Word Mover's Distance angewendet auf die Paraphrasenextraktion im Altgriechischen

Marcus Pöckelmann / Jörg Ritter / Paul Molitor

Abstract In diesem Beitrag wird eine Heuristik zur Paraphrasenextraktion vorgestellt, die auch bei großen Textkorpora in interaktiven Anwendungen eingesetzt werden kann. Es wird gezeigt, wie der in 2015 von Kusner et al. eingeführte Word Mover's Distance Ansatz zur Dokumentenklassifikation auf die Paraphrasensuche angepasst werden kann, so dass nur für verhältnismäßig wenige Textsegmente aus dem Textkorpus die teure Word Mover's Distance zu einer Suchanfrage berechnet werden muss. Die Heuristik wurde auf dem aus 38 Millionen Wörtern (ohne Stoppwörter) bestehenden altgriechischen Textkorpus Thesaurus Linguae Graecae (TLG)[1] evaluiert, in dem nach Paraphrasen aus Platons Werk gesucht wurde. Die experimentellen Ergebnisse zeigen, dass die Qualität der Heuristik nur minimal von der Qualität des sehr rechenzeitintensiven Brute-Force-Ansatzes abweicht, bei dem zu jedem Textsegment aus dem Textkorpus die Word Mover's Distance zu der Suchanfrage berechnet wird. Die Laufzeiten der Heuristik bewegen sich in einem Bereich, der ein effektives flüssiges Arbeiten der geisteswissenschaftlichen Nutzer erlaubt.

Keywords Paraphrasen, Paraphrasenextraktion, Word Mover's Distance, WMD, Word2Vec, Griechische Literatur, Platon

1 Einleitung

Kusner et al.[2] stellten kürzlich eine neue Distanzfunktion, die *Word Mover's Distance* (WMD), vor, um Textdokumente zu klassifizieren. Das von ihnen vorgestellte Maß führt zu weit besseren Klassifikationen als die bisher aus der Literatur bekannten Ansätze wie beispielsweise BOW (*Bag-Of-Words*)[3], TF-IDF (*Term Frequency - Inverse*

1 Das für die Evaluierung der Heuristik verwendete Textkorpus ist der Thesaurus Linguae Graecae in der Version E (TLG-E), im Folgenden abgekürzt als TLG zitiert.

2 Kusner (2015).

3 Zellig (1954).

Document Frequency)[4], BM25 Okapi[5], LSI (*Latent Semantic Indexing*)[6] oder LDA (*Latent Dirichlet Allocation*)[7]. Eine detaillierte Analyse der Methoden in Bezug auf das Problem der Dokumentenklassifikation findet man in der Arbeit von Kusner et al.

Eine der zentralen Anwendungen der Dokumentenklassifikation ist das Finden von Dokumenten, die zu einem gegebenen Dokument semantisch ähnlich sind. Da die Laufzeit der Berechnung der WMD asymptotisch mehr als kubisch[8] in der Anzahl des Vokabulars der Dokumente ist, stellen Kusner et al. im zweiten Teil ihrer Arbeit zwei bei der genannten Anwendung effizient zu berechnende untere Schranken der WMD vor, mit denen es möglich ist, die Berechnung der WMD zwischen den meisten Dokumenten zu vermeiden.

Ein ähnliches Problem zur Dokumentenklassifikation ist das Auffinden von Paraphrasen zu einer gegebenen Textpassage in einem Korpus. Dieses Problem, oder besser gesagt, das Problem, gute Kandidaten für Paraphrasen zu finden,[9] kann man derart angehen, dass man nach Textsegmenten im Korpus sucht, die zu dem gegebenen Textsegment eine große semantische Ähnlichkeit haben. Um alle guten Kandidaten für Paraphrasen eines gegebenen Textsegments S der Länge m, d. h. mit m Wörtern, zu finden, muss für jedes im Korpus enthaltene Textsegment Ω einer bestimmten Länge die Distanz zu S berechnet oder abgeschätzt werden. Die Textsegmente mit den kleinsten Distanzen zu S werden als Kandidaten für Paraphrasen von S vorgeschlagen. Die Herausforderung besteht bei dieser Anwendung in der Anzahl der Textsegmente, für welche die Distanz zu berechnen ist. Wenn die Länge m der Suchanfrage S klein ist – und das ist in der Regel der Fall – so haben wir bei einem aus N Wörtern bestehenden Korpus bis zu N WMD-Berechnungen durchzuführen, was zu sehr hohen Berechnungszeiten führen kann.

In dem durch die VolkswagenStiftung geförderten Projekt *Platon Digital: Tradition und Rezeption*, an dem wir beteiligt sind, arbeitet eine Gruppe von Geisteswissenschaftler(innen) zusammen mit Korpuslinguist(inn)en und Informatiker(innen), die Nachwirkungen von Platons Werk in der altgriechischen Literatur heraus. Es werden hierbei alle im TLG enthaltenen griechischen Texte betrachtet. Die im TLG enthaltenen Texte umfassen circa 75 Millionen griechische Wörter bzw. 38 Millionen Wörter ohne Stoppwörter. Textpassagen aus Platons Werk, für

4 Salton (1988).
5 Robertson (1994).
6 Deerwester (1990).
7 Blei (2003).
8 Die Berechnungskomplexität liegt in $O(n^3 \cdot \log n)$, wobei n die Größe des Vokabulars ist.
9 Letztendlich haben am Ende des Tages Fachwissenschaftler(innen) zu entscheiden und inhaltlich zu begründen, ob ein durch einen Algorithmus vorgeschlagenes Textsegment eine Paraphrase der gegebenen Textpassage ist.

die Paraphrasen im TLG gesucht werden, haben typischerweise eine Länge m von 3 bis 50 Wörtern. Um gute (bzw. alle guten) Kandidaten für Paraphrasen zu einer gegebenen Textpassage S zu finden, sind die Distanzen zwischen jedem Textsegment Ω einer bestimmten Länge aus dem TLG und der Suchanfrage S zu berechnen beziehungsweise abzuschätzen. Dies führt zu bis zu 38 Millionen Distanz-Berechnungen, was bei weitem zu viel für interaktive Anwendungen ist. In diesem Beitrag werden wir einen heuristischen Ansatz vorstellen, mit dem man die Paraphrasenberechnung in „Realzeit" durchführen kann. Die durch die Heuristik jeweils berechneten besten 500 Kandidaten für eine gegebene Paraphrase entsprechen in der Regel den durch den Brute-Force-Ansatz jeweils berechneten besten 500 Kandidaten.

Der vorliegende Beitrag ist wie folgt strukturiert: Nach einer kurzen Einführung in die Word Mover's Distance (WMD) und ihrer Berechnung (Kapitel 3) wird die von uns vorgeschlagene Anpassung der WMD für das Problem der Paraphrasensuche in Kapitel 4 vorgestellt. Die experimentellen Ergebnisse zu der Güte des neuen Ansatzes werden in Kapitel 5 beschrieben und diskutiert. Der Beitrag schließt mit einem Fazit und einem Ausblick.

2 Notationen

In diesem Beitrag wird das Symbol S die gegebene Textpassage bezeichnen, für welche Paraphrasen gesucht werden. Mit m bezeichnen wir die Länge der Suchanfrage S, d. h. die Anzahl der Wörter in S. Das Symbol K steht für das Korpus, in dem nach Paraphrasen gesucht wird, und Ω für ein Textsegment des Korpus K, für das geprüft wird, ob es ein guter Kandidat für eine Paraphrase von S ist. N bezeichnet die Länge des Korpus K, ebenfalls in Wörtern gemessen, und n die Größe des Vokabulars des Korpus K. Wörter aus dem Vokabular werden in der Regel mit den Buchstaben i, j, u, v und w angegeben. S_r bzw. K_r steht für das r-te Wort der Suchanfrage S bzw. des Korpus K. Ist j ein Wort des Vokabulars, so bezeichnet $\vec{j}$ eine Einbettung von j im Euklidischen Raum (siehe Kapitel 3.1). Für je zwei Wörter u und v des Vokabulars gibt $\|\vec{u}-\vec{v}\|_2$ die Euklidische Distanz zwischen den beiden Vektoren $\vec{u}$ und $\vec{v}$ an. Die Anzahl der Kandidaten für Paraphrasen zu der gegebenen Suchanfrage, die der Algorithmus zurückgeben und dem / der Fachwissenschaftler / in vorlegen soll, bezeichnen wir mit q. Die Zahl p, die in den Algorithmen verwendet wird, ist ein konstantes Vielfaches von q, d. h. $p = \beta \cdot q$ für eine von der Länge m der Suchanfrage abhängige Konstante $\beta \geq 1$.

3 Word2Vec und Word Mover's Distance

Unser Ansatz zum Auffinden guter Kandidaten für Paraphrasen von einer gegebenen Textpassage S in einem gegebenen Korpus K wendet die von Kusner et al. für das Problem der Dokumentenklassifikation eingeführte *Word Mover's Distance* (WMD) an.[10] Die WMD zwischen zwei Dokumenten gibt die Kosten an, um das eine Dokument in das andere zu überführen. Zur Berechnung der Distanz zwischen zwei Wörtern werden die Wörter des Vokabulars in Bezug auf das Gesamtkorpus in den Euklidischen Raum eingebettet. Hier benutzen Kusner et al. das von Mikolov et al. eingeführte *Word2Vec*.[11] Die Distanz zwischen zwei Wörtern ergibt sich durch die Euklidische Distanz zwischen den beiden dazugehörigen Einbettungen.

Um unsere Anpassung von WMD für die Paraphrasenextraktion zu verstehen, müssen wir auf einige Details von *Word2Vec* und der Berechnung der WMD eingehen.

3.1 Worteinbettungen mittels Word2Vec

Word2Vec wurde in 2013 durch Mikolov et al. eingeführt.[12] Es ist eine Methode, die jedem Wort j des Vokabulars eines Korpus K einen mehrdimensionalen Vektor $\vec{j}$ derart zuweist, dass Einbettungen von Wörtern, die im Korpus in gleichen Kontexten auftreten, nahe beieinander liegen. Je größer das Korpus K, auf dem die Worteinbettungen gelernt wurden, um so besser sind die Worteinbettungen in Bezug auf semantische Ähnlichkeit. Eine naheliegende Möglichkeit, zwei Wörter u und v miteinander zu vergleichen, besteht dann darin, die Euklidische Distanz $\|\vec{u}-\vec{v}\|_2$ zwischen den Worteinbettungen $\vec{u}$ und $\vec{v}$ oder die Kosinusähnlichkeit der beiden Vektoren $\vec{u}$ und $\vec{v}$ zu berechnen.

Neben der semantischen Ähnlichkeit von Wörtern mit nahe beieinander liegenden Einbettungen hat *Word2Vec* die Eigenschaft, dass man mit den Worteinbettungen rechnen kann. Speziell kann man sich über arithmetische Operationen semantische Analogien berechnen lassen. Beispielsweise fragt der Ausdruck $\overrightarrow{\text{König}} - \overrightarrow{\text{Mann}} + \overrightarrow{\text{Frau}}$ nach einem Wort, das zu dem Wort ‚Frau' steht wie das Wort ‚König' zu dem Wort ‚Mann', und berechnet (bei geeignetem Korpus K, auf dem die Worteinbettungen gelernt worden sind) einen Vektor, der sehr nahe an der

10 Kusner (2015).
11 Mikolov (2013a).
12 Mikolov (2013a).

Einbettung $\overrightarrow{\text{Königin}}$ liegt.[13] Der Ausdruck $\overrightarrow{\text{schlecht}} - \overrightarrow{\text{gut}} + \overrightarrow{\text{hübsch}}$ gibt einen Vektor zurück, der nahe an der Einbettung $\overrightarrow{\text{hässlich}}$ liegt, d. h.

$$\left\| (\overrightarrow{\text{schlecht}} - \overrightarrow{\text{gut}} + \overrightarrow{\text{hübsch}}) - \overrightarrow{\text{hässlich}} \right\|_2 \approx 0.$$

Worteinbettungen sind nicht neu. *Word2Vec* unterscheidet sich jedoch von anderen Techniken zur Worteinbettung dadurch, dass es auf Korpora mit einigen Milliarden von Wörtern und einem Korpus-Vokabular mit einigen Millionen von Wörtern angewendet werden kann. Details zu *Word2Vec* findet man in den Arbeiten von Mikolov et al.[14] und Wehrung[15].

3.2 Word Mover's Distance[16]

Die *Word Mover's Distance* (WMD) ist eine neue Distanz zwischen zwei Textdokumenten. Sie wurde in 2015 durch Kusner et al. vorgestellt.[17]

Die WMD kann mit einem beliebigen Distanzmaß auf Wörtern arbeiten, sofern das Maß die Eigenschaft hat, dass zwei Wörter, deren Distanz zueinander klein ist, zu einem gewissen Grad semantisch ähnlich sind. Kusner et al. benutzen in ihrer Arbeit die Euklidische Distanz zwischen den *Word2Vec*-Einbettungen der Wörter.

Die Word Mover's Distance $WMD(A, B)$ zwischen zwei Textdokumenten A und B ist definiert als die minimale kumulative Distanz, um die Wörter des Textdokuments A derart zu transformieren, dass das Dokument B entsteht. Jedes Wort i des Dokuments A wird im Ganzen oder nur partiell in ein Wort j des Dokuments B „transportiert". Bezeichnet T_{ij} den Anteil, wie viel von Wort i in das Wort j transportiert wird, so belaufen sich die dazugehörigen Überführungskosten auf $T_{ij} \cdot \left\| \vec{i} - \vec{j} \right\|_2$. Das Optimierungsziel besteht darin, die Gesamtkosten zu minimieren, um das Dokument A in das Dokument B zu transportieren.

Um dies formal zu modellieren, wird ein Textdokument D durch einen sogenannten normalisierten *bag-of-words*-Vektor (BOW-Vektor)[18] $d = (d_1, \ldots, d_n)$ dargestellt, dessen Dimension n gleich der Größe des Vokabulars ist. Wenn das Wort i des Vokabulars c_i-mal im Dokument D enthalten ist, so hat d_i den Wert $d_i = c_i / \sum_{j=1}^{n} c_j$.

13 Mikolov (2013b).
14 Mikolov (2013a).
15 Wehrung (2017).
16 Wir folgen bei den Ausführungen in Kapitel 3.2 der Originalarbeit von Kusner (2015).
17 Kusner (2015).
18 Zellig (1954).

Die Word Mover's Distance $WMD(A, B)$ zwischen zwei Textdokumenten A und B (mit ihren BOW-Vektoren a und b) ist dann durch das lineare Programm

$$\min_{T \geq 0} \sum_{i=1}^{n} \sum_{j=1}^{n} T_{ij} \cdot \left\| \vec{i} - \vec{j} \right\|_2$$

mit

$$\forall i \in \{1, \ldots, n\} \quad \sum_{j=1}^{n} T_{ij} = a_i$$

$$\forall j \in \{1, \ldots, n\} \quad \sum_{i=1}^{n} T_{ij} = b_j$$

gegeben. Die erste Nebenbedingung besagt, dass jedes Wort des Dokuments A in Gänze nach B transportiert werden muss, die zweite Nebenbedingung, dass jedes Wort des Dokuments B in Gänze aus A heraus „erzeugt" werden muss. Ein solches Optimierungsproblem ist ein Spezialfall eines Transportproblems, das in der Literatur als *Earth Mover's Distance Metrik* bekannt ist.[19] Die beste bekannte asymptotische mittlere Laufzeit, um lineare Programme dieser Art zu lösen, liegt in $O(n^3 \cdot \log n)$,[20] was je nach Probleminstanz bereits zu recht hohen Laufzeiten führen kann.

Um das Problem, zu einem gegebenen Textdokument S die q „ähnlichsten" Textdokumente eines Korpus effizient zu berechnen, zu lösen, führen Kusner et al.[21] zwei untere Schranken für das obige Transportproblem ein: die *Word Centroid Distance* (WCD) und die *Relaxierte Word Mover's Distance* (RWMD). Die unteren Schranken führen dazu, dass auf die Berechnung der WMD zwischen der Mehrzahl der Textdokumente und dem gegebenen Textdokument S verzichtet werden kann. Jedes Dokument Ω für das wenigstens eine der beiden unteren Schranken für $WMD(\Omega, S)$ echt größer als die WMD-Distanzen von q bereits gefundenen Textdokumenten ist, braucht nicht weiter betrachtet zu werden.

Der Ansatz zur Berechnung der q zu einem gegebenen Textdokument S ähnlichsten Textdokumente aus einem gegebenen Korpus besteht darin, zuerst die von der Laufzeit her billige WCD zwischen jedem Textdokument Ω aus dem Korpus und Dokument S zu berechnen, dann die Dokumente des Korpus nach ihrer WCD

19 Rubner (1998).

20 Pele (2009).

21 Kusner (2015).

zu S in aufsteigender Reihenfolge zu sortieren und für die q ersten Dokumente die exakte WMD zu S zu berechnen. Dann werden die restlichen Textdokumente, deren WCD kleiner als die WMD der bis dahin q besten Dokumente ist, betrachtet. Für jedes dieser Dokumente wird die RWMD zu S berechnet. Ist diese untere Schranke größer als das bis dahin q-te beste Dokument, so braucht das Textdokument nicht weiter betrachtet zu werden. Wenn nicht, dann wird die exakte WMD zwischen dem Dokument und S berechnet und die Liste der q besten Textdokumenten muss gegebenenfalls aktualisiert werden.

3.3 Relaxierte Word Mover's Distance (RWMD)

Die Grundidee von RWMD ist recht einfach.[22] Entfernt man die zweite Nebenbedingung oder die erste Nebenbedingung aus dem obigen linearen Programm, so führt dies offensichtlich jeweils zu einer unteren Schranke für WMD. Die Lösung $T^{(1)}$ des ersten relaxierten Problems, bei dem nur die erste Nebenbedingung zu erfüllen ist, ist durch

$$T_{ij}^{(1)} = \begin{cases} a_i, & \text{wenn } j = \underset{u \in B}{\operatorname{argmin}} \left\| \vec{i} - \vec{u} \right\|_2 \\ 0, & \text{sonst} \end{cases}$$

gegeben, die Lösung $T^{(2)}$ des zweiten relaxierten Problems durch

$$T_{ij}^{(2)} = \begin{cases} b_j, & \text{wenn } i = \underset{u \in A}{\operatorname{argmin}} \left\| \vec{u} - \vec{j} \right\|_2 \\ 0, & \text{sonst.} \end{cases}$$

Dies führt dann zu den beiden unteren Schranken

$$lb_1(A, B) = \sum_{i=1}^{n} \sum_{j=1}^{n} T_{ij}^{(1)} \cdot \left\| \vec{i} - \vec{j} \right\|_2$$

$$lb_2(A, B) = \sum_{i=1}^{n} \sum_{j=1}^{n} T_{ij}^{(2)} \cdot \left\| \vec{i} - \vec{j} \right\|_2.$$

22 Wir verwenden in unserer Anwendung nur RWMD. Die zweite untere Schranke WCD ist bei der in unserer Arbeit betrachteten Anwendung der Paraphrasenextraktion keine wirklich gute untere Schranke. Zudem ist WCD aufgrund des großen in der altgriechischen Literatur verwendeten Vokabulars recht zeitaufwändig zu berechnen.

Die beiden unteren Schranken können berechnet werden, indem jeweils nach dem in dem Euklidischen *Word2Vec*-Raum nächst liegenden im Gegendokument enthaltenen Wort gesucht wird. Die relaxierte Word Mover's Distance $RWMD(A, B)$ zwischen zwei Texten A und B ist dann das Maximum

$$RWMD(A, B) = \max\{lb_1(A, B), lb_2(A, B)\}$$

der beiden unteren Schranken $lb_1(A, B)$ und $lb_2(A, B)$ und ist natürlich auch eine untere Schranke für die WMD zwischen A und B.

4 WMD basierte Paraphrasenextraktion

Lassen Sie uns nun detaillierter auf den Anwendungsfall der Paraphrasenextraktion eingehen. Wir wollen in einem gegebenen großen[23] Korpus K zu einer gegebenen, aus m Wörtern bestehenden Textpassage S die q vielversprechendsten Textstellen finden, an denen S paraphrasiert sein könnte. Wir beschränken uns der Einfachheit halber in diesem Artikel auf Textsegmente aus K, deren Länge der Länge m der Suchanfrage S entspricht.[24]

Ein eher naiver Ansatz, dieses Paraphrasenextraktionsproblem zu lösen, besteht darin, dass wir N-m+1 viele WMD-Berechnungen durchführen, nämlich eine WMD-Berechnung zu S für jedes im Korpus K enthaltene Textsegment Ω der Länge m und dann die Textsegmente nach ihrer WMD-Distanz zu S sortieren. Da die Textpassagen, für die Paraphrasen in K gesucht werden, in der Praxis typischerweise kurz sind – in der Regel haben sie eine Länge m von 3 bis 50 Wörtern –, führt dieser Ansatz größenordnungsmäßig zu N WMD-Berechnungen. Bei einem großen Korpus, d. h. bei großem N, benötigen diese N WMD-Berechnungen Laufzeiten, die in interaktiven Anwendungen nicht akzeptabel sind.

Bevor wir uns überlegen, wie wir diesen naiven Ansatz beschleunigen können, wollen wir auf die folgenden zwei Beobachtungen hinweisen, die in Bezug auf ein schnelleres Verfahren berücksichtigt werden müssen bzw. sollten:

---

23 In dem durch die VolkswagenStiftung geförderten Projekt *Platon Digital: Tradition und Rezeption* verwenden wir als Korpus K den TLG, der aus ungefähr 38 Millionen Wörtern besteht, nachdem die Stoppwörter entfernt worden sind. Wir betrachten also in diesem Projekt ein Korpus, dessen Länge N im zweistelligen Millionenbereich liegt. Die Größe n des im TLG verwendeten Vokabulars ist in etwa eine Million.

24 Eine Verallgemeinerung der in diesem Beitrag vorgestellten Heuristik auf variable Längen ist möglich und führt zu einer Heuristik, die in etwa die gleiche Laufzeit hat wie der hier betrachtete Spezialfall.

a) Es ist speicherplatzaufwändig, die Euklidischen Distanzen $\|\vec{i}-\vec{j}\|_2$ für jedes Paar von Wörtern i und j des Vokabulars des Korpus K vorzuberechnen. Bei einem Vokabular bestehend aus n Wörtern, sind $\frac{1}{2} \cdot n^2$ Einträge abzuspeichern. Besteht das Vokabular, wie bei dem altgriechischen Textkorpus aus einer Million oder mehr Wörtern, so führt dies zu einem Gesamtspeicherplatzbedarf von 2 bis 4 und mehr Terabyte.[25]

b) Es ist wesentlich einfacher, zu einer Suchanfrage S die untere Schranke $lb_1(\Omega, S)$ für alle Textsegmente Ω der Länge m aus dem Korpus K zu berechnen als alle $lb_2(\Omega, S)$. Schiebt man ein Fenster der Länge m jeweils um eine Position über das Korpus K, dann kann man die Berechnung der ersten unteren Schranke $lb_1(\Omega_{\text{neu}}, S)$ des unter dem neuen Fenster ab Position $r{+}1$ liegenden Textsegmentes $\Omega_{\text{neu}} = (K_{r+1}, \ldots, K_{r+m})$ aus der unteren Schranke $lb_1(\Omega_{\text{alt}}, S)$ des vorherigen Fensters $\Omega_{\text{alt}} = (K_r, \ldots, K_{r+m-1})$ berechnen, indem man den Summanden des links aus dem Fenster herausfallenden Wortes K_r von der unteren Schranke subtrahiert und den Summanden des rechts in das Fenster erscheinenden Wortes K_{r+m} auf die untere Schranke addiert. Die restlichen Summanden bleiben unverändert, da sich die Suchanfrage S selbst nicht verändert. Die Berechnung der zweiten unteren Schranke lb_2 ist aufwändiger, da alle in die Berechnung von lb_2 eingehenden Summanden durch das Verschieben des Fensters aktualisiert werden müssen, auch wenn das Fenster nur um eine Position weitergerückt wird.

4.1 Algorithmus zur exakten WMD basierten Paraphrasensuche

Wir verstehen unter *exakter* WMD basierter Paraphrasensuche einen Algorithmus, welcher für eine gegebene Suchanfrage S der Länge m und einer gegebenen Zahl q die q in Bezug auf die WMD zu S besten im Korpus K enthaltenen Textsegmente $\Omega_1, \ldots, \Omega_q$ der Länge m berechnet. Es sind Textsegmente $\Omega_1, \ldots, \Omega_q$ der Länge m aus dem Korpus K zu berechnen, so dass es im Korpus K kein Textsegment $\Omega \notin \{\Omega_1, \ldots, \Omega_q\}$ der Länge m gibt mit

$$WMD(\Omega, S) < \max_{1 \le k \le q} WMD(\Omega_k, S).$$

Die zwei oben gemachten Beobachtungen führen direkt zu einem Ansatz zum Lösen dieses Problems, der auf den ersten Blick effizienter sein sollte als der naive Brute-Force-Ansatz, der zu jedem Textsegment Ω des Korpus K die WMD zu S

25 Jeder Eintrag besteht aus mindestens einer Gleitkommazahl, benötigt also 4 oder 8 Byte. Da das Distanzmaß symmetrisch ist, müssen statt n^2 „nur" $\frac{1}{2} \cdot n^2$ Einträge gespeichert werden, so dass sich der Platzbedarf auf $4 \cdot \frac{1}{2} \cdot n^2$ Byte bzw. $8 \cdot \frac{1}{2} \cdot n^2$ Byte beläuft. Ist $n = 10^6$, so ergibt sich ein Platzbedarf von $2 \cdot 10^{12}$ bzw. $4 \cdot 10^{12}$ Byte.

berechnet und die Textsegmente dann in aufsteigender Reihenfolge nach ihrer WMD zu S sortiert.

Sei $S = S_1 S_2 \ldots S_m$ die gegebene Suchanfrage bestehend aus den Wörtern S_1, S_2, …, S_m:

1. Berechne die Euklidischen Distanzen $\|\vec{j} - \vec{S_k}\|_2$ zwischen jedem Wort j aus dem Vokabular des Korpus K und jedem Wort S_k aus der Suchanfrage S. Wenn für ein Wort S_k aus der Suchanfrage S die Distanzen zu allen Wörtern aus dem Vokabular schon berechnet worden sind, lade den entsprechenden Wertevektor aus dem Cache[26]. Als Ergebnis dieses ersten Schrittes erhält man m Vektoren der Dimension n. Diese m Vektoren werden für die Dauer der aktuellen Suchanfrage im Hauptspeicher gehalten.[27]
2. Berechne für jedes Wort j aus dem Vokabular des Korpus K das ähnlichste Wort S_{k_j} aus S, d. h.

   $$k_j = \underset{1 \leq k \leq m}{\operatorname{argmin}} \|\vec{j} - \vec{S_k}\|_2.$$

   Sei $dist_s(j)$ die entsprechende minimale Euklidische Distanz $\|\vec{j} - \vec{S_{k_j}}\|_2$. Das Feld $dist_s$ wird für die Dauer der Berechnung der aktuellen Suchanfrage im Hauptspeicher abgelegt.[28]
3. Berechne die untere Schranke

   $$lb_1(\Omega, S) = \sum_{j \in \Omega} \omega_j \cdot dist_s(j).$$

   für jedes Textsegment Ω (mit seinem *bag-of-words*-Vektor ω) aus dem Korpus K der Länge m. Schiebt man ein Fenster der Länge m wortweise über das Korpus K, so benötigt die Berechnung der unteren Schranke $lb_1(\Omega, S)$ des aktuell unter dem Fenster erscheinenden Textsegments Ω, wie oben in Beobachtung (b) beschrieben, konstante Zeit.
4. Sortiere die Textsegmente der Länge m des Korpus K in aufsteigender Reihenfolge nach ihren lb_1-Schranken. Sei $List_{lb_1}$ diese Liste.

---

26 Ein Cache ist in diesem Zusammenhang ein auf der Festplatte realisierter logischer Speicher, in den bereits berechnete Werte hinterlegt werden, um eine Neuberechnung zu vermeiden.

27 Bei einem Vokabular, das ungefähr 1.000.000 Wörter enthält, und Suchanfragen, die eine Länge von bis zu 50 Wörtern haben, belegen diese Vektoren jeweils Speicherplatz von ungefähr einem halben Gigabyte.

28 Das Feld $dist_s$ belegt bei einem Vokabular mit ungefähr 1.000.000 Wörtern um die 8 Megabyte Speicherplatz.

5. Berechne die (exakte) WMD zu S für die ersten p Textsegmente der Liste $List_{lb_1}$[29] und sortiere diese p Textsegmente aufsteigend nach ihrer WMD zu S. Sei $List_{\text{WMD}}$ diese Liste.
6. Für die restlichen Textsegmente Ω der Liste $List_{lb_1}$ überprüfe in aufsteigender Reihenfolge ihrer lb_1-Schranken jeweils, ob seine lb_1-Schranke $lb_1(\Omega, S)$ größer gleich der WMD-Distanz des letzten Listeneintrages der Liste $List_{\text{WMD}}$ ist. Wenn ja, gehe zu Schritt 7. Wenn nein, so berechne die lb_2-Schranke $lb_2(\Omega, S)$ zwischen Ω und S. Ist diese größer gleich der WMD des letzten Listeneintrages der Liste $List_{\text{WMD}}$, so gehe zum nächsten Textsegment der Liste $List_{lb_1}$. Ansonsten berechne die WMD zwischen Ω und S. Falls dieser Wert echt kleiner ist als die WMD des letzten Listeneintrages der Liste $List_{\text{WMD}}$, so lösche den letzten Listeneintrag aus $List_{\text{WMD}}$ und füge Ω in die geordnete Liste $List_{\text{WMD}}$ ein.
7. Entferne sich überlappende Textsegmente aus $List_{\text{WMD}}$. Genauer: Überlappen sich zwei in der Liste $List_{\text{WMD}}$ enthaltene Textsegmente, so wird das Textsegment mit der höheren WMD zu S, also das zweite der beiden, gelöscht.
8. Gib die q ersten Textsegmente aus der Liste $List_{\text{WMD}}$ zurück.

Leider hat sich bei den experimentellen Untersuchungen (siehe Kapitel 5) ergeben, dass bei vielen Suchanfragen sowohl die lb_1-Schranke als auch die lb_2-Schranke nicht wirklich nah an der WMD liegen. Dies hat zur Folge, dass für sehr viele, um nicht zu sagen für die meisten Textsegmente des Korpus die WMD in Schritt 6 berechnet werden muss und sich der Ansatz von der Laufzeit her nur wenig von dem Brute-Force-Algorithmus, in dem einfach für jedes Textsegment des Korpus die WMD berechnet wird, unterscheidet.

4.2 Heuristik zur WMD basierten Paraphrasensuche

Um die hohen Laufzeiten zu vermeiden, ersetzen wir unter Aufgabe der beweisbaren Optimalität die Schritte 3 bis 6 aus dem obigen Algorithmus wie folgt:

3. Berechne die beiden unteren Schranken $lb_1(\Omega, S)$ und $lb_2(\Omega, S)$ für die ersten p Textsegmente Ω der Länge m des Korpus K und bestimme jeweils die relaxierte Word Mover's Distance $RWMD(\Omega, S)$.

29 Im Schritt 7 des Ansatzes werden Textsegmente aus der Liste entfernt, wenn sie sich mit anderen Textsegmenten, die eine kleinere WMD zu S haben, überlappen. Damit im Ergebnis q Textstellen durch den Algorithmus ausgegeben werden können, müssen vor dem Streichen der sich überlappenden Textstellen Listen mit mehr als q Textstellen geführt werden. Die Anzahl p wird in Abhängigkeit der Länge m der typischen Suchanfragen empirisch bestimmt. Mehr dazu im Kapitel zu den experimentellen Ergebnissen.

4. Sortiere diese p Textsegmente in aufsteigender Ordnung nach ihren RWMDs. Sei $List_{\text{RWMD}}$ diese Liste.
5. Für die restlichen Textsegmente Ω der Länge m des Korpus K berechne jeweils die schnell zu berechnende untere Schranke $lb_1(\Omega, S)$ und überprüfe, ob diese untere Schranke größer als die RWMD des letzten Listeneintrages der Liste $List_{\text{RWMD}}$ ist. Wenn ja, gehe zum nächsten Textsegment des Korpus. Wenn nein, berechne $lb_2(\Omega, S)$ und $RWMD(\Omega, S)$. Ist diese größer gleich der RWMD des letzten Listeneintrages der Liste $List_{\text{RWMD}}$, so gehe zum nächsten Textsegment des Korpus. Andernfalls, entferne den letzten Eintrag der Liste $List_{\text{RWMD}}$ und füge Ω in die geordnete Liste $List_{\text{RWMD}}$ ein.
6. Berechne die (exakte) WMD für alle p Textsegmente der Liste $List_{\text{RWMD}}$ und sortiere die p Textsegmente aufsteigend nach ihren WMDs. Sei $List_{\text{WMD}}$ diese Liste.

Somit wird die laufzeitaufwändige exakte WMD nur noch in Schritt 6 für p Textsegmente berechnet, was zu hohen Laufzeiteinsparungen führt. Falls sich die relaxierte WMD in Bezug auf die entsprechend geordnete Reihenfolge der Textsegmente mehr oder weniger wie die exakte WMD verhält, so sollte dieser heuristische Ansatz zu einer Suchanfrage S in etwa die gleiche Liste von q Textsegmenten berechnen wie der im vorherigen Abschnitt beschriebene exakte Algorithmus angesetzt auf S berechnet.

5 Evaluierung der Heuristik

Die Evaluierung konzentriert sich darauf, unsere Vermutung experimentell zu bestätigen, dass die in Kapitel 4.2 beschriebene Heuristik zur Paraphrasenextraktion in etwa die gleiche Liste an Paraphrasenkandidaten zurückgibt, wie wenn man die exakte WMD zwischen allen Textstellen des Korpus K und der Suchanfrage berechnet und dann die besten q sich nicht überlappenden Textstellen zurückgibt. Darüber hinaus wollen wir nachweisen, dass der heuristische Ansatz um ein Vielfaches schneller als der exakte Algorithmus ist und in interaktiven Anwendungen eingesetzt werden kann.

Zur Evaluierung des Ansatzes haben wir als Korpus K den Thesaurus Linguae Graecae (TLG) gewählt, in dem wir nach Paraphrasen von Textpassagen aus Platons Werk suchen. Der TLG besteht aus ungefähr 38 Millionen Wörtern, nachdem die Stoppwörter entfernt worden sind, d. h. $N \approx 38.000.000$. Die Größe n des im TLG verwendeten Vokabulars ist in etwa eine Million.[30]

30 Wörter (und Eigennamen), die nur einige wenige Male im TLG enthalten sind, werden entgegen der üblichen Praxis auf Verlangen der geisteswissenschaftlichen Partner mitbetrachtet, da sich die Schreibweise einzelner griechischer Wörter über die Zeit geändert hat.

Pöckelmann et al. haben anhand von Beispielen die Ergebnisse des Brute-Force-Algorithmus' angesetzt auf den TLG inhaltlich diskutiert und die hohe Qualität der vorgeschlagenen Textstellen bestätigt.[31] Wir gehen aus diesem Grunde hier nicht auf die Güte der durch den exakten Algorithmus vorgeschlagenen Textstellen ein und verweisen diesbezüglich auf den oben angegebenen Artikel.

Zur Evaluierung werden zwei verschiedene Experimente durchgeführt. In dem ersten Experiment überprüfen wir, ob der heuristische Ansatz angewandt auf die drei durch Pöckelmann et al. detailliert besprochenen Textsegmente von Platon jeweils die gleiche Liste von Paraphrasenkandidaten wie der Brute-Force-Algorithmus zurückgibt. Das zweite Experiment gibt genauere Informationen zu den Berechnungszeiten des heuristischen Ansatzes im Vergleich zu dem exakten Verfahren.

5.1 Verhalten der Heuristik im Vergleich zum exakten Verfahren

Um das Verhalten der Heuristik und das Verhalten des exakten Verfahrens bei einer gegebenen Suchanfrage zu vergleichen, haben wir in einem ersten Test beide Ansätze auf drei ausgewählte Textsegmente von Platon angewendet. Die durch den exakten Algorithmus vorgeschlagenen Textstellen möglicher Paraphrasen dieser drei Textsegmente wurden sehr detailliert durch Gräzist(inn)en und Historikerinnen diskutiert und ihre gute Qualität bestätigt:[32]

- **Textsegment 1:** τῷ μὲν θείῳ καὶ ἀθανάτῳ καὶ νοητῷ καὶ μονοειδεῖ καὶ ἀδιαλύτῳ καὶ ἀεὶ ὡσαύτως κατὰ ταὐτὰ ἔχοντι ἑαυτῷ ὁμοιότατον εἶναι ψυχή (Platon *Phaidon* 80 b1–3); in dieser Textpassage beschreibt Platon in seiner ihm eigenen Art die Gegenüberstellung von Körper und Geist.[33]
- **Textsegment 2:** τοῦτο δέ, ὡς ἔοικεν, οὐκ ὀστράκου ἂν εἴη περιστροφή, ἀλλὰ ψυχῆς περιαγωγή, ἐκ νυκτερινῆς τινος ἡμέρας εἰς ἀληθινὴν τοῦ ὄντος οὖσαν ἐπάνοδον, ἣν δὴ φιλοσοφίαν ἀληθῆ φήσομεν εἶναι (Platon *Politeia* 521 c5–8); hier geht es um die ‚Umlenkung der Seele' durch die wahre Philosophie.[34]

31 Pöckelmann (2017).

32 Pöckelmann (2017).

33 Übersetzung Schleiermacher: ‚Sieh nun zu, sprach er, o Kebes, ob aus allem gesagten uns dieses hervorgeht, *daß dem göttlichen, unsterblichen, vernünftigen, eingestaltigen, unauflöslichen, und immer einerlei und sich selbst gleich verhaltenden am ähnlichsten ist die Seele,* dem menschlichen und sterblichen und unvernünftigen und vielgestaltigen und auflöslichen und nie einerlei und sich selbst gleichbleibenden diesem wiederum der Leib am ähnlichsten ist?' Die Hervorhebungen der Übersetzungen dieses Beitrages stammen von den Verfassern.

34 Übersetzung Schleiermacher: *‚Das ist nun freilich, scheint es, nicht wie sich eine Scherbe umwendet, sondern es ist eine Umlenkung der Seele, welche aus einem gleichsam nächtlichen*

– **Textsegment 3:** καὶ δὴ καὶ Σοφοκλεῖ ποτε τῷ ποιητῇ παρεγενόμην ἐρωτωμένῳ ὑπό τινος: ‘πῶς,’ ἔφη, ‘ὦ Σοφόκλεις, ἔχεις πρὸς τἀφροδίσια; ἔτι οἷός τε εἶ γυναικὶ συγγίγνεσθαι’; καὶ ὅς, ‘εὐφήμει,’ ἔφη, ‘ὦ ἄνθρωπε: ἀσμενέστατα μέντοι αὐτὸ ἀπέφυγον, ὥσπερ λυττῶντά τινα καὶ ἄγριον δεσπότην ἀποδράς.’ (Platon *Politeia* 329 b6–c4); in diesem Textausschnitt gibt Platon eine Anekdote über den Tragödiendichter Sophokles wieder.[35]

Die geisteswissenschaftlichen Mitarbeiterinnen gaben an, dass eine Liste von 500 Textstellen, die ihnen als Vorschläge durch die Algorithmen vorgelegt werden, praktikabel sei, um sie in angemessener Zeit zu sichten. Aus diesem Grunde haben wir in unseren Experimenten die Länge q der auszugebenen Liste auf 500 gesetzt. Es zeigt sich, dass

– bei Wahl von p = 10.000 die Heuristik bei den Textsegmenten 1 und 2 die gleiche Liste wie der exakte Algorithmus ausgibt und sich im Falle des Textsegments 3 die Listen sich (nur) um 21 Einträge unterscheiden.
– bei Wahl von p = 20.000 die Heuristik auch bei Textsegment 3 die gleiche Liste wie der exakte Algorithmus ausgibt.

Die Laufzeit der Heuristik belief sich auf einem Rechensystem mit 16 Kernen auf circa 30 Sekunden bei Textsegment 1, auf 40 Sekunden bei Textsegment 2 und auf 100 Sekunden bei Textsegment 3.

5.2 Laufzeit der Heuristik im Vergleich zum exakten Verfahren

Um die Laufzeit der Heuristik mit der des exakten Verfahrens zu vergleichen, haben wir das exakte Verfahren und die Heuristik auf jeweils 5 Suchanfragen der Länge 10, d.h. mit 10 Wörtern, der Länge 20, der Längen 30 bis 31, der Längen 40 bis 41 und der Längen 48 bis 52 für p gleich 20.000 angewendet. Die Suchanfragen stammen aus dem durch die geisteswissenschaftlichen Mitarbeiter(innen) im Rahmen des Projektes *Platon Digital* ausgearbeiteten ‚Goldstandard‘.[36]

Tage zu dem wahren Tage des Seienden jene Auffahrt antritt, welche wir eben die wahre Philosophie nennen wollen.‘

35 Übersetzung Teuffel: ‚*Namentlich war ich einmal dabei, wie jemand an den Dichter Sophokles die Frage richtete: »Wie sieht’s bei dir aus, Sophokles, mit der Liebe? Vermagst du noch einem Weibe beizuwohnen?« Der antwortete: »Nimm deine Zunge in acht, Mensch; bin ich doch herzlich froh, daß ich davon erlöst bin, wie ein Sklave, der von einem tobsüchtigen und wilden Herrn erlöst worden ist!«‘*

36 In dem im Rahmen des Projektes entwickelten ‚Goldstandard‘ (siehe den Beitrag „Ein Parallelkorpus von Paraphrasen auf Platon: Der ‚Goldstandard‘ des Projekts *Platon Digital*“ von Wöckener-Gade et al. auf den Seiten 275–323 in diesem Band) sind keine fünf

Tabelle 1 gibt die gemittelten Laufzeiten der beiden Verfahren in Sekunden bei Nutzung von 16 Rechenkernen an. Die Rechenkerne werden in der Heuristik zur Parallelisierung der RWMD-Berechnungen in Schritt 5 eingesetzt, indem der Korpus K in gleichgroße Teile zerlegt und dann an die Kerne verteilt wird, und zur Parallelisierung der p WMD-Berechnungen in Schritt 6. In dem exakten Verfahren werden die Rechenkerne zur Beschleunigung der Schritte 5 und 6 verwendet. Die Zeiten entsprechen den Wartezeiten des Nutzers und nicht der Summe der CPU-Zeiten. Wenn auch die reale Wartezeit des Nutzers ein eher unübliches Maß für die Laufzeit von Algorithmen ist – insbesondere hängt sie von Parametern ab, die nur schwer beeinflussbar sind, wie beispielsweise von der aktuellen Rechenlast der eingesetzten Prozessoren –, so ist es das Maß, das für den Nutzer am relevantesten ist. Es gibt unmittelbare Auskunft, ob er flüssig mit dem System arbeiten kann.

Das exakte Verfahren benötigt bei kurzen Suchanfragen der Länge 10 und 20 durchschnittlich bereits 40 bzw. 80 Minuten. Die mittleren Laufzeiten bei längeren Suchanfragen liegen bei einigen Stunden. So beträgt die mittlere Laufzeit bei Suchanfragen der Länge 50 ungefähr 8,7 Stunden. Die Laufzeit wird hier dominiert durch die Laufzeit der vielen WMD-Berechnungen. Die Laufzeiten der Heuristik bewegen sich im Bereich von 3 (bei Suchanfragen der Länge 10) bis 16 Minuten (bei Suchanfragen der Länge 50) und zeigen in den Experimenten ein in Bezug auf die Länge der Suchanfrage annähernd lineares Verhalten.

Noch deutlicher wird der Vergleich, wenn beide Ansätze auf Suchanfragen noch größerer Länge angewendet werden. Wir haben in einem Test zum Vergleich der Laufzeiten sowohl den exakten Algorithmus wie auch die Heuristik auf eine Suchanfrage aus Platons *Nomoi* bestehend aus 189 Wörtern angewendet. Beide Berechnungen wurden auf dem gleichen Rechnersystem bestehend aus 150 Rechenkernen durchgeführt. Die Berechnung der WMDs zwischen der Suchanfrage und allen Textstellen im TLG benötigte auf diesem Rechnersystem 518.203 Sekunden, was in etwa 6 Tage sind. Die Laufzeit der Heuristik bewegte sich im Bereich von wenigen Minuten.

Diese Laufzeittests zeigen, dass der exakte Algorithmus aufgrund der hohen Laufzeiten kein geeignetes Arbeitsmittel für Geisteswissenschaftler(innen) ist, die Heuristik aber, die kaum schlechtere Ergebnisse als der exakte Algorithmus liefert, durchaus (auch) für interaktive Anwendungen geeignet ist.

Textpassagen mit der exakten Länge 30 enthalten, so dass wir für unsere Tests Textpassagen der Länge 30 und 31 verwendet haben. Gleiches gilt für die Längen 40 und 50.

6 Fazit und Ausblick

Der auf der Word Mover's Distance basierende Ansatz ist geeignet, um (bisher nicht entdeckte) Paraphrasen zu Texten Platons zu finden.[37] Die Laufzeit zur Berechnung der Word Mover's Distance ist jedoch zu hoch, um sie zwischen jeder Textpassage der altgriechischen Literatur und einem gegebenen Textsegment von Platon online berechnen zu können. Der in diesem Beitrag vorgestellte heuristische Ansatz behebt diesen Mangel. Er gibt bei geeigneter Festlegung der Parameter die gleiche Liste an Textstellen zurück wie der exakte Algorithmus. Die Laufzeiten der Heuristik bewegen sich im unteren Minutenbereich, so dass der Ansatz auch in interaktiven Szenarien eingesetzt werden kann.

Um die Qualität des Verfahrens weiter zu steigern, muss als Nächstes die Erweiterung der Heuristik implementiert werden, mit der auch Textstellen variabler Länge, d. h. Textstellen, die kürzer oder länger als die Länge m der gegebenen Suchanfrage sind, gefunden werden können. Die entsprechenden Berechnungen lassen sich in etwa der gleichen Zeit durchführen wie der in dieser Arbeit betrachtete Spezialfall, in denen nur nach Textstellen gleicher Länge wie die der Suchanfrage gesucht wird.

Anhang

Tabelle 1. Laufzeiten der Heuristik im Vergleich zu den Laufzeiten des exakten Verfahrens

Länge der Suchanfragen	Gemittelte Laufzeit des exakten Verfahrens	Gemittelte Laufzeit der Heuristik	Beschleunigung durch die Heuristik
10	2.417 Sek.	214 Sek.	11,3
20	5.622 Sek.	315 Sek.	17,8
30/31	11.168 Sek.	492 Sek.	22,7
40/41	20.924 Sek.	802 Sek.	26,1
48–52	31.285 Sek.	936 Sek.	33,4

37 Pöckelmann (2017).