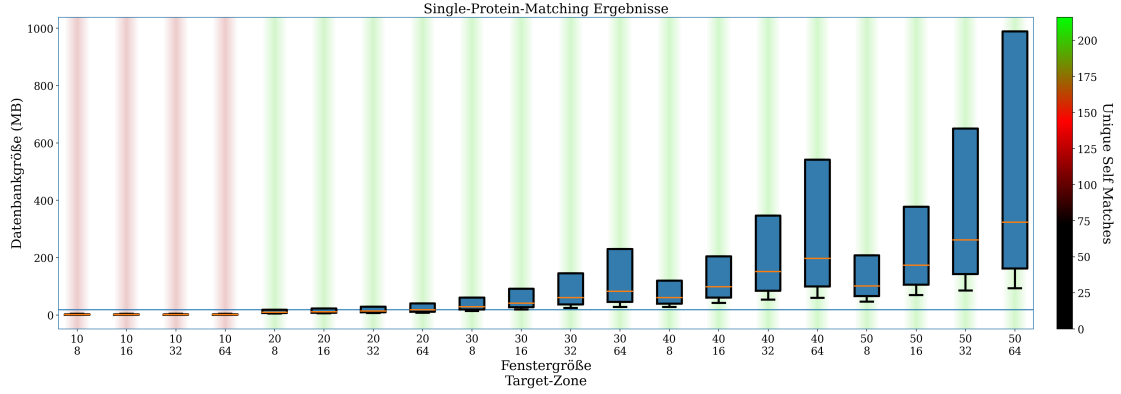


Abbildung 13: Single-Protein-Matching-Ergebnisse: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten. Die Werte für diesen Plot sind in den Dateien `Target-Zone/*` im Anhang (Seite 37) zu finden.

4.4 Selection-Method

Da das Exp. 3 (Seite 17) in Unterabschnitt 4.3 (Seite 24) aufgrund zu hoher Laufzeit vorzeitig abgebrochen wurde, wurde bei der Berechnung folgender Ergebnisse die Datenbankerstellung bei zu hoher Größe abgebrochen. Es wurden beide Matching-Varianten (beschrieben auf Seite 12) für die in Tabelle 2 (Seite 19) angegebenen Parameter durchgeführt und dabei drei verschiedene Ansätze der Selektion verfolgt.

4.4.1 Ansatz 1: Wahl der Frequenzen nur über Grenzwerte

Der erste Ansatz verwendet die durch das Signifikanzniveau α definierten Randbereiche. Es werden nur die Frequenzen ausgewählt, deren Amplituden durch die Grenzwerte zu Ausreißern werden. In Abbildung 14 (Seite 26) sind die Ergebnisse des Single-Protein-Matchings abgebildet. Für $\alpha = 5\%$ gibt es keine Werte, weil die Datenbanken zu groß waren. Ebenfalls zu groß waren sie für alle $k = 0$ und bei $\alpha = 0.1$ zusätzlich bei $k = 1$. Die besten Ergebnisse erzielten jedes Mal die Kombination aus $\alpha = 0.001\%$ mit $k \in \{2, 3\}$, wo die Datenbanken auf Höhe der Eingabegröße liegen und alle Proteine eindeutig erkannt wurden. Die Schärfe liegt zudem jedes Mal bei nahezu 100%.

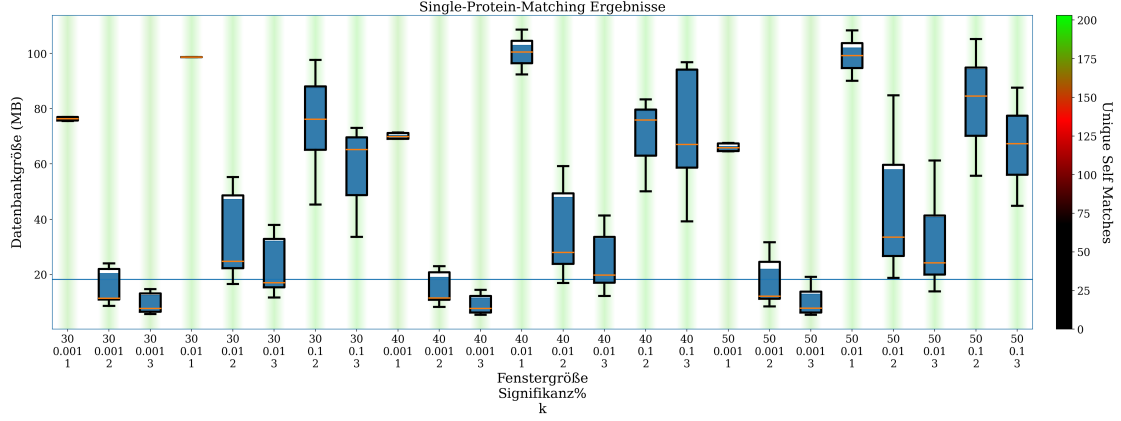


Abbildung 14: Single-Protein-Matching Exp. 4, Ansatz 1: Wahl nur mit Grenzwerten: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten. Die Daten für die Abbildungen sind in den Dateien **Selection-Method/*** im Anhang (Seite 37) abgelegt (Werte bei Spalte **Selection_Method=none**).

Die Ergebnisse des Family-Matchings sind in Abbildung 15 (Seite 27) dargestellt. Wie auch beim Single-Protein-Matching in Abbildung 14 (Seite 26) ist die Performanz für die getesteten Quantile unter 5% für $k \in \{2, 3\}$ immer die beste. Der höchste F1-Score von etwa 33×10^{-4} wird für FG 50 mit 50% Überlappung, $n_peaks = 3$ und $k = 3$ erzielt. Die Schärfe liegt bei allen Werten unter 50%.

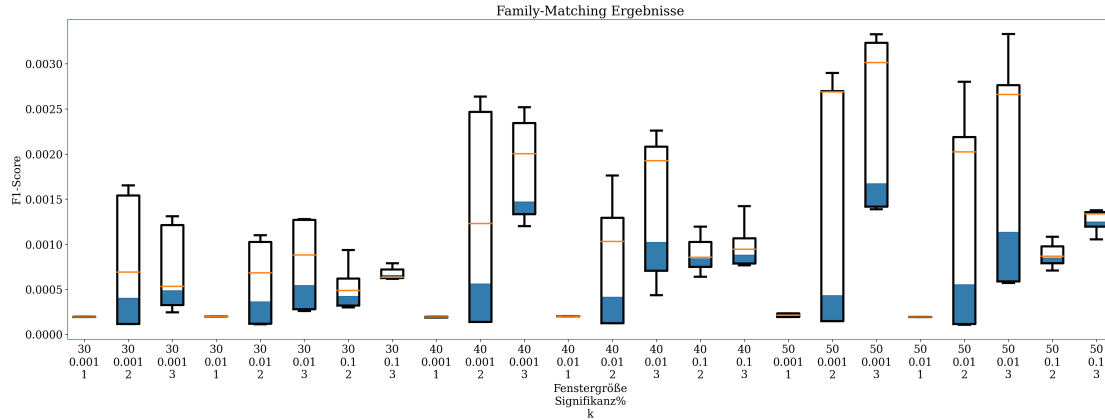


Abbildung 15: Family-Matching Exp. 4, Ansatz 1: Wahl nur mit Grenzwerten: Die Boxen bilden die mittleren F1-Scores (siehe Gleichung 2 (Seite 15)) für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die originalen Werte für diesen Plot sind in `Selection-Method/summary_match_family.csv` im Anhang (Seite 37) zu finden (Werte bei Spalte `Selection_Method=none`).

4.4.2 Ansatz 2: Wahl der Frequenzen über Ausreißer in den Extrempunkten

Über den zweiten Ansatz der Frequenzwahl werden die Frequenzen in zwei Schritten selektiert. Zuerst werden diejenigen behalten, deren Amplituden ein lokales Extremum darstellen. Anschließend werden von diesen wiederum nur welche behalten, deren Amplituden durch das entsprechende Signifikanzniveau α als Ausreißer definiert wird. Demzufolge erweitert dieser Ansatz den ersten (Unterunterabschnitt 4.4.1 (Seite 25)) um die Vorselektion durch Extremwerte. Hierfür wurden beide Matching-Ansätze (beschrieben auf Seite 12) für die in Tabelle 2 (Seite 19) angegebenen Parameter getestet. Die Single-Protein-Matching-Ergebnisse sind in Abbildung 16 (Seite 28) zusammengefasst.

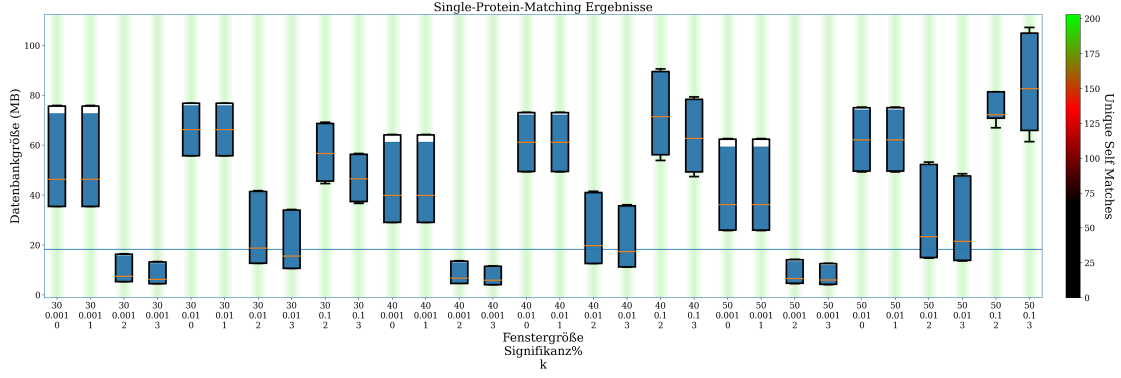


Abbildung 16: Single-Protein-Matching Exp. 4, Ansatz 2: Wahl der Ausreißer der lokalen Maxima/Minima: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten. Die Daten für die Abbildungen sind in den Dateien **Selection-Method/*** im Anhang (Seite 37) abgelegt (Werte bei Spalte **Selection_Method=absolute**).

Die Ergebnisse sind dem ersten Ansatz in Abbildung 14 (Seite 26) sehr ähnlich: Das 5%-Quantil ist nicht vertreten, aufgrund zu großer Datenbanken. Die besten Werte erzielen $k \in \{2, 3\}$ für die restlichen Quantile, wobei das 0.1%-Quantil für k vertreten ist. Die Unterschiede liegen in einer geringeren Streuung der Datenbankgrößen und der Tatsache, dass $k = 0$ Ergebnisse besitzt. Die Werte für $k \in \{0, 1\}$ sind dabei identisch, da Frequenz 0 als Randfrequenz nicht als Extremwert infrage kommt und bei $k = 1$ folglich nicht entfernt werden kann. Es wurden jedes Mal alle Proteine mit einer nahezu 100%-igen Schärfe wiedererkannt.

Die Ergebnisse des Family-Matchings sind in der nachfolgenden Abbildung 17 (Seite 29) zu sehen und den Ergebnissen aus dem ersten Ansatz ebenfalls ähnlich (Abbildung 15 (Seite 27)): Die besten Werte erzielen $k \in \{2, 3\}$ für die restlichen Quantile, wobei das 0.1%-Quantil für k vertreten ist, aber generell die Streuung der F1-Scores geringer ist. Der höchste F1-Score von etwa 33×10^{-4} wird hier ebenfalls für FG 50 mit 50% Überlappung und $n_peaks = 3$ erzielt, allerdings mit $k = 2$. Bei allen Werten wird ebenfalls eine Schärfe von unter 0.5 erreicht.

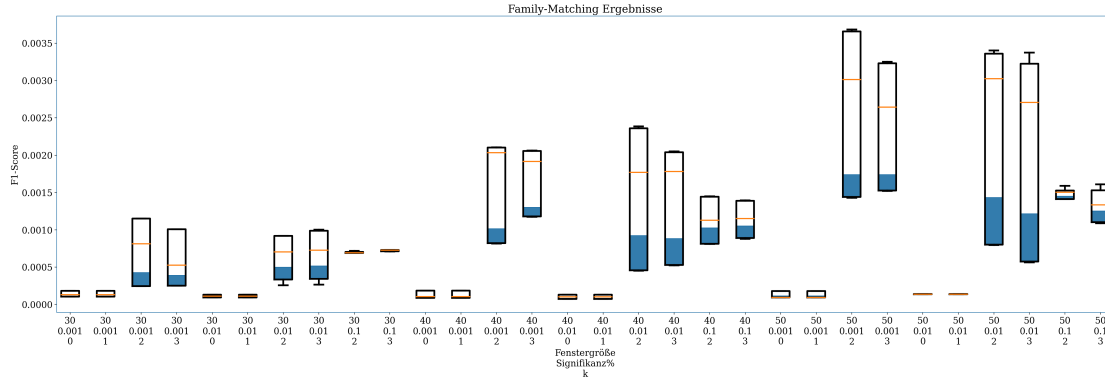


Abbildung 17: Family-Matching Exp. 4, Ansatz 2: Wahl der Ausreißer der lokalen Maxima/Minima: Die Boxen bilden die mittleren F1-Scores (siehe Gleichung 2 (Seite 15)) für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die originalen Werte für diesen Plot sind in `Selection-Method/summary_match_family.csv` im Anhang (Seite 37) zu finden (Werte bei Spalte `Selection_Method=absolute`).

4.4.3 Ansatz 3: Wahl nach Grenzwertabweichung

Der dritte Ansatz der Frequenzselektion ist dem zweiten ähnlich (Unterunterabschnitt 4.4.1 (Seite 25)). Hier wird ebenfalls eine Vorselektion durch Extrema getätigt und anschließend nur die Frequenzen behalten, deren Amplituden die Grenzwerte des entsprechenden α überschreiten. Der Unterschied liegt darin, dass nicht die Extrema in den Amplitudenwerten identifiziert werden, sondern in deren absolute Abweichung zum jeweiligen Grenzwert, sodass nur die stärksten Ausreißer gewählt werden (siehe Beschreibung des dritten Ansatzes in Exp. 4 (Seite 18)). Auch für diesen Ansatz werden für die verschiedenen Parameter aus Tabelle 2 (Seite 19) sowohl das Single-Protein-Matching als auch das Family-Matching durchgeführt. Die Ergebnisse für Ersteres ist in Abbildung 18 (Seite 30) dargestellt.

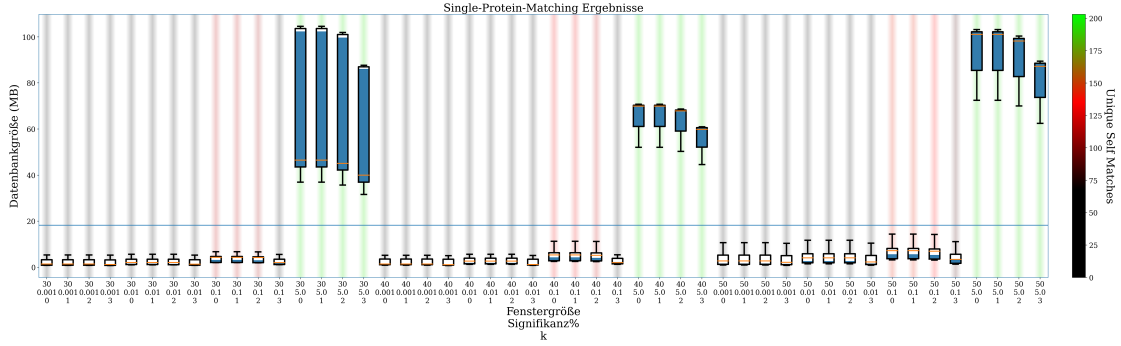


Abbildung 18: Single-Protein-Matching Exp. 4, Ansatz 3: Wahl nach Grenzwertabweichung: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten. Die Daten für die Abbildungen sind in den Dateien **Selection-Method/*** im Anhang (Seite 37) abgelegt (Werte bei Spalte **Selection_Method=deviation**).

Es gibt beim Single-Protein-Matching für alle k und für alle getesteten Quantile Ergebnisse. Für jedes dieser Quantile unter 5% sind die Datenbanken medial unter der Hälfte der Eingabegröße. Gleichmaßen sind die Schärfe und Anzahl eindeutig identifizierter Proteine (Unique Self Matches) allerdings sehr niedrig. Für diesen Ansatz hat lediglich das 5%-Quantil eine hohe Schärfe mit vollständiger Identifikationsrate, wobei die Datenbanken immer zwischen etwa 35 und 100 MB liegen. Bezüglich der Größe hat $k = 3$ hierbei das beste Ergebnis.

Die Ergebnisse des Family-Matchings mit diesem dritten Selektionsansatz sind in Abbildung 19 (Seite 31) abgebildet. Für das 5%-Quantil gibt es in jedem Fall die besten Ergebnisse hinsichtlich Schärfe und F1-Score, abgesehen von Quantil 0.1%, das bei $k = 3$ mit einer Schärfe nahe null dennoch bessere F1-Scores erhält. Die Schärfe der

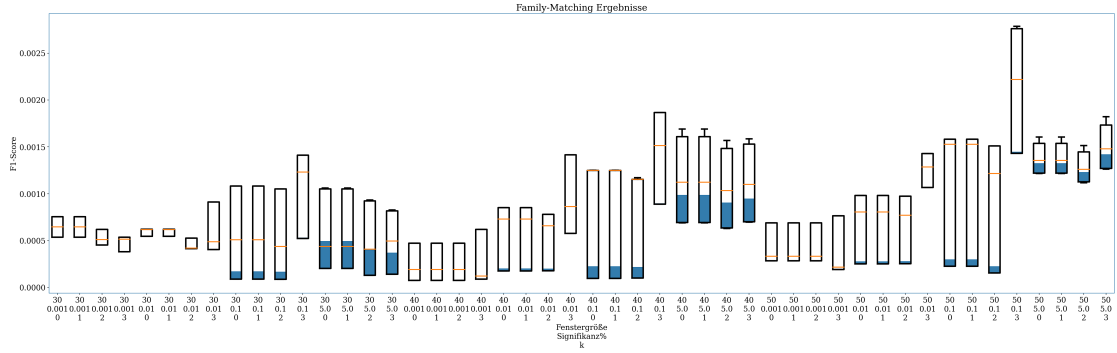


Abbildung 19: Family-Matching Exp. 4 (Seite 18), Ansatz 3: Wahl nach Grenzwertabweichung: Die Boxen bilden die mittleren F1-Scores (siehe Gleichung 2 (Seite 15)) für die verschiedenen Parameter aus Tabelle 2 (Seite 19) ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 (Seite 15)), sofern sie einen positiven Wert hat. Das heißt, je mehr die Box prozentual gefüllt ist, desto höher ist die Schärfe (bei Schärfe 0.5 wäre sie halb voll). Die originalen Werte für diesen Plot sind in `Selection-Method/summary_match_family.csv` im Anhang (Seite 37) zu finden (Werte bei Spalte `Selection_Method=deviation`).

5 Diskussion

Die Entwicklung von prot-fin hat das Ziel, eine mögliche Alternative zu Alignment-basierten Verwandtschaftsanalysen zwischen Proteinen zu erhalten. Der zugrundeliegende Algorithmus ist dabei von SHAZAM inspiriert, welches Musik erkennt, indem es Tonaufnahmen nicht auf direkte Ähnlichkeit vergleicht, wie es bei einem Alignment der Fall wäre, sondern durch Vergleich der periodischen Signale innerhalb der Aufnahmen. Dies wurde zum Teil erfolgreich bei prot-fin umgesetzt, denn die Erkennung von Proteinen funktioniert zwar, dafür ist aber die Performanz hinsichtlich Speicherbedarf der Datenbanken und daraus resultierenden hohen Laufzeiten nicht gut. Außerdem fehlt bisher eine robuste und zuverlässige Möglichkeit, verwandte Proteine zu identifizieren. Um diesen Mängeln entgegenzuwirken, wurden vier verschiedene Experimente entwickelt, die zum Ziel haben, nur möglichst signifikante Signale in die Datenbank einfließen zu lassen.

Verglichen mit den Ergebnissen der vorigen Version von prot-fin in Abbildung 2 (Seite 3), wurde in allen angesetzten Experimenten eine Verbesserung in den Datenbankgrößen erzielt, abgesehen von Fenstergröße (FG) 10, welche an Eindeutigkeit der Ergebnisse verloren hat. Es ist hier aber wichtig zu bemerken, dass in der Vorversion das Matching lediglich für nur einen Kidera-Faktor (KF) durchgeführt wurde. Daher ergibt der Verlust bei FG 10 Sinn, da die so wenigen infrage kommenden Frequenzen trotzdem auf über 100 MB aufgebläht wurden. Bezüglich der Schärfe lässt sich hier leider kein Vergleich vornehmen, da dieses Maß in der Vorversion noch nicht implementiert war. Ebenso stammt der Ansatz des Family-Matchings auch erst aus dieser Arbeit.

Das Sampling scheint eine gute Idee gewesen zu sein, um die Frequenzselektion signifikanter zu gestalten, ohne dabei willkürlich Information zu entfernen. In Abbildung 9 (Seite 21) ist der Effekt auf die Wahl deutlich zu erkennen. Die Reduktion der Häufigkeit, mit der eine Frequenz gewählt wird, im Vergleich zu den vorigen Ergebnissen bei nur einem betrachteten KF ist hier sehr stark erfolgt. So liegt der Maximalwert bei $\alpha = 5\%$ und allen KFs bei etwa 45000 (Abbildung 9a (Seite 21)) und in der Vorversion mit nur einem KF bei fast dem Zehnfachen (Abbildung 8 (Seite 20)). Dennoch ähneln sich beide Varianten in der Hinsicht, dass Frequenz 0.1 am häufigsten auftaucht, also einer Periode von jeder 10. Aminosäure (Reziproke der Frequenz ist Periodenweite).

Die geringere Anzahl Frequenzen zeigt sich in Abbildung 10 (Seite 22) in deutlich kleineren Datenbankgrößen als vorher (Abbildung 2 (Seite 3)).

Bezüglich der Werte für $\alpha \leq 0.1\%$ ist es zweifelhaft, ob bei diesen niedrigen Häufigkeiten unter 200 wirklich alle Trainings-Protein (TP) abgedeckt werden können, zumindest mit dem in Exp. 1 verwendeten Selektionsansatz. Für $\alpha = 5\%$ ist das Single-Protein-Matching in Abbildung 10 (Seite 22) jedenfalls noch vielversprechend.

Da die Datenbankgrößen dennoch recht hoch sind, wäre hier nur eine Reduktion über andere Ansätze denkbar, wie zum Beispiel das Filtern von Hashes in Exp. 2 (Seite 17). Die Ergebnisse dort zeigen, dass auf diese Weise effektiv kleinere Datenbanken erzielt werden, wobei die Schärfe und Identifikationsrate verlustfrei bleiben. Da bei dem Experiment zudem die gelernten Grenzwerte das alleinige Selektionskriterium darstellen, im

Gegensatz zur Kombination mit Wahl lokaler Maxima wie bei Exp. 1 (Seite 16), sollte dies für die Ergebnisse in Abbildung 10 (Seite 22) zu einer Verbesserung führen.

Das Single-Protein-Matching bei Exp. 2 hat gute Ergebnisse hervorgebracht. In Abbildung 11 (Seite 23) ist klar zu erkennen, wie das Filterquantil lediglich die Datenbankgröße beeinflusst. Es hat sich somit durch die kleinen Datenbanken bei kleinem Quantil gezeigt, dass die Hashes, die in sehr vielen Proteinen vorkommen, wenig Relevanz für das Single-Protein-Matching haben.

Auch beim Family-Matching stechen die kleinen Quantile hervor. Das in Abbildung 12 (Seite 24) kleinste Quantil der Hashes von 10% hat hier den besten Score. Allerdings ist der steigende F-Score mit kleiner werdendem Quantil dadurch zu erklären, dass die Wahrscheinlichkeit auf falsche Treffer sinkt, wenn weniger Hashes behalten werden. Die Mitglieder der Familie sind schließlich garantiert enthalten, sofern es Hashes gibt, die sie sich teilen.

In Exp. 3 (Seite 17) wird dieselbe Selektionsmethode wie in Exp. 1 (Seite 16) verwendet. Es zeigt sich, dass eine Target-Zone (TZ) von 8, also einem Bedarf von 3 Bit, vollkommen ausreichend für die Identifikation ist. Wie auch in den Ergebnissen zu Exp. 2 in Unterabschnitt 4.2 (Seite 23) ist hier beim Single-Protein-Matching kein Verlust an Präzision bei der Erkennung von Proteinen sichtbar (siehe Abbildung 13 (Seite 25)). Insofern scheint die TZ von 8 die beste Wahl zu sein, um eine kleine Datenbank zu erzielen. Lediglich FG 10 hat eine schlechte Performanz, doch das hängt mit dem verwendeten Signifikanzniveau von 5% zusammen, welches schon in dem repräsentativen Single-Protein-Matching in Unterabschnitt 2.2 (Seite 5) von Exp. 1 eine schlechte Identifikation lieferte.

... Selection-Method ...

Was in den Ergebnissen fehlt, ist beim Single-Protein-Matching der Bezug zur Familienähnlichkeit. Aktuell wird lediglich betrachtet, ob das Protein selbst identifiziert wurde und wie weit sich der Score von den Treffern abhebt, die nicht in der Familie sind. Letzteres wird durch die Schärfe abgebildet, die den Abstand prozentual angibt, also wie viel höher der Score ist. Dieser Abstand soll möglichst hoch sein, der zu den Familienmitgliedern hingegen nicht. Die Schärfe müsste um diese Information erweitert werden, was sich folgendermaßen formulieren lässt:

$$\begin{aligned}
scores_trp &= \{SI(t) \cdot JSI(t) \mid t \in TrP\} \\
scores_fp &= \{SI(f) \cdot JSI(f) \mid f \in FP\} \\
dist_nicht_familie &= \frac{\max(scores_trp) - \max(scores_fp)}{\max(scores_trp)} \\
dist_familie &= \frac{\max(scores_trp) - \text{mean}(scores_trp \setminus \{\max(scores_trp)\})}{\max(scores_trp)} \\
Schärfe &= dist_nicht_familie \cdot (1 - dist_familie)
\end{aligned} \tag{4}$$

TrP sind die Treffer innerhalb der Familie und FP die anderen. Der Abstand ($dist$) zu den TrP ist, wie sehr sich der beste Score der Familie prozentual von allen anderen

Familienmitgliedern abhebt. Damit dieser Abstand in die Schärfe minimierend einfließt, wird diese wie bisher berechnet und anschließend mit der Umkehrung des Familienabstands multipliziert, also wie nah der beste Score den *TrP* ist. Auf diese Weise wäre ein hoher Abstand zu *FP* bei ebenso hohem Abstand zu *TrP* trotzdem schlecht bewertet. Gleiches gilt umgekehrt, dass eine hohe Nähe zu den Familienmitgliedern auch schlecht bewertet wird, wenn die *FP* ebenso nah sind. Gegebenenfalls sollte der Abstand zu den *FP* auch über den Mittelwert berechnet werden anstelle des Maximums, damit die Schärfe robuster gegenüber Ausreißern der *FP* ist.

Ausblick

SHAZAM ist zweifellos sehr erfolgreich in der Erkennung von Musik mittels Tonaufnahmen, die aufgrund schlechter Mikrofonqualität sehr verzerrt sind. Durch das Aufteilen in mehrere Fenster, die einen Umfang von unter einer Sekunde haben, wird dies ermöglicht, da somit nicht relevant ist, wo die Aufnahme startet.

Wie in der Einleitung (Seite 1) erwähnt, sind Vektoren der digitalen Musik deutlich länger als Proteinsequenzen. Ein Fenster von einer halben Sekunde hätte etwa 20000 Werte. Bei einem angenommenen Originalsong von drei Minuten Länge (180 Sekunden) und 50% Überlappung der Fenster wären das $\frac{180 \cdot (20000 \cdot 2)}{0.5 \cdot 20000} = 720$ Fenster.

Die Umsetzung dieses Verfahrens auf Proteine führt aktuell in prot-fin allerdings zu Fenstern von maximaler Länge 50 (siehe Tabelle 2 (Seite 19)), was bei einer medialen Aminosäuresequenzlänge von 300 (Abbildung 3 (Seite 5)) bei auch 50% Überlappung zu lediglich $\frac{300}{0.5 \cdot 50} = 12$ Fenstern führt.

Die kurzen Fenster führen zu einer geringen Anzahl überhaupt möglicher Frequenzen und die wenigen Fenster zu eingeschränkter Hashbildung. Es ist nicht auszuschließen, dass dieser aktuelle Ansatz der Übersetzung einer Aminosäuresequenz in einen numerischen Vektor nicht zielführend ist.

Sollten weitere Experimente weiterhin eine schlechte Performanz hinsichtlich der Erkennung funktional Verwandter aufweisen, gibt es noch alternative Ansätze, den numerischen Vektor zu generieren:

1. Ein noch nicht weiterverfolgter Weg ist, die Sequenzen in echte Musik zu übersetzen, sodass SHAZAM direkt darauf angewandt werden könnte. **A0A1D8EJF9.wav** im Anhang (Seite 37) ist eine Beispielmusikdatei, die in einem anfänglichen Test aus einer Aminosäuresequenz generiert wurde, wobei jede Aminosäure in einem Akkord aus KFs entsprach.

Ein solcher Akkord besteht folglich aus 10 Tönen, dessen Lautstärke durch den jeweiligen Wert eines KF beeinflusst wird (näheres dazu siehe **README.txt** im Anhang (Seite 37) im Abschnitt **Reproduzierbarkeit** zu **A0A1D8EJF9.wav**). Problem hierbei war ebenso die Datenbankgröße, da Musik komplexer als Text ist. Dennoch ist der Kern der Idee vielleicht trotzdem richtig, da die Aminosäuresequenz so den Umfang von tausenden Werten hätte, wie auch Musik, und SHAZAM direkt auf

diese synthetische Aufnahme angewandt werden könnte. Die exakte Methode, wie die Übersetzung am besten erfolgt, müsste dann ausgetestet werden.

2. Ebenso möglich wäre, anstelle ein synthetische Musik zu generieren, stattdessen den aktuell erstellten Vektor, wie in Abbildung 4 (Seite 7) dargestellt, durch lineare Interpolation um weitere Werte zu ergänzen. Auf diese Weise wären ebenfalls mehr Fenster möglich, abhängig davon, um wie viele Werte der Vektor erweitert wird.

Es gibt allerdings auch noch Wege, wie die aktuelle Implementierung weiterentwickelt werden kann. So ist zum Beispiel das Family-Matching ausbaufähig. Neben den oben genannten Änderungen, könnte ebenfalls die Bewertungsmethode der Treffer erweitert werden. Aktuell ist deren Score lediglich die Anzahl Hashes, die mit den Hashes der Familie übereinstimmen, wie auf Seite 15 erläutert. Im Gegensatz zum Single-Protein-Matching wird hier die Position der Hashes also nicht einbezogen, was die Wahrscheinlichkeit auf Übereinstimmung deutlich erhöht. Der Grund ist, dass nicht klar ist, ob die Hashes auch innerhalb der Familie dieselben Positionen haben. Das müsste experimentell ermittelt werden. Andernfalls wäre es auch ein möglicher Ansatz, alle Positionen P zu speichern und bei der Identifizierung eines Treffers einen Hash nur als Übereinstimmung zu bewerten, wenn seine Position Teil von P ist.

Das Speichern aller Positionen der Hashes wäre ebenso eine Option für das Single-Protein-Matching. Zwar würde das Ermitteln des S1-Score, siehe Abbildung 7 (Seite 14), deutlich aufwändiger sein, aber möglicherweise wäre dieser Score dann gar nicht mehr notwendig. Wenn bei der Generierung der Hashes in Algorithmus 3 (Seite 9) nicht die Position selbst, sondern die Anzahl der Positionen gespeichert werden würde, wäre die Summe der Werte der Hashes die Gesamtzahl an Hashes des Proteins. Werden die Hashes zweier Proteine verglichen, so bildet die Summe der absoluten Differenzen der geteilten Hashes die Übereinstimmung beider Proteine.

Das Verhältnis beider wäre konzeptuell dem JSI ähnlich, nur dass die beiden Mengen Duplikate enthalten dürfen. Das Problem, dass der JSI positionsunspezifisch ist, wird hier dadurch ausgeglichen, dass wirklich alle Hashes in den Score einfließen. Denn die Kombination aller Hashes beinhaltet deren Positionen in abstrakter Form, da die Hashes prinzipiell Kanten in einem fast vollständigen Graph sind, wie in Abbildung 5 (Seite 10) dargestellt. Hat man eine Kante, folgt zwangsläufig daraus, dass von den Enden ebenfalls weitere Kanten ausgehen, und mit der Vorgabe, nach welchem System die Punkte verbunden werden, entsteht trotzdem der ursprüngliche Graph. Demzufolge wäre es auch möglich, mit diesem Score die TZ wieder zu erweitern, um einen vollständigeren Graph aus der Constellation-Map (CM) zu erstellen, wobei die Performanz des Scorings davon unberührt bliebe.

Ein Problem, das bei den kleinen FGs angegangen werden muss, ist die Hashbildung. Bei einer maximalen FG von 50 gibt es 26 Frequenzen, die als Minimum oder Maximum ausgewählt werden können. Bei einer TZ von 8 sind das $(26 \cdot 2)^2 \cdot 8 = 21632$ mögliche Kombinationen, die ein Hash annehmen kann. Davon ausgehend, dass prot-fin mit öffentlichen Datenbanken wie der UniProtKB [Con20] werden soll, welche in Version 2024_04 vom 24.07.2024 über 200 Millionen von Einträgen in TrEMBL hat und über 500 Tausend

in Swiss-Prot, ist das viel zu unspezifisch.

Sollte es bei diesen kleinen Fenstern bleiben, muss der Informationsgehalt [Sha48] eines Hashes deutlich erhöht werden. Hierfür könnten zum Beispiel statt Paaren von Punkten der CM größere n -Tupel kombiniert werden, wobei hier die Gefahr zu hoher Spezifität besteht. Dennoch würde die Anzahl möglicher Kombinationen in Abhängigkeit von n exponentiell steigen $((26 * 2)^n \cdot 8)$ und entspräche zum Beispiel für Quadrupel schon 7311616, was deutlich mehr ist. Das oben beschriebene Einbeziehen aller Hash-Positionen wäre vielleicht auch eine Option für das Problem oder die Erweiterung der Vektoren, weil Ersteres die Häufigkeit eines Hashes einbringt, welche beim Scoring relevant ist, und Letzteres die Anzahl Fenster oder maximal möglicher Frequenzen erhöht.

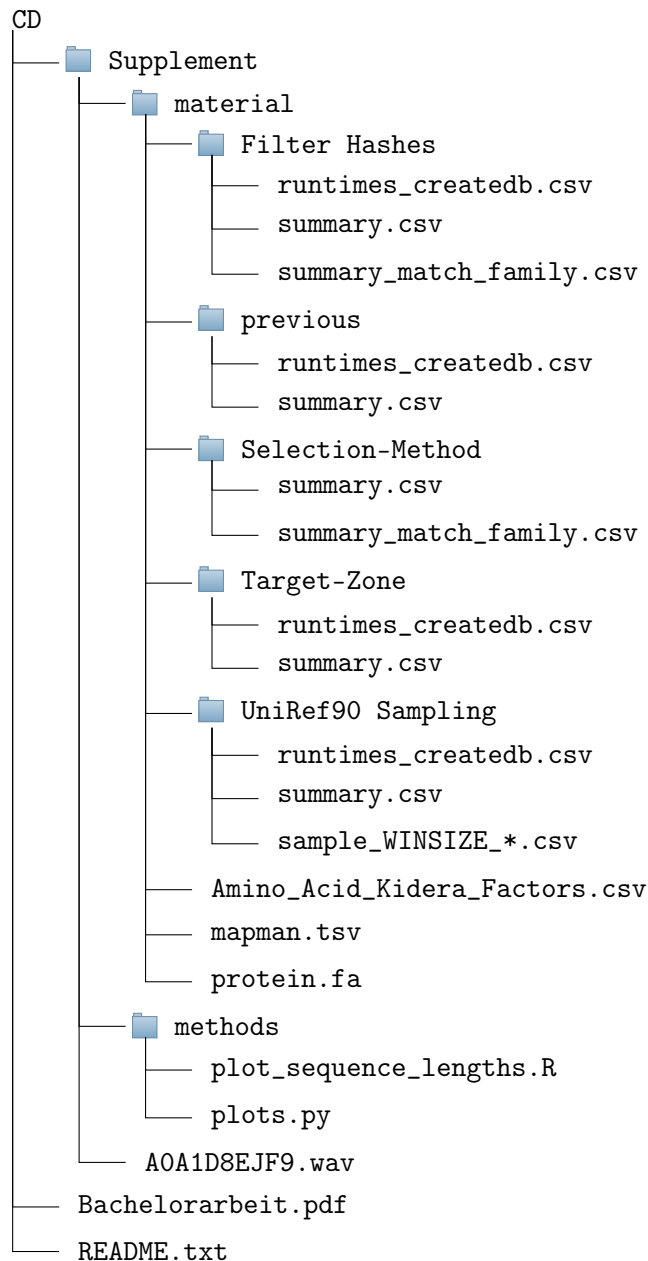
Diese Betrachtungen zeigen, dass noch einige Experimente möglich sind, bevor das Konzept, die von SHAZAM inspirierte Spektralanalyse auf Proteinsequenzen anzuwenden, als ungeeignet abzulehnen wäre. Die Identifikation einzelner Proteine funktioniert sehr gut, was allerdings bisher nur mit vollständigen Suchsequenzen getestet wurde. Das Erkennen von funktioneller Ähnlichkeit ist hingegen noch schwach. Doch wird ein Weg gefunden, besonders Letzteres zu verbessern, so bietet der Algorithmus eine ganz neue Alternative für die sequenzbasierte Verwandtschaftsanalyse.

6 Anhang

Sollte die CD nicht verfügbar sein, ist diese Arbeit mit Anhang auf GitHub verfügbar:

<https://github.com/qwerdenkerXD/Bachelorarbeit-Bioinformatik/tree/master/CD>

Alle nötigen Informationen zu den Dateien, der Reproduzierbarkeit der Ergebnisse und Plots und der Verfügbarkeit des Codes der Implementierung der Algorithmen in prot-fin sind in `README.txt` nachzulesen.



Literatur

- [Con20] The UniProt Consortium. “UniProt: the universal protein knowledgebase in 2021”. In: *Nucleic Acids Research* 49.D1 (Nov. 2020), S. D480–D489. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa1100. eprint: <https://academic.oup.com/nar/article-pdf/49/D1/D480/35364103/gkaa1100.pdf>. URL: <https://doi.org/10.1093/nar/gkaa1100>.
- [Doo81] Russell F Doolittle. “Similar amino acid sequences: chance or common ancestry?” In: *Science* 214.4517 (1981), S. 149–159.
- [Jum+21] John Jumper u. a. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596.7873 (Aug. 2021), S. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: <https://doi.org/10.1038/s41586-021-03819-2>.
- [Kid+85] Akinori Kidera u. a. “Statistical analysis of the physical properties of the 20 naturally occurring amino acids”. In: *Journal of Protein Chemistry* 4.1 (Feb. 1985), S. 23–55. ISSN: 1573-4943. DOI: 10.1007/BF01025492. URL: <https://doi.org/10.1007/BF01025492>.
- [LOH+14] MARC LOHSE u. a. “Mercator: a fast and simple web server for genome scale functional annotation of plant sequence data”. In: *Plant, Cell & Environment* 37.5 (2014), S. 1250–1258. DOI: <https://doi.org/10.1111/pce.12231>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/pce.12231>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/pce.12231>.
- [ORT15] Daniel Osorio, Paola Rondón-Villarreal und Rodrigo Torres. “Peptides: A Package for Data Mining of Antimicrobial Peptides”. In: *The R Journal* 7 (1 2015). <https://rjournal.github.io/>, S. 4–14. ISSN: 2073-4859.
- [Pea13] William R. Pearson. “An Introduction to Sequence Similarity (“Homology”) Searching”. In: *Current Protocols in Bioinformatics* 42 (2013), S. 3.1.1–3.1.8. DOI: <https://doi.org/10.1002/0471250953.bi0301s42>. eprint: <https://currentprotocols.onlinelibrary.wiley.com/doi/pdf/10.1002/0471250953.bi0301s42>. URL: <https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/0471250953.bi0301s42>.
- [Sch+19] Rainer Schwacke u. a. “MapMan4: A Refined Protein Classification and Annotation Framework Applicable to Multi-Omics Data Analysis”. In: *Molecular Plant* 12.6 (2019). Plant Systems Biology, S. 879–892. ISSN: 1674-2052. DOI: <https://doi.org/10.1016/j.molp.2019.01.003>. URL: <https://www.sciencedirect.com/science/article/pii/S1674205219300085>.
- [Sha48] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.4 (1948), S. 623–656. DOI: 10.1002/j.1538-7305.1948.tb00917.x.

- [Str21] Michael Strauss. “How Shazam Works - An explanation in Python”. In: (Jan. 2021). Zugriff: 17.08.2024. URL: <https://michaelstrauss.dev/shazam-in-python>.
- [Suz+14] Baris E. Suzek u. a. “UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches”. In: *Bioinformatics* 31.6 (Nov. 2014), S. 926–932. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu739. eprint: https://academic.oup.com/bioinformatics/article-pdf/31/6/926/49011550/bioinformatics_31_6_926.pdf. URL: <https://doi.org/10.1093/bioinformatics/btu739>.
- [Var+23] Mihaly Varadi u. a. “AlphaFold Protein Structure Database in 2024: providing structure coverage for over 214 million protein sequences”. In: *Nucleic Acids Research* 52.D1 (Nov. 2023), S. D368–D375. ISSN: 0305-1048. DOI: 10.1093/nar/gkad1011. eprint: <https://academic.oup.com/nar/article-pdf/52/D1/D368/55039845/gkad1011.pdf>. URL: <https://doi.org/10.1093/nar/gkad1011>.
- [Vir+20] Pauli Virtanen u. a. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), S. 261–272. DOI: 10.1038/s41592-019-0686-2.
- [Wan03] Avery Wang. “An Industrial Strength Audio Search Algorithm.” In: Jan. 2003.

Eigenständigkeitserklärung

Ich bestätige, dass die eingereichte Arbeit eine Originalarbeit ist und von mir ohne weitere Hilfe verfasst wurde. Die Arbeit wurde nicht geprüft, noch wurde sie widerrechtlich veröffentlicht. Die eingereichte elektronische Version ist die einzige eingereichte Version.

Unterschrift

Ort und Datum

Erklärung zu Eigentum und Urheberrecht

Ich erkläre hiermit mein Einverständnis, dass die Technische Hochschule Bingen diese Arbeit Studierenden und interessierten Dritten zur Einsichtnahme zur Verfügung stellen und unter Nennung meines Namens (Franz-Eric Sill) veröffentlichen darf.

Unterschrift

Ort und Datum