



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

Dozent: Prof. Dr. Asis Hallab

Zusammenfassung

...

Abstract

...

Literatur

- [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>.
- [Str21] Michael Strauss. “How Shazam Works - An explanation in Python”. In: (Jan. 2021). Zugriff: 17.08.2024. URL: <https://michaelstrauss.dev/shazam-in-python>.
- [Wan03] Avery Wang. “An Industrial Strength Audio Search Algorithm.” In: Jan. 2003.

Abbildungsverzeichnis

1	Spektralanalyse eines Fensters der STFT mit Markierung lokaler Maxima	1
2	Bisherige Ergebnisse von prot-fin	2
3	Beispielsequenz als numerische Repräsentation	5
4	Constellation-Map und Hashing	7
5	Schematischer Aufbau eines Hashes	8
6	Ermittlung des S1-Score	11
7	Häufigkeit gewählter Frequenzen über alle Trainings-Proteine Exp. 1 . . .	16
8	Single-Protein-Matching Exp. 1	17
9	Single-Protein-Matching Exp. 2	18
10	Family-Matching Exp. 2	18
11	Single-Protein-Matching Exp. 3	19
12	Mittlere Frequenzzahl pro Fenster einer Sequenz	20
13	Single-Protein-Matching Exp. 4	21
14	Family-Matching Exp. 4	21

Tabellenverzeichnis

1	Kidera-Faktoren	5
2	Experiment-Parameter	15

Abkürzungsverzeichnis

ASCII	“American Standard Code for Information Interchange”	20
CSV	Comma Separated Values	9
Exp.	Experiment	13
FG	Fenstergröße	15
JSI	Jaccard Similarity Index	11
KF	Kidera-Faktor	1
MB	Megabyte	
STFT	Short Time Fourier Transformation	1
TP	Trainings-Protein	8
TZ	Target-Zone	14

Algorithmenverzeichnis

1	Übersetzen einer Aminosäuresequenz in einen numerischen Vektor	4
2	Sammeln von Strukturdaten	6
3	Hashing	6
4	Erstellung der Datenbank	8
5	Treffer-Bewertung beim Single-Protein-Matching	10

Inhaltsverzeichnis

Abstract	II
Literatur	III
Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Abkürzungsverzeichnis	VI
Algorithmenverzeichnis	VII
1 Einleitung	1
2 Material	4
2.1 Trainings-Proteine	4
2.2 UniRef90	4
3 Methoden	4
3.1 Grundalgorithmus	4
Übersetzen einer Aminosäuresequenz in einen numerischen Vektor	4
Sammeln von Strukturdaten	6
Hashing	6
Erstellung der Datenbank	8
Single-Protein-Matching	9
Family-Matching	12
3.2 Experiment 1: UniRef90 Sampling	13
3.3 Experiment 2: Filter Hashes	13
3.4 Experiment 3: Target-Zone	14
3.5 Experiment 4: Selection-Method	14
3.6 Durchführung	15
4 Ergebnisse	16
4.1 UniRef90 Sampling	16
4.2 Filter Hashes	18
4.3 Target-Zone	19
4.4 Selection-Method	19
5 Diskussion	22
6 Anhang	26

1 Einleitung

... Wissenschaftlicher Kontext, zufällige Ähnlichkeit in Alignments ...

Diese Bachelorarbeit beinhaltet die Entwicklung von Version 0.4 des Projekts prot-fin¹ und den zugehörigen Experimenten. prot-fin stellt sich dem Problem zufälliger Ähnlichkeit und beschäftigt sich daher mit der Frage, ob es möglich ist, funktionsähnliche Proteine über ihre physikalischen Eigenschaften zu identifizieren, anstelle der lediglichen Buchstaben ihrer Aminosäuren, und ob das die Problematik umgeht.

Eine Grundlage hierfür bildet die Arbeit von Akinori Kidera *et. al.*, 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.

Der Einfluss eines jeden KF in einer Aminosäure lässt sich numerisch darstellen, sodass eine Aminosäuresequenz in 10 Vektoren übersetzt werden kann, welche nun 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 Short Time Fourier Transformation (STFT) dar, welche in dem musikalischen Spektrum intervall-/fensterweise periodisch auftretende Signale analysiert, wodurch auch die Störgeräusche eine geringe Relevanz haben.

In Abbildung 1 wird dieser Sachverhalt für ein Fenster der STFT dargestellt. 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. Diese Signalstärke oder Amplitude der Frequenz wird über Summen der Originaldaten ermittelt. Frequenz 0 ist lediglich die Summe aller Eingabewerte, also hier allen KF-Werten, die den numerischen Vektor der Eingabe-Aminosäurekette darstellen.

Damit die Musikerkennung funktioniert,

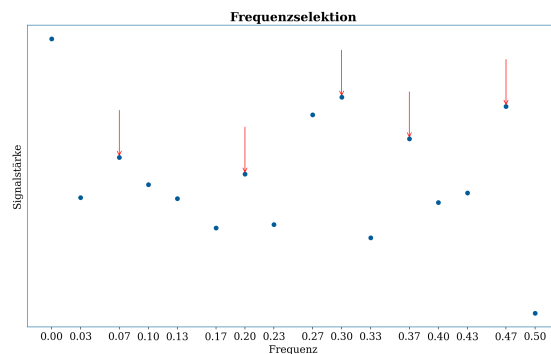


Abbildung 1: Spektralanalyse eines Fensters der STFT mit Markierung lokaler Maxima

¹https://github.com/usadellab/prot-fin/releases/tag/v0.4/experiments/recog_with_fft bei Commit 86ea260

wird nun vorher eine Datenbank erstellt, welche die Periodizitäten der Eingabesongs 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 wird das mittels der lokalen Maxima der Amplituden erreicht.

Für die Anwendung auf Proteine werden statt des musikalischen Spektrums die numerischen Vektoren der Aminosäuresequenzen verwendet. Nun gibt es in prot-fin zwei verschiedene Anwendungsansätze:

1. **Single-Protein:** 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 rechten Abbildung 2 dargestellt. Die Wahl der Frequenzen der STFT Fenster und deren Abspeicherung müssen folglich noch verbessert werden.

Hierzu werden in dieser Arbeit mehrere Experimente angegangen.

1. **UniRef90-Sampling:** Je weniger Frequenzen in einem Fenster ausgewählt werden, desto weniger muss gespeichert werden. 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 das Wahlkriterium für eine Frequenz sein, was zu möglicherweise weniger selektierten Frequenzen mit dennoch

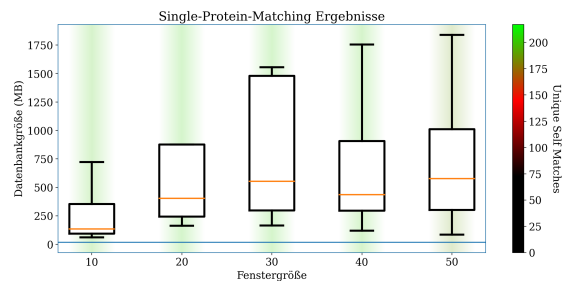


Abbildung 2: Bisherige Ergebnisse von prot-fin: Überblick der Ergebnisse für die Hydrophobizität mit Verwendung verschiedener Parameter, die die STFT betreffen (siehe Unterabschnitt 3.6). Die Farbe stellt dar, wie viele Eingabeproteine eindeutig identifiziert wurden, die blaue Linie die Größe der Eingabedaten.

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 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. Dabei wird die Datenbankerstellung abgebrochen, sobald deren Größe die Eingabe um ein 6-faches übersteigt.

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 [LOH+14].

Diese Bachelorarbeit ist auf [GitHub](#)² verfügbar. Bei Fragen oder Anmerkungen zur Kontaktaufnahme bitte die dortige Issue-Funktion verwenden.

²<https://github.com/qwerdenkerXD/Bachelorarbeit-Bioinformatik>

2 Material

2.1 Trainings-Proteine

...

2.2 UniRef90

...

3 Methoden

3.1 Grundalgorithmus

Algorithmus 1 Übersetzen einer Aminosäuresequenz in einen numerischen Vektor

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

Vorbedingung $0 \leq kf \leq 10$

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

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

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

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

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

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 das Spektrum einer 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 Akinori Kidera, welches 1985 publiziert wurde. 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 [vgl. Kid+85]. In Tabelle 1 sind diese dargestellt.

Tabelle 1: Kidera-Faktoren

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.

Da für die Fourier Transformation negative Werte problematisch sind, wird der Vektor anschließend dahingehend normalisiert, dass das absolute Minimum aller Werte aus Tabelle 1 aufaddiert wird. Das absolute Minimum ist 2.33, sodass die Übersetzung der Beispielsequenz EVKEFDGQGCFC für die Hydrophobizität folgendermaßen geschehe:

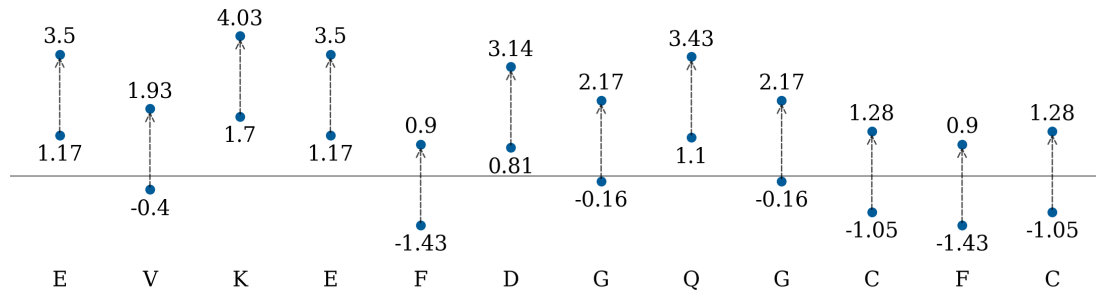


Abbildung 3: Beispielsequenz als numerische Repräsentation: Die horizontale Linie ist die Nulllinie

Algorithmus 2 Sammeln von Strukturdaten

Vorbedingung *aa_vector* aus Algorithmus 1

▷ *aa* = Aminosäure

Nachbedingung *constellation_map* is an Array of Arrays of Floats

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

Der erhaltene Vektor aus Algorithmus 1 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 oder in der Musik ein Refrain oder dem Rhythmus. Für jedes Fenster werden die Frequenzen der auffälligsten Signale ausgewählt, wie in Abbildung 1 dargestellt mittels der lokalen Maxima, sodass über alle Intervalle eine sogenannte Constellation-Map entsteht. Diese Map wird dabei als Array repräsentiert, wobei jedes Element hierbei eine Liste darstellt, die ihrem Index entsprechend die gewählten Frequenzen des jeweiligen Fensters beinhaltet. In Abbildung 4 ist im linken Teil die visuelle Darstellung einer Constellation-Map als Scatter-Plot abgebildet.

Algorithmus 3 Hashing

Vorbedingung *constellation_map* aus Algorithmus 2

Vorbedingung *protein_id* is a String

Vorbedingung $0 \leq kf \leq 10$

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

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

Die erhaltene Map wird elementweise gehashed, um einen effizienten Vergleich mit anderen Maps 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 Map 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 wird sich folgend für den Hash die Position der Kante in der Constellation-Map gemerkt, sowie die ID des Proteins, für die diese Map erstellt wurde. Sollte ein Hash dabei mehrfach vorkommen, verbleibt lediglich seine letzte Position. Dieses Verfahren wird im rechten Teil von Abbildung 4 repräsentativ dargestellt, wobei rote Linien die ignorierten Kanten abbilden.

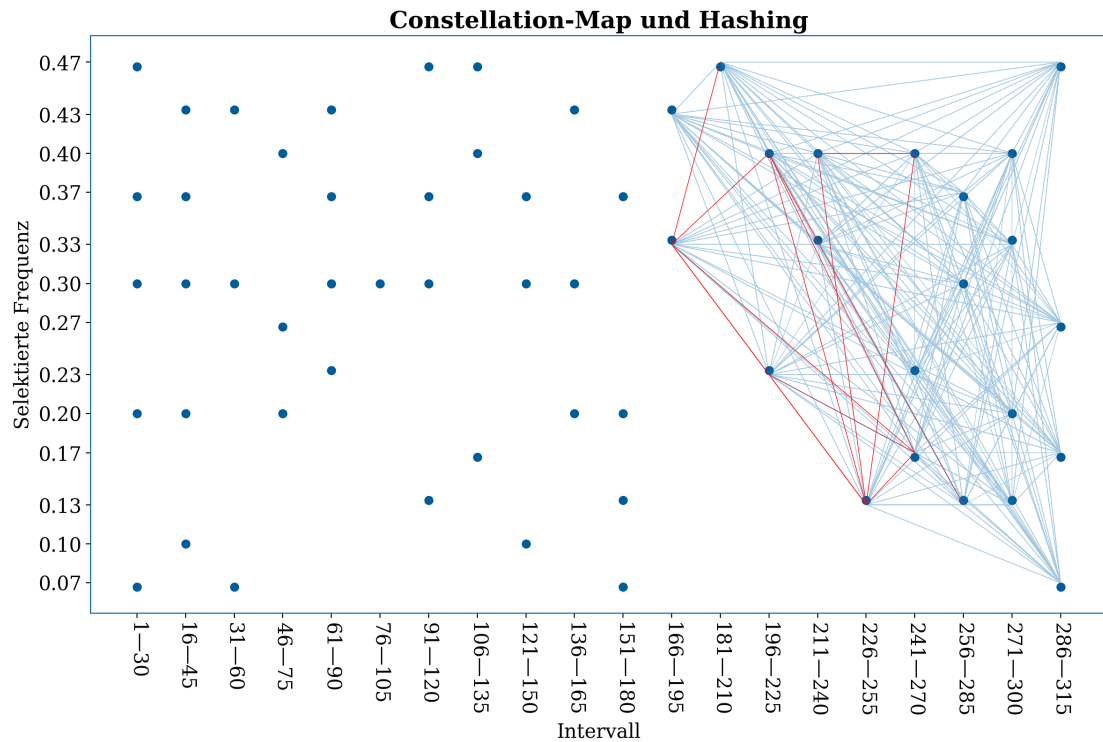


Abbildung 4: Die Punkte bilden die Constellation-Map. Die zur Übersicht nur rechts eingezeichneten Kanten repräsentieren die Hash-Bildung, wobei rote Kanten ignorierte Hashes darstellen, weil sie mehrfach auftauchen.

Die Hashfunktion ist bijektiv gestaltet, sodass aus einem Hash all seine Bestandteile, die für die Erstellung verwendet wurden, eindeutig abgeleitet werden können. Das hängt damit zusammen, dass diese Bestandteile auf Bit-Ebene hintereinandergereiht werden, nämlich nach folgendem Schema:

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

Abbildung 5: Schematischer Aufbau eines Hashes

Für 10 Kidera-Faktor reichen 4 Bit. 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 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 von prot-fin beliebig belegt werden.

Algorithmus 4 Erstellung der Datenbank

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                                      $\triangleright kf = Kidera\ Faktor$ 
4:     aa_vector  $\leftarrow$  get_aa_vector(sequence, kf)                 $\triangleright aa = Aminosäure$ 
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 ersten drei beschriebenen Algorithmen 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 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 4.0 von prot-fin ist in Python implementiert. Von daher wird für die Persistierung der Datenbank (`save_to_file`) zur Einfachheit das `pickle`-Modul verwendet, welches 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 Constellation-Maps von Treffer und Suchsequenz. Je höher der Rang eines Treffers, desto funktionsähnlicher sollte das entsprechende Protein sein. Die Ausgabe 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)
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

Vorbedingung *hashes* aus Algorithmus 3

Vorbedingung *database* aus Algorithmus 4

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 protein, offsets in matches_per_tp do
19:   score  $\leftarrow$  get_most_common_offset(offsets)
20:   match_protein_hashes  $\leftarrow$  database.get_hashes(protein)
21:   jsi  $\leftarrow$  get_jsi(hashes, match_protein_hashes)
22:   match_scores[protein]  $\leftarrow score \cdot jsi$ 
23: end for
```

Um den Score zu bestimmen, also die Ähnlichkeit der Constellation-Map 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 der Kante in der Eingabe-Map zur Kante der jeweiligen TP-Map, also wie weit die Eingabe-Map 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.

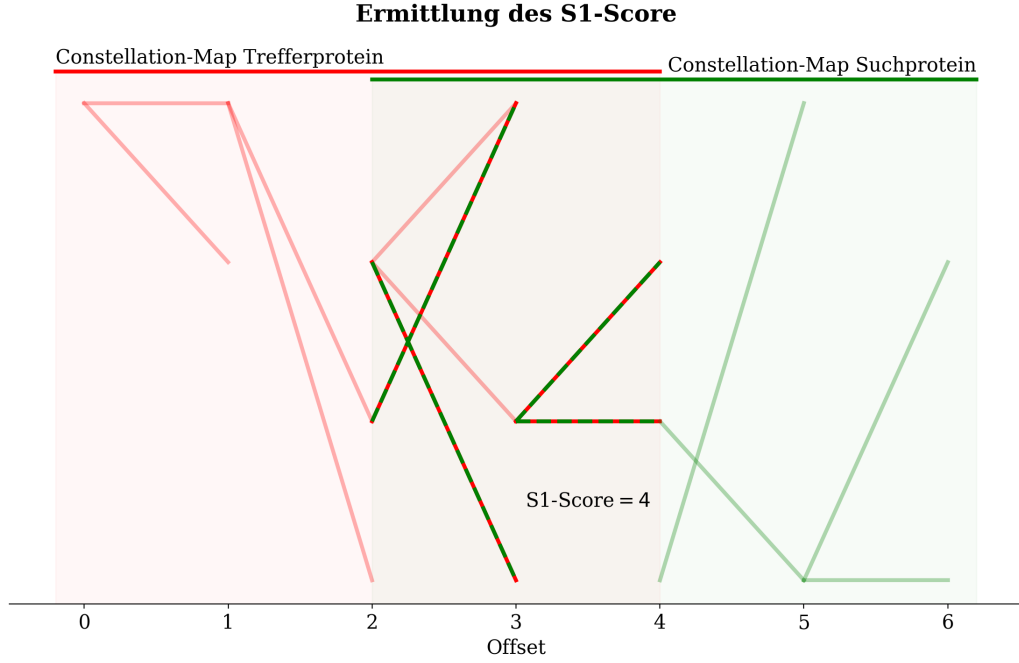


Abbildung 6: Die Constellation-Maps werden aneinander verschoben. Die maximale Überschneidung ist der Score.

Da es 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-Map übereinstimmen würden, sodass trotz Mis-Match der nahezu maximale Score erreicht werden würde. Bezogen auf das Beispiel zu S1 in Abbildung 6, 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 6 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. 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, die mit den Familienhashes übereinstimmen. Der Vorteil dieses Ansatzes ist, dass durch die Reduktion der betrachteten Hashes die Laufzeit verringert wird.

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:

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

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 als CSV-Tabelle beinhaltet aktuell diese Zusammenfassung zu Entwicklungszwecken. Sollte prot-fin ausgereift sein, wird die Ausgabe vermutlich die Treffer je Suchfamilie enthalten. In den Spalten wird 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 \rightarrow Anzahl der Hashes, die sich die Mitglieder teilen

3.2 Experiment 1: UniRef90 Sampling

Ein Bestandteil der Strukturanalyse in Algorithmus 2 ist die Selektion signifikanter Frequenzen zur Erstellung der Constellation-Map. 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.

Diesem Problem soll durch ein Sampling-Experiment abgeholfen werden. Darin werden 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, werden die Grenzquantile einer jeden Frequenz ermittelt, um signifikant seltene Signalstärken zu ermitteln. Folglich ist es möglich, für die Constellation-Map nur diejenigen Frequenzen zu behalten, welche in den Randzonen 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 wird daher anschließend insofern angepasst, dass bei der Frequenz-Selektion in Zeile 4 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.

Die zu klärende Frage ist, welche Grenzquantile verwendet werden müssen, um eine möglichst gute Wahl zu erzielen.

3.3 Experiment 2: Filter Hashes

Die Experimente zu vorigen Versionen von prot-fin haben zu sehr großen Datenbanken und langsamem Matching geführt. Unter der Annahme, dass es möglich ist, die Datenbank auf Eingabegröße zu reduzieren, sollte darin je Protein nur das Notwendigste gespeichert werden. In Experiment (Exp.) 1 wird dafür die Frequenzwahl angegangen, indem die Grenzwerte mit der Wahl der Maxima kombiniert werden. Alternativ soll in dem hiesigen Experiment die Größe durch einen Filter ermöglicht werden, wobei hier die Sampling-Grenzwerte als alleiniges Selektionskriterium dienen. Zum Filtern werden nach der Datenbankerstellung nur die Einträge der quantilmäßig seltensten Hashes behalten. Die entfernten Hashes an sich werden zudem in einer Blacklist gespeichert, um sie vor dem Matching auch aus den Hashes der Suchsequenzen zu entfernen, da ansonsten das Ergebnis verfälscht werden würde, aufgrund von fehlenden übereinstimmenden Hashes,

die eigentlich da gewesen wären.

Auch in diesem Experiment muss 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 gilt einen guten Kompromiss zu finden.

3.4 Experiment 3: Target-Zone

Die Target-Zone (TZ) beschreibt in Abbildung 4 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 5), also 4096, für Aminosäuresequenzen nahezu unbeschränkt ist, wird in diesem Experiment geprüft, wie viele Bit tatsächlich notwendig sind, um eine echte Einschränkung darzustellen und trotzdem ein gutes Matching zu gewährleisten. Umgesetzt wird dies in Zeile 6 von Algorithmus 3, wo durch die Nachfolger iteriert wird. Um die TZ einzubeziehen, werden die hard-coded $2^{12} - 1$ mit einer zusätzlichen Variable zu $2^{target_zone_bits} - 1$ erweitert. Als Selektionsmethode wird der Ansatz aus Exp. 1 verwendet.

3.5 Experiment 4: Selection-Method

Dieses letzte Experiment dieser Arbeit knüpft an Exp. 1 an. Dort ist die Intention, die ermittelten Grenzwerte mit einer anschließenden Maxima-Wahl zu kombinieren. Allerdings gibt es einige andere Ansätze für die Auswahl.

1. Es wird eine Vorselektion über die bisherige Methode der Maxima getätigt und dann auf die daraus resultierenden Frequenzen der Grenzwertfilter angewandt. Zusätzlich werden auch Minima einbezogen.
2. Es wird eine Vorselektion mittels Maxima durchgeführt, wobei nicht die Maxima in den rohen Amplituden der Frequenzen gesucht werden, sondern in deren absoluten Abweichungen vom Grenzwert. Da die Amplituden je Frequenz möglicherweise unterschiedlich streuen, werden diese Abweichungen zur Vergleichbarkeit vorher durch die jeweilige Standardabweichung der Amplituden geteilt.

Diesen beiden Methoden werden zusätzlich durch einen Parameter k erweitert, welcher bestimmt, dass die ersten k Frequenzen (entsprechend ihrer Reihenfolge wie im Beispiel in Abbildung 1) im Nachhinein aus der Auswahl entfernt werden. Das hat den Hintergrund, dass niedrige Frequenzen hier wenig über Periodizität aussagen, da in dieser Periode nur eine oder zwei Aminosäuren betrachtet werden. Da die Amplituden auf Summen basieren, kommt es dadurch leicht zu hohen Werten.

Um die Ansätze effizient zu testen, wird die Datenbankerstellung in Algorithmus 4 so angepasst, dass sie abgebrochen wird, sobald der Speicherbedarf das 6-fache der Eingabedaten übersteigt.

3.6 Durchführung

Die der Durchführung der Experimente läuft nach einem immergleichen Schema ab. Zuerst werden mit den MapMan-Proteinen aus Unterabschnitt 2.1 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 nämlich möglichst funktional divers zu halten, werden je Wurzelknoten der Bins sieben zufällige Sequenzen ausgewählt. Zumindest mit den Trainings-Proteine als Eingabe sollten bei korrekten Parametern erwartbar gute Ergebnisse erzielt werden, sodass hier Konzeptfehler sehr schnell auffallen. Für das Family-Matching werden alle die MapMan-Bins mit ihren Proteinen einbezogen, die durch mehr als einem Protein vertreten werden.

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. Im Gegensatz zu k aus Exp. 4 limitiert n_{peaks} nachträglich die Gesamtzahl gewählter Frequenzen, anstatt konkrete zu entfernen. Alle Parameter mit ihren getesteten Werten sind in Tabelle 2 aufgelistet:

Tabelle 2: Experiment-Parameter

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%	0.1%	0.01%	0.001%		1
			0.01%	0.001%		4
	5%					2, 3

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

4 Ergebnisse

4.1 UniRef90 Sampling

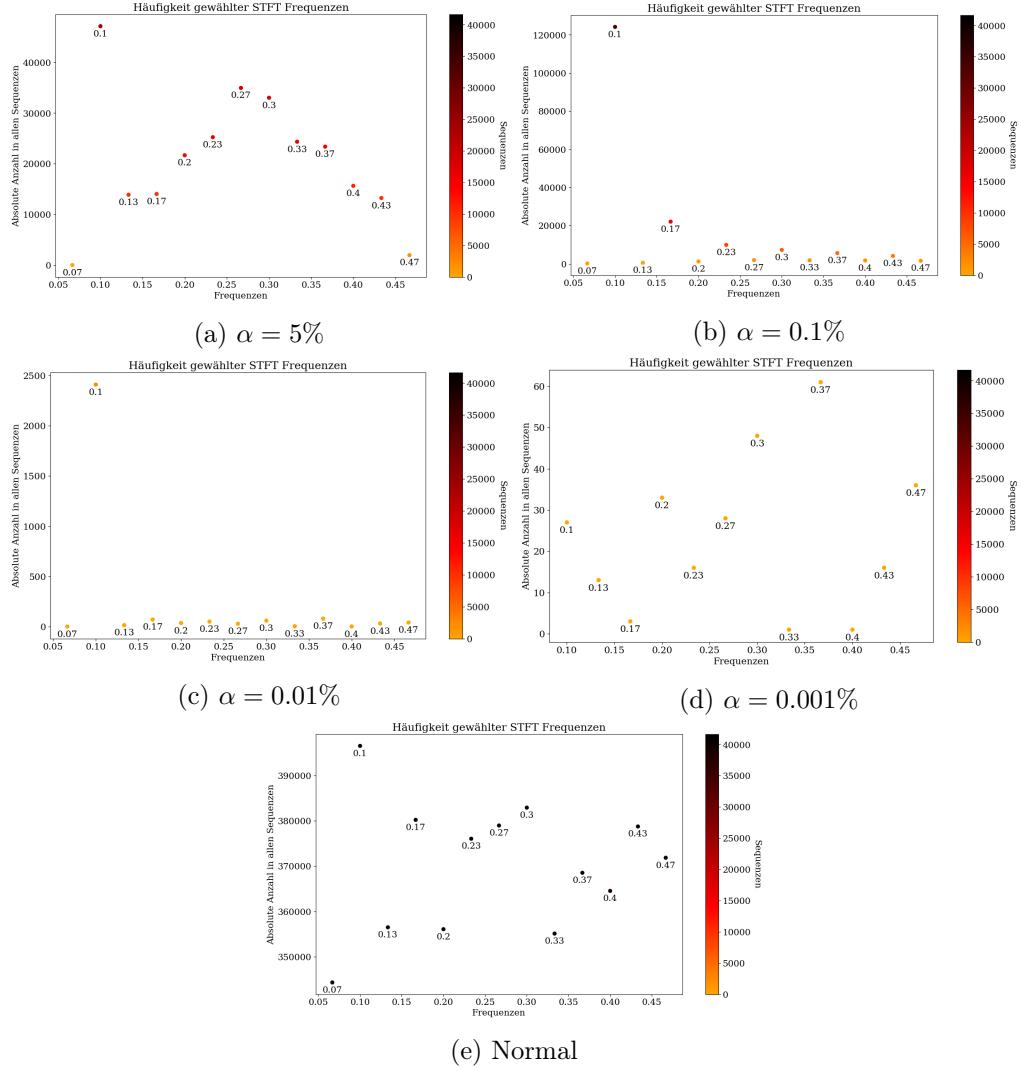


Abbildung 7: Häufigkeit gewählter Frequenzen über alle Trainings-Proteine: Die Häufigkeiten, wie oft eine Frequenz in einer Sequenz gewählt wird, werden über alle TP summiert (y-Achse). Die Farbe der Punkte gibt an, in wie vielen Sequenzen die jeweilige Frequenz vorkam.

Beim UniRef90 Sampling wurden die Amplituden-Grenzwerte für vier verschiedene Signifikanzniveaus ermittelt. Abbildung 7 stellt den folglich Einfluss auf die Frequenzselektion bezüglich Häufigkeit dar, exemplarisch für Hydrophobizität bei FG 30 mit Überlappung von 15 und $n_peaks = alle$.

In Abbildung 7e ist die Auswahl vor dem Sampling dargestellt. Die Häufigkeiten der Frequenzen liegen im Mittel etwa bei 360000 und sind in allen Frequenzen vertreten, was bei circa 40000 Proteinen bedeutet, dass in einer Sequenz jede Frequenz etwa 9-mal vorkommt. Frequenz 0.1, also eine Periode von jeder 10. Aminosäure, scheint zudem besonders häufig gewählt zu werden. Bis auf Abbildung 7d scheint das auch für die Frequenzselektion zu gelten, die auf dem Sampling basiert.

Die Häufigkeiten werden mit schrumpfendem α deutlich kleiner. So kommt eine Frequenz mit 5% Grenze nur etwa in jeder zweiten Sequenz vor, bei 0.1% in jeder 20., und bei noch strengeren Grenzen seltener als in jeder 200. Sequenz.

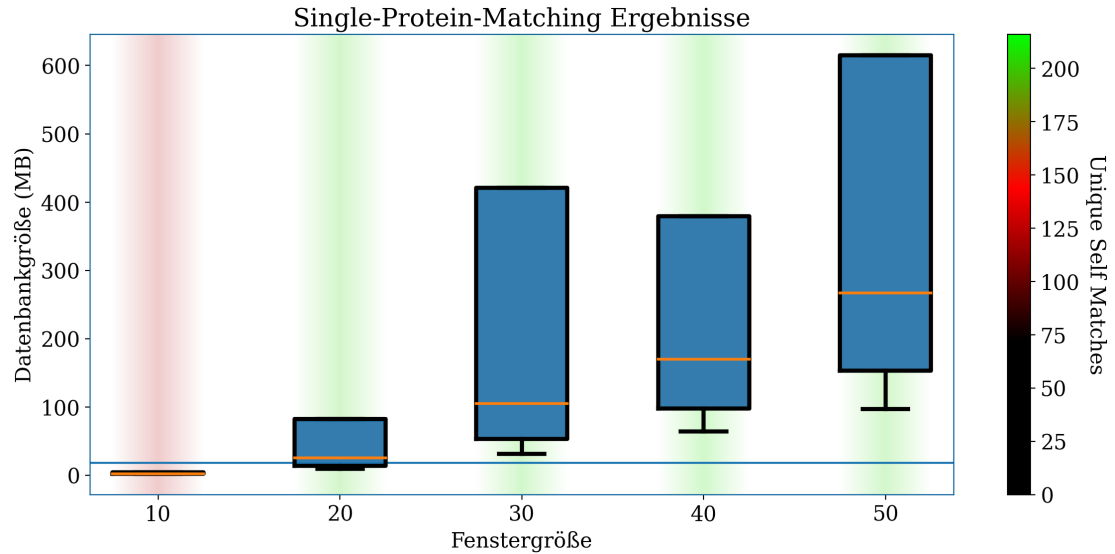


Abbildung 8: Matching-Ergebnisse für $\alpha = 5\%$. Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 Seite 12), sofern sie einen positiven Wert hat. Die farbige Fläche über und unter den Boxen stellt die Anzahl 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.

Ursprünglich war Exp. 1 so konzipiert, dass nur das ein 5%-alpha ermittelt wird. Da die Datenbanken aber weiterhin zu groß sind, wie in Abbildung 8, Unterabschnitt 4.2 und Unterabschnitt 4.3 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, scheitert aber bei der Identifikation und wurde daher nicht für alle Experimente betrachtet.

4.2 Filter Hashes

Aufgrund eines Eingabefehlers wurde der Parameter $Quantil = 0.8$ ausgelassen.

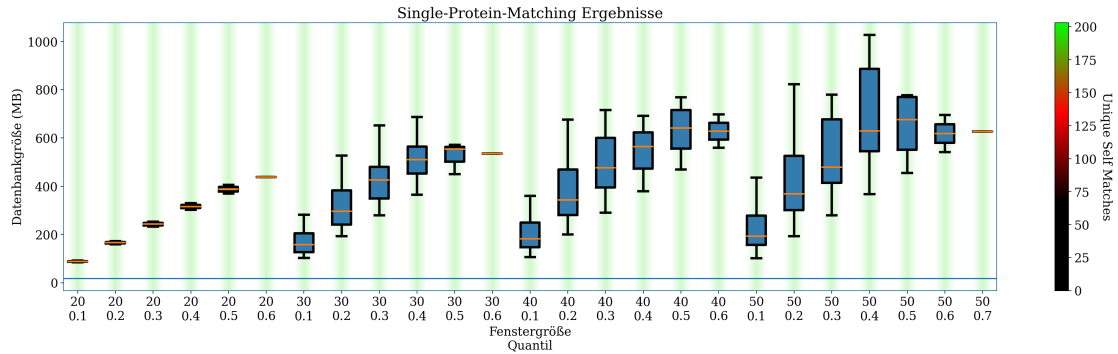


Abbildung 9: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 Seite 12), sofern sie einen positiven Wert hat. Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten.

Aufgrund zu langer Laufzeit wurde das Single-Protein-Matching vorzeitig abgebrochen, sodass die Daten der pro FG jeweils letzten Quantile nicht vollständig sind. Abbildung 9 zeigt für die Suchproteine für alle getesteten Parameter eine hohe Identifikationsrate mit hoher Schärfe. Mit sinkendem Quantil schrumpft auch die Datenbankgröße scheinbar linear. Dennoch ist keine davon unter dem Limit der Eingabe.

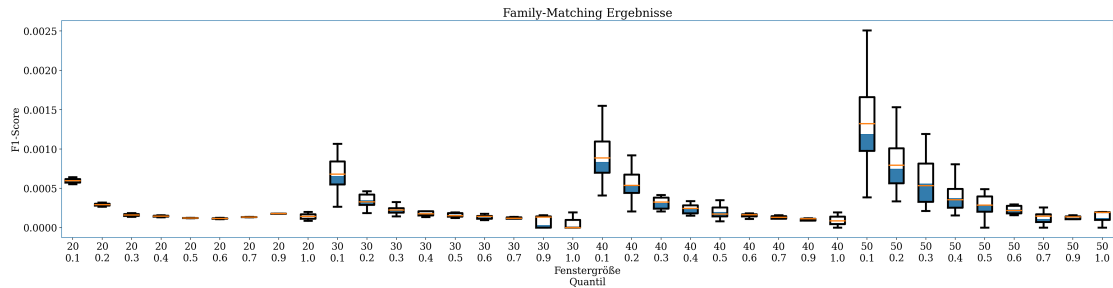


Abbildung 10: Die Boxen bilden die F-Scores für die verschiedenen Parameter aus Tabelle 2 ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 Seite 12), sofern sie einen positiven Wert hat.

Der Family-Matching-Ansatz in Abbildung 10 hat eine geringe Schärfe von unter 0.5, wobei sich für die getesteten Parameter zeigt, dass kleinere Quantile einen höheren F-

Score haben, der aber deutlich unter 1 liegt.

4.3 Target-Zone

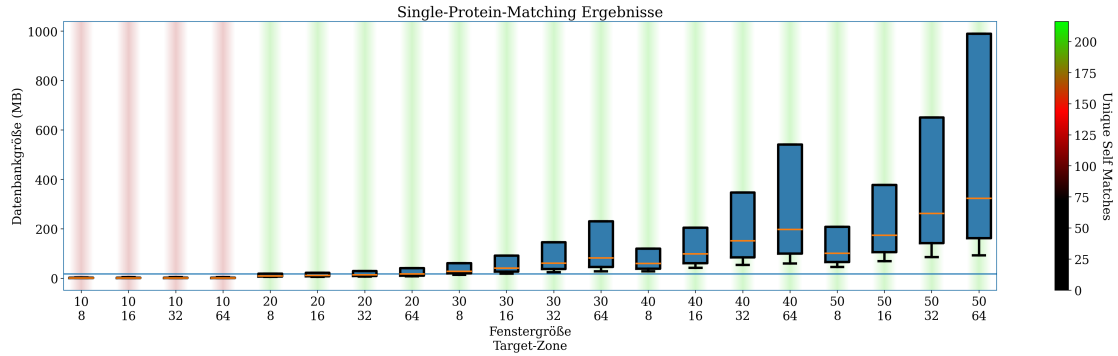


Abbildung 11: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 Seite 12), sofern sie einen positiven Wert hat. Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten.

Für das Testen verschiedener Target-Zones ist in Abbildung 11 eine Tendenz zu exponentiellem Wachstum der Datenbankgröße bei ansteigender TZ erkennbar. Wie beim Sampling in Abbildung 8 scheitert die Identifikation bei FG 10, im Gegensatz zu den anderen Größen, welche auch eine hohe Schärfe haben.

4.4 Selection-Method

Da das Exp. 3 in Unterabschnitt 4.3 aufgrund zu hoher Laufzeit vorzeitig abgebrochen wurde, wurde in dem hiesigen Experiment die Datenbankerstellung bei zu hoher Größe abgebrochen. Lediglich der Selektionsansatz, der die Maxima der absoluten Grenzwertabweichungen auswählt, hat zur Fertigstellung geführt, sodass das Matching nur für diese Datenbanken erfolgen konnte.

Die vorigen Ergebnisse haben veranlasst, wie in Unterabschnitt 4.1 erwähnt, das Signifikanzniveau von 5% zu verringern, sodass zusätzlich die Grenzwerte für $\alpha \in \{0.1, 0.01, 0.001\}$ ermittelt wurden. Um die Laufzeit dieses Experiments durch zu viele Parameterwerte nicht unnötig zu erhöhen, wurden vor der Durchführung die pro TP selektierten Frequenzen gezählt und die mittlere Anzahl Frequenzen pro Fenster ermittelt, um die zu testenden α -Werte zu begrenzen.

In Abbildung 12 wird deutlich, dass die Verringerung der Signifikanz die Anzahl der

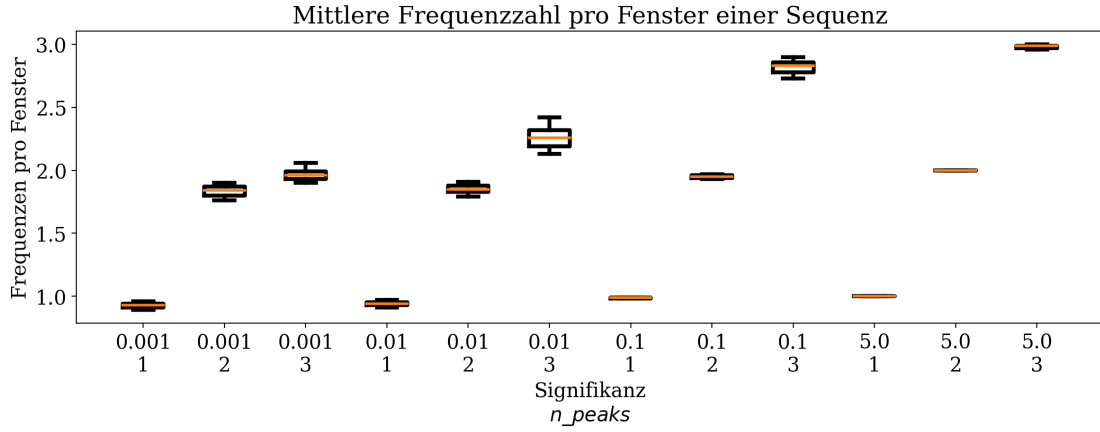


Abbildung 12: Mittlere Frequenzzahl pro Fenster einer Sequenz

gewählten Frequenzen reduziert. Für $\alpha = 5\%$ sind es deutlich zu viele Frequenzen, da die Anzahl aufgrund der geringen Streuung offenbar lediglich durch n_peaks limitiert wird. Wenn eine Datenbank nicht 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:

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

Selbst bei einer Frequenz pro Fenster wäre die Datenbank nach dieser Abschätzung doppelt so groß wie die Eingabe. Von daher sollte α die Frequenzwahl möglichst in Richtung einer Anzahl von 1 bringen, weshalb $\alpha = 0.1$ nicht verwendet wird, da die Streuung hier auf meistens 2 Frequenzen pro Fenster schließen lässt. Das Matching wird von daher für die beiden kleinsten α -Werte durchgeführt.

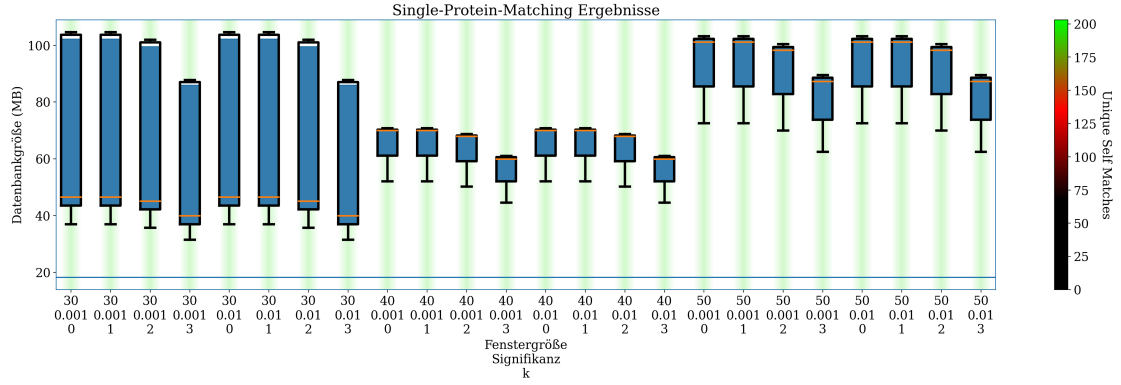


Abbildung 13: Die Boxen bilden die Datenbankgrößen für die verschiedenen Parameter aus Tabelle 2 ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 Seite 12), sofern sie einen positiven Wert hat. Die farbige Fläche über und unter den Boxen stellt die Anzahl dar, wie viele der Suchproteine als Treffer mit alleinigem besten Score identifiziert wurden (hier “Unique Self Matches”). Die blaue horizontale Linie kennzeichnet die Größe der Eingabedaten.

In Abbildung 13 zeigt sich für das Single-Protein-Matching, dass die Identifikation bei den getesteten Parametern mit hoher Schärfe erfolgreich funktionierte. Beim Wechsel von $k = 2$ zu $k = 3$ ist zudem ein kleiner Sprung nach unten zu erkennen.

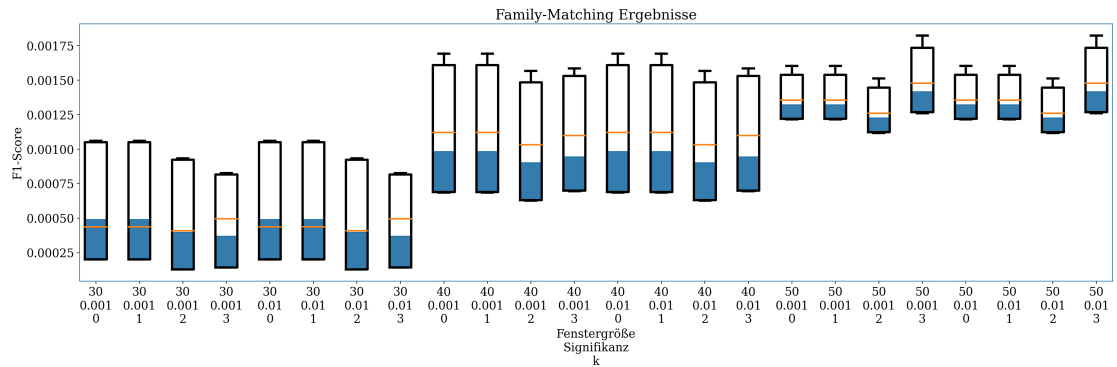


Abbildung 14: Die Boxen bilden die F-Scores für die verschiedenen Parameter aus Tabelle 2 ab. Die blaue Füllung repräsentiert die Schärfe (Gleichung 2 Seite 12), sofern sie einen positiven Wert hat.

Beim Family-Matching scheinen die höheren FGs ab 40 besser abzuschneiden. Für $k = 2$ sinkt der F-Score ein wenig. Die Schärfe liegt wie bei Abbildung 10 überall unter 0.5.

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 nun halbwegs erfolgreich bei prot-fin umgesetzt. Halbwegs deswegen, weil die Erkennung von Proteinen zwar funktioniert, dafür aber die Performanz hinsichtlich Speicherbedarf der Datenbanken und daraus resultierenden hohen Laufzeiten nicht gut ist. Außerdem fehlt eine robuste und zuverlässige Möglichkeit, verwandte Proteine zu identifizieren. Um diesen Mängeln entgegenzuwirken, wurden 4 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, 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 7 ist der Effekt auf die Wahl deutlich zu erkennen. Interessant hierbei ist, dass es für mehrere Signifikanzniveaus in jeder 10. Aminosäure ein periodisches Signal gibt. Das betrifft bei einem Fenster der Größe 30 zwar nur drei und ist vielleicht in den Trainings-Protein (TP) begründet, aber könnte möglicherweise auf ein konzeptuelles Problem hindeuten. Für zukünftige Ergebnisse sollte das im Hinterkopf behalten werden.

Bezüglich der Werte für $\alpha \leq 0.01\%$ ist es zweifelhaft, ob bei diesen niedrigen Häufigkeiten unter 100 wirklich alle TP abgedeckt werden können. Für $\alpha = 5\%$ sieht das Matching in Abbildung 8 jedenfalls noch in Ordnung aus. 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. Die Ergebnisse dort zeigen, dass auf diese Weise effektiv kleinere Datenbanken erzielt werden, wobei die Schärfe und Identifikationsrate scheinbar 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, sollte dies für die Ergebnisse in Abbildung 8 zu einer Verbesserung führen. Abbildung 10 betrachtend, scheint wohl ein zu behaltendes Quantil der Hashes von 10% geeignet zu sein. 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 auf jeden

Fall enthalten, sofern es Hashes gibt, die sie sich teilen. Die Auswertung der Ergebnisse des Family-Matchings sollte dahingehend erweitert werden, dass angegeben wird, wie viele Familien keine Hashes hatten und was darin die jeweilige Mindestzahl an Hashes ist, die ein Mitglied hat, damit die Darstellung der Ergebnisse weniger irreführend gestaltet werden kann. Ebenso ist es denkbar, dass das Verhältnis von mittlerer Hashanzahl in der Familie und der tatsächlichen Menge geteilter Hashes als Bewertungsmaß neben dem F-Score einfließt.

In Exp. 3 wird dieselbe Selektionsmethode wie in Exp. 1 verwendet. Es zeigt sich, dass eine Target-Zone (TZ) von 8, also einem Bedarf von 3 Bit, vollkommen ausreichend für die Identifikation ist, welche in ihrer Bewertung keine Einbußen hat und eine Datenbankgröße von medialen etwa 100 MB zur Folge hat.

Bei Exp. 4 hat sich offenbar nur eine Methode als tauglich erwiesen, nämlich die Frequenzen danach zu wählen, wie stark deren Amplituden von den gelernten Grenzwerten in Exp. 1 abweichen. Die Datenbankgrößen sind sehr gut, wenn man bedenkt, dass hier keine Hashes herausgefiltert wurden, sondern das lediglich über diese Selektion mit TZ 8 erzielt wurde. Der Parameter $k = 3$ scheint ganz geeignet zu sein, die Werte beim Single-Protein-Matching sind da am besten. Beim Family-Matching beeinflusst k scheinbar nicht allzu viel, wenn die Ergebnisse nicht irreführend sind, nur bei FG 50 gibt es offenbar einen erhöhten F-Score. Für die weitere Entwicklung sollte diese Selektionsmethode auf jeden Fall in Betracht gezogen werden. Bei Durchführungen, die weiterhin k -Werte austesten, reicht es, nur ein $k \in \{0, 1\}$ zu testen, da die Ergebnisse immer identisch sind. Dies liegt daran, dass die Frequenzen nach lokalen Maxima ausgewählt werden, was bedeutet, dass die Randfrequenzen niemals infrage kommen können, da ihr Extremverhalten nur von einer Seite betrachtet werden kann. Wenn also $k = 1$ gilt, wird eine Frequenz ignoriert, die sowieso niemals gewählt wird.

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:

$$\begin{aligned}
scores_trp &= \{S1(t) \cdot JSI(t) \mid t \in TrP\} \\
scores_fp &= \{S1(f) \cdot JSI(f) \mid f \in FP\} \\
dist_nicht_familie &= \frac{\max(scores_trp) - \max(scores_fp)}{\max(scores_trp)} \\
dist_familie &= \frac{\max(scores_trp) - \text{mean}(scores_trp \setminus \{\max(scores_trp)\})}{\max(scores_trp)} \\
Schärfe &= dist_nicht_familie \cdot (1 - dist_familie)
\end{aligned} \tag{4}$$

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

Den bisherigen Ergebnissen nach, hat prot-fin das Potential, eines Tages seinem Zweck gerecht zu werden. Aktuell ist dies zwar noch nicht der Fall, aber die Experimente zeigen, dass es Wege gibt, dem Ziel nahezukommen. Auch möglich ist, dass der aktuelle Ansatz einfach nicht funktioniert, weil die aus den Sequenzen erhaltenen Vektoren zu kurz sind oder ein anderer noch ungesehener Fall vorliegt, der die Verwandtschaftserkennung verhindert.

Andere Ansätze gibt es auf jeden Fall:

1. Der für mich anfangs intuitivste Weg war, die Sequenzen in echte Musik zu übersetzen, sodass SHAZAM direkt darauf angewandt werden könnte. Im Anhang (siehe Abschnitt 6) ist eine Beispielmusikdatei, die aus einer Aminosäuresequenz generiert wurde, wobei jede Aminosäure in einem Akkord aus KFs entsprach. Problem hierbei war ebenso die Datenbankgröße, da Musik komplexer als Text ist. Dennoch ist der Kern der Idee vielleicht trotzdem richtig, nur die Umsetzung noch nicht ideal.
2. Ebenso möglich wäre, anstelle ein echtes musikalisches Spektrum zu erstellen, stattdessen den aktuell erstellten Vektor, wie in ?? dargestellt, durch lineare Interpolation um weitere Punkte zu ergänzen. Mit anderen Worten, die Punkte in ?? werden verbunden. Auf diese Weise hätte die STFT wesentlich mehr Spielraum.
3. Auch denkbar wäre, mehrere KFs in den Vektor einzubeziehen. Allerdings hätte das möglicherweise den Nebeneffekt, dass beim Matching nicht KF-spezifisch gesucht wird. Denn prot-fin hätte theoretisch die Option, funktionelle Ähnlichkeit in Bezug auf spezielle physikalische Eigenschaften zu identifizieren.

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 12 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. Ansonsten 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 wäre ebenso eine Option für das Single-Protein-Matching. Zwar würde das ermitteln des S1-Score, siehe Abbildung 6, deutlich aufwändiger sein, aber möglicherweise wäre dieser Score dann gar nicht mehr notwendig. Wenn bei der Generierung der Hashes in Algorithmus 3 nicht die Position, 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 beim JSI ist, dass er positionsunspezifisch ist. Dieser alternative Score ist ihm zwar ähnlich, aber da hier wirklich alle Hashes einfließen, da die Kombination aller Hashes die Positionen in abstrakter Form beinhaltet, da die Hashes prinzipiell Kanten in einem fast vollständigen Graph sind, wie in Abbildung 4 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 zu erstellen, wobei die Performanz des Scorings davon unberührt bliebe.

Ein Problem, das bei den kleinen FG 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 mal mit Millionen von Proteinen verwendet werden soll, ist das viel zu unspezifisch. Sollte es bei diesen kleinen Fenstern bleiben, muss der Informationsgehalt eines Hashes deutlich erhöht werden. Hierfür könnten zum Beispiel statt Paaren von Punkten der Constellation-Map größere n-Tupel kombiniert werden, wobei hier die Gefahr zu hoher Spezifität besteht. Das oben beschriebene Einbeziehen aller Hash-Positionen wäre vielleicht auch eine Option für das Problem oder die Erweiterung der Vektoren.

Es gibt also noch viel Entwicklungsbedarf und -möglichkeiten für prot-fin. Die Identifikation einzelner Proteine funktioniert sehr gut, das Erkennen von funktioneller Ähnlichkeit hingegen weniger, wobei die Speicher-Komplexität des Trainings auch deutlich verringert werden muss. Doch wird ein Weg gefunden, das zu erreichen, so bietet der Algorithmus eine ganz neue Alternative für die sequenzbasierte Verwandtschaftsanalyse.

6 Anhang

...

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