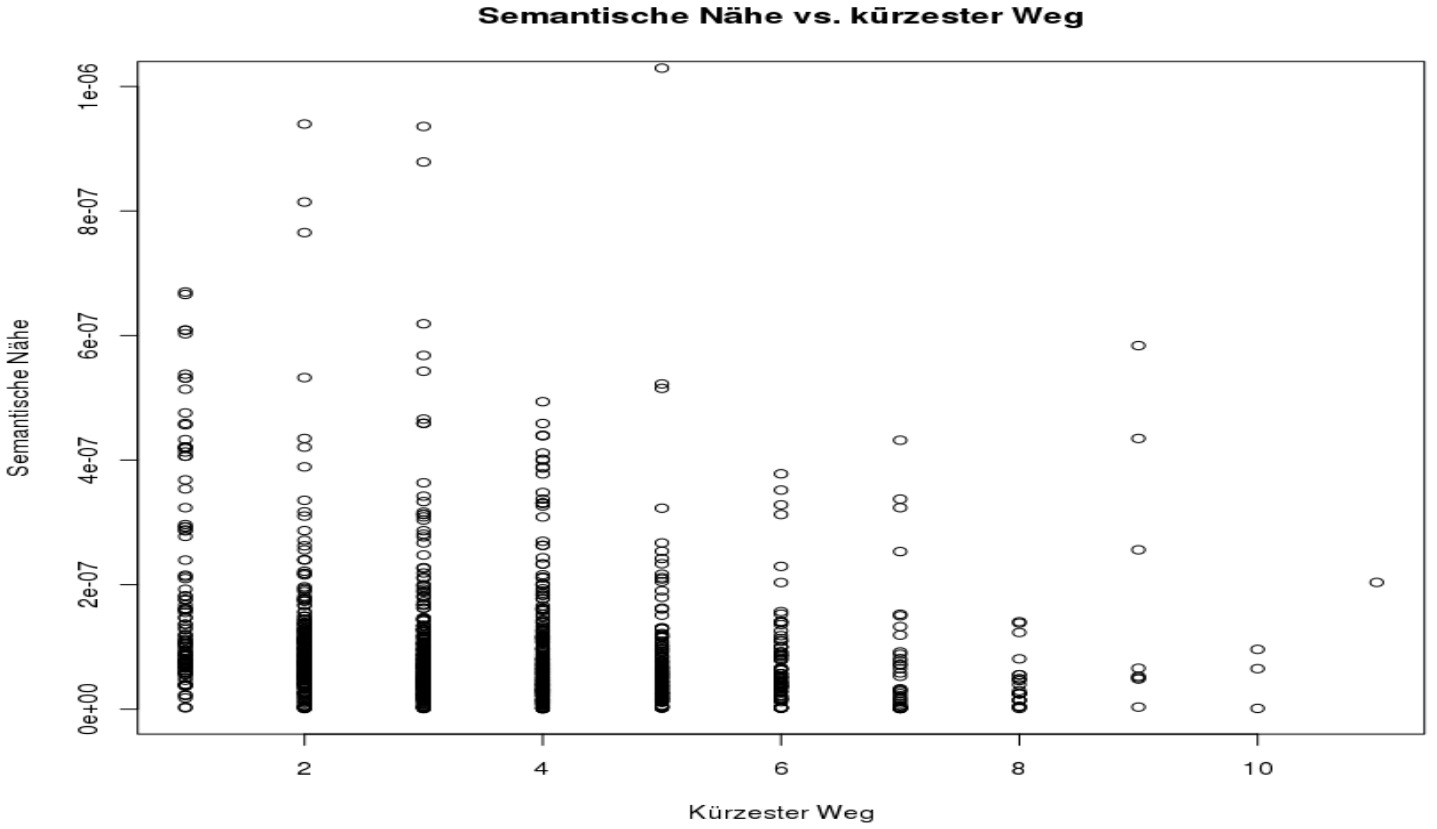
Kürzeste Wege und semantische Nähe von Wikipedia Artikeln



Vektorisierung

- Vektorisierungspotential:
  - Berechnung der semantischen Nähe
    - Matrix die die Häufigkeiten aller in Wikipedia vorkommenden Wörter pro Artikel speichert
      - (Zeilen: Wörter; Reihen: Artikel)
    - Metrik: zweifach normalisiertes Skalarprodukt
  - Es gibt Algorithmen mit vektorisierter Graphensuche
- Herausforderungen:
  - Spalten der dünnbesetzten Matrix liegen mit unterschiedlichen Längen hintereinander im Speicher
- Vergleich der Lösungsansätze:
  - Idee: Skalarprodukt oder Reihensumme als Matrix Vektor Multiplikation statt geschachtelter Schleifen (Eigen Bibliothek)

Parallelisierung

- Graphensuche:
  - Äußere Schleife die alle Knoten abläuft
    - #pragma omp for
  - Funktion selbst
    - #pragma omp task; atomic vector
- Bag-of-words:
  - Berechnung der Spalten- und Reihensummen und Skalarprodukte, erstellen der Matrix und normalisieren
    - #pragma omp for; #pragma omp critical

MPI

- Graphen:
  - Zwei Vektoren die Graphen abbilden
  - MPI_Bcast; MPI_Gatherv
  - Overhead zu reduzieren: Nur Teile des Graphen verschicken
  - Gleichmäßigere Auslastung der Prozesse durch z.B. Verteilung der Kanten berücksichtigen

- Bag-of-words:
  - Aufteilung der Spalten- und Reihensummen und Normierung

Speed-Up

- Dict size: 239.000 Wörter
- Untersuchte Knotenpaare: 3.089.870
- Graphensuche:
  - MPI
    - 40 sek. (n = 1, 8 Threads)
    - 22 sek. (n = 2, 16 Threads)
    - 10 sek. (n = 6, all Threads)
  - OpenMP
    - 21 sek. (Host, 16 Threads)
    - 25 sek. (Host, 8 Threads)
    - 43 sek. (Host, 4 Threads)
    - 135 sek. (Host, 1 Thread)
- Bag-of-words:
  - OpenMP
    - 40 sek. (16 Threads)
    - 70 sek. (8 Threads)
    - 93 sek. (4 Threads)
    - 373 sek. (1 Thread)

Herausforderungen

- Filtern von Wörtern und Links aus xml file (REGEX !!!)
- Eigen Bibliothek optimal ausnutzen
- Aufteilung des Graphen in verteilbare Vektoren
- Zeitmanagement