Handout: Holisitic Ontology Matching

Testat 2

Programm-Architektur

- Der gesamte **Prozessablauf** wird mit der Klasse HolomaProcessControl geregelt.[1] U.a. wird hier zu Anfang mit ExecutionEnvironment.getExecutionEnvironment() der Kontext zur Programmausführung (lokal vs. Cluster) bestimmt.[2] Der Prozess beinhaltet folgende Teilprozesse:

  1. Graph von Ontologien und Mapping erstellen
  2. Zusammenhangskomponenten berechnen
  3. Analyse der Zusammenhangskomponenten
  4. Pagerank-Berechnung: Iteration über Komponenten und falls die Komponente eine "kritische" Größe hat, also in einem vorher festgelegten Intervall liegt, dann ...
     (a) Komponente mit zusätzlicher Struktur anreichern
     (b) Personalized Page Rank berechnen
     (c) Komponente mit Pagerank-Werten auswerten

- Grah erstellen und Zusammenhangskomponenten berechnen/ analysieren wurde bereits besprochen.

- Die **Anreicherung der Komponenten** erfolgt in connComp.ConnCompEnrichment. Als Input wird eine Anreicherungstiefe depth, der Graph graph, eine Zuordnung von Kantentyp zu Kantengewicht mapWeight und die ExcecutionEnvironment env benötigt. Die Zusammenhangskomponente wird der Methode getEnrichedConnComp als Menge von Knoten-IDs übergeben. Diese Methode sieht wie folgt aus:

  1. Berechnnung des Subgraphens von graph, welcher der angereicherten Komponente entspricht. Dies erfolgt durch die Methode extractSubgraph(), welche folgendes tut:
     (a) Die Menge der Knoten-IDs wird um die IDs erweitert, deren Knoten genau eine Kante (Richtung egal) entfernt sind, siehe Methode addNextHopp(). Dies wird depth-mal gemacht.
     (b) Mithilfe dieser Knoten-IDs wird der Subgraph aus graph extrahiert.
     (c) Es wird jeder Knoten mit dem komplexen Wert VertexValue versehen, welcher Ontologie-Name und Pagerank-Wert umfasst.
  2. Mapping der Kantentypen auf Gewichte. Hier wird der komplexe Kanten-Wert EdgeValue eingeführt, welcher Kantentyp und Kantengewicht umfasst.
  3. Das Ergebnis ist ein Graph mit Knoten-IDs vom Typ String, und *user-defined data types* für Knoten-Wert (VertexValue) und Kanten-Wert (EdgeValue).

- Der **Personalized Page Rank** Algorithmus ist in ppr.PersonalizedPageRank mithilfe von *vertex-centric iteration* implementiert.[3] Über eine set-Methode wird die angereicherte Komponente übergeben. Die Methode start() startet die Berechnung.

  1. Es wird eine List von Quellknoten sources als alle Knoten in der Komponente erstellt.
  2. Dann wird über jeden Quellknoten iteriert:
     (a) Initialisierung der Pagerank-Werte aller Knoten mit 1, falls Knoten gleich dem aktuellen Quellknoten ist, sonst 0.
     (b) Berechnung der Pagerank-Werte für den aktuellen Quellknoten mit calculateOneSource().
        – Vertex-centric Iteration mit Messaging- und Update-Schritt.

[1] Das Projekt findet sich unter https://github.com/agataba/HolOMa
[2] https://ci.apache.org/projects/flink/flink-docs-release-0.7/api/java/org/apache/flink/api/java/ExecutionEnvironment.html
[3] https://ci.apache.org/projects/flink/flink-docs-release-0.10/libs/gelly_guide.html#vertex-centric-iterations

- **Update** erfolgt gemäß $(1-\epsilon)\cdot \text{sum} + \epsilon \cdot \delta$, wobei sum die Summe aller Werte ist, die der Knoten per Messaging erhält.
- **Messaging** sendet zu allen Knoten, zu welchen eine ausgehende Kante existiert, einen Float-Wert, der sich berechnet nach $\text{pr} \cdot \frac{\text{weight}}{\text{sumWeight}}$, wobei pr der Pagerank-Wert des sendenden Knoten ist, weight das Kantengewicht vom sendenden zum empfangenden Knoten und sumWeight die Summe aller Kantengewichte ausgehend vom sendenden Knoten ist.

(c) In einer globalen Map wird der aktuelle Quellknoten und der dazugehörige Pagerank-Vektor abgespeichert.

- Die **Auswertung** der Pagerank-Werte erfolgt in der Klasse ppr.PPREvaluation. Dazu stehen u.a. die folgenden Methoden zur Verfügung:
  - getTrueBestFriends() gibt eine Map von Quellknoten zur Menge aller echten besten Freunde zurück. Ein Knoten ist ein echter bester Freund, wenn es keinen anderen Knoten mit einem höheren Pagerank-Wert gibt und der Knoten nicht der Quellknoten selbst ist. Der Pagerank-Wert muss größer als 0 sein.
  - getWorstFriends() gibt eine Map von Quellknoten zur Menge aller schlechtesten Freund zurück. Ein Knoten ist ein schlechtester Freund, wenn es keinen anderen Knoten gibt, der einen niedrigen Pagerank-Wert hat.
  - getStatistMeans() gibt eine Map von Quellknoten zum arithmetischen Mittel aller Pagerank-Werte in dessen Pagerank-Vektor.

Auswertung

- Zur Auswertung gibt es zwei **Datensätze**: einen kleinen Beispiel-Datensatz "Farbe", welcher aus drei Ontologien besteht und insgesamt 16 Knoten umfasst, siehe Abb. 1; sowie den echten Datensatz "Bio" bestehend aus neun Ontologien aus dem Life-Science Bereich. Aus letzterem wird genau eine Komponente mit acht Knoten ausgewählt, welche angereichert wird und auf welcher der Personalized Page Rank Algorithmus läuft.

- Die **Teleportationswahrscheinlichkeit** ϵ wird auf $0,15$ gesetzt. Für den Datensatz "Farbe" ist die **Anzahl der Iterationen** bei der Pagerank-Berechnung 10. Als **Tiefe** wird 1 bzw. 2 gewählt. Für den Datensatz "Bio" wird die Tiefe auf 1 gesetzt und die Anzahl der Iterationen auf 1, 3 bzw. 6. Die Berechnung für eine Tiefe von 2 wurde nach 30min abgebrochen (das Programm fand sich zu diesem Zeitpunk in der Pagerank-Berechnung).

- Die **programm-interne Analyse** wird in Dateien Y_analysisX_PPR.txt geschrieben. Die erste Zeile führt die eingestellten Parameter auf. Danach folgt für jede ausgewertete Zusammenhangskomponente die Kanten und Knoten der angereicherten Komponenten; gefolgt von den Pagerank-Vektoren, also eine Tabelle mit dem Quellknoten als erste Spalte sowie abhängig von diesem Quellknoten der Pagerank-Wert (Spalte 3, inkl. Ontologie-Name) für jeden Knoten in der Komponenten (Spalte 2). Danach kommt der durchschnittliche Pagerank-Wert pro Quellknoten und eine Übersicht zu *best true friends* und *worst friends*. Diese sind Ausschnitte aus den bereits oben vollständig dargestellten Pagerank-Vektoren. Die Dateien sind im Ordner analysis_ppr gegeben.

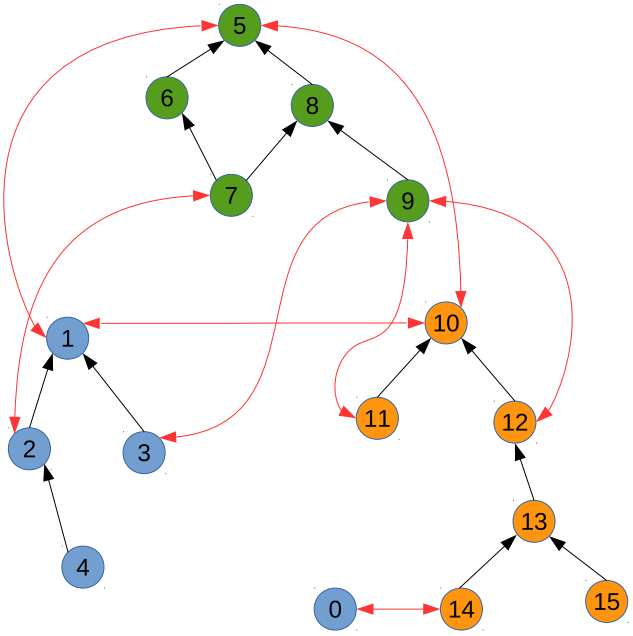

Abbildung 1: Beispiel-Graph.