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

prot-fin, ein toller Titel

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.

[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 Constellation-Map und Hashing: 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. 7
3 Ermittlung des S1-Score: Die Constellation-Maps werden aneinander verschoben. Die maximale Überschneidung ist der Score. 10

Tabellenverzeichnis

1 Kidera-Faktoren 5

Abkürzungsverzeichnis

STFT Short Time Fourier Transformation 1
TP Trainings-Protein 8
CSV Comma Separated Values 8
JSI Jaccard Similarity Index 10
TZ Target-Zone 13

Algorithmenverzeichnis

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

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 . . . 5
Hashing . . . 6
Erstellung der Datenbank . . . 8
Single-Protein-Matching . . . 8
Family-Matching . . . 11
3.2 Experiment 1: UniRef90 Sampling . . . 12
3.3 Experiment 2: Filter Hashes . . . 12
3.4 Experiment 3: Target-Zone . . . 13
3.5 Experiment 4: Selection-Method . . . 13

4 Ergebnisse 14

5 Diskussion 15

1 Einleitung

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

Diese Bachelorarbeit beinhaltet die Entwicklung von Version 0.4 des Projekts prot-fin[1] 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 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 Faktors 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 Kidera-Fakor-Werten, die den numerischen Vektor der Eingabe-Aminosäurekette darstellten.

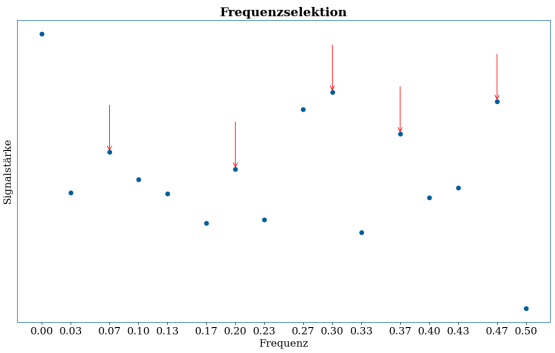

Abbildung 1: Spektralanalyse eines Fensters der STFT mit Markierung lokaler Maxima

Damit die Musikerkennung funktioniert,

[1] `https://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 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 allerdings eine schlechte Performanz hinsichtlich Speicher- und Laufzeitkomplexität. 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 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 → 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 $sequence \in \{\text{A-Z}, \text{'}\Psi\text{'}, \text{'}\Omega\text{'}, \text{'}\Phi\text{'}, \text{'}\zeta\text{'}, \text{'}\Pi\text{'}, \text{'}+\text{'}, \text{'-'}\}^*$
Vorbedingung $0 \leq kf \leq 10$
Nachbedingung $|aa\_vector| = |sequence|$
Nachbedingung $v \geq 0 \; \forall \; v \in aa\_vector$

```
1: aa_vector ← array()
2: for each aa in sequence do
3:     kf_value ← get_kf_value(aa, kf)
4:     min_kf_value ← get_min_kf_value()
5:     aa_vector.append(kf_value + 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 Faktor in einen numerischen Vektor übersetzt werden, wobei ein höherer absoluter Wert für mehr Relevanz des Faktors steht. Hätte man also die Beispielsequenz `EVKEFDGQGCFC`, wäre die Übersetzung für die Hydrophobizität die folgende:

E	V	K	E	F	D	G	Q	G	C	F	C
1.17	-0.4	1.7	1.17	-1.43	0.81	-0.16	1.1	-0.16	-1.05	-1.43	-1.05

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, weshalb der normalisierte Ergebnisvektor der Beispielsequenz so aussieht:

$$\begin{aligned} raw\_vec &= [1.17\ {-0.4}\ \ 1.7\ \ 1.17\ {-1.43}\ \ 0.81\ {-0.16}\ \ 1.1\ {-0.16}\ {-1.05}\ {-1.43}\ {-1.05}] \\ normalized\_vec &= raw\_vec + 2.33 \\ normalized\_vec &= [3.5\ \ 1.93\ \ 4.03\ \ 3.5\ \ 0.9\ \ 3.14\ \ 2.17\ \ 3.43\ \ 2.17\ \ 1.28\ \ 0.9\ \ 1.28] \end{aligned} \tag{1}$$

Algorithmus 2 Sammeln von Strukturdaten

Vorbedingung *aa_vector* aus Algorithmus 1
Nachbedingung *constellation_map* is an Array of Arrays of Floats

```
constellation_map ← array()
stft_result ← stft_transform(aa_vector)
for each window in stft_result do
    selected_frequencies ← select_maxima(window)
    constellation_map.append(selected_frequencies)
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 2 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, length(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 2 repräsentativ dargestellt, wobei rote Linien die ignorierten Kanten abbilden.

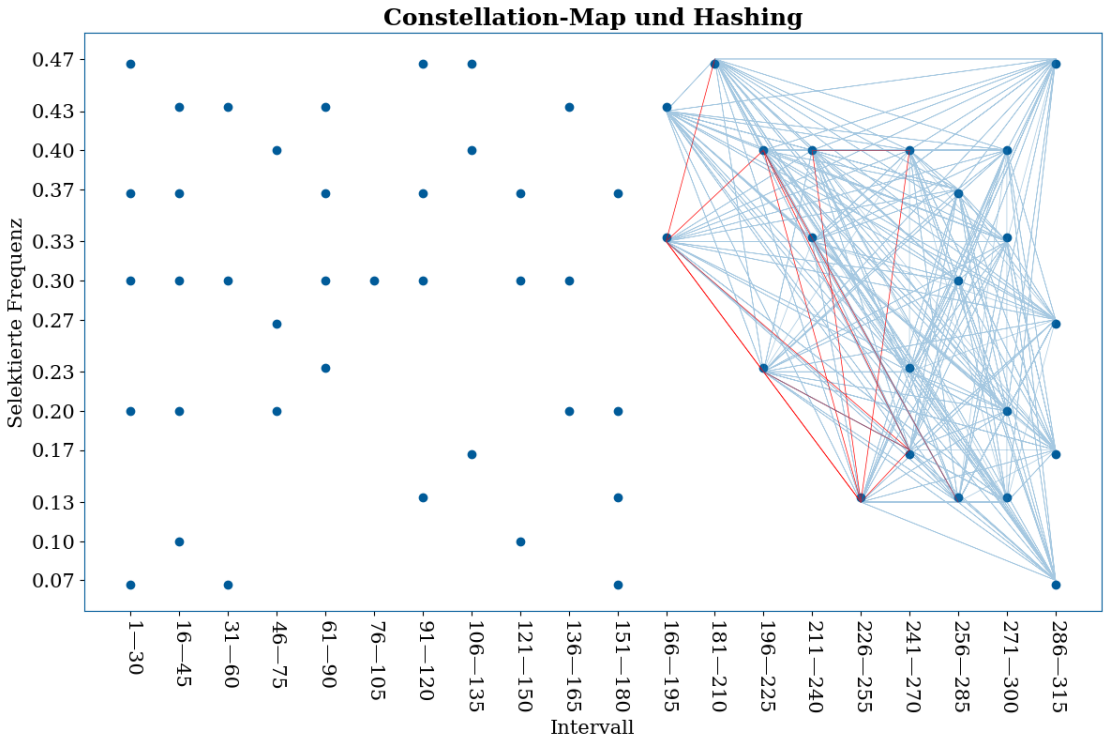

Abbildung 2: Constellation-Map und Hashing: Die Punkte bilden die Constellation-Map. Die zur Übersicht nur rechts eingezeichneten Kanten repräsentieren die Hash-Bildung, wobei rote Kanten ignorierte Hashes darstellen, weil sie mehrfach auftauchen.

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
unbelegt	Kidera Faktor	Kantenlänge/Fensterabstand	Frequenz Nachfolger	Frequenz Ursprung

Für 10 Faktoren 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 ← hashmap()
2: for each protein_id, sequence in fasta do
3:     for kf ← 0 to 10 do
4:         aa_vector ← get_aa_vector(sequence, kf)
5:         constellation_map ← get_constellation_map(aa_vector)
6:         hashes ← get_hashes(constellation_map, protein_id)
7:         for each hash in hashes do
8:             if hash not in database then
9:                 database[hash] ← 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.

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 → Rang
   2. Match_Protein_ID → Protein-ID des Treffers
   3. JSI → Jaccard Similarity Index (siehe Algorithmus 5)

4. Score → Score (Übereinstimmung der Constellation-Map)
5. Input_Protein_ID → Protein-ID der Suchsequenz
6. Input_Sequence_Length → Sequenzlänge der Suchsequenz
7. Input_Found_Hashes → 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 ← hashmap()
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] ← hashmap()
7:             end if
8:             offset ← tp_position − position
9:             if offset not in matches_per_tp[protein_id] then
10:                matches_per_tp[protein_id][offset] ← 0
11:            end if
12:            offset_count ← matches_per_tp[protein_id][offset]
13:            matches_per_tp[protein_id][offset] ← offset_count + 1
14:        end for
15:    end if
16: end for
17: match_scores ← hashmap()
18: for each protein, offsets in matches_per_tp do
19:    score ← get_most_common_offset(offsets)
20:    match_protein_hashes ← database.get_hashes(protein)
21:    jsi ← get_jsi(hashes, match_protein_hashes)
22:    match_scores[protein] ← score · 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.

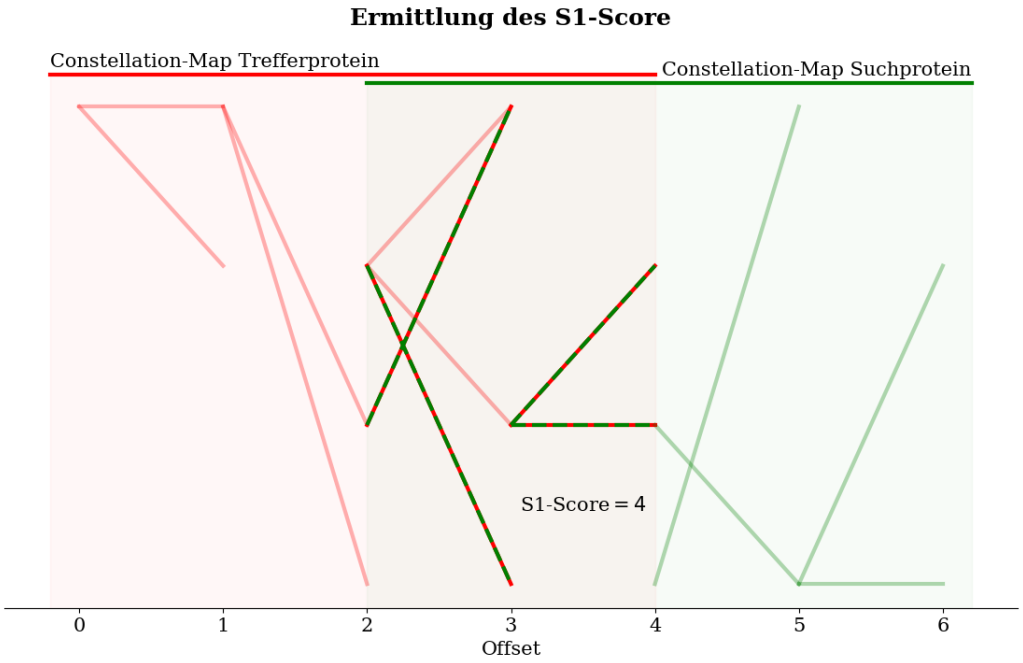

Abbildung 3: Ermittlung des S1-Score: Die Constellation-Maps werden aneinander verschoben. Die maximale Überschneidung ist der Score.

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 3, 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:

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

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 3 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:

$$
\begin{aligned}
t &= \texttt{max\_score}(TrP) \\
Sharpness &= \begin{cases} \frac{t - \texttt{max\_score}(FP)}{t}, & \text{if t} > 0 \\ \text{-1}, & \text{sonst} \end{cases} \\
Precision &= \frac{|TrP|}{|TrP| + |FP|} \\
Recall &= \frac{|TrP|}{|TrP| - member\_count} \\
F\_Score &= \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}
\end{aligned}
\tag{2}
$$

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 → 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 anstelle mittels Maxima über die über- oder unterschrittenen Grenzwerte identifiziert werden. Zudem wird einem Hash je Frequenz noch mit einem Bit die Information hinzugefügt, ob sie besonders stark oder schwach 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 1 wird dafür die Frequenzwahl angegangen. Alternativ soll in dem hiesigen Experiment die Größe durch einen Filter ermöglicht werden. Dazu werden nach der Datenbankerstellung nur die Einträge der quantilsmäßig 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 2 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, also 4096, für Aminosäuresequenzen 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 $2^{target\_zone} - 1$ erweitert.

3.5 Experiment 4: Selection-Method

Dieses letzte Experiment dieser Arbeit knüpft an Unterabschnitt 3.2 an. Dort ist die Intention, die ermittelten Grenzwerte als alleiniges Auswahlkriterium zu verwenden. Allerdings gibt es einige Ansätze, dies zu erweitern, um noch strenger zu wählen.

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.

4 Ergebnisse

...

5 Diskussion

...

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