Humboldt-Universität zu Berlin
Institut für Informatik

Lehrstuhl für Algorithmen und Komplexität

Diplomarbeit

Ein Approximationsalgorithmus zur Berechnung der Ähnlichkeit dreidimensionaler Punktmengen

Stefan Kirchner

12. September 2003

Gutachter: Dr. Stefan Hougardy
Prof. Dr. Anusch Taraz

Inhaltsverzeichnis

1 Einleitung

Diese Diplomarbeit entstand im Rahmen eines Projektes mit der Arbeitsgruppe Frömmel aus dem Institut für Biochemie des Universitätsklinikums Charité der Humboldt-Universität zu Berlin. Mit Ausnahme dieser Einleitung, in welcher der biologische Hintergrund kurz skizziert wird, werden jedoch keine weiteren biologischen Kenntnisse benötigt.
Die Arbeitsgruppe Frömmel verwaltet eine Datenbank DIP (Dictionary of Interfaces in Proteins), welche etwa 10^7 dreidimensionale Strukturen, sog. Patche, enthält. Patche sind Teile von Proteinen und können mit anderen Molekülen wechselwirken. In einem Protein können solche Regionen für gewisse Zellprozesse verantwortlich sein und sind daher für Biologen interessant. Eine weitere Datenbank enthält die dreidimensionale Struktur von Molekülen, die als Wirkstoffe für Medikamente in Deutschland zugelassen sind.
Dabei sind oft sowohl für erwünschte als auch unerwünschte Wechselwirkungen kleine Bereiche in einem Molekül verantwortlich. Daraus ergibt sich die Idee, durch den Austausch von gewissen Bereichen in einem Molekül einige Funktionsweisen zu erhalten und andere zu zerstören. Idealerweise sollte z.B. ein Medikament seine Hauptfunktion weiterhin ausüben, aber eine unangenehme Nebenwirkung gezielt ausgeschaltet werden. Bei Proteinen ist ähnliches denkbar, z.B. dass bestimmte unerwünschte Zellprozesse ausgeschaltet werden.
Die Hoffnung ist, dass dieses Vorhaben am ehesten funktioniert, wenn eine Struktur durch eine andere, geometrisch ähnliche Struktur ersetzt wird. Das führt zu der Frage, wie zwei Moleküle oder Teile von ihnen nach einer noch zu definierenden Ähnlichkeitsfunktion möglichst schnell auf Ähnlichkeit überprüft werden können.

Mit dieser Fragestellung befasst sich diese Arbeit, die sowohl theoretische als auch praktische Teile enthält und wie folgt gegliedert ist. In Kapitel 2 wird die Modellierung beschrieben und das Ähnlichkeitsmaß vorgestellt. In Kapitel 3 wird zunächst ein optimaler Algorithmus mit exponentieller Laufzeit entwickelt. Anschließend wird ein polynomielles Approximationsschema für das Problem vorgestellt und dessen Korrektheit bewiesen.
Da die Laufzeit dieser Approximation in der Praxis jedoch nicht akzeptabel ist, werden diejenigen Teile, die für die hohe Laufzeit verantwortlich sind, mit der Hoffnung modifiziert, in der Praxis die Laufzeit ohne größeren Qualitätsverlust deutlich zu reduzieren. Die näheren Details zu der Implementation sind in Kapitel 4 beschrieben. Um die Qualität des implementierten Algorithmus einschätzen zu können, wird er mit anderen Algorithmen verglichen. Die Ergebnisse finden sich in Kapitel 5 wieder. Kapitel 6 gibt einen Ausblick auf weiterführende Fragestellungen, die im Rahmen dieser Arbeit nicht beantwortet werden konnten.

2 Problembeschreibung

In diesem Kapitel sollen Definitionen und Begriffe eingeführt werden, die es erlauben, ein geeignetes Maß festzulegen, nach der die Ähnlichkeit zweier Moleküle in dieser Arbeit gemessen wird. Das Ähnlichkeitsmaß ist von der Arbeitsgruppe Frömmel aus dem Institut für Biochemie der Charité entwickelt worden und soll zwei Moleküle als ähnlich klassifizieren, wenn diese eine „ähnliche räumliche Struktur“ aufweisen.
Zu jedem gegebenen Molekül sind unter anderem bekannt:

1. die Koordinaten der Atome im dreidimensionalem Raum,
2. die Atomtypen (z.B. Sauerstoff, Kohlenstoff, Wasserstoff),
3. die Bindungen zwischen zwei Atomen.

Wir werden im Rahmen der vorliegenden Arbeit uns lediglich für die Koordinaten der Nichtwasserstoffatome interessieren und die Atomtypen und Bindungen ausblenden.
Seien $A_1, \ldots, A_m$ die dreidimensionalen Koordinaten aller m Nichtwasserstoffatome eines Moleküls. Diese Punkte können zu einer Matrix $A \in \mathbb{R}^{3\times m}$ zusammengefasst werden, wobei die i-te Spalte genau der Vektor A_i ist. Die Matrix A repräsentiert dann dieses Molekül. Es sei angemerkt, dass in dieser reduzierten Sichtweise Information verloren geht, da zwei verschiedene Moleküle die gleiche Matrix besitzen können.
Die Spalten einer Matrix werden im Folgenden auch Punkte und die Matrix selbst Punktmenge genannt. Um zwei Punkte aus verschiedenen Matrizen aufeinander abbilden zu können, dient zur Vorbereitung die folgende Definition.

Definition 1 *Seien $m, n \in \mathbb{N} \setminus \{0\}$ gegeben. Eine* Überlagerung *f ist eine Funktion mit den Eigenschaften:*

1. $f : \{1, \ldots, m\} \longrightarrow \{0, \ldots, n\}$
2. $\forall i, j \in \{1, \ldots, m\} : f(i) = f(j) \Rightarrow i = j \vee f(i) = 0$

Weiterhin sei $f^* := \{i | f(i) \neq 0\}$ die Menge, deren Elemente durch f nicht auf Null abgebildet werden. Eine Überlagerung $\tilde{f}$ ist Teilüberlagerung von f, falls gilt

$$\forall i \in \{1, \ldots, m\} : \tilde{f}(i) > 0 \Rightarrow \tilde{f}(i) = f(i).$$

Bemerkung: Sofern nicht anders geschrieben, werden im Folgenden mit A und B immer reelle Matrizen der Form $3 \times m$ bzw. $3 \times n$ und mit A_i bzw. B_i die i-te Spalte von A bzw. B gemeint sein.
Zu gegebenen A und B kann eine Überlagerung wie folgt interpretiert werden. Gilt $f(i) > 0$, so hat der Punkt A_i den Partnerpunkt $B_{f(i)}$, und A_i und

$B_{f(i)}$ heißen überlagert. Andernfalls hat der Punkt A_i keinen Partnerpunkt aus B. Beachte, dass f bijektiv ist, wenn der Definitionsbereich auf f^* und der Wertebereich auf $Bild(f^*)$ eingeschränkt wird. Daher kann jeder Punkt aus B mit höchstens einem Punkt aus A überlagert sein.

Definition 2 *Unter dem* mittleren quadratischen Abstand *(root mean squared distance) von A, B und einer Überlagerung f wird die Funktion*

$$rms(A, B; f) := \sqrt{\frac{\sum_{i \in f^*} ||A_i - B_{f(i)}||^2}{|f^*|}} \tag{1}$$

verstanden,[1] *wobei mit der Notation $||.||$ der euklidische Abstand bezeichnet sei.*

Da Moleküle im Raum beweglich sind, soll dieses in der Modellierung auch berücksichtigt werden. Dazu wird ein relativer Abstandsbegriff eingeführt, der es erlaubt, die Punktmengen derart zu bewegen, so dass bei gegebener Überlagerung der mittlere quadratische Abstand minimiert wird. Zunächst soll geklärt werden, was hierbei genau als Bewegung zu verstehen ist. Aus der linearen Algebra ist bekannt, dass die Menge der Rotationen $O^+(3)$ genau diejenigen Matrizen $R \in \mathbb{R}^{3\times 3}$ sind, deren transponierte Matrix R^T gleich ihrer Inversen (sog. orthogonale Matrizen) und deren Determinante zusätzlich Eins ist. Eine Translation ist ein beliebiger Vektor aus dem $\mathbb{R}^3$.

Definition 3 *Eine Abbildung $M : \mathbb{R}^3 \to \mathbb{R}^3$ ist eine* Bewegung*, falls es eine Rotationsmatrix $R \in \mathbb{R}^{3\times 3}$ und einen Translationsvektor $t \in \mathbb{R}^3$ gibt, so dass für alle $x \in \mathbb{R}^3$*

$$M(x) = Rx + t \tag{2}$$

gilt.

Mit diesem Bewegungsbegriff sollen insbesondere Spiegelungen ausgeschlossen werden. Die Bewegungen bilden eine Gruppe, die mit $AO^+(3)$ bezeichnet wird. Damit lässt sich jetzt folgender Abstandsbegriff zwischen A und B bei gegebener Überlagerung f einführen, der berücksichtigt, dass A und B bewegt werden dürfen:

$$rms^*(A, B; f) := \inf_{\substack{(R_A,t_A)\in AO^+(3)\\(R_B,t_B)\in AO^+(3)}} \sqrt{\frac{\sum_{i \in f^*} ||(R_A A_i + t_A) - (R_B B_{f(i)} + t_B)||^2}{|f^*|}} \tag{3}$$

Bemerkung: Eine Bewegung $M = (R, t)$ kann auf eine Punktmenge $B \in \mathbb{R}^{3\times n}$ erweitert werden, indem M auf jede einzelne Spalte von B angewandt wird. Dazu diene die Notation

$$M(B) := RB + t\,\mathbf{1_n}, \tag{4}$$

[1] Gilt $|f^*| = 0$, so soll $rms(A, B; f) := 0$ gelten.

wobei $\mathbf{1_n}$ ein Zeilenvektor mit n Einträgen ist, welche alle Eins sind.
Da die Bewegungen eine Gruppe bilden, ist rms^* invariant bezüglich Bewegungen von A und B. Es gilt also für alle Überlagerungen f und alle Bewegungen M_1 und M_2

$$rms^*(A, B; f) = rms^*(M_1(A), M_2(B); f). \tag{5}$$

Daher kann im Folgenden ohne Einschränkung davon ausgegangen werden, dass die Punktmenge A fest ist und nur B bewegt werden darf.
Zu einer gegebenen Überlagerung f kann dann die Anzahl der überlagerten Punkte und deren rms^* in Beziehung gesetzt werden. Dies soll durch die Funktion

$$score^*(A, B; f) := \frac{|f^*|}{\min(m, n)} \cdot e^{-rms^*(A,B;f)} \tag{6}$$

geschehen, wobei e die Eulersche Zahl[2] ist.
Das Optimierungsproblem, mit dem sich diese Arbeit beschäftigt, ist eine Überlagerung f zu finden, die $score^*(A, B; f)$ maximiert, also

$$\boxed{score^*(A, B) := \max_f score^*(A, B; f).} \tag{7}$$

Da es nur endlich viele Überlagerungen gibt, existiert das Maximum. Dieser Funktionswert soll zu gegebenen Punktmengen A und B die Ähnlichkeit messen. Je höher der Wert ist, der im Folgenden auch *Scorewert* genannt wird, umso ähnlicher gelten A und B. Beide Faktoren aus (6) nehmen für $|f^*| \neq 0$ Werte aus dem Intervall $(0, 1]$ an, daher hat der Scorewert denselben Wertebereich. Die beiden Faktoren verhalten sich dabei gegenläufig zueinander. Auf der einen Seite kann für den ersten Faktor der Wert Eins erreicht werden, indem alle Punkte aus A oder aus B überlagert werden. Je mehr Punkte aber überlagert sind, ein desto größerer mittlerer quadratischer Abstand muss in der Regel in Kauf genommen werden, womit der zweite Faktor kleiner wird. Auf der anderen Seite ist in der Regel der zweite Faktor größer, wenn nur wenige geeignete Punkte überlagert sind. Im Extremfall wird nur ein Punkt aus A gewählt, der mit Abstand Null mit einem beliebigen Punkt aus B überlagert ist. Sind wenige Punkte überlagert, wird zwangsläufig der erste Faktor klein sein. Emprische Beobachtungen zeigen, dass es im Allgemeinen günstig ist, eine „mittelgroße“ Menge zu wählen, deren mittlerer quadratischer Abstand nach geeigneter Bewegung von B relativ klein ist.
Hohe Scorewerte lassen sich daher nur erzielen, wenn beide Faktoren nahe Eins sind. In diesem Fall sind zum einen viele Punkte überlagert und zum anderen der mittlere quadratische Abstand relativ klein. Moleküle mit einem hohen Scorewert weisen daher eine ähnliche räumliche Struktur auf.

[2]Statt e^x wird im Folgenden auch $\exp(x)$ verwendet.

Die Scorefunktion ist symmetrisch in ihren Argumenten, d.h. es gilt

$$score^*(A, B) = score^*(B, A). \tag{8}$$

Deshalb soll im Folgenden o.B.d.A. immer $m \leq n$ gelten.

Lemma 4 *Es seien $A \in \mathbb{R}^{3\times m}$ und $B \in \mathbb{R}^{3\times n}$ gegeben. Für jede Überlagerung f gilt*

$$\exists M \in AO^+(3) : rms^*(A, B; f) = rms(A, M(B); f).$$

Beweis: Da für $||t|| \to \infty$ mit $t \in \mathbb{R}^3$ der Wert $rms(A, RB + t; f)$ mit beliebiger Rotation R gegen unendlich konvergiert, gibt es eine Translation t_0 mit der Eigenschaft

$$\forall R\, \forall ||t|| > ||t_0|| : rms(A, RB + t; f) > rms(A, B; f).$$

Da die Menge der Rotationen im $\mathbb{R}^{3\times 3}$ abgeschlossen ist, ist auch die Menge

$$\mathcal{M} := \{(R, t) |\, R \text{ ist Rotation und } t \in \mathbb{R}^3 \text{ mit } ||t|| \leq ||t_0||\}$$

in $\mathbb{R}^{3\times 3} \times \mathbb{R}^3$ abgeschlossen. Die Funktion $rms(A, RB + t; f)$ ist stetig in R und t und nimmt daher auf $\mathcal{M}$ ihr Minimum an. Damit gibt es eine Bewegung $M \in \mathcal{M}$ mit

$$rms^*(A, B; f) = rms(A, M(B); f).$$

□

M wird im Folgenden die zu A, B und f *optimale Bewegung* genannt. Insofern kann das Infimum in (3) durch das Minimum ersetzt werden.
Bemerkung: Die Scorefunktion ist im Allgemeinen nicht skalierungsinvariant. Zum einen gilt

$$\lim_{\lambda\to 0} score^*(\lambda A, \lambda B) = 1,$$

denn der Abstand zweier beliebiger Punkte aus A und B strebt für $\lambda \to 0$ gegen Null. Der obige Grenzwert ergibt sich, indem alle Punkte aus A überlagert werden. Betrachte nun den Wert

$$\lim_{\lambda\to\infty} score^*(\lambda A, \lambda B).$$

Zunächst gilt für eine Überlagerung f die Implikation

$$rms^*(A, B; f) > 0 \Longrightarrow \lim_{\lambda\to\infty} rms^*(\lambda A, \lambda B; f) = \infty,$$

womit dann $\lim_{\lambda\to\infty} score^*(\lambda A, \lambda B; f) = 0$ folgt.

Ist f mit $|f^*| < m$ eine Überlagerung mit $rms^*(A, B; f) = 0$, und für alle Überlagerungen $\tilde{f}^*$ mit $|\tilde{f}^*| > |f^*|$ gilt $rms^*(A, B; f) > 0$, so folgt

$$1 > \lim_{\lambda \to \infty} score^*(\lambda A, \lambda B) = \frac{|f^*|}{m} \geq \frac{1}{m}.$$

Also gilt $\lim_{\lambda \to \infty} score^*(\lambda A, \lambda B) = \lim_{\lambda \to 0} score^*(\lambda A, \lambda B)$ genau dann, wenn alle Punkte aus A mit Abstand Null mit Punkten aus B überlagert werden können.
Daher macht die Scorefunktion nur Sinn, wenn die Punkte in einer festen Einheit vorliegen. In den verwendeten Daten werden die Abstände in der Einheit Ångstrøm (Å) gemessen. 1 Ångstrøm sind 10^{-10}m, also der ungefähre Durchmesser eines Wasserstoffatoms.

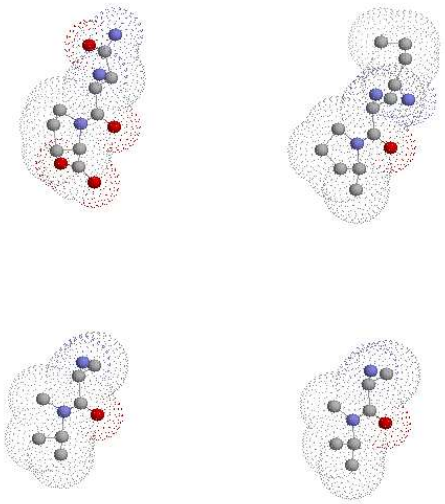

Abbildung 1: Oben links bzw. oben rechts sind zwei verschiedene Patche abgebildet. Darunter sind die jeweiligen überlagerten Atome aus der optimalen Überlagerung abgebildet.

Ein Beispiel für eine Überlagerung ist in der Abbildung 1 zu erkennen. In der oberen Abbildung sind zwei Patche abgebildet, die aus je sechzehn Atomen bestehen. Direkt darunter sind die überlagerten Teilmengen der jeweiligen Patche aus der optimalen Überlagerung abgebildet. Die verschiedenen Graustufen stellen lediglich verschiedene Atomtypen dar.

3 Algorithmen zur Berechnung der Scorefunktion

Nachdem das Überlagerungsproblem im vorangegangenen Abschnitt eingeführt wurde, sollen in diesem Abschnitt zwei Algorithmen zur Lösung obiger Problemstellung vorgestellt werden. Der eine Algorithmus liefert eine exakte Lösung mit exponentieller Laufzeit. Der andere Algorithmus approximiert die Scorefunktion in polynomieller Laufzeit beliebig genau.

3.1 Ein exakter Überlagerungsalgorithmus

Der Beweis von Lemma 4 ist ein reiner Existenzbeweis. Für eine algorithmische Lösung stellt sich daher die Frage, wie bei einer gegebenen Überlagerung die optimale Bewegung effizient berechnet werden kann. Dieses Problem wird auch als *absolute orientation problem* bezeichnet und kann durch ein Verfahren gelöst werden, das in [14] beschrieben und in dem hier vorgestellten Algorithmus implementiert ist.

3.1.1 Berechnung einer optimalen Bewegung bei bekannter Überlagerung

Das folgende Lemma und das anschließende Theorem gestatten es, zu einer gegebenen Überlagerung die optimale Bewegung algorithmisch zu bestimmen.

Lemma 5 (Singulärwertzerlegung, [12]) *Jede $k \times k$ Matrix $K \in \mathbb{R}^{k\times k}$ lässt sich in eine Form UDV zerlegen. Dabei sind U und V orthogonale Matrizen und D eine eindeutige Diagonalmatrix* $\mathrm{diag}(d_1, \ldots, d_k)$ *mit nichtnegativen Einträgen und mit $d_i \leq d_j$ für $i < j$. Die Diagonaleinträge von D werden* Singulärwerte *von K genannt.*

Theorem 1 ([14]) *Seien $A, B \in \mathbb{R}^{3\times m}$ mit der Identität f als Überlagerung gegeben. Weiter sei $s_A := \frac{1}{m}\sum_{i=1}^{m} A_i$ der Schwerpunkt von A. s_B sei analog definiert. Dann gelten die beiden folgenden Aussagen.*

1. *In jeder optimalen Bewegung werden die Schwerpunkte aufeinander abgebildet, es gilt also für den Translationsvektor $t = s_A - Rs_B$.*
2. *Sei UDV eine Singulärwertzerlegung von AB^T. Für die Rotation R in einer optimalen Bewegung gilt $R = USV$, wobei S wie folgt definiert ist:*

$$S = \begin{cases} \mathrm{diag}(1,1,+1) & \text{falls } \det(AB^T) \geq 0 \\ \mathrm{diag}(1,1,-1) & \text{falls } \det(AB^T) < 0 \end{cases} . \tag{9}$$

Damit kann jetzt bei gegebenen A, B und einer Überlagerung f die optimale Bewegung bestimmt und $score^*(A, B; f)$ berechnet werden, indem nur

die überlagerten Punkte betrachtet werden. Der Algorithmus 1 berechnet $score^*(A, B; f)$ und gibt eine optimale Bewegung zurück.

Algorithmus 1: Umeyama

Input: $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}, f, m \leq n$

Output: eine zu f optimale Bewegung (R, t) und $score^*(A, B; f)$

1 Seien $A_{j_1}, \ldots, A_{j_{|f^*|}}$ die überlagerten Punkte aus A.
Definiere $A'_i := A_{j_i}$ und $B'_i := B_{f(j_i)}$ mit $1 \leq i \leq |f^*|$.

2 Bestimme Singulärwertzerlegung $UDV = A'B'^T$.

3 Bestimme Schwerpunkt $s_{A'}$ und $s_{B'}$ von A' bzw. B'.

4 $S = \begin{cases} \text{diag}(1, 1, +1) & \text{falls } \det(A'B'^T) \geq 0 \\ \text{diag}(1, 1, -1) & \text{falls } \det(A'B'^T) < 0 \end{cases}$

5 $R = USV; \quad t = s_{A'} - Rs_{B'};$

6 **return** $\left(\frac{|f^*|}{m} \cdot \exp\left(-\sqrt{\frac{\sum_{i=1}^{|f^*|} ||A'_i - (RB'_i + t)||^2}{|f^*|}} \right), (R, t) \right)$

Wegen $\det(D) \geq 0$ und $\det(A'B'^T) = \det(U)\det(D)\det(V)$ bietet es sich an, die Vorzeichen der Determinanten von U und V zu bestimmen, um daraus S zu erhalten. Da der Betrag der Determinante von orthogonalen Matrizen Eins ist, sollte diese Methode numerisch stabiler sein, als die Determinante von $A'B'^T$ direkt zu bestimmen. Letzteres führt zu Problemen, wenn die Determinante vom Betrag sehr klein ist, da dann evtl. für S die ‚falsche' Diagonalmatrix gewählt wird.

Bemerkung: Das Lemma 5 kann auf endlichdimensionale Vektorräume über $\mathbb{R}$ erweitert werden, wobei in dieser Arbeit es nur für den dreidimensionalen Raum benötigt wird. Eine stabile Berechnung der Singulärwerte ist nicht trivial. Daher wird für die Singulärwertzerlegung ein Algorithmus von Forsythe verwendet, der in den Numerical Recipes [10] als C-Code angegeben ist.

Wenn die Singulärwerte alle verschieden sind, so ist die Bewegung eindeutig. Kommen hingegen Singulärwerte doppelt vor, so können mehrere optimale Bewegungen existieren, da die orthogonalen Matrizen U und V dann nicht eindeutig sind [8].

3.1.2 Enumeration von Überlagerungen

Mit dem Algorithmus 1 kann das Überlagerungsproblem als ein endliches kombinatorisches Optimierungsproblem aufgefasst werden. Zu gegebenen Punktmengen A und B gibt es eine endliche Anzahl an Überlagerungen, wobei jeder Überlagerung f der Wert $score^*(A, B; f)$ zugeordnet wird. Folglich

kann die Scorefunktion exakt berechnet werden,[3] indem alle Überlagerungen enumeriert werden und jeweils für jede einzelne Überlagerung der Algorithmus 1 aufgerufen wird. Der größte gefundene Wert wird am Ende ausgegeben und ist dann gerade $score^*(A, B)$.
Dieser Ansatz ist jedoch nur für relativ kleine Mengen praktikabel, da die Anzahl an Überlagerungen bei größeren Mengen stark anwächst. Die genaue Anzahl lässt sich mit Hilfe elementarer Kombinatorik leicht bestimmen. Seien $A \in \mathbb{R}^{3\times m}$ und $B \in \mathbb{R}^{3\times n}$ gegeben. Sind genau k Punktpaare überlagert, so können unabhängig k-elementige Teilmengen aus A und aus B gewählt und beliebig bijektiv aufeinander abgebildet werden. Indem über alle k mit $1 \leq k \leq m \leq n$ summiert wird, ergeben sich dann genau

$$\sum_{k=1}^{m} \binom{m}{k}\binom{n}{k} k! \tag{10}$$

verschiedene Überlagerungen, bei denen mindestens ein Atom überlagert ist. Einen Überblick für die Anzahl an Überlagerungen mit $m = n$ gibt die Tabelle 1 wieder.

m	$\sum_{k=1}^{m} \binom{m}{k}^2 k!$	m	$\sum_{k=1}^{m} \binom{m}{k}^2 k!$
1	1	11	3405357681
2	6	12	53334454416
3	33	13	896324308633
4	208	14	16083557845278
5	1545	15	306827170866105
6	13326	16	6199668952527616
7	130921	17	132240988644215841
8	1441728	18	2968971263911288998
9	17572113	19	69974827707903049153
10	234662230	20	1727194482044146637520

Tabelle 1: Anzahl verschiedener Überlagerungen mit $m = n$

Um alle Überlagerungen zu enumerieren, bietet es sich an, einen gewichteten Suchbaum $T_{m,n}$ zu verwenden, der die Eigenschaft hat, dass jeder Pfad von der Wurzel zu den Blättern eineindeutig einer Überlagerung entspricht. Dabei sei $T_{i,n}$ wie folgt induktiv über i definiert.

- $T_{i,n} = (\{p_0\}, \emptyset)$ mit Wurzelknoten p_0 und $i = 0$.
- $T_{i,n}$ mit $i \geq 1$ entsteht aus $T_{i-1,n}$ wie folgt.

[3] Der Rotationsmatrix und damit auch die Scorefunktion lassen sich im dreidimensionalen Raum durch algebraische Gleichungen darstellen.

> Für jedes Blatt v aus $T_{i-1,n}$ und für jedes $c \in \{1, \ldots, n\}$ gilt: Sofern es keine Kante mit Kantengewicht c auf dem Pfad von der Wurzel nach v gibt, füge einen neuen Knoten w_c und die Kante $\{v, w_c\}$ mit Gewicht c hinzu. Zusätzlich füge für jedes alte Blatt v einen neuen Knoten w_0 und die Kante $\{v, w_0\}$ mit Gewicht 0 hinzu.

Jeder Pfad $P = [p_0, \ldots, p_m]$ in $T_{m,n}$ von der Wurzel p_0 bis zu einem beliebigen Blatt p_m entspricht eineindeutig einer Überlagerung f, indem das Kantengewicht der Kante $\{p_{i-1}, p_i\}$ von P genau als $f(i)$ interpretiert wird. Um alle Blätter von $T_{m,n}$ zu besuchen und nicht den gesamten Baum im Speicher zu halten, bietet sich eine Tiefensuche an, bei dem im Wesentlichen nur die Pfadgewichte von der Wurzel bis zum aktuellen Knoten gespeichert werden müssen.
Bei jedem Blatt wird Algorithmus 1 mit der korrespondierenden Überlagerung aufgerufen. Der zurückgegebene Wert wird mit dem bisher besten Scorewert verglichen, welcher dann gegebenenfalls aktualisiert wird.

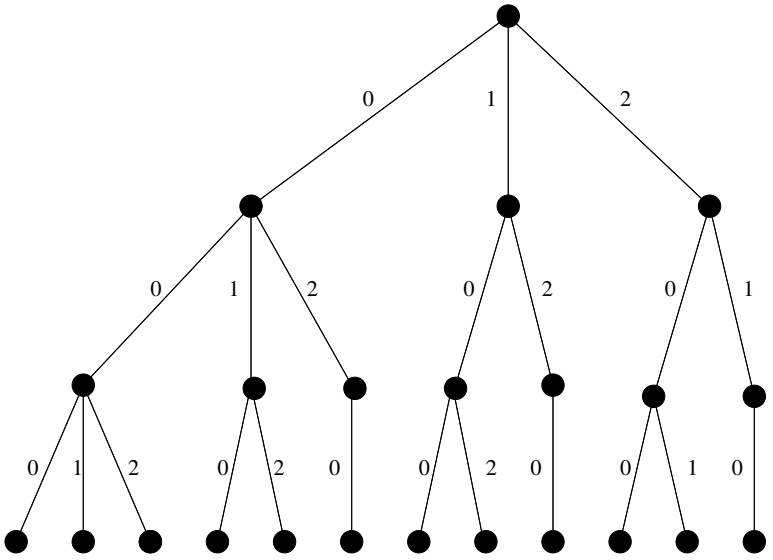

Abbildung 2: Überlagerungsbaum $T_{m,n}$ mit $m = 3$ und $n = 2$

Um $T_{10,10}$ vollständig zu enumerieren und an den Blättern Algorithmus 1 aufzurufen, werden etwa 45 Minuten benötigt.[4]
Mit einem Verfahren, das im Allgemeinen mit Branch-and-Cut bezeichnet wird, ist eine deutliche Verbesserung möglich. Sei $[p_0, \ldots, p_k]$ ein Pfad von der Wurzel p_0 zu einem Knoten p_k, wobei jede Kante $\{p_{i-1}, p_i\}$ mit $1 \leq i \leq k$ das Gewicht c_i habe. Für eine Überlagerung f gilt $f(i) = c_i$ mit $1 \leq i \leq k$ genau dann, wenn der zu f korrespondierende Pfad von der Wurzel zu einem Blatt den Knoten p_k enthält.

[4] Alle Zeitangaben beziehen sich in dieser Arbeit auf einen Athlonprozessor mit 1.3GHz.

$\tilde{f}$ sei die Überlagerung

$$\tilde{f}(i) = \begin{cases} c_i & \text{falls } 1 \leq i \leq k \\ 0 & \text{sonst} \end{cases} .$$

Damit ist $\tilde{f}$ eine Teilüberlagerung jeder Überlagerung, die obigen Bedingungen genügt. Weiterhin seien M' und M Bewegungen mit $rms^*(A, B; \tilde{f}) = rms(A, M'(B); \tilde{f})$ bzw. $rms^*(A, B; f) = rms(A, M(B); f)$. Es ergibt sich nun folgende Abschätzung

$$\begin{aligned} rms^*(A, B; f) &= \sqrt{\frac{\sum_{i \in f^*} ||A_i - M(B_{f(i)})||^2}{|f^*|}} \\ &\geq \sqrt{\frac{\sum_{i \in f^* \cap \{1,\ldots,k\}} ||A_i - M(B_{f(i)})||^2}{|f^*|}} \\ &\geq \sqrt{\frac{\sum_{i \in \tilde{f}^*} ||A_i - M'(B_{\tilde{f}(i)})||^2}{|f^* \cap \{1, \ldots, k\}| + |f^* \cap \{k+1, \ldots, m\}|}} \\ &\geq \sqrt{\frac{\sum_{i \in \tilde{f}^*} ||A_i - M'(B_{\tilde{f}(i)})||^2}{|\tilde{f}^*| + m - k}}, \end{aligned} \tag{11}$$

woraus die Ungleichung

$$score^*(A, B; f) \leq \frac{|\tilde{f}^*| + m - k}{m} \cdot \exp\left(-\sqrt{\frac{\sum_{i \in \tilde{f}^*} ||A_i - M'(B_{\tilde{f}(i)})||^2}{|\tilde{f}^*| + m - k}} \right) \tag{12}$$

folgt. Der rechte Term ist eine obere Schranke für $score^*(A, B; f)$ und kann effizient berechnet werden, wenn der Algorithmus sich im Knoten p_k befindet. Ist die obere Schranke kleiner als der bisher beste gefundene Scorewert, so kann beim Knoten p_k der rekursive Abstieg abgebrochen werden.
Mit dieser Schranke terminiert der Algorithmus bei zwei zehnelementigen Punktmengen in etwa zwei Sekunden, was eine deutliche Verbesserung gegenüber den 45 Minuten bei vollständiger Enumeration darstellt. Immerhin können mit diesem Verfahren zwei Punktmengen jeweils der Größe 16 in wenigen Minuten optimal überlagert werden. Bei größeren Punktmengen steigt die Laufzeit rapide an, so dass im Allgemeinen jeweils 20 Punkte nicht in akzeptabler Zeit optimal überlagert werden können.
Um den Suchbaum klein zu halten, sollte also möglichst früh erkannt werden, dass eine angefangene Teilüberlagerung nicht zu einer Überlagerung erweitert werden kann, die einen größeren Scorewert als den besten bislang bekannten liefert.
Dazu sind zwei Ansätze verfolgt worden. Zum einen sind zu einer gegebenen Teilüberlagerung zusätzliche obere Schranken für den Scorewert entwickelt

worden, die den Suchbaum ein wenig reduziert haben. Der zweite Ansatz bestand darin, die Enumerationsreihenfolgen zu ändern. Diese Technik kann in der Praxis bei Branch-and-Cut-Algorithmen sehr effizient sein. Ein Beispiel ist der Algorithmus von Brown zum Färben von Graphen [3]. Es erwies sich als vorteilhaft, die Punkte aus A in einem Preprozessing geeignet zu sortieren, womit die Laufzeit sich um einen Faktor von etwa zwei bis drei verkürzen ließ.
Je höher der bereits bekannte Scorewert ist, umso besser schneidet die Schranke den Baum ab. Dem exakten Überlagerungsalgorithmus kann daher am Anfang ein Parameter p mitgegeben werden, so dass bei jedem Knoten im Baum der rekursive Abstieg abgebrochen wird, wenn die rechte Seite von (12) kleiner als das Maximum von p und dem besten bisher gefundenen Scorewert ist.
Nachdem der Algorithmus terminiert, gibt es zwei Möglichkeiten. Gilt $score^*(A, B) < p$, dann hat der Algorithmus keine optimale Überlagerung gefunden, liefert dafür aber ein Zertifikat, dass $score^*(A, B) < p$ gilt. Die andere Möglichkeit ist $score^*(A, B) \geq p$. Dann hat der Algorithmus den optimalen Scorewert gefunden.
Ein approximatives Verfahren, das im Laufe dieser Arbeit vorgestellt wird, liefert für $score^*(A, B)$ eine gute untere Schranke und kann als Preprozessing benutzt werden, um ein geeignetes p zu wählen. Damit ist sichergestellt, dass der optimale Scorewert gefunden wird, wenn der exakte Überlagerungsalgorithmus anschließend mit p aufgerufen wird.
Empirische Untersuchungen mit $p \gtrapprox 0.7$ zeigen, dass bei Molekülen mit 20 bis 25 Atomen die Schranke den Enumerationsbaum so stark reduziert, dass der Algorithmus nach etwa einer Minute terminiert. Dieses Vorgehen ist in Kapitel 5 benutzt worden, um den optimalen Scorewert zweier sehr ähnlicher Moleküle zu bestimmen.
Weiter deuten die Untersuchungen darauf hin, dass mit dieser Methode der optimale Scorewert von zwei sehr ähnlichen Molekülen mit einer grösseren Anzahl an Atomen schneller bestimmt werden kann, als bei zwei recht unterschiedlichen Molekülen, die aus weniger Atomen bestehen.

3.2 Ein Approximationsalgorithmus zur Berechnung der Scorefunktion

Ausgangspunkt für einen exakten Überlagerungsalgorithmus war Theorem 1, welches ein konstruktives Verfahren beschreibt, um zu einer gegebenen Überlagerung eine optimale Bewegung zu erhalten. Im Folgenden betrachten wir die umgekehrte Fragestellung:

> Gegeben sei eine Bewegung mit Rotationsmatrix R und Translationsvektor t. Gibt es einen polynomiellen Algorithmus, der bezüglich dieser Bewegung eine optimale Überlagerung f findet,

also die Funktion

$$score(A, RB + t; f) := \frac{|f^*|}{m} \cdot e^{-rms(A,RB+t;f)} \tag{13}$$

maximiert?

Definition 6 *Mit $score(A, B)$ sei der maximale Score bezeichnet, der bei festen Matrizen A und B erreicht werden kann, also*

$$score(A, B) := \max_f score(A, B; f).$$

Der nachfolgende Abschnitt zeigt, dass $score(A, B)$ in polynomieller Zeit berechnet werden kann. Anschließend wird gezeigt, dass es ausreicht, polynomiell viele Bewegungen zu betrachten, um die Scorefunktion beliebig genau approximieren zu können.

3.2.1 Berechnung einer optimalen Überlagerung bei bekannter Bewegung

Wir bilden zu gegebenen $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}$ und einer Bewegung $(R, t) \in AO^+(3)$ einen vollständigen bipartiten gewichteten Graphen $K^w_{m,n}$ mit den Knotenpartitionen $\{A_1, \ldots, A_m\}$ und $\{B_1, \ldots, B_n\}$ und einer Kantengewichtsfunktion w, für die gilt

$$\forall\, i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\} : w(\{A_i, B_j\}) := ||A_i - (RB_j + t)||^2.$$

Offensichtlich ist $K^w_{m,n}$ eindeutig und kann in $\mathcal{O}(mn)$ konstruiert werden.

Definition 7 *Sei ein Graph $G = (V, E)$ gegeben. Ein* Matching *$M \subseteq E$ ist eine Kantenmenge, bei der keine zwei Kanten zueinander inzident sind. Die Knoten, die zu keiner Kante aus M inzident sind, heißen* unüberdeckt*, alle anderen Knoten heißen* überdeckt*.*

Korollar 8 *Seien $A, B, (R, t)$ und $K^w_{m,n}$ gegeben. Jede Überlagerung f kann eineindeutig als Matching M_f mit*

$$M_f := \bigcup_{i\in f^*} \{A_i, B_{f(i)}\} \tag{14}$$

in $K^w_{m,n}$ aufgefasst werden.

Für das Gewicht $w(M_f)$ eines Matchings M_f gilt

$$\begin{aligned} w(M_f) &:= \sum_{e\in M_f} w(e) \\ &= \sum_{i\in f^*} w(\{A_i, B_{f(i)}\}) \\ &= \sum_{i\in f^*} ||A_i - (RB_{f(i)} + t)||^2. \end{aligned} \tag{15}$$

Lemma 9 *Seien A, B und eine Bewegung (R,t) gegeben. Sei f eine Überlagerung mit $score(A, RB+t) = score(A, RB+t; f)$. Dann gilt:*

1. *M_f ist ein Matching, das unter allen Matchings in $K^w_{m,n}$ mit genau $|f^*|$ Kanten minimales Gewicht hat.*
2. *Ist $w(M_{\tilde{f}}) = w(M_f)$ und $|\tilde{f}^*| = |f^*|$ dann folgt $score(A, RB+t; \tilde{f}) = score(A, RB+t; f)$.*

Beweis: Zum Beweis der ersten Aussage betrachten wir die Kontraposition. Angenommen es sei $M_{\tilde{f}}$ ein Matching mit $w(M_{\tilde{f}}) < w(M_f)$ und $|\tilde{f}^*| = |f^*|$. Dann gilt

$$\begin{aligned}
score(A, RB+t; \tilde{f}) &= \frac{|\tilde{f}^*|}{m} \cdot \exp\left(-\sqrt{\frac{\sum_{i\in\tilde{f}^*} ||A_i - (RB_{\tilde{f}(i)} + t)||^2}{|\tilde{f}^*|}}\right) \\
&\overset{(15)}{=} \frac{|\tilde{f}^*|}{m} \cdot \exp\left(-\sqrt{\frac{w(M_{\tilde{f}})}{|\tilde{f}^*|}}\right) \\
&> \frac{|f^*|}{m} \cdot \exp\left(-\sqrt{\frac{w(M_f)}{|f^*|}}\right) \\
&\overset{(15)}{=} \frac{|f^*|}{m} \cdot \exp\left(-\sqrt{\frac{\sum_{i\in f^*} ||A_i - (RB_{f(i)} + t)||^2}{|f^*|}}\right) \\
&= score(A, RB+t; f), \qquad (16)
\end{aligned}$$

also insbesondere $score(A, RB+t) \neq score(A, RB+t; f)$.
Die zweite Aussage folgt durch analoge Rechnung, indem „>“ durch „=“ ersetzt wird. □

Damit ist das Problem, zu einer gegebenen Bewegung eine optimale Überlagerung zu bestimmen, auf ein Matchingproblem im gewichteten bipartiten Graphen $K^w_{m,n}$ zurückgeführt worden. Es muss lediglich die Menge $\{M_1, \ldots, M_m\}$ berechnet werden, wobei M_i ein Matching mit genau i Kanten ist, das unter allen Matchings mit genau i Kanten minimales Gewicht hat. Anschließend wird der Index i mit

$$\max_i \left\{ \frac{i}{m} \cdot \exp\left(-\sqrt{\frac{w(M_i)}{i}}\right) \right\} \qquad (17)$$

bestimmt. Für die zum Matching M_i korrespondierende Überlagerung f gilt dann $score(A, RB+t) = score(A, RB+t; f)$.
Im Folgenden soll beschrieben werden, wie $\{M_1, \ldots, M_m\}$ effizient berechnet werden kann. Dazu müssen zunächst ein paar Begriffe eingeführt werden.

Definition 10 *Seien ein Graph $G = (V, E, w)$ mit $w : E \to \mathbb{R}$ und ein Matching $M \subseteq E$ gegeben. Ein Pfad p in G heißt* alternierend *bezüglich M, falls er abwechselnd Kanten aus M und aus $E \setminus M$ benutzt. p heißt* M-augmentierend*, falls er alternierend ist und dessen Anfangs- und Endknoten unüberdeckt sind. Das Gewicht $w(p)$ ist definiert als*

$$w(p) := \sum_{e \in E(p) \setminus M} w(e) - \sum_{e \in E(p) \cap M} w(e). \tag{18}$$

Diese Definition gestattet es, den folgenden Satz zu formulieren.

Satz 11 *Sei G ein kantengewichteter Graph und M_k ein Matching mit genau k Kanten in G, das unter allen Matchings mit genau k Kanten minimales Gewicht hat. Sei p_0 ein M_k-augmentierender Pfad mit minimalem Gewicht. Dann ist die symmetrische Differenz $M_k \triangle p_0$ ein Matching in G mit genau $k+1$ Kanten, das unter allen Matchings mit genau $k+1$ Kanten minimales Gewicht hat.*

Beweis: Es ist offensichtlich, dass $M_k \triangle p_0$ ein Matching mit genau $k+1$ Kanten ist. Es bleibt zu zeigen, dass es minimales Gewicht hat. Betrachte dazu ein beliebiges Matching M' mit genau $k+1$ Kanten, das unter allen Matchings mit genau $k+1$ Kanten minimales Gewicht hat. $M_k \triangle M'$ hat Maximalgrad zwei und zerfällt daher disjunkt in die Mengen

- P der M_k-augmentierenden Pfade in $M_k \triangle M'$,
- Q der M'-augmentierenden Pfade in $M_k \triangle M'$ und
- R der Pfade und Kreise gerader Längen in $M_k \triangle M'$.

Angenommen es gilt $w(r \cap M_k) < w(r \cap M')$ mit $r \in R$. Dann ist $M' \triangle r$ ein Matching mit genau $k+1$ Kanten, für das $w(M' \triangle r) < w(M')$ gilt, im Widerspruch zur Minimalität von M'. Analog erhält man für $w(r \cap M_k) > w(r \cap M')$ mit $M_k \triangle r$ ein Matching mit genau k Kanten, für das $w(M_k \triangle r) < w(M_k)$ gilt, im Widerspruch zur Minimalität von M_k. Also gilt für alle $r \in R : w(r \cap M_k) = w(r \cap M')$.
Weiterhin gilt $|P| - |Q| = 1$. Dies führt zur folgenden Fallunterscheidung:
Fall 1: $|P| = 1$. Dann gibt es genau einen augmentierenden Pfad p, und wegen $w(r \cap M_k) = w(r \cap M')$ für alle $r \in R$ gilt $w(M') = w(M_k \triangle p)$. Damit gilt $w(M_k \triangle p) \leq w(M_k \triangle p_0)$, weil $w(M')$ minimal ist. Andererseits gilt $w(M_k \triangle p_0) \leq w(M_k \triangle p)$, da p_0 ein augmentierender Pfad in G mit minimalen Gewicht ist. Also gilt $w(M_k \triangle p_0) = w(M_k \triangle p)$ und damit $w(M_k \triangle p_0) = w(M')$.
Fall 2: $|P| > 1$. Seien $p \in P$ und $q \in Q$ in $M_k \triangle M'$ augmentierende Pfade.
a) Angenommen $w(p) > w(q)$.
Dann ist $M' \triangle (p \cup q)$ ein Matching mit genau $k+1$ Kanten mit Gewicht

$w(M') + w(q) - w(p) < w(M')$, im Widerspruch zur Minimalität von M'.
b) Angenommen $w(p) < w(q)$.
Dann ist $M_k \triangle (p \cup q)$ ein Matching mit genau k Kanten mit Gewicht $w(M') + w(p) - w(q) < w(M_k)$, im Widerspruch zur Minimalität von M_k.
c) Also haben in $M_k \triangle M'$ alle M_k- und M'-augmentierenden Pfade dasselbe Gewicht c. Daraus folgt

$$\begin{aligned} w(M') - w(M_k) &= \sum_{r \in R} \big(w(M' \cap r) - w(M_k \cap r)\big) + \\ &\quad + \sum_{p \in P} (w(M' \cap p) - w(M_k \cap p)) + \\ &\quad + \sum_{q \in Q} (w(M' \cap q) - w(M_k \cap q)) \\ &= 0 + \sum_{p \in P} w(p) - \sum_{q \in Q} w(q) \\ &= |P|\, c - |Q|\, c \\ &= (|Q| + 1)\, c - |Q|\, c \\ &= c. \end{aligned} \tag{19}$$

Für ein beliebiges $p \in P$ gilt $w(M_k \triangle p) = w(M_k) + w(p) = w(M_k) + c$ und wegen (19) folgt $w(M_k \triangle p) = w(M')$. Analog zum Fall 1 gilt dann $w(M_k \triangle p_0) = w(M')$. □

Satz 11 gilt für allgemeine Graphen. In bipartiten Graphen können augmentierende Pfade mit minimalem Gewicht durch eine geschickte Anwendung des Algorithmus von Dijkstra [4] gefunden werden [6, 9, 13], wie im Folgenden erläutert wird.
Sei $G = (V_1 \cup V_2, E, w)$ ein bipartiter gewichteter Graph und $M \subseteq E$ ein Matching in G, das minimales Gewicht unter allen Matchings mit $|M|$ Kanten hat. V_1^u, V_2^u seien die überdeckten Knoten von V_1 bzw. V_2. G wird in einen gerichteten Graphen $\vec{G}_M$ überführt, indem alle Kanten aus $E \setminus M$ in $\vec{G}_M$ gerichtete Kanten sind, die von V_1 nach V_2 verlaufen und ihr Gewicht behalten. Alle Matchingkanten $e \in M$ verlaufen in $\vec{G}_M$ von V_2 nach V_1 und erhalten das Gewicht $-w(e)$.
Zusätzlich werden in $\vec{G}_M$ zwei Knoten s und t eingefügt. Es verlaufen von s zu allen Knoten aus $V_1 \setminus V_1^u$ gerichtete Kanten mit Gewicht 0, und von allen Knoten aus $V_2 \setminus V_2^u$ verlaufen gerichtete Kanten nach t mit Gewicht 0.
Formal besteht $\vec{G}_M$ also aus der Knotenmenge

$$V' := \{s, t\} \cup V_1 \cup V_2 \text{ mit } \{s, t\} \cap \{V_1 \cup V_2\} = \emptyset \tag{20}$$

und der Kantenmenge

$$\vec{E} := \bigcup_{x \in V_1 \setminus V_1^u} (s, x) \cup \bigcup_{y \in V_2 \setminus V_2^u} (y, t) \cup \bigcup_{\substack{x \in V_1, y \in V_2, \\ \{x,y\} \in E \setminus M}} (x, y) \cup \bigcup_{\substack{x \in V_1, y \in V_2, \\ \{x,y\} \in M}} (y, x) \tag{21}$$

mit den Gewichten

$$w'((x,y)) := \begin{cases} 0 & \text{falls } x = s \text{ oder } y = t \\ w(\{x,y\}) & \text{falls } \{x,y\} \in E \setminus M \text{ mit } x \in V_1 \\ -w(\{x,y\}) & \text{falls } \{x,y\} \in M \text{ mit } x \in V_2 \end{cases} . \quad (22)$$

Ein M-augmentierender Pfad p mit minimalem Gewicht in G entspricht somit in $\vec{G}_M$ eineindeutig einem st-Pfad p_{st} mit minimalem Gewicht.
Der Algorithmus von Dijkstra kann auf $\vec{G}_M$ nicht angewendet werden, um p_{st} zu finden, da Kanten mit negativen Gewicht vorkommen können. Im Folgenden werden wir daher die Kantengewichte von $\vec{G}_M$ so verändern, dass alle Kantengewichte nichtnegativ sind und st-Pfade mit minimalem Gewicht in $\vec{G}_M$ im modifizierten Graphen minimal bleiben und umgekehrt.

Satz 12 *Sei $G = (V_1 \cup V_2, E, w)$ gegeben. Sofern ein Matching in G der Kardinalität k existiert, gibt es in G ein Matching M_k mit genau k Kanten, das minimales Gewicht hat, und einen Graphen $\vec{G}^+_{M_k} := (V', \vec{E}, w^+)$ mit*

- $\forall e \in \vec{E} : w^+(e) \geq 0$,
- *jeder Pfad mit minimalen Gewicht in $\vec{G}_{M_k} = (V', \vec{E}, w')$ ist minimal in $\vec{G}^+_{M_k}$ und umgekehrt.*

Beweis: Der Beweis wird induktiv über die Kardinalität von M_k geführt.
IA) Ist $k = 0$, so gilt $M_k = \emptyset$. Sei i das geringste Gewicht einer Kante in $\vec{G}_{M_0}$. Dann erfüllt für alle Kanten $(u,v) \in \vec{E}$ die Funktion

$$w^+((u,v)) := \begin{cases} 0 & \text{falls } u = s \vee v = t \\ w'((u,v)) + i & \text{sonst} \end{cases} \quad (23)$$

die angegebenen Bedingungen.
IS) Nach Induktionsvoraussetzung existieren $M_k, \vec{G}_{M_k}$ und $\vec{G}^+_{M_k}$. Wenn es in G ein Matching der Kardinalität $k+1$ gibt, dann existiert nach Satz 11 ein M_k-augmentierender Pfad p mit minimalen Gewicht und in $\vec{G}^+_{M_k}$ der korrespondierende st-Pfad p_{st}, so dass $M_k \triangle p$ minimales Gewicht unter allen Matchings in G mit genau $k+1$ Kanten hat.
Betrachte eine Funktion F, die jedem Knoten $u \in V'$ das Gewicht eines su-Pfades mit minimalem Gewicht in $\vec{G}^+_M$ zuordnet. Führe einen Graphen $\vec{G}^{++}_{M_k}$ ein, der aus $\vec{G}^+_{M_k}$ entsteht, indem alle Kantengewichte durch

$$w^{++}((u,v)) := w^+((u,v)) + F(u) - F(v) \quad (24)$$

ersetzt werden. Dadurch ändert sich das Gewicht jedes st-Pfades um $F(s) - F(t)$. Somit sind st-Pfade in $\vec{G}^+_{M_k}$ minimal genau dann, wenn sie es in $\vec{G}^{++}_{M_k}$ sind. Wegen $F(v) \leq w^+((u,v)) + F(u)$ sind alle Kanten in $\vec{G}^{++}_{M_k}$ nichtnegativ.

Insbesondere haben die Kanten aus p_{st} in $\vec{G}^{++}_{M_k}$ Gewicht 0. Betrachte den Graphen $\vec{G}^{+}_{M_{k+1}}$, der entsteht, indem in $\vec{G}^{++}_{M_k}$ die Kanten aus p_{st} umgedreht und die Anfangs- und Endkante von p_{st} gelöscht werden.
Alle Kanten haben nichtnegatives Gewicht, und jeder minimale st-Pfad in $\vec{G}^{+}_{M_{k+1}}$ ist minimal in $\vec{G}_{M_{k+1}}$ und umgekehrt. Weiterhin gilt $\vec{G}_{M_{k+1}} = \vec{G}_{M_k \triangle p}$, womit der Induktionsschritt vollzogen ist. □

Der Beweis liefert ein konstruktives Verfahren, um für $0 \leq k \leq m$ inkrementell Matchings mit genau k Kanten und minimalem Gewicht zu berechnen, sofern diese existieren. Um in $\vec{G}^{+}_{M_k}$ einen minimalen st-Pfad p_{st} und F zu berechnen, kann nun der Algorithmus von Dijkstra verwendet werden,[5] da alle Kantengewichte nichtnegativ sind. Mit F kann zunächst $\vec{G}^{++}_{M_k}$ gebildet und daraus durch Invertierung von p_{st} und Entfernung der Anfangs- und Endkante $\vec{G}^{+}_{M_{k+1}}$ konstruiert werden. Dieses Verfahren kann so lange iteriert werden, bis es keinen st-Pfad mehr gibt.
Laufzeit: Der Algorithmus von Dijkstra benötigt eine Laufzeit von $\mathcal{O}(|E| + |V_1 \cup V_2| \ln(|V_1 \cup V_2|))$, wenn geeignete Datenstrukturen (Fibonacci-heaps, siehe [7]) verwendet werden. Die Kantenmodifizierung und das Invertieren eines minimalen st-Pfades kann in linearer Zeit durchgeführt werden. Da nach maximal $|V_1 \cup V_2|/2$ Iterationen kein st-Pfad mehr existiert, ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(|V_1 \cup V_2| \cdot (|E| + |V_1 \cup V_2| \ln(|V_1 \cup V_2|)))$.

Korollar 13 *Der Algorithmus 2 berechnet zu gegebenen A und B in $\mathcal{O}(n^3)$ den Wert score(A, B).*

[5] Dieser Algorithmus kann so implementiert werden, dass er sowohl einen minimalen st-Pfad als auch alle Distanzen zwischen s und beliebigen Knoten $u \in V'$ berechnet.

Algorithmus 2: Scoremaximierung bei festen Matrizen A und B

Input: $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}, m \leq n$

Output: $score(A, B)$ und die dazugehörende Überlagerung

$M_0 = \emptyset; val = 0;$

G sei der bipartite Graph mit den Partitionen A_i, $1 \leq i \leq m$, und B_j, $1 \leq j \leq n$, und den Kanten mit Gewicht $w(\{A_i, B_j\}) = ||A_i - B_j||^2$

for $k = 1$ **to** m **do**

  berechne M_{k-1}-augmentierenden Pfad p_k in G mit minimalem Gewicht

  $M_k = M_{k-1} \triangle p_k$

  if $\frac{k}{m} \cdot \exp\left(-\sqrt{\frac{w(M_k)}{k}}\right) > val$ **then**

    $val = \frac{k}{m} \cdot \exp\left(-\sqrt{\frac{w(M_k)}{k}}\right)$

    $\forall i \in \{1, \ldots, m\} : f(i) = \begin{cases} j & \text{falls } \exists\, j \text{ mit } \{A_i, B_j\} \in M_k \\ 0 & \text{sonst} \end{cases}$

  end

end

return (val, f)

3.2.2 Ein PTAS für das Überlagerungsproblem

Die Tatsache, effizient zu festen Punktmengen eine optimale Überlagerung berechnen zu können, gestattet es nun, für das Überlagerungsproblem einen Approximationsalgorithmus zu entwickeln. Die Grundidee besteht darin, die Menge aller Bewegungen zu diskretisieren. Für eine geeignet gewählte endliche Teilmenge $\mathcal{M}$ aller Bewegungen kann dann für jedes $M \in \mathcal{M}$ der Algorithmus 2 aufgerufen werden, um $score(A, M(B))$ zu berechnen. Da es überabzählbar viele Bewegungen gibt, ist es nicht überraschend, dass dieses Verfahren keine optimale Überlagerung garantiert. Andererseits wird es sich zeigen, dass eine geeignete Diskretisierung ausreicht, um eine Überlagerung zu erhalten, deren Scorewert dem optimalen Scorewert „sehr nahe“ kommt. Das soll im Folgenden näher präzisiert werden, indem zunächst die Güte eines Algorithmus definiert wird.

Definition 14 *Sei $OPT(I)$ der optimale Wert einer zu maximierenden Zielfunktion mit Eingabe I. Unter der* Güte *eines Algorithmus Alg wird der Wert*

$$\rho(ALG) := \inf_I \left\{ \frac{ALG(I)}{OPT(I)} \;\middle|\; \text{für alle Instanzen } I \right\} \tag{25}$$

verstanden.

Die Güte eines Algorithmus ist demnach eine reelle Zahl aus dem Intervall $[0, 1]$. Gilt $\rho(ALG) = 1$, so ist der Algorithmus optimal. Es kann jedoch sehr aufwendig sein, eine optimale Lösung zu berechnen. Daher stellt sich die Frage, ob es Approximationsalgorithmen mit polynomieller Laufzeit und $\rho(ALG) > 0$ gibt. Eine Klasse von Algorithmen, die in polynomieller Zeit beliebig nahe an die optimale Lösung gelangen, wird im Folgenden definiert.

Definition 15 *Ein Algorithmus ALG ist ein* PTAS *(polynomial time approximation scheme), wenn er zu jedem festen $\epsilon > 0$ polynomielle Laufzeit hat und eine Güte von*

$$\rho(Alg) \geq 1 - \epsilon \tag{26}$$

besitzt.

Im Folgenden wird gezeigt, dass für das Überlagerungsproblem ein PTAS existiert. Zur Vorbereitung werden jedoch noch einige Definitionen und Lemmata benötigt.

Definition 16 *Seien $x \in \mathbb{R}^3$ und $r > 0$ gegeben. Dann sei mit $\mathcal{K}(x; r)$ eine Kugel mit Mittelpunkt x und Radius r, also die Menge $\{y \in \mathbb{R}^3 : ||x - y|| \leq r\}$, bezeichnet.*

Definition 17 *$B^* := M(B)$ mit $M \in AO^+(3)$ sei diejenige Matrix, für die*

$$score^*(A, B) = score(A, M(B)) \tag{27}$$

gilt.

Definition 18 *Seien $x \in \mathbb{R}^3$ und eine Funktion $M : \mathbb{R}^3 \to \mathbb{R}^3$ gegeben. Dann sei*

$$d_M(x) := ||M(x) - x||. \tag{28}$$

Lemma 19 *Es gilt für $\epsilon \geq 0$ und $x_i \geq 0$ die Ungleichung*

$$\sqrt{\frac{\sum_{i=1}^n (x_i + \epsilon)^2}{n}} \leq \sqrt{\frac{\sum_{i=1}^n x_i^2}{n}} + \epsilon. \tag{29}$$

Beweis: Es gilt

$$\begin{array}{rccc} & \sqrt{\frac{\sum_{i=1}^n (x_i+\epsilon)^2}{n}} & \leq & \sqrt{\frac{\sum_{i=1}^n x_i^2}{n}} + \epsilon \\ \iff & \frac{\sum_{i=1}^n (x_i+\epsilon)^2}{n} & \leq & \frac{\sum_{i=1}^n x_i^2}{n} + \epsilon^2 + 2\epsilon\sqrt{\frac{\sum_{i=1}^n x_i^2}{n}} \\ \iff & \frac{\sum_{i=1}^n x_i}{n} & \leq & \sqrt{\frac{\sum_{i=1}^n x_i^2}{n}}. \end{array} \tag{30}$$

Auf der linken Seite der letzten Ungleichung steht das arithmetische Mittel, welches kleiner gleich dem quadratischem Mittel auf der rechten Seite ist. □

Lemma 20 *Der maximale (euklidische) Abstand zweier überlagerter Punkte ist in jeder optimalen Überlagerung durch* $\sqrt{m} \cdot \ln(m)$ *nach oben beschränkt.*

Beweis: Eine untere Schranke für den Score ist $\frac{1}{m}$, wenn nur ein Punkt überlagert wird. Angenommen in einer optimalen Überlagerung gibt es einen Abstand größer als $\sqrt{m} \cdot \ln(m)$. Dann ergibt sich für den optimalen Scorewert von A und B eine obere Schranke von

$$score^*(A,B) < exp\left(-\sqrt{\frac{(\sqrt{m}\ln(m))^2}{m}}\right) = \frac{1}{m} \tag{31}$$

□

Algorithmus 3: Ein Überlagerungsalgorithmus mit Güte $1-\epsilon'$.

Input: $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}, 0 < \epsilon' < 1, m \leq n$

Output: Ein Wert ALG für den gilt: $ALG \geq (1-\epsilon') \cdot score^*(A,B)$

1 $score =$ bestmöglicher Score unter allen Überlagerungen mit maximal zwei überlagerten Punkten

2 $\epsilon = -\ln(1-\epsilon')$ // $(\Rightarrow \epsilon > 0 \wedge \epsilon > \epsilon')$

3 **for** $i = 1$ **to** m **do**

4 $\quad G_i := \mathcal{K}(A_i; \sqrt{m}\ln(m)) \cap \{A_i + \frac{\epsilon}{24}\begin{pmatrix}\lambda_1\\ \lambda_2\\ \lambda_3\end{pmatrix} \mid \lambda_1,\lambda_2,\lambda_3 \in \mathbb{Z}\}$

end

5 **foreach** $\begin{pmatrix}a_1\\ a_2\\ a_3\end{pmatrix} \in \binom{\{1,\ldots,m\}}{3}$ *mit* $1 \leq a_1 < a_2 < a_3 \leq m$ **do**

6 $\quad$ **foreach** $\begin{pmatrix}b_1\\ b_2\\ b_3\end{pmatrix}$ *mit* $1 \leq b_i \leq n;\ b_i \neq b_j$ *für* $i \neq j$ $(i,j \in \{1,2,3\})$ **do**

7 $\quad\quad$ **foreach** $g_1 \in G_{a_1}, g_2 \in G_{a_2}, g_3 \in G_{a_3}$ **do**

8 $\quad\quad\quad rms = \min\limits_{M' \in AO^+(3)} \sqrt{\frac{\sum_{\nu=1}^{3} ||g_\nu - M'(B_{b_\nu})||^2}{3}}$

9 $\quad\quad\quad$ **if** $(rms \leq \frac{\epsilon}{24})$ **then** $score' := score(A, M'(B))$

10 $\quad\quad\quad$ **if** $(score < score')$ **then** $score := score'$

$\quad\quad$ **end**

$\quad$ **end**

end

11 **return** $score$;

Im Folgenden sei f eine optimale Überlagerung, die also $score^*(A,B) = score^*(A,B;f)$ erfüllt. Um zu zeigen, dass der Algorithmus 3 auf Seite 21 ein PTAS für das Überlagerungsproblem liefert, kann ohne Einschränkung vorausgesetzt werden, dass $|f^*| \geq 3$ gilt, denn andernfalls findet der Algorithmus in Zeile 1 den optimalen Scorewert. Weiterhin sei zunächst vorausgesetzt, dass die überlagerten Punkte aus B nicht auf einer Geraden liegen. In jeder optimalen Überlagerung gibt es ein Tripel $T_3 = \{i,j,k\} \subseteq f^*$ und ein überlagertes Dreieck $((A_i, B^*_{f(i)}), (A_j, B^*_{f(j)}), (A_k, B^*_{f(k)}))$ mit folgenden Eigenschaften:

1. $||B^*_{f(i)} - B^*_{f(j)}|| = \max\limits_{\nu_1,\nu_2 \in f^*} \{||B^*_{f(\nu_1)} - B^*_{f(\nu_2)}||\}$,
2. $B^*_{f(k)}$ maximiert die Höhe $h_{B^*_{f(k)}}$ im Dreieck $\Delta(B^*) = \Delta(B^*_{f(i)}, B^*_{f(j)}, B^*_{f(k)})$.

Lemma 21 *Für die Bewegung M', die in Zeile 8 berechnet wird, gilt*

$$\forall x \in T_3 : ||M'(B_{f(x)}) - B^*_{f(x)}|| < \frac{\epsilon}{8}. \tag{32}$$

Beweis: Der Algorithmus 3 betrachtet in den Zeilen 5 und 6 alle Möglichkeiten ein Dreieck mit Punkten aus A mit einem Dreieck mit Punkten aus B zu überlagern. Daher werden im Laufe des Algorithmus die Werte

$$a_1 = i, \quad a_2 = j, \quad a_3 = k \quad \text{und} \quad b_1 = f(i), \quad b_2 = f(j), \quad b_3 = f(k)$$

angenommen. Weiterhin gilt

$$\forall x \in T_3 : B^*_{f(x)} \in \mathcal{K}\left(A_x; \sqrt{m} \cdot \ln(m)\right) \subset \bigcup_{g \in G_x} \mathcal{K}\left(g; \frac{\epsilon}{24}\right), \tag{33}$$

woraus

$$\forall x \in T_3 \; \exists g'_x \in G_x \quad \text{mit} \quad ||g'_x - B^*_{f(x)}|| \leq \frac{\epsilon}{24} \tag{34}$$

folgt. In Zeile 7 werden alle Tripel $[g_i, g_j, g_k]$ mit $g_i \in G_i, g_j \in G_j$ und $g_k \in G_k$ betrachtet, darunter also auch ein Tripel $[g'_i, g'_j, g'_k]$, das die Aussage (34) erfüllt. Die rechte Seite der Ungleichung in (34) ist dabei auch eine obere Schranke für den in Zeile 8 berechneten rms, denn es gilt

$$\begin{aligned} rms &= \min_{M' \in AO^+(3)} \sqrt{\frac{\sum_{x \in T_3} ||g'_x - M'(B_{f(x)})||^2}{3}} \\ &\leq \sqrt{\frac{\sum_{x \in T_3} ||g'_x - B^*_{f(x)}||^2}{3}} \\ &\overset{(34)}{\leq} \frac{\epsilon}{24}. \end{aligned} \tag{35}$$

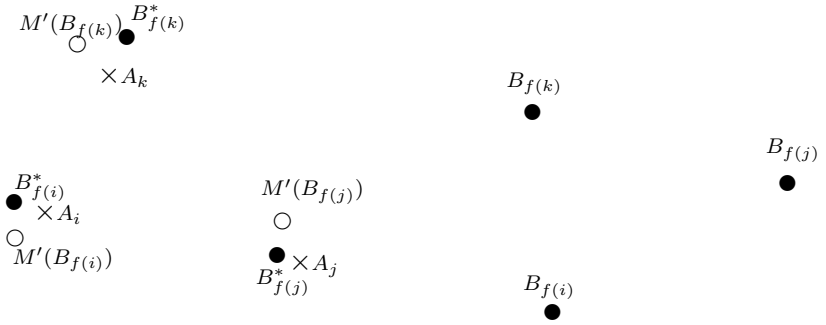

Abbildung 3: schematische Darstellung der Punkte $A_x, B^*_{f(x)}$ und $M'(B_{f(x)})$ mit $x \in T_3$

Dabei gilt für den maximalen Abstand

$$\max_{x \in T_3}\{||g'_x - M'(B_{f(x)})||\} \leq \sqrt{3} \cdot \frac{\epsilon}{24}, \tag{36}$$

denn ansonsten gilt $rms > \frac{\epsilon}{24}$, was ein Widerspruch zu (35) ist. Unter Ausnutzung der Dreiecksungleichung ergibt sich dann $\forall x \in T_3$ die Abschätzung

$$\begin{aligned} ||M'(B_{f(x)}) - B^*_{f(x)}|| &= ||M'(B_{f(x)}) - g'_x + g'_x - B^*_{f(x)}|| \\ &\leq ||M'(B_{f(x)}) - g'_x|| + ||g'_x - B^*_{f(x)}|| \\ &\leq \sqrt{3} \cdot \frac{\epsilon}{24} + \frac{\epsilon}{24} \\ &< \frac{\epsilon}{8}. \end{aligned} \tag{37}$$

□

Definition 22 *M sei diejenige Bewegung, die $B^*_{f(x)}$ nach $M'(B_{f(x)})$ für jedes $x \in T_3$ überführt. Es soll also gelten:*

$$\forall x \in T_3 : M(B^*_{f(x)}) = M'(B_{f(x)}).$$

Bemerkung: M ist eindeutig, da die Punkte aus $B^*_{f(x)}$ mit $x \in T_3$ nach Annahme nicht auf einer Geraden liegen.
Das Lemma 21 hat gezeigt, dass für alle $x \in T_3$ der Wert $d_M(B^*_{f(x)})$ durch $\frac{\epsilon}{8}$ beschränkt ist, wie die folgende Rechnung nachweist.

$$\begin{aligned} \forall x \in T_3 : \quad d_M(B^*_{f(x)}) &= ||M(B^*_{f(x)}) - B^*_{f(x)}|| \\ &= ||M'(B_{f(x)}) - B^*_{f(x)}|| \\ &\overset{L.21}{<} \frac{\epsilon}{8} \end{aligned} \tag{38}$$

Eine ähnliche Aussage gilt für alle anderen überlagerten Punkte aus der optimalen Überlagerung.

Lemma 23 *Es gilt*

$$\forall \nu \in f^* : d_M(B^*_{f(\nu)}) \leq \epsilon. \tag{39}$$

Damit sind die notwendigen Hilfsmittel zusammen, um nachzuweisen, dass für das Überlagerungsproblem ein PTAS existiert.

Theorem 2 *Sei* $0 < \epsilon' < 1$ *und ALG der in Zeile 11 zurückgegebene Wert. Dann gilt*

$$ALG \geq (1 - \epsilon') \cdot score^*(A, B). \tag{40}$$

Beweis: Zunächst erhalten wir durch Anwendung von Lemma 23 und Lemma 19 die Abschätzung

$$\begin{aligned} rms(A, M(B^*); f) &= \sqrt{\frac{\sum_{\nu \in f^*} ||A_\nu - M(B^*_{f(\nu)})||^2}{|f^*|}} \\ &\overset{L.23}{\leq} \sqrt{\frac{\sum_{\nu \in f^*} (||A_\nu - B^*_{f(\nu)}|| + \epsilon)^2}{|f^*|}} \\ &\overset{L.19}{\leq} \sqrt{\frac{\sum_{\nu \in f^*} ||A_\nu - B^*_{f(\nu)}||}{|f^*|}} + \epsilon \\ &= rms(A, B^*; f) + \epsilon. \end{aligned} \tag{41}$$

Da die **if**-Bedingung für M' in Zeile 9 wegen (35) erfüllt ist, wird der bezüglich M' beste Score berechnet. Damit gilt

$$\begin{aligned} ALG &\geq score(A, M'(B)) \\ &= score(A, M(B^*)) \\ &\geq \frac{|f^*|}{m} \cdot e^{-rms(A, M(B^*); f)} \\ &\overset{(41)}{\geq} \frac{|f^*|}{m} \cdot e^{-(rms(A, B^*; f) + \epsilon)} \\ &= score^*(A, B) \cdot e^{-\epsilon}. \end{aligned} \tag{42}$$

Da in der zweiten Zeile $\epsilon = -\ln(1 - \epsilon')$ gesetzt wird, erhalten wir

$$ALG \geq e^{-\epsilon} \cdot score^*(A, B) = (1 - \epsilon') \cdot score^*(A, B) \tag{43}$$

und damit eine Approximationsgüte von $1 - \epsilon'$.

Laufzeit: Die für die Laufzeit relevanten Zeilen sind 5–11. Zeile 5 wird $\mathcal{O}(m^3)$, die Zeile 6 wird $\mathcal{O}(n^3)$ mal durchlaufen. Für jedes i mit $1 \leq i \leq m$ gilt $|G_i| = \mathcal{O}((\frac{\sqrt{m}\ln(m)}{\epsilon})^3)$, daher wird Zeile 7 insgesamt $\mathcal{O}((\frac{\sqrt{m}\ln(m)}{\epsilon})^9)$ mal durchlaufen. Zeile 8 kann durch den Aufruf von Algorithmus 1 in konstanter

Zeit und Zeile 9 durch den Algorithmus 2 in $\mathcal{O}(n^3)$ berechnet werden. Wegen $0 < \epsilon' < \epsilon$ ergibt sich eine Gesamtlaufzeit von $\mathcal{O}\left(\left(n \cdot \frac{\sqrt{m}\ln(m)}{\epsilon'}\right)^9\right)$. □

Beweis von Lemma 23 Zunächst können wir o.B.d.A. annehmen, dass

$$M'(B_{f(i)}) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \qquad M'(B_{f(j)}) = \begin{pmatrix} ||B_{f(i)} - B_{f(j)}|| \\ 0 \\ 0 \end{pmatrix}$$

$$\text{und } M'(B_{f(k)}) = \begin{pmatrix} \xi \\ h_{B^*_{f(k)}} \\ 0 \end{pmatrix} \tag{44}$$

mit $0 < \xi < ||B_{f(i)} - B_{f(j)}||$ gilt. Andernfallls bewegen wir A, B und $M'(B_{f(x)})$ mit $x \in T_3$ derart, so dass $M'(B_{f(x)})$ auf obige Vektoren abgebildet werden. Die Ungleichung (37) bleibt dabei weiter gültig.
Die Beweisstruktur für das Lemma besteht aus drei Schritten. Im ersten Schritt betrachten wir die Translation, im zweiten und dritten Schritt die Rotation der (eindeutigen) Bewegung M. Die Rotation wird dabei in zwei Rotationen zerlegt. Die einzelnen Teilbewegungen erlauben es, Abschätzungen vorzunehmen, mit denen dann das Lemma folgt.
Schritt 1: Zunächst führen wir die Translation

$$T : x \mapsto x - B^*_{f(i)} \quad \text{mit} \quad x \in \{B^*_{f(\nu)} |\ \nu \in f^*\} \tag{45}$$

aus. Damit ist schon einmal der Punkt $M'(B_{f(i)})$ zur Deckung gebracht worden, denn es gilt $M'(B_{f(i)}) = T(B^*_{f(i)})$. Außerdem haben sich die Punkte $B^*_{f(\nu)}$ mit $\nu \in f^*$ nicht allzu weit entfernt, denn es gilt

$$\begin{aligned} \forall x \in \{B^*_{f(\nu)} | \nu \in f^*\}: \quad d_T(x) &= ||T(x) - x|| \\ &= ||(x - B^*_{f(i)}) - x|| \\ &= || - B^*_{f(i)}|| \\ &= ||M'(B_{f(i)}) - B^*_{f(i)}|| \\ &\overset{L.21}{<} \frac{\epsilon}{8}. \end{aligned} \tag{46}$$

Unter Ausnutzung der Dreiecksungleichung und (37) erhalten wir

$$\begin{aligned} ||T(B^*_{f(j)}) - M'(B_{f(j)})|| &= ||T(B^*_{f(j)}) - B^*_{f(j)} + B^*_{f(j)} - M'(B_{f(j)})|| \\ &\leq ||T(B^*_{f(j)}) - B^*_{f(j)}|| + ||B^*_{f(j)} - M'(B_{f(j)})|| \\ &\leq 2 \cdot \frac{\epsilon}{8} \\ &= \frac{\epsilon}{4}. \end{aligned} \tag{47}$$

Schritt 2: Jetzt liegen die Punkte $T(B^*_{f(j)})$ und $M'(B_{f(j)})$ auf der Sphäre von $\mathcal{K}(0; M'(B_{f(j)}))$, denn es gilt

$$|| \underbrace{B^*_{f(j)} - B^*_{f(i)}}_{T(B^*_{f(j)})} || = ||M'(B_{f(j)}) - \underbrace{M'(Bf(i))}_{0}||. \tag{48}$$

Alle anderen Punkte $T(B^*_{f(\nu)})$ mit $\nu \in f^*$ liegen nicht außerhalb dieser Späre, denn ansonsten ist das ein Widerspruch zu der Eigenschaft 1 von $\Delta(B^*)$.
Fall 1: Die Punkte $T(B^*_{f(j)})$ und $M'(B_{f(j)})$ sind nicht antipodal[6].
Betrachte dann den auf der Sphäre von $\mathcal{K}(0; M'(B_{f(j)}))$ liegenden eindeutigen Großkreis K, der durch die beiden Punkte $T(B^*_{f(j)})$ und $M'(B_{f(j)})$ verläuft, und die Gerade g, die durch den Ursprung geht und orthogonal zur Ebene $E \supset K$ steht. Dann gibt es eine eindeutige Rotationsmatrix R mit Drehachse g und

$$R(T(B^*_{f(j)})) = M'(B_{f(j)}). \tag{49}$$

Andererseits hat der Punkt $T(B^*_{f(j)})$ unter allen Punkten $T(B^*_{f(\nu)})$ mit $\nu \in f^*$ maximalen Abstand zu g.[7] Damit gilt

$$d_R(T(B^*_{f(j)})) = \max_{\nu \in f^*}\{d_R(T(B^*_{f(\nu)}))\}. \tag{50}$$

Aus (47), (49) und (50) folgt dann

$$\forall \nu \in f^* : \quad d_R(T(B^*_{f(\nu)})) \leq \frac{\epsilon}{4}. \tag{51}$$

Fall 2: $T(B^*_{f(j)})$ und $M'(B_{f(j)})$ sind antipodal.
Dann existieren mehrere Rotationsmatrizen R, die $R(T(B^*_{f(j)})) = M'(B_{f(j)})$ erfüllen, z.B. die Rotation mit Drehachse $\lambda(0,0,1)^T$ und Drehwinkel π. Die Aussage (51) bleibt jedoch für jedes R, das (49) erfüllt, gültig.

Es gilt somit

$$R(T(B^*_{f(i)})) = M'(B_{f(i)}) \quad \text{und} \quad R(T(B^*_{f(j)})) = M'(B_{f(j)}). \tag{52}$$

Aus (47) und (51) folgt

$$\forall \nu \in f^* : d_{R \circ T}(B^*_{f(\nu)}) \leq \frac{\epsilon}{2}. \tag{53}$$

Schritt 3: Im letzten Schritt muss nur noch der Punkt $R(T(B^*_{f(k)}))$ auf $M'(B_{f(k)})$ abgebildet werden, was sich durch eine eindeutige Rotations R_x mit Drehachse $\lambda(1,0,0)^T$ erreichen lässt. Dabei hat $R(T(B^*_{f(k)}))$ unter allen

[6] Zwei Punkte auf einer Kugeloberfläche sind antipodal, wenn deren Strecke durch den Mittelpunkt der Kugel verläuft.

[7] Der Abstand eines Punktes x zu einer Geraden Y sei $\min_{y \in Y}\{||x - y||\}$.

Punkten $R(T(B^*_{f(\nu)}))$ mit $\nu \in f^*$ maximalen Abstand zur Drehachse, denn andernfalls ergibt sich ein Widerspruch zur Eigenschaft 2 von $\Delta(B^*)$. Somit gilt für R_x die Ungleichung

$$d_{R_x}(R(T(B^*_{f(\nu)}))) \leq \frac{\epsilon}{2} \quad \text{mit} \quad \nu \in f^*. \tag{54}$$

Nachdem T, R und R_x ausgeführt sind, gilt mit den Abschätzungen (53) und (54)

$$\forall x \in f^* : d_M(B^*_{f(x)}) = d_{R_x \circ R \circ T}(B^*_{f(x)}) \leq \epsilon. \tag{55}$$

Damit ist das Lemma 23 bewiesen, sofern $B^*_{f(i)}, B^*_{f(j)}$ und $B^*_{f(k)}$ nicht auf einer Geraden liegen.
Liegen diese Punkte auf einer Geraden, so ist die Bewegung M nicht eindeutig. Die Argumentation in Schritt 1 und Schritt 2 bleibt jedoch gültig. Nach dem zweiten Schritt liegen alle Punkte $(R(T(B^*_{f(\nu)})))$ mit $\nu \in f^*$ auf der Geraden $\lambda(1,0,0)^T$. Diese Punkte bleiben fix, wenn im dritten Schritt um diese Gerade mit einem beliebigen Drehwinkel gedreht wird. Damit bleibt die Abschätzung (54) für jeden Drehwinkel gültig, und damit auch die Aussage in (55).
Damit ist das Lemma bewiesen, und daraus folgt mit dem Beweis zu Theorem 2 die Korrektheit des Algorithmus 3. □

Dieser Ansatz funktioniert auch bei einigen Varianten der Scorefunktion. Werden Spiegelungen zugelassen, so kann der Algorithmus jeweils mit Input (A, B) und $(A, s(B))$ mit einer beliebigen Spiegelung s ausgeführt und dann das bessere Ergebnis genommen werden. Sollen nur gleiche Atomtypen überlagert werden, so sind im Matchingalgorithmus nur Kanten zwischen gleichen Atomtypen zugelassen.
Die meisten Abschätzungen bezüglich Konstanten und dem maximalen Abstand zweier überlagerter Punkte in einer optimalen Überlagerung sind vermutlich nicht scharf und können durch eine genauere Analyse verbessert werden.
In Zeile 9 kann neben dem Scorewert auch die dazugehörige Überlagerung berechnet werden, die dann gegebenenfalls in Zeile 10 gespeichert wird. In Zeile 11 kann diese zusätzlich zurückgegeben werden, um als Zertifikat für den Scorewert zu dienen.

4 Implementation eines effizienten Algorithmus zur Überlagerung

Der Algorithmus 3 ist in erster Linie von theoretischem Interesse, da dessen Analyse zeigt, dass für das Überlagerungsproblem ein PTAS existiert. Die Laufzeit ist in der Praxis jedoch nicht akzeptabel, da der Grad des Polynoms sehr groß ist. Das liegt im Wesentlichen an folgenden drei Anweisungen, die ineinander geschachtelt aufgerufen werden.

1. In den Zeilen 5–6 von Algorithmus 3 werden alle Möglichkeiten betrachtet, um alle Spaltentripel aus A auf alle Spaltentripel aus B abzubilden.
2. Die Feinheit der Diskretisierung, welche die Anzahl der Schleifenaufrufe in Zeile 7 bestimmt, ist von dem Eingabeparameter ϵ abhängig und trägt maßgeblich zur Laufzeit bei.
3. Zu zwei gegebenen festen Matrizen wird in Zeile 9 ein gewichtetes bipartites Matching berechnet.

Für einen Algorithmus, der in der Praxis ein effizientes Laufzeitverhalten besitzen soll, müssen daher einige Modifikationen vorgenommen werden. In diesem Kapitel sollen einige Änderungen, welche die drei obigen Punkte betreffen, vorgestellt werden, mit dem Ziel, gute Überlagerungen in kurzer Zeit zu erhalten. Da mit diesen Änderungen leider keine Güteaussage mehr getroffen werden kann, werden im nächsten Kapitel experimentelle Ergebnisse vorgestellt, um die Qualität des Algorithmus einschätzen zu können.

4.1 Auswahl der Dreiecksüberlagerung

Die Zeilen 5 und 6 in Algorithmus 3 werden zusammen $6\binom{m}{3}\binom{n}{3}$-mal durchlaufen. Es ist jedoch zu erwarten, dass zwei Dreiecke, die sich nur mit einem sehr hohen mittleren quadratischen Abstand überlagern lassen, nicht als Teilüberlagerung in einer optimalen Überlagerung vorkommen. Diese Annahme wird durch die gewonnenen Erfahrungen gestützt.
Daher besteht die erste Modifikation darin, nur eine Teilmenge an Paaren von Spaltentripel zu betrachten, deren mittlerer quadratischer Abstand relativ klein ist und die algorithmisch effizient gefunden werden können. Die genaue Auswahl wird durch die folgende Definition spezifiziert.

Definition 24 *Zu einer Überlagerung f heißt eine Menge $\{x, y\} \in \binom{|f^*|}{2}$ d-beschränkt, falls*

$$\big|\,||A_x - A_y|| - ||B_{f(x)} - B_{f(y)}||\,\big| < d \tag{56}$$

gilt. Eine Dreiecksüberlagerung ist eine Überlagerung f mit $|f^*| = 3$ und heißt d-beschränkt, wenn alle $\{x, y\} \in \binom{|f^*|}{2}$ d-beschränkt sind.

Bemerkung: Falls $\{x, y\}$ d-beschränkt ist, reden wir im Folgenden auch von dem d-beschränkten Kantenpaar $\{\{A_x, B_{f(x)}\}, \{A_y, B_{f(y)}\}\}$.
Zu gegebenen A und B ist die Anzahl der d-beschränkten Dreiecksüberlagerungen monoton steigend in d. Mit dem Parameter d kann daher gesteuert werden, wie viele Dreiecksüberlagerungen betrachtet werden sollen. Die genauere Wahl für d wird später diskutiert. Zunächst soll darauf eingegangen werden, wie alle d-beschränkten Dreiecke effizient gefunden werden können. Dafür erweisen sind folgende Datenstrukturen als zweckmäßig.
Interpretiere $A \in \mathbb{R}^{3\times m}$ als vollständigen gewichteten Graphen mit Knoten $A_1, \ldots, A_m$, dessen Kantengewichte durch den euklidischen Abstand zweier Knoten gegeben seien.
Die Kanten werden durch ein Feld `dist` mit der Datenstruktur

```
struct dist {
      int  von;
      int  nach;
      int  gewicht;
}
```

repräsentiert, das aufsteigend nach dem Feld `gewicht` sortiert ist. Zu einer Kante $\{A_x, A_y\}$ mit $x < y$ und i-kleinstem Gewicht sei `dist[`i`].von` $:= A_x$, `dist[`i`].nach` $:= A_y$ und `dist[`i`].gewicht` $:= ||A_x - A_y||$. Zusätzlich existiere für jeden Knoten A_i ein Feld `dist_a`, das die nach dem entsprechenden Kantengewicht aufsteigend sortierten Nachbarn beinhaltet. `dist_a[`i`][`j`].node` sei der j-te Nachbar von Knoten A_i, `dist_a[`i`][`j`].gewicht` sei das Gewicht der korrespondierenden Kante.
Die beiden Datenstrukturen `dist` und `dist_a` können in $\mathcal{O}(m^2 \ln m)$ berechnet werden.
Nun sind wir in der Lage, den folgenden Algorithmus anzugeben.

Theorem 3 *Zu jedem $d > 0$ findet der Algorithmus 4 auf Seite 31 alle d-beschränkten Dreiecke.*

Beweis: Sei $((A_{\tilde{a}_1}, B_{\tilde{b}_1}), (A_{\tilde{a}_2}, B_{\tilde{b}_2}), (A_{\tilde{a}_3}, B_{\tilde{b}_3}))$ ein d-beschränktes Dreieck und o.B.d.A. gelte $1 \leq \tilde{b}_1 < \tilde{b}_2 < \tilde{b}_3 \leq n$.
Betrachte den Schleifendurchlauf mit $b_1 = \tilde{b}_1$ und $b_2 = \tilde{b}_2$.
Sofern es zu $\{B_{b_1}, B_{b_2}\}$ in A eine d-beschränkte Kante gibt, ist dist$[i_1]$ in Zeile 3 diejenige Kante aus A mit dem kleinsten Gewicht, die zu $\{B_{b_1}, B_{b_2}\}$ d-beschränkt ist. Andernfalls ist $i_1 = \binom{m}{2} + 1$ und erfüllt die anschließende **while**-Bedingung in Zeile 5 nicht. $\{B_{b_1}, B_{b_2}\}$ d-beschränkt ist, sofern sie existiert. In der **while**-Schleife in Zeile 5 werden daher alle Kanten aus A durchlaufen, die zu $\{B_{b_1}, B_{b_2}\}$ d-beschränkt sind, da das Feld `dist` nach Gewicht aufsteigend sortiert ist und i_1 in Zeile 19 inkrementiert wird.
Daher nimmt i_1 auch jenen Wert an, für den die Kante `dist[`i_1`]` genau der Kante $\{A_{\tilde{a}_1}, A_{\tilde{a}_2}\}$ entspricht. Je nach Orientierung der Kante `dist[`i_1`]` sind jetzt zwei Fälle zu unterscheiden:

Fall 1: Es gilt $\tilde{a}_1 < \tilde{a}_2$.
Betrachte den Durchlauf mit $r = 1$ in Zeile 6. In diesem Fall gilt nach Zeile 8 $a_1 = \tilde{a}_1$ und $a_2 = \tilde{a}_2$. Betrachte in Zeile 10 den Durchlauf mit $b_3 = \tilde{b}_3$. In Zeile 11 ist `dist_a[`a_2`][`i_2`]` diejenige Kante mit minimalem Gewicht, die zu $A_{\tilde{a}_2}$ inzident ist und mit der Kante $\{B_{\tilde{b}_2}, B_{\tilde{b}_3}\}$ d-beschränkt ist, sofern eine solche existiert. Andernfalls gilt $i_2 = m$, und i_2 erfüllt nicht die Bedingung der **while**-Schleife in Zeile 12. Da `dist_a[`a_2`][.]` nach der zweiten Komponente aufsteigend sortiert ist, werden in der **while**-Schleife in Zeile 12 alle Kanten aus A durchlaufen, die zu $A_{\tilde{a}_2}$ inzident und zu $\{B_{\tilde{b}_2}, B_{\tilde{b}_3}\}$ d-beschränkt sind, darunter also auch die Kante $\{A_{\tilde{a}_2}, A_{\tilde{a}_3}\}$. Die Bedingung in Zeile 14 ist erfüllt, wenn $\tilde{a}_3$ nicht in der Menge $\{\tilde{a}_1, \tilde{a}_2\}$ vorkommt und die verbleibenden Kanten $\{A_{\tilde{a}_1}, A_{\tilde{a}_3}\}$ und $\{B_{\tilde{b}_1}, B_{\tilde{b}_3}\}$ aus der Dreiecksüberlagerung d-beschränkt sind. In diesem Fall sind $\tilde{a}_1, \tilde{a}_2$ und $\tilde{a}_3$ paarweise verschieden, und $((A_{\tilde{a}_1}, B_{\tilde{b}_1}), (A_{\tilde{a}_2}, B_{\tilde{b}_2}), (A_{\tilde{a}_3}, B_{\tilde{b}_3}))$ ist eine d-beschränkte Dreieckecksüberlagerung.

Fall 2: Es gilt $\tilde{a}_2 < \tilde{a}_1$.
Die Argumentation verläuft analog zum ersten Fall, nur dass der Durchlauf mit $r = 2$ in Zeile 6 betrachtet werden muss.
Andererseits werden in Zeile 20 auch nur d-beschränkte Dreiecksüberlagerungen ausgegeben, da bei allen drei Kantenpaaren geprüft wird, ob sie d-beschränkt sind (Zeilen 4, 12 und 14). □

Bemerkung: Im praktischen Einsatz ist dieser Algorithmus im Vergleich zu dem Algorithmus, der einfach alle Dreiecksüberlagerungen ausprobiert, deutlich schneller, sofern es relativ wenige d-beschränkte Dreiecksüberlagerungen gibt. Im worst-case haben beide Algorithmen jedoch dieselbe Laufzeit.

Algorithmus 4: d-beschränkte Dreiecksüberlagerungen finden

Input: $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}, d > 0, m \leq n$
Output: Alle d-beschränkten Dreiecksüberlagerungen

1 $\mathcal{F} = \emptyset$
2 **for** $b_1 = 1$ **to** $n-2$ **do**
3 **for** $b_2 = b_1 + 1$ **to** $n-1$ **do**
4 $i_1 = \min(\{x : \mid ||B_{b_1} - B_{b_2}|| - \texttt{dist[}x\texttt{].gewicht}| < d\}$
 $\cup \{\binom{m}{2} + 1\})$
5 **while** $(i_1 \leq \binom{m}{2}$ **and** $|\texttt{dist[}i_1\texttt{].gewicht} - ||B_{b_1} - B_{b_2}|| \mid <$
 $d)$ **do**
6 **for** $r = 1$ **to** 2 **do**
7 **if** $r = 1$ **then**
8 $a_1 = \texttt{dist[}i_1\texttt{].nach}$; $a_2 = \texttt{dist[}i_1\texttt{].von}$
 else
9 $a_1 = \texttt{dist[}i_1\texttt{].von}$; $a_2 = \texttt{dist[}i_1\texttt{].nach}$
 end
10 **for** $b_3 = b_2 + 1$ **to** n **do**
11 $i_2 = \min(\{x : \mid ||B_{b_2} - B_{b_3}||$
 $- \texttt{dist\_a[}a_2\texttt{][}x\texttt{].gewicht}| < d\} \cup \{m\})$
12 **while** $i_2 < m$ **and**
 $\mid \texttt{dist\_a[}a_2\texttt{][}i_2\texttt{].gewicht} - ||B_{b_2} - B_{b_3}|| \mid < d$ **do**
13 $a_3 = \texttt{dist\_a[}a_2\texttt{][}i_2\texttt{].node}$
14 **if** $i_2 \leq m$ **and** $a_3 \notin \{a_1, a_2\}$ **and**
 $\mid ||B_{b_1} - B_{b_3}|| - ||A_{a_1} - A_{a_3}|| \mid < d$ **then**
15 $\forall i \in \{1, \ldots, m\} : f(i) = 0$
16 $\forall i \in \{1, 2, 3\} : f(a_i) = b_i$
17 $\mathcal{F} = \mathcal{F} \cup f$
 end
18 $i_2 = i_2 + 1$
 end
 end
 end
19 $i_1 = i_1 + 1$
 end
 end
end
20 **return** $\mathcal{F}$

4.2 Lokale Optimierung von Überlagerungen

Die Diskretisierung der Umgebungen der einzelnen Punktmengen in Algorithmus 3 in den Zeilen 3, 4 und 7 wird aufgrund der Laufzeitnachteile nicht

implementiert. Stattdessen werden die Verfahren aus den Abschnitten 3.1.1 und 3.2.1 kombiniert, um eine gegebene Überlagerung lokal zu verbessern. Diese Technik findet sich auch in [1] wieder. Angefangen mit $i = 1$ und einer Dreiecksüberlagerung können die Schritte

1. Berechne zu festen Matrizen A und B mit Algorithmus 2 die optimale Überlagerung f_i und damit $s_i^{(1)} := score(A, B; f_i)$.
2. Berechne zu f_i mit Algorithmus 1 die optimale Bewegung M für B und setze $s_i^{(2)} := score^*(A, B; f_i)$.
3. $B := M(B); i = i + 1;$

solange wiederholt werden, bis für ein i_0 die Gleichung $s_{i_0+1}^{(1)} = s_{i_0}^{(2)}$ gilt. Nach dem 2. Schritt gilt in jedem Durchlauf i die Ungleichung $s_i^{(1)} = score(A, B; f_i) \leq score^*(A, B; f_i) = s_i^{(2)}$. Direkt nachdem B dann in Schritt 3 aktualisiert wurde, gilt $s_i^{(2)} = score^*(A, B; f_i) = score(A, B; f_i)$. Da im anschließenden Schritt 1 des nächsten Durchlaufs die Überlagerung f_{i+1} gewählt wird, die zu A und B optimal ist, gilt $score(A, B; f_i) \leq score(A, B; f_{i+1})$ und damit $s_i^{(2)} \leq s_{i+1}^{(1)}$.
Damit ergibt sich eine monoton wachsende Folge $s_1^{(1)}, s_1^{(2)}, s_2^{(1)}, s_2^{(2)}, \ldots$, für die es einen Index geben muss, ab dem sie konstant ist, da es nur endlich viele Überlagerungen gibt. Die Schritte 1–3 können so lange iteriert werden, bis zum ersten Mal zwei Folgenglieder konstant sind. Dazu waren in den meisten Fällen weniger als fünf Iterationen notwendig. Eine plausible Erklärung dafür ist, dass durch die anfängliche Dreiecksüberlagerung, die grobe Ausrichtung schon recht gut fixiert ist.
Ist $s_1^{(1)}$ weit vom Scorewert der bisher besten Überlagerung entfernt, so tritt nach empirischen Untersuchungen nur sehr selten der Fall ein, dass am Ende der Iteration eine Überlagerung entsteht, deren Scorewert den bisher besten Scorewert übertrifft. Zwecks Zeitersparnis bietet es sich daher an, nur dann eine Überlagerung lokal zu verbessern, wenn $s_1^{(1)}$ nicht allzu weit vom Scorewert der bisher besten Überlagerung entfernt ist.

4.3 Hashing von Überlagerungen

Bei Testläufen stellte sich heraus, dass viele Überlagerungen mehrfach berechnet werden. Da die Matchingalgorithmen für einen Großteil der Laufzeit verantwortlich sind, sollte eine Überlagerung nicht weiter lokal verbessert werden, wenn das schon zu einem früheren Zeitpunkt geschehen ist. Da es sehr aufwändig ist, alle bisherigen Überlagerungen zu speichern und eine gegebene Überlagerung darin wiederzufinden, bietet es sich stattdessen an, diese zu hashen. Zu jeder Überlagerung f mit m Einträgen sei $h : f \to h_f$

die Hashfunktion mit

$$h(i) = \begin{cases} 0 & \text{falls } f(i) = 0 \\ 1 & \text{falls } f(i) > 0 \end{cases} . \tag{57}$$

Die Hashfunktion kann in einem Binärbaum gespeichert werden, analog wie der Enumerationsbaum im Abschnitt 3.1.2. Ein Pfad von der Wurzel zu einem Blatt im Binärbaum entspricht eineindeutig einem h. Falls zu einer Überlagerung f der zum Hashwert h_f korrespondierende Pfad im Binärbaum noch nicht existiert, wird er in den Baum entsprechend eingefügt. Am Blatt wird eine Liste $L(h_f)$ angelegt und der Scorewert von f in $L(h_f)$ gespeichert. Existiert zu einer Überlagerung f der Hashwert bereits, so wird der zu f gehörende Scorewert mit allen Werten aus $L(h_f)$ verglichen. Ist dieser in $L(h_f)$ enthalten[8], so gilt f als eine bereits zu einem früheren Zeitpunkt aufgetretende und gehashte Überlagerung, andernfalls wird er in $L(h_f)$ eingefügt.
Anzumerken ist, dass eine vorher noch nicht gehashte Überlagerung nicht zwangsläufig auch als solche erkannt wird. Es seien f_1 und f_2 zwei verschiedene Überlagerungen, aber mit gleichem Hashwert und gleichem Scorewert. f_1 sei bereits gehasht und daher in $L(h_{f_1})$ enthalten. Wenn danach f_2 zum ersten Mal auftritt, wird es nicht in $L(h_{f_1})$ gespeichert, sondern irrtümlich als die schon früher aufgetretende Überlagerung f_1 identifiziert.
In durchgeführten Untersuchungen wurde eine solche Situation jedoch nicht beobachtet. Verschiedene Überlagerungen mit gleichem Hashwert traten zwar gelegentlich auf, aber nur bei relativ geringen Scorewerten, die zudem noch verschieden waren. Solche sind jedoch meistens nicht von großem Interesse. Mit Hashing konnte die Laufzeit im Durchschnitt um etwa ein Fünftel gesenkt werden, so dass der Einsatz gerechtfertigt erscheint. Abschließend sei erwähnt, dass bei der Berechnung ähnlicher Moleküle Mehrfachüberlagerungen häufiger aufgetreten sind.

4.4 Approximatives Matching

Die Motivation, sich mit approximativen Matchingalgorithmen zu befassen, besteht darin, dass der Algorithmus 2 eine Laufzeit von $\mathcal{O}(n^3)$ hat, die nicht akzeptabel ist, wenn er sehr oft aufgerufen wird.
Während es Approximationsalgorithmen mit linearer Laufzeit und konstanter Güte für ein schwerstes Matching in gewichteten Graphen gibt [5], sind solche Approximationsalgorithmen für das Minimierungsproblem bezüglich Matchings mit fester Kardinalität in gewichteten Graphen nicht bekannt. Da die hier auftretenden Graphen metrisch sind, können eventuell zufriedenstellende Approximationen erreicht werden. Im Folgenden werden zwei

[8] Da in der Implementation die Scorewerte Fließkommazahlen sind und daher mit geringen Rundungsfehlern behaftet sind, werden bei diesem Vergleich zwei Zahlen a und b als gleich angesehen, wenn $|a - b| < \epsilon$ mit $\epsilon \approx 10^{-5}$ gilt.

Matchingalgorithmen diskutiert, die beide im Rahmen dieses Projekts implementiert wurden.

4.4.1 Entfernung von Kanten mit hohem Gewicht

Bei den zugrundeliegenden Daten ist der zu A und B in der Matchingroutine konstruierte Graph vollständig bipartit, wobei die meisten Kanten ein hohes Gewicht haben. Eine Kante mit hohem Gewicht wird mit hoher Wahrscheinlichkeit nicht in einem minimalen Matching M_i mit genau i Kanten auftreten, dessen korrespondierende Überlagerung optimal ist. Denn es ist dann meistens besser, die teure Kante wegzulassen. Dadurch werden zwar nur $i-1$ Punkte überlagert, jedoch reduziert sich auch der mittlere quadratische Abstand, was im Endeffekt zu einem höherem Scorewert führt.
Sei f eine Überlagerung mit $score(A,B) = score(A,B;f)$. Werden nur Kanten mit Gewicht größer als

$$\delta := \max_{e \in \bigcup_{i=1}^{|f^*|} M_i} w(e) \tag{58}$$

entfernt, so existieren die augmentierenden Pfade $p_1, \ldots, p_{|f^*|}$ mit jeweils minimalem Gewicht aus Algorithmus 2 im augedünnten Graphen weiterhin. f wird weiterhin ausgegeben, wenn nur Kanten in G zugelassen sind, deren Kantengewicht durch δ nach oben beschränkt ist. Um δ zu bestimmen, müsste jedoch die gerade zu berechnenden Matchings M_i schon bekannt sein. Im Folgenden ist daher δ lediglich ein Eingabeparameter eines entsprechenden Approximationsalgorithmus Sparse-Matching(A, B, δ). Graphen mit $\delta > 2$ wurden nur sehr selten beobachtet und auch nur dann, wenn $score(A,B;f) \ll score^*(A,B)$ galt. Werden Kanten mit Gewicht größer zwei entfernt, hat jeder Knoten in der Regel nur noch konstanten Grad,[9] und damit bleiben dann nur noch wenige Kanten (in der Größenordnung $\mathcal{O}(n)$) übrig. In Graphen, in denen die Kantenanzahl linear bezüglich der Knotenanzahl ist, bieten sich für die Implementation des Algorithmus von Dijkstra binäre Heaps an, womit die Laufzeit auf $\mathcal{O}(n^2 \ln(n))$ reduziert werden kann.

4.4.2 Greedy-Matching

Da auch eine Laufzeit von $\mathcal{O}(n^2 \ln(n))$ noch recht hoch ist, soll hier eine weitere Vereinfachung vorgestellt werden, um $score(A,B)$ zu approximieren. Wie in 4.4.1 sind in dem bipartiten Graphen also nur Matchingkanten enthalten, die ein Gewicht kleiner als δ haben, wobei δ ein Parameter des Algorithmus ist. Gestartet wird mit einem leeren Matching M. Alle Kanten mit Kantengewicht kleiner als der Eingabeparameter δ werden nach Gewicht aufsteigend sortiert und dann der Reihe nach zu M hinzugefügt,

[9] Zwischen je zwei Atomen gibt es einen Mindestabstand.

sofern dadurch M weiterhin ein Matching bleibt. Bei jeder hinzugefügten Kante wird die zum Matching korrespondierende Überlagerung f bestimmt und $score(A, B; f)$ berechnet. Der Algorithmus ist auf Seite 35 angegeben.

Algorithmus 5: Greedy-Matching

Input: $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}, \delta > 0, m \leq n$

Output: eine Überlagerung f und dessen Scorewert

1 $M = \emptyset$; $val = 0$; $k = 0$;
2 $E_{\leq\delta} = \{\{A_i, B_j\} | \, i \in \{1, \ldots, m\} \wedge j \in \{1, \ldots, n\} \wedge ||A_i - B_j||^2 \leq \delta\}$
3 **while** $E_{\leq\delta} \neq \emptyset$ **do**
4 $\quad e$ sei eine Kante mit minimalem Gewicht aus $E_{\leq\delta}$
5 $\quad$ entferne e und alle zu e inzidenten Kanten aus $E_{\leq\delta}$
6 $\quad M = M \cup e$; $k = k + 1$;
7 $\quad$ **if** $\frac{k}{m} \cdot \exp\left(-\sqrt{\frac{w(M)}{k}}\right) > val$ **then**
8 $\quad\quad val = \frac{k}{m} \cdot \exp\left(-\sqrt{\frac{w(M)}{k}}\right)$
9 $\quad\quad \forall i \in \{1, \ldots, m\} : \; f(i) = \begin{cases} j & \text{falls } \exists\, j \text{ mit } \{A_i, B_j\} \in M \\ 0 & \text{sonst} \end{cases}$
$\quad$ **end**
end
10 **return** (val, f)

Dieser Algorithmus bietet sich wegen seiner kurzen Laufzeit an. Die Menge $E_{\leq d}$ in Zeile 2 kann in $\mathcal{O}(mn)$ berechnet werden. In der Implementation wird bei zwei Punkten $A_x := (x_1, x_2, x_3)$ und $B_y := (y_1, y_2, y_3)$ zunächst geprüft, ob $(x_1 - y_1)^2 > d$ gilt. Es stellt sich heraus, dass bei den meisten Koordinatenpaaren diese Ungleichung erfüllt ist, so dass die Kante sofort verworfen werden kann, ohne die anderen Komponenten noch anzuschauen. Wie in 4.4.1 angeführt, hat $E_{\leq d}$ für $\delta \approx 2$ bezüglich n etwa lineare Größe. Die Laufzeit von Algorithmus 5 beträgt offensichtlich $\mathcal{O}(mn + |E_{\leq d}| \cdot \ln |E_{\leq d}|)$ und erweist sich meistens als deutlich schneller als das Verfahren aus 4.4.1. Ohne die Zeilen 7–9 ist der Algorithmus unter anderem in [2] beschrieben und dort mit anderen approximativen Matchingalgorithmen verglichen worden, jedoch um perfekte Matchings minimalen Gewichts in vollständigen metrischen Graphen zu finden. Für diesen Fall ist die Approximationsgüte bekannt und beträgt

$$\frac{3}{4}\, n^{-\log_2 \frac{3}{2}}. \tag{59}$$

Der Beweis ist in [11] zu finden, und die dort angegebene Instanz lässt sich für vollständig bipartite Graphen anpassen.

Es gibt ein einfaches hinreichendes Kriterium um festzustellen, ob bei gleichen Eingaben Greedy-Matching die gleiche Überlagerung zurückgibt wie Sparse-Matching.

Lemma 25 *Sei $(A \cup B, E, c)$ ein bipartiter gewichteter Graph mit paarweise verschiedenen Kantengewichten. Sei f = Sparse-Matching(A, B, δ). Wenn die ersten $|f^*|$ augmentierenden Pfade jeweils aus genau einer Kante bestehen, dann gilt f = GreedyMatching(A, B, δ).*

Beweis: Der Beweis kann über Induktion geführt werden. Da p_1 die Kante mit minimalen Gewicht ist, wählt auch Greedy-Matching in Zeile 4 die Kante p_1. Im Induktionsschritt habe Greedy-Matching und Sparse-Matching die Kanten $p_1, \ldots, p_i$ mit $i < |f^*|$ ausgewählt. Da Sparse-Matching die Kante p_{i+1} wählt, ist sie nicht zu den bereits gewählten Kanten inzident, und damit in der aktuellen Menge $E_{\leq\delta}$ aus Greedy-Matching enthalten. p_{i+1} ist die Kante mit dem kleinsten Gewicht in $E_{\leq\delta}$, da andernfalls p_{i+1} kein augmentierender Pfad mit minimalen Gewicht ist. Das Kantengewicht von p_{i+1} ist eindeutig, und der Greedy-Matching wählt daher p_{i+1}. □

4.4.3 Kombination der beiden Matchingverfahren

Die beiden approximativen Matchingverfahren können kombiniert werden. Empirische Beobachtungen haben gezeigt, dass in immerhin etwa 70 % der Fälle die Voraussetzungen von Lemma 25 erfüllt waren. In der Implementation wird aus diesem Grund das Verfahren aus 4.4.1 nur bei Überlagerungen verwendet, deren Scorewerte relativ nahe an dem bisher besten Scorewert sind.

4.5 Anmerkungen zu den verwendeten Parametern

Nachdem die einzelnen Prozeduren vorgestellt worden sind, ist auf Seite 38 der implementierte Algorithmus abgebildet. Dabei besitzt der Algorithmus einige Parameter, die angepasst werden können, um entweder sehr schnell ein Ergebnis mit geringerer Qualität zu erhalten oder bessere Ergebnisse zu erzielen und eine Zunahme der Laufzeit in Kauf zu nehmen. Die folgenden Werte für die Parameter haben sich hinsichtlich Laufzeit und Qualität bewährt, und die in Kapitel 5 vorgestellten Resultate sind damit erzielt worden.

- Mit dem Parameter d kann die Anzahl der Dreiecksüberlagerungen variiert werden, die der Algorithmus 4 zurückgibt. Bei Molekülgrößen von 16 bis etwa 35 Atomen ist ein Wert von $d = 0.2$ verwendet worden. Bei Molekülen mit 60 Atomen steigt die Anzahl stark an, so dass $d = 0.1$ gesetzt werden sollte.

- Die Parameter δ_G und δ_S dünnen die Graphen bei den Matchingalgorithmen aus. Es sind speziell die Werte $\delta_G = 1.5$ und $\delta_S = 2.0$ verwendet worden.
- Die Formulierung „$score_{akt}$ `nicht zu klein`“ in Zeile 5 ist dann erfüllt, wenn der Scorewert der aktuell betrachteten Überlagerung größer als die Hälfte von $score_{best}$ ist.
- Ein Scorewert gilt in Zeile 9 als „`sehr hoch`“, wenn er weniger als 10% von dem bisher besten Scorewert entfernt ist.

Mit diesen Parametern dauert eine Überlagerung bei Punktmengen bestehend aus je 25 Punkten etwa eine Sekunde. Die Laufzeit kann ohne größeren Qualitätsverlust weiter verkürzt werden, wenn die erste Bedingung in Zeile 5 verschärft wird, z.B. durch $score_{akt} \geq score_{best}/1.5$.
Auf die beste gefundene Überlagerung kann am Ende ein lokales Suchverfahren angewandt werden, indem als Nachbarschaft einer Überlagerung alle diejenigen Überlagerungen betrachtet werden, die sich in genau einem Funktionswert unterscheiden. Jede Überlagerung f erhält dabei den Wert $score^*(A, B; f)$. Bei den durchgeführten Untersuchungen war damit nur in sehr seltenen Fällen eine Verbesserung möglich, und dann war diese auch nur gering.

Algorithmus 6: Approximatives Überlagern

Input: $A \in \mathbb{R}^{3\times m}, B \in \mathbb{R}^{3\times n}, m \leq n, d > 0, \delta_G, \delta_S$

Output: Eine Überlagerung und dessen Scorewert

```
1  score_best = 0
2  F ⟵ d-beschränkte Dreiecksüberlagerung(A, B, d)
3  foreach f ∈ F do
4  |  (score_akt, f) ⟵ Greedy-Matching(A, B, δ_G)
5  |  while score_akt nicht zu klein und f noch nicht gehasht
   |  do
6  |  |  f hashen
7  |  |  (score_akt, (R, t)) ⟵ Umeyama(A, B, f) // Algorithmus 1
8  |  |  B := RB + t
9  |  |  if score_akt sehr hoch then
10 |  |  |  (score_neu, f) ⟵ Sparse-Matching(A, B, δ_S)
   |  |  else
11 |  |  |  (score_neu, f) ⟵ Greedy-Matching(A, B, δ_G)
   |  |  end
12 |  |  if score_neu ≤ score_akt then
13 |  |  |  break // while-Schleife verlassen
   |  |  else
14 |  |  |  score_akt = score_neu
   |  |  end
   |  end
15 |  if score_akt > score_best then
16 |  |  score_best = score_akt
17 |  |  f_best = f
   |  end
   end
18 return (score_best, f_best )
```

5 Experimentelle Ergebnisse

In diesem Kapitel sollen Ergebnisse vorgestellt werden, die es erlauben, die Qualität des implementierten Algoritmus einzuschätzen. Es gibt zwei „Gegner“, mit denen sich der Algorithmus gemessen hat: zum einen den exakten Überlagerungsalgorithmus und zum anderen einen von Martin Thimm entwickelten Algorithmus, der einen anderen Ansatz verfolgt, um approximativ zu überlagern. Dabei zeigt sich, dass der implementierte Algorithmus trotz einer fehlenden Gütegarantie, sehr gute Resultate erzielt.

5.1 Überlagerung von Patchen

Die folgenden Daten stammen aus der von der Arbeitsgruppe Frömmel verwalteten Datenbank DIP (Dictionary of Interfaces in Proteins), in der sogenannte Patche gespeichert sind. Ein Patch ist ein Teil eines Proteins, das mit anderen Molekülen in Wechselwirkung treten kann. In DIP finden sich etwa 10^7 Patche, wobei die meisten Patche zwischen zehn und sechzig Atomen besitzen.
Da mit dem exakten Überlagerungsalgorithmus maximal 16 Punkte in akzeptabler Zeit überlagert werden können, wurden zufällig 120 Patche mit je 16 Atomen ausgewählt und deren Scorewert paarweise sowohl mit dem exakten Überlagerungalgorithmus als auch mit dem approximativen Algorithmus berechnet.
Zu jedem der $\binom{120}{2} = 7140$ Patchvergleich wurde der Quotient aus der approximativen Lösung und dem Optimum gebildet. Die Patchvergleiche wurden derart nummeriert, dass eine monoton wachsende Funktion entsteht, die in Abbildung 4 zu sehen ist. Davon sind nur die ersten 200 Vergleiche aufgetragen, da dies genau diejenigen Patchvergleiche sind, bei denen der approximatve Algorithmus das Optimum verfehlt hat. Von diesen sind deutlich über die Hälfte nur 2% von der optimalen Lösung entfernt, und nur weniger als 10 Patchvergleiche verfehlen die optimale Lösung um mehr als 5%. Insgesamt wurde in nur 3% der Fälle das Optimum verfehlt, was zeigt, dass trotz der fehlenden Gütegarantie, der implementierte Algorithmus die Scorefunktion auf dieser Stichprobe sehr gut approximiert.

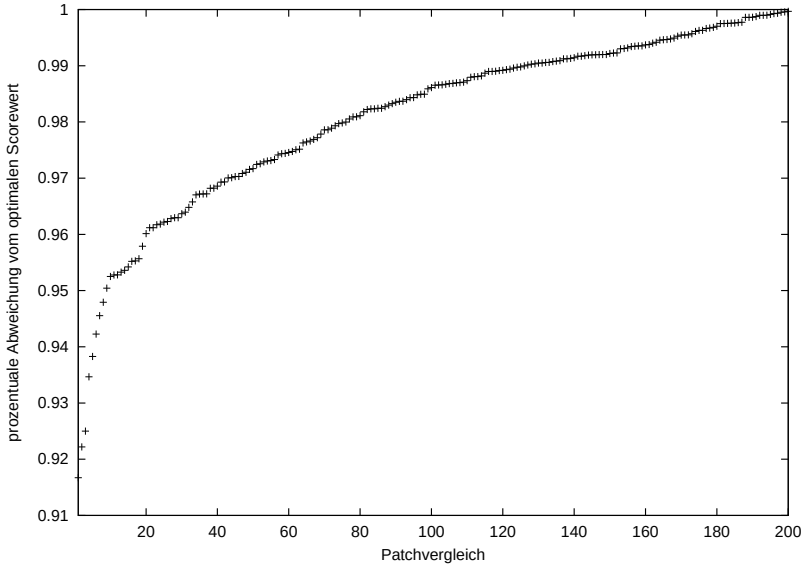

Abbildung 4: Die mit dem approximativen Algorithmus gewonnenen paarweisen Vergleiche von den Patchen sind derart sortiert, dass die Funktion monoton steigend ist. Es gibt 7140 Vergleiche, und damit ist in weniger als 3% das Optimum verfehlt worden.

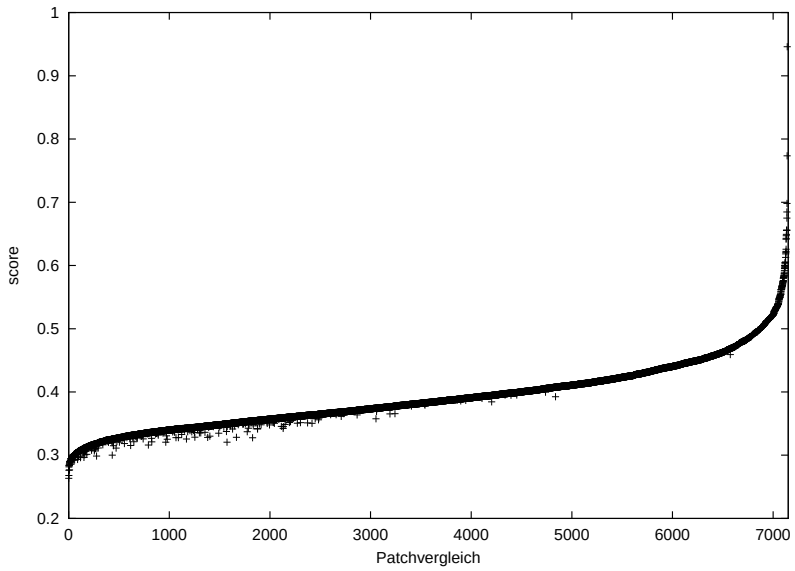

Abbildung 5: Die optimale Überlagerung ist nach aufsteigendem Scorewert sortiert worden und als breite Linie erkennbar. Die Kreuze (+) zeigen die Vergleiche an, bei denen das Optimum mit dem approximativen Algorithmus verfehlt wurde.

In der Abbildung 5 sind die 7140 Patchvergleiche nach ihrem optimalem Scorewert aufsteigend sortiert und als breite Linie erkennbar. Die 200 Patchvergleiche aus Abbildung 4, bei denen der approximative Algorithmus die optimale Überlagerung verfehlt hat, sind durch ein Pluszeichen (+) gekennzeichnet, die sich ein wenig unterhalb der breite Linie befinden. Es zeigt sich, dass der approximative Algorithmus signifikant häufiger die optimale Lösung verfehlt, wenn die Patchpaare einen geringen Ähnlichkeitswert haben.
Werden für einen exakten Patchvergleich, wobei beide Patche je 16 Atome besitzen, im Durchschnitt drei Minuten Rechenzeit zu Grunde gelegt, so ist ein Computer 15 Tage beschäftigt, um alle 7140 Vergleiche durchzurechnen. Der approximative Algorithmus ist deutlich schneller und berechnet in einer Sekunde etwa acht Überlagerungen, und ist daher nach etwa 15 Minuten fertig. Somit ist er etwa einen Faktor 1440 schneller und findet bei den hier verwendeten Daten immerhin in über 97% der Fälle die optimale Überlagerung.
Daher ist er auf jeden Fall geeignet, um beim exakten Überlagerungsalgorithmus als Preprozessing eingesetzt zu werden, damit dieser dann mit einer guten unteren Schranke für den Scorewert gestartet werden kann.

5.2 Überlagerung von Neuroleptika

Eine weitere Klasse an Daten, welche die Arbeitsgruppe Frömmel zur Verfügung gestellt hat, sind sogenannte Neuroleptika. Neuroleptika sind Moleküle, die zur Behandlung von Psychosen eingesetzt werden und in der Regel aus drei Kohlenstoffringen und einigen Seitenketten bestehen. Da einige Seitenketten flexibel sind und sich um eine feste Achse drehen können, existieren zu einem Neuroleptikum mehrere räumliche Strukturen. Diese werden Konformationen genannt. Die Arbeitsgruppe Frömmel ist insbesondere am maximalen Scorewert interessiert, der bei einem paarweisen Vergleich der Konformationsmengen zweier Neuroleptika auftritt.
Die Daten bestehen aus 26 Neuroleptika, wobei es zu jedem einzelnen etwa fünfzig Konformationen gibt. Die i-te Zeile und i-te Spalte bezeichnen dasselbe Neuroleptikum N_i.[10]
Die obere Dreiecksmatrix entsteht wie folgt. Sei i eine Zeile und j eine Spalte mit $i \leq j$. Alle Konformationen aus N_i werden mit allen Konformationen aus N_j überlagert. Der größte mit dem hier vorgestellten Approximationsalgorithmus gefundene Scorewert wird mit 100 multipliziert, auf die nächstkleinere ganze Zahl abgerundet und dann in die Tabellenzelle $[i, j]$ eingetragen. Die Diagonale erhält den Wert 100, da unter anderem identische Konformationen miteinander verglichen werden.
Mit Ausnahme der ersten beiden und den letzten sechs Zeilen bzw. Spalten wurden die gleichen Neuroleptikakonformationen mit einem von Martin Thimm entwickelten Algorithmus überlagert, wobei seine Ergebnisse sich in der unteren Dreiecksmatrix wiederfinden. Ist der Eintrag $[j, i]$ leer, so stimmt sein gefundener Wert mit dem Eintrag in $[i, j]$ überein. Andernfalls ist die Differenz von seinem Ergebnis und dem Eintrag aus $[i, j]$ angegeben. In der unteren Dreiecksmatrix sind keine positiven Werte enthalten, d.h. der zu zwei Neuroleptika gehörende maximale Scorewert, der mit dem Algorithmus von Martin Thimm berechnet wurde, ist in keinem Falle größer als der entsprechende maximale Scorewert, der mit dem hier vorgestellten Algorithmus erzielt wurde.
In Abbildung 6 sind alle Konformationsüberlagerungen aufgetragen, die auch Martin Thimm berechnet hat. Jedes Kreuz entspricht einem Vergleich. Diejenigen Vergleiche, deren Kreuze sich oberhalb der Winkelhalbierenden befinden, erzielen mit dem von Martin Thimm entwickelten Algorithmus einen höheren Scorewert als mit dem hier vorgestellten Verfahren.
Die meisten Punkte befinden sich jedoch unterhalb der Winkelhalbierenden, was die Vermutung zulässt, dass der in dieser Arbeit vorgestellte Algorithmus in der Regel höhere Scorewerte erzielt, als der Algorithmus von Martin Thimm.
Insgesamt sind 882456 Vergleiche ausgewertet worden. Zu jedem Vergleich wurde der Scorewert mit Algorithmus 6 approximiert und danach sortiert.

[10] Aus historischen Gründen existieren zu den meisten Neuroleptika verschiedene Namen.

Damit ergibt sich eine monoton steigende Folge, die in Abbildung 7 abgebildet ist.
Bei 30542 Vergleichen wurde ein Scorewert größer als 0.6736 erzielt. Mit dem exakten Überlagerungsalgorithmus konnte verifiziert werden, dass der approximative Algorithmus die optimale Lösung fast immer gefunden hatte. In Tabelle 2 sind alle 19 Vergleiche angegeben, bei denen der approximative Algorithmus einen Wert von über 0.6736 erzielt und den optimalen Scorewert verfehlt hat.

berechneter Score	exakter Score	relativer Fehler	berechneter Score	exakter Score	relativer Fehler
0.683571	0.709905	0.962905	0.680541	0.681939	0.997950
0.697507	0.678467	0.972703	0.685984	0.686077	0.999864
0.675089	0.686488	0.983395	0.742475	0.742565	0.999879
0.701558	0.691533	0.985710	0.691509	0.691577	0.999902
0.681380	0.690751	0.986434	0.753986	0.754039	0.999930
0.692731	0.700986	0.988224	0.731807	0.731832	0.999966
0.676505	0.684172	0.988794	0.693246	0.693266	0.999971
0.691018	0.685560	0.992102	0.685826	0.685834	0.999988
0.681592	0.685998	0.993577	0.718255	0.718259	0.999994
0.681136	0.683228	0.996938			

Tabelle 2: Bei 30542 Vergleichen, bei denen der Algorithmus einen Wert von über 0.6736 erzielt hat, ist nur in 19 Fällen das Optimum verfehlt worden.

Vergleiche der Neuroleptika

	0189	0349	0361	0454	1023	1295	1317	1363	1629	1683	1711	1759	1963	2092	2590	2593	2602	2713	2804	2852	—	—	—	—	—
D0000000	100	52	51	50	56	57	50	60	47	48	50	47	47	57	53	52	52	46	51	51	55	72	48	47	50	49
D0000001	x	100	52	47	50	50	50	53	47	49	50	52	45	65	56	50	43	41	53	44	60	60	48	49	53	51
D0000002	x	x	100	77	94	56	99	88	56	89	87	84	88	54	72	57	88	76	94	77	55	51	51	54	55	53
D0000003	x	x		100	78	59	74	76	56	72	77	74	66	49	71	51	88	61	87	55	47	50	45	54	47	45
D0000004	x	x			100	50	78	88	55	94	82	88	86	55	63	54	87	72	87	74	52	49	50	54	55	52
D0000005	x	x		-1	-1	100	53	49	78	52	52	52	54	58	58	51	60	57	53	52	54	59	49	62	53	51
D0000006	x	x				-3	100	87	55	85	95	84	90	53	77	54	91	75	93	76	50	51	49	51	53	51
D0000007	x	x				-1		100	49	76	94	83	77	53	70	50	93	66	88	73	50	52	49	54	51	50
D0000008	x	x		-1			-2	-1	100	52	49	54	52	51	52	48	57	55	54	52	50	55	48	79	49	49
D0000009	x	x		-1				-1		100	77	88	77	50	73	51	84	72	85	68	47	48	45	49	49	49
D0000010	x	x		-3	-1	-2			-2		100	83	77	45	76	51	96	63	86	65	48	49	44	53	46	44
D0000011	x	x	-3		-1		-5	-3	-3			100	83	52	65	57	84	76	82	74	50	48	51	51	53	54
D0000012	x	x		-2								-1	100	49	75	53	75	65	88	66	48	45	44	54	49	50
D0000013	x	x		-4	-2	-4	-2		-2	-2			-2	100	51	60	41	41	58	46	45	45	45	54	60	55
D0000014	x	x	-1		-3				-2			-1			100	55	74	68	66	66	65	51	50	50	51	48
D0000015	x	x	-1			-4			-3	-1	-1			-3		100	50	51	56	49	50	49	47	47	54	52
D0000016	x	x		-4	-4	-1	-3		-1				-2	(0/−1)[a]			100	63	87	63	46	44	43	53	40	42
D0000017	x	x		-3								-4			-4	-1		100	77	95	43	47	40	52	40	45
D0000018	x	x			-4	-3			-3					-4	-1			100	75	52	50	51	57	57	57	55
D0000019	x	x		-4		-1			-5			-1				-1			100	49	48	43	52	45	44
D0000020	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100	70	42	47	43	45
D0000021	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100	46	50	47	47
D0000022	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100	50	50	64
D0000023	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100	48	51
D0000024	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100	48
D0000025	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100

[a] Der Algorithmus von Martin Thimm ist nicht symmetrisch bezüglich A und B. Daher ergeben sich an dieser Stelle zwei verschiedene Werte.

Molekülnamen der einzelnen Konformationen

D00	haloperidol	D01	pimozide	D02	promazine_hydrochloride
D04	alimemazine_tartrate	D05	tiapride_hydrochloride	D06	chlorpromazine_hydroc
D08	sulpiride	D09	levomepromazine_maleate	D10	perphenazine_maleate
D12	thioridazine_hydrochloride	D13	risperidone	D14	chlorprothixen
D16	fluphenazin	D17	Flupentixol	D18	Prothipendyl
D20	fluspirilene	D21	penfluridol	D22	perospirone
D24	sertindole	D25	ziprasidone		

D03	NCI0014959
D07	perazine_fendizoate
D11	promethazine_hydrochloride
D15	clozapin
D19	Zuclopenthixol
D23	remoxipride

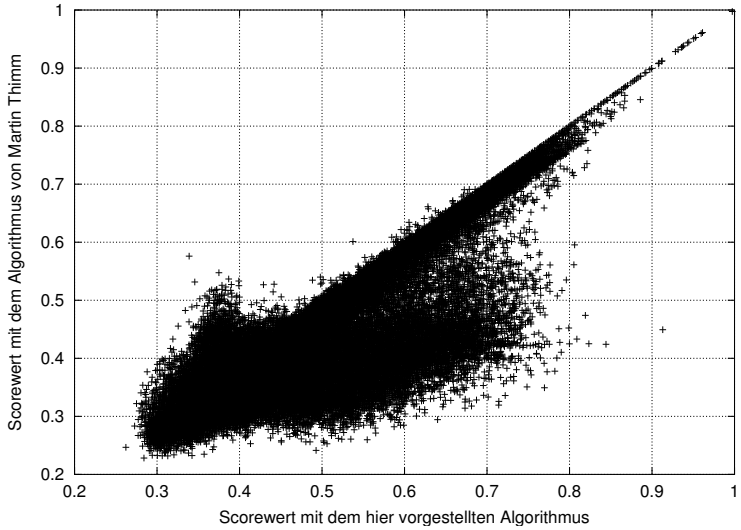

Abbildung 6: Jeder Punkt entspricht einem Konformationsvergleich. Da die meisten Punkte unterhalb der Winkelhalbierenden liegen, liefert der hier vorgestellte Algorithmus in der Regel bessere Werte als der Algorithmus von Martin Thimm

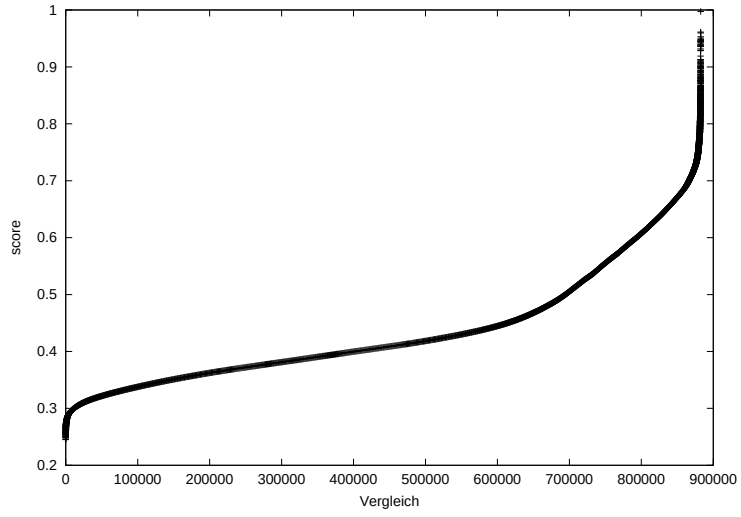

Abbildung 7: Die Konformationsvergleiche sind aufsteigend nach ihrem approximierten Scorewert sortiert.

6 Ausblick

Das theoretische Hauptergebnis in dieser Arbeit besteht sicherlich darin, dass die Existenz eines PTAS bewiesen werden konnte. Da sich dieses jedoch für die praktische Anwendung als nicht effizient erwiesen hat, wurde ein Algorithmus entworfen und implementiert, der sowohl hinsichtlich der Laufzeit als auch der Qualität der Ergebnisse gute Resultate liefert.
Am Ende dieser Arbeit sollen einige offene Fragestellungen vorgestellt werden, deren Beantwortung zu einem besseren Verständnis des Überlagerungsproblems führen könnte.

1. Ist das Überlagerungsproblem $\mathcal{NP}$-vollständig? Falls dies bewiesen werden könnte, so ist die Schwierigkeit des Problems vom komplexitätstheoretischen Standpunkt aus ziemlich genau klassifiziert. Es blieb noch die Frage offen, ob auch ein FPTAS existiert. Der Kehrwert des ϵ-Terms geht zwar zur neunten Potenz in die Laufzeit ein, hängt aber immerhin nicht von n ab, wie es bei manchen anderen Algorithmen, die ein PTAS liefern, der Fall ist.

   Die Schwierigkeit, die $\mathcal{NP}$-Vollständigkeit nachzuweisen, liegt in der unhandlichen Scorefunktion. Ein Ansatz wäre, einen Spezialfall des Problems als $\mathcal{NP}$-vollständig nachzuweisen. Denkbar wären z.B.

   - dass die Punkte aus dem zweidimensionalem Raum stammen oder
   - dass statt Bewegungen nur Translationen erlaubt sind.

2. Der implementierte Algorithmus besitzt keine Gütegarantie. Lässt sich der Algorithmus derart modifizieren, dass die Laufzeit in der Praxis noch akzeptabel bleibt und eine brauchbare Güte bewiesen werden kann?

3. Im Rahmen des Projekts mit der Arbeitsgruppe Frömmel ist von Martin Thimm ein vollkommen anderer Ansatz entwickelt worden, um approximativ zu überlagern. Es ist denkbar, dass sich die Vorteile aus beiden Verfahren kombinieren lassen, um schnellere oder auch qualitativ bessere Lösungen zu erhalten.

4. Gibt es schärfere Schranken für einen Branch-and-Cut-Algorithmus, so dass das Überlagerungsproblem in vertretbarer Zeit mit mehr als 16 Atomen exakt berechnet werden kann? Es ist denkbar, dass solche Schranken auch bei approximativen Verfahren hilfreich wären.

5. Können Strukturinformationen in den Daten ausgenutzt werden (z.B. C_6-Ringe), um ohne Qualitätseinbußen schneller zu überlagern?

6. Das Überlagerungsproblem lässt sich auf den $\mathbb{R}^n$ erweitern. Dann fixieren Dreiecksüberlagerung die Bewegung nicht mehr. Wenn stattdessen

n-Simplizes überlagert werden, so dürfte das einen raschen Anstieg der Laufzeit nach sich ziehen. Was für Techniken sind hier Erfolg versprechend?

Literatur

[1] T. Akutsu, *Protein Structure Alignment Using a Graph Matching Technique*, Genome Informatics 6, 1–8, (1995)
`http://www.genome.ad.jp/manuscripts/GIW95/GIW95.html`

[2] D. Avis, *A Survey of Heuristics for the Weighted Matching Problem*, Networks, Vol.13 (1983), 475–493

[3] J.R. Brown, *Chromatic scheduling and the chromatic number problem*, Management Science, Volume 19, 1972, 456–463

[4] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, Vol. 1, pp.269–271, 1959

[5] D. Drake and S. Hougardy, *A Simple Approximation Algorithm for the Weighted Matching Problem*, Information Processing Letters 85 (2003), 211-213

[6] J.Edmonds, R.M. Karp, *Theoretical improvements in the algorithmic efficiency for network flow problems*, Journal of the ACM, 19:248–264, 1972

[7] M.L. Fredman and R.E. Tarjan, *Fibonacci heaps and their uses in improve network algorithms*, Journal of the Association for Computing Machinery, 34:595–615, 1987

[8] C. Gröpl, *Algorithmen in der Bioinformatik*, Vorlesungsskript 2002/2003, 17–18, Humboldt-Universität zu Berlin
`http://www.informatik.hu-berlin.de/Institut/struktur/algorithmen/lehre/lvws0203/script_algbioinf/`

[9] E.W. Mayr, *Praktikum Algorithmen-Entwurf (Teil 6)*, Nov. 2002, 6–11, Technische Universität München
`http://wwwmayr.in.tum.de/lehre/2002WS/algoprak/part6.ps.gz`

[10] W.H. Press, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes in C*, Second Edition, Kapitel 2.6, 59–70, auch online verfügbar unter
`http://www.library.cornell.edu/nr/bookcpdf/c2-6.pdf`

[11] E.M. Reingold and R.E. Tarjan, *On a greedy heuristics for complete matching*, SIAM J.Comput. 10 (1981), 676–681

[12] J. Stoer, R. Bulirsch, *Einführung in die Numerische Mathematik II*, Springer-Verlag, 2. Auflage, 1978, S. 18–20

[13] R.E. Tarjan, *Data Structure and Network Algorithms*, vol. 44. SIAM, 1983

[14] S. Umeyama, *Least-Squares Estimation of Transformation Parameters Between Two Point Patterns*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, No. 4; April 1991, 676–681

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen, mich bei allen Leuten zu bedanken, die direkt oder indirekt zu dieser Arbeit beigetragen haben.
Zunächst seien meine Eltern Christiane und Gerhard Kirchner erwähnt, die mich im Laufe der letzten fünf Jahren nicht nur in finanzieller Hinsicht bei meinem Studium unterstützt und so mir die Möglichkeit gegeben haben, mich darauf zu konzentrieren.
Meinen weiteren Dank gilt Stefan Hougardy, der mit seiner Vorlesung „Graphen und Algorithmen“ vor knapp drei Jahren mein Interesse für diskrete Mathematik im Allgemeinen und im Besonderen für das Teilgebiet der Graphentheorie geweckt hat. In meiner Zeit als studentische Hilfskraft und in den vergangenen Monaten habe ich seine ständige und hilfsbereite Betreuung, insbesondere bei meinen Arbeiten über Steinerbäume als auch bei dem Bioprojekt, aus dem das Thema dieser Arbeit entsprungen ist, immer sehr zu schätzen gewusst.
Ihm und Anusch Taraz seien gedankt, dass sie diese Diplomarbeit begutachten.
Weiter seien Martin Thimm, Clemens Gröpl und Till Nierhoff gedankt, die jederzeit und besonders im Bioprojekt stets hilfsbereite Ansprechpartner waren und die viele Fragen meinerseits beantworten konnten und zusätzlich neue Fragen aufgeworfen haben.
Zuletzt möchte ich mich bei Xenia Rendtel, Peter Liske, Mike Löffler, Andreas Schlick und insbesondere Dirk Schlatter bedanken, dass sie diese Arbeit durchgelesen und wertvolle Hinweise gegeben haben.

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet zu haben.
Ich bin damit einverstanden, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin öffentlich ausgelegt wird.

Berlin, den 12. September

Stefan Kirchner