

# Über die Klassifizierung von Knoten in dynamischen Netzwerken mit Inhalt

Martin Thoma

Betreuer: Christopher Oßner

**Zusammenfassung** In dieser Arbeit wird der DYCOS-Algorithmus, wie ihn Charu Aggarwal und Nan Li in [AgLi11] vorgestellt haben, erklärt. Er klassifiziert automatisch Knoten in Netzwerken, die bereits teilweise mit Labels versehen sind. Zur Klassifizierung kann er textuelle Informationen, die den Knoten zugeordnet sind, nutzen. Er ist auch für für viele 10 000 Knoten große, dynamische Netzwerke geeignet. Zusätzlich wird auf Schwächen von [AgLi11] hingewiesen und mögliche Verbesserungen vorgeschlagen.

**Keywords:** DYCOS, Label Propagation, Knotenklassifizierung

## 1 Einleitung

### 1.1 Motivation

Teilweise gelabelte Netzwerke sind allgegenwärtig. Publikationsdatenbanken mit Publikationen als Knoten, Literaturverweisen und Zitaten als Kanten sowie Tags oder Kategorien als Labels; Wikipedia mit Artikeln als Knoten, Links als Kanten und Kategorien als Labels sowie soziale Netzwerke mit Eigenschaften der Benutzer als Labels sind drei Beispiele dafür. Häufig sind Labels nur teilweise vorhanden und es ist wünschenswert die fehlenden Labels zu ergänzen.

### 1.2 Problemstellung

Gegeben ist ein Graph, der teilweise gelabelt ist. Zusätzlich stehen zu einer Teilmenge der Knoten Texte bereit. Gesucht sind nun Labels für alle Knoten, die bisher noch nicht gelabelt sind.

**Definition 1 (Knotenklassifizierungsproblem)** Sei  $G_t = (V_t, E_t, V_{L,t})$  ein gerichteter Graph, wobei  $V_t$  die Menge aller Knoten,  $E_t$  die Kantenmenge und  $V_{L,t} \subseteq V_t$  die Menge der gelabelten Knoten jeweils zum Zeitpunkt  $t$  bezeichne. Außerdem sei  $L_t$  die Menge aller zum Zeitpunkt  $t$  vergebenen Labels und  $f : V_{L,t} \rightarrow L_t$  die Funktion, die einen Knoten auf sein Label abbildet.

Weiter sei für jeden Knoten  $v \in V$  eine (eventuell leere) Textmenge  $T(v)$  gegeben.

Gesucht sind nun Labels für  $V_t \setminus V_{L,t}$ , also  $\tilde{f} : V_t \rightarrow L_t$  mit  $\tilde{f}|_{V_{L,t}} = f$ .

### 1.3 Herausforderungen

Die Graphen, für die dieser Algorithmus konzipiert wurde, sind viele 10 000 Knoten groß und dynamisch. Das bedeutet, es kommen neue Knoten und eventuell auch neue Kanten hinzu bzw. Kanten oder Knoten werden entfernt. Außerdem stehen textuelle Inhalte zu den Knoten bereit, die bei der Klassifikation genutzt werden können. Bei kleinen Modifikationen sollte nicht alles nochmals berechnet werden müssen, sondern basierend auf zuvor berechneten Labels sollte die Klassifizierung modifiziert werden.

## 2 DYCOS

### 2.1 Überblick

DYCOS (DYnamic Classification algorithm with cOntent and Structure) ist ein Knotenklassifizierungsalgorithmus, der Ursprünglich in [AgLi11] vorgestellt wurde. Er klassifiziert Knoten, indem mehrfach Random Walks startend bei dem zu

klassifizierenden Knoten gemacht werden und die Labels der besuchten Knoten gezählt werden. Das Label, das am häufigsten vorgekommen ist, wird als Label gewählt. DYCOS nutzt also die sog. Homophilie, d. h. die Eigenschaft, dass Knoten, die nur wenige Hops von einander entfernt sind, häufig auch ähnlich sind [BhCM11].

Der DYCOS-Algorithmus nimmt jedoch nicht einfach den Graphen für dieses Verfahren, sondern erweitert ihn mit Hilfe der zur Verfügung stehenden Texte.

Für diese Erweiterung wird zuerst das Vokabular  $W_t$  bestimmt, das charakteristisch für eine Knotengruppe ist. Wie das gemacht werden kann und warum nicht einfach jedes Wort in das Vokabular aufgenommen wird, wird in Abschnitt 2.4 erläutert.

Nach der Bestimmung des Vokabulars wird für jedes Wort im Vokabular ein Wortknoten zum Graphen hinzugefügt. Alle Knoten, die der Graph zuvor hatte, werden nun „Strukturknoten“ genannt. Ein Strukturknoten  $v$  wird genau dann mit einem Wortknoten  $w \in W_t$  verbunden, wenn  $w$  in einem Text von  $v$  vorkommt.

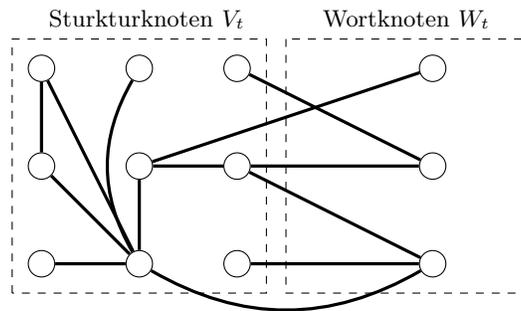


Abbildung 1. Erweiterter Graph

Entsprechend werden zwei unterschiedliche Sprungtypen unterschieden, die strukturellen Sprünge und inhaltliche Mehrfachsprünge:

**Definition 2** Sei  $G_{E,t} = (V_t, E_{S,t} \cup E_{W,t}, V_{L,t}, W_t)$  der um die Wortknoten  $W_t$  erweiterte Graph.

Dann heißt das zufällige wechseln des aktuell betrachteten Knoten  $v \in V_t$  zu einem benachbartem Knoten  $w \in V_t$  ein **struktureller Sprung**.

Im Gegensatz dazu benutzen inhaltliche Mehrfachsprünge tatsächlich die Grapherweiterung:

**Definition 3** Sei  $G_t = (V_t, E_{S,t} \cup E_{W,t}, V_{L,t}, W_t)$  der um die Wortknoten  $W_t$  erweiterte Graph.

*Dann heißt das zufällige wechseln des aktuell betrachteten Knoten  $v \in V_t$  zu einem benachbartem Knoten  $w \in W_t$  und weiter zu Nachbar von  $v' \in V_t$  von  $w$  ein **inhaltlicher Mehrfachsprung**.  $v'$  ist also genau einen Sprung über einen Wortknoten  $w$  von  $v$  entfernt.*

Der DYCOS-Algorithmus betrachtet die Texte, die einem Knoten zugeordnet sind, als eine Multimenge von Wörtern. Das heißt, zum einen wird nicht auf die Reihenfolge der Wörter geachtet, zum anderen wird bei Texten eines Knotens nicht zwischen verschiedenen Texten unterschieden. Jedoch wird die Anzahl der Vorkommen jedes Wortes berücksichtigt.

## 2.2 Datenstrukturen

Zusätzlich zu dem gerichteten Graphen  $G_t = (V_t, E_t, V_{L,t})$  verwaltet der DYCOS-Algorithmus zwei weitere Datenstrukturen:

- Für jeden Knoten  $v \in V_t$  werden die vorkommenden Wörter, die auch im Vokabular  $W_t$  sind, und deren Anzahl gespeichert. Das könnte z. B. über ein assoziatives Array geschehen. Wörter, die nicht in Texten von  $v$  vorkommen, sind nicht im Array. Für alle vorkommenden Wörter ist der gespeicherte Wert zum Schlüssel „Wort“ die Anzahl der Vorkommen von „Wort“ in den Texten von  $v$ .
- Für jedes Wort des Vokabulars  $W_t$  wird eine Liste von Knoten verwaltet, in deren Texten das Wort vorkommt.
- An einigen Stellen macht ein assoziatives Array, auch „dictionary“ oder „map“ genannt, sinn. Zusätzlich ist es nützlich, wenn diese Datenstruktur für unbekannte Schlüssel keinen Fehler ausgibt, sondern für diese Schlüssel den Wert 0 annimmt. Eine solche Datenstruktur wird in Python `defaultdict` genannt und ich werde im Folgenden diese Benennung beibehalten.

## 2.3 Sprungtypen

Die beiden bereits definierten Sprungtypen, der strukturelle Sprung sowie der inhaltliche Mehrfachsprung werden im folgenden erklärt.

Der strukturelle Sprung entspricht einer zufälligen Wahl eines Nachbarknotens. Hier gibt es nichts besonderes zu beachten.

Bei inhaltlichen Mehrfachsprüngen sieht die Sache schon anders aus: Es ist nicht sinnvoll, direkt von einem strukturellem Knoten  $v \in \mathcal{N}_t$  zu einem mit  $v$  verbundenen Wortknoten  $w$  zu springen und von diesem wieder zu einem verbundenem strukturellem Knoten  $v' \in \mathcal{N}_t$ . Würde man dies machen, wäre zu befürchten, dass aufgrund von Homonymen die Qualität der Klassifizierung verringert wird. So hat „Brücke“ im Deutschen viele Bedeutungen. Gemeint sein können z. B.

das Bauwerk, das Entwurfsmuster der objektorientierten Programmierung oder ein Teil des Gehirns.

Deshalb wird für jeden Knoten  $v$ , von dem aus man einen inhaltlichen Mehrfachsprung machen will folgendes vorgehen gewählt:

1. Gehe alle in  $v$  startenden Random Walks der Länge 2 durch und erstelle eine Liste  $L$ , der erreichbaren Knoten  $v'$ . Speichere außerdem, durch wie viele Pfade diese Knoten  $v'$  jeweils erreichbar sind.
2. Betrachte im folgenden nur die Top- $q$  Knoten, wobei  $q \in \mathbb{N}$  eine zu wählende Konstante des Algorithmus ist.
3. Wähle mit Wahrscheinlichkeit  $\frac{\text{ANZAHL}(v')}{\sum_{w \in L} \text{ANZAHL}(w')}$  den Knoten  $v'$  als Ziel des Mehrfachsprungs.

Konkret könnte also ein Inhaltlicher Mehrfachsprung sowie wie in Algorithmus 1 beschrieben umgesetzt werden.

---

#### Algorithmus 1 Inhaltlicher Mehrfachsprung

---

```

1: function INHALTLICHERMEHRFACHSPRUNG(Knoten  $v$ )
2:   //Alle Knoten bestimmen, die von  $v$  aus über Pfade der Länge 2 erreichbar sind
3:   //Zusätzlich wird für diese Knoten die Anzahl der Pfade der Länge 2 bestimmt,
4:   //durch die sie erreichbar sind
5:    $reachableNodes \leftarrow$  defaultdict
6:   for each Wortknoten  $w$  in  $v.GETWORDNODES()$  do
7:     for each Strukturknoten  $x$  in  $w.GETSTRUCTURALNODES()$  do
8:        $reachableNodes[x] \leftarrow reachableNodes[x] + 1$ 
9:   //Im folgenden gehe ich davon aus, dass ich über Indizes wahlfrei auf Elemente
10:  //aus  $M_H$  zugreifen kann. Dies muss bei der konkreten Wahl der Datenstruktur
11:  //berücksichtigt werden
12:   $M_H \leftarrow \text{MAX}(reachableNodes, q)$   $\triangleright$  Also:  $|M_H| = q$ , falls  $|reachableNodes| \geq q$ 
13:  //Generate dictionary with relative frequencies
14:   $s \leftarrow 0$ 
15:  for each Knoten  $x$  in  $M_H$  do
16:     $s \leftarrow s + reachableNodes[x]$ 
17:   $relativeFrequency \leftarrow$  Dictionary
18:  for each Knoten  $x$  in  $M_H$  do
19:     $relativeFrequency \leftarrow \frac{reachableNodes[x]}{s}$ 
20:   $random \leftarrow \text{RANDOM}(0, 1)$ 
21:   $s \leftarrow 0$ 
22:   $i \leftarrow 0$ 
23:  while  $s < random$  do
24:     $s \leftarrow s + relativeFrequency[i]$ 
25:     $i \leftarrow i + 1$ 
26:   $v \leftarrow M_H[i - 1]$ 
27:  return  $v$ 

```

---

**Algorithmus 2** Struktureller Sprung

---

```

1: function STURKTURELLERSPRUNG(Knoten  $v$ , Anzahl  $q$ )
2:    $n \leftarrow v$ .NEIGHBORCOUNT           ▷ Wähle aus der Liste der Nachbarknoten
3:    $r \leftarrow \text{RANDOMINT}(0, n - 1)$        ▷ einen zufällig aus
4:    $v \leftarrow v$ .NEXT( $r$ )                 ▷ Gehe zu diesem Knoten
5:   return  $v$ 

```

---

**2.4 Vokabularbestimmung**

Da die Größe des Vokabulars die Datenmenge signifikant beeinflusst, liegt es in unserem Interesse so wenig Wörter wie möglich ins Vokabular aufzunehmen. Insbesondere sind Wörter nicht von Interesse, die in fast allen Texten vorkommen, wie im Deutschen z. B. „und“, „mit“ und die Pronomen.

Nun kann man manuell eine Liste von zu beachtenden Wörtern erstellen oder mit Hilfe des Gini-Koeffizienten automatisch ein Vokabular erstellen. Der Gini-Koeffizient ist ein statistisches Maß, das die Ungleichverteilung bewertet. Er ist immer im Intervall  $[0, 1]$ , wobei 0 einer Gleichverteilung entspricht und 1 der größt möglichen Ungleichverteilung.

Sei nun  $n_i(w)$  die Häufigkeit des Wortes  $w$  in allen Texten mit dem  $i$ -ten Label.

$$p_i(w) := \frac{n_i(w)}{\sum_{j=1}^{|\mathcal{L}_i|} n_j(w)} \quad (\text{Relative Häufigkeit des Wortes } w) \quad (1)$$

$$G(w) := \sum_{j=1}^{|\mathcal{L}_i|} p_j(w)^2 \quad (\text{Gini-Koeffizient von } w) \quad (2)$$

In diesem Fall ist  $G(w) = 0$  nicht möglich, da zur Vokabularbestimmung nur Wörter betrachtet werden, die auch vorkommen.

Ein Vorschlag, wie die Vokabularbestimmung implementiert werden kann, ist als Pseudocode mit Algorithmus 3 gegeben. Dieser Algorithmus benötigt neben dem Speicher für den Graphen, die Texte sowie die  $m$  Vokabeln noch  $\mathcal{O}(|\text{Verschiedene Wörter in } S_t| \cdot (|\mathcal{L}_t| + 1))$  Speicher. Die Average-Case Zeitkomplexität beträgt  $\mathcal{O}(|\text{Wörter in } S_t|)$ , wobei dazu die Vereinigung von Mengen  $M, N$  in  $\mathcal{O}(\min |M|, |N|)$  sein muss.

**Algorithmus 3** Vokabularbestimmung**Input:**

- 1:  $\mathcal{T}_t$  (Knoten mit Labels),
- 2:  $\mathcal{L}_t$  (Labels),
- 3:  $f : \mathcal{T}_t \rightarrow \mathcal{L}_t$  (Label-Funktion),
- 4:  $m$  (Gewünschte Vokabulargröße)

**Output:**  $\mathcal{M}_t$  (Vokabular)

- 5:
- 6:  $S_t \leftarrow \text{SAMPLE}(\mathcal{T}_t)$  ▷ Wähle eine Teilmenge  $S_t \subseteq \mathcal{T}_t$  aus
- 7:  $\mathcal{M}_t \leftarrow \bigcup_{v \in S_t} \text{GETTEXTASSET}(v)$  ▷ Menge aller Wörter
- 8:  $cLabelWords \leftarrow (|\mathcal{L}_t| + 1) \times |\mathcal{M}_t|$ -Array, mit 0en initialisiert
- 9:
- 10: **for each**  $v \in \mathcal{T}_t$  **do** ▷ Gehe jeden Text Wort für Wort durch
- 11:      $i \leftarrow \text{GETLABEL}(v)$
- 12:     **for each**  $(word, occurrences) \in \text{GETTEXTASMULTISET}(v)$  **do**
- 13:          $cLabelWords[i][word] \leftarrow cLabelWords[i][word] + occurrences$
- 14:          $cLabelWords[i][|\mathcal{L}_t|] \leftarrow cLabelWords[i][|\mathcal{L}_t|] + occurrences$
- 15:
- 16: **for each** Wort  $w \in \mathcal{M}_t$  **do**
- 17:      $p \leftarrow$  Array aus  $|\mathcal{L}_t|$  Zahlen in  $[0, 1]$
- 18:     **for each** Label  $i \in \mathcal{L}_t$  **do**
- 19:          $p[i] \leftarrow \frac{cLabelWords[i][w]}{cLabelWords[i][|\mathcal{L}_t|]}$
- 20:      $w.gini \leftarrow \text{SUM}(\text{MAP}(\text{SQUARE}, p))$
- 21:  $\mathcal{M}_t \leftarrow \text{SORTDESCENDINGBYGINI}(\mathcal{M}_t)$
- 22: **return**  $\text{TOP}(\mathcal{M}_t, m)$

Die Menge  $S_t$  kann durch Aus der Menge aller Dokumenten, deren Knoten gelabelt sind, mithilfe des in [Vitt85] vorgestellten Algorithmus bestimmt werden.

## 2.5 Der Algorithmus

Der DYCOS-Algorithmus verwendet nun für jeden Knoten der gelabelt wird  $r$  Random Walks der Länge  $l$ , wobei mit einer Wahrscheinlichkeit  $p_S$  ein struktureller  $l$ -Sprung und mit einer Wahrscheinlichkeit von  $(1 - p_S)$  ein inhaltlicher  $l$ -Mehrfachsprung gemacht wird.

Die Vokabularbestimmung kann zu jedem Zeitpunkt  $t$  durchgeführt werden, muss es aber nicht.

Im Folgenden werde ich den DYCOS-Algorithmus als Pseudocode vorstellen. Dafür benötigt man die beiden Hilfsfunktionen für den strukturellen Sprung sowie den inhaltlichen Mehrfachsprung:

---

**Algorithmus 4** DYCOS-Algorithmus

---

**Input:**

- 1:  $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{A}_t, \mathcal{T}_t)$  (Netzwerk),
- 2:  $r$  (Anzahl der Random Walks),
- 3:  $l$  (Länge eines Random Walks),
- 4:  $p_s$  (Wahrscheinlichkeit eines strukturellen Sprungs),
- 5:  $q$  (Anzahl der betrachteten Knoten nach der Aggregatanalyse)

**Output:** Klassifikation von  $\mathcal{N}_t \setminus \mathcal{T}_t$ 

```

6:
7:
8: for each Knoten  $v$  in  $\mathcal{N}_t \setminus \mathcal{T}_t$  do
9:    $d \leftarrow$  defaultdict
10:  for  $i$  von 1 bis  $r$  do
11:     $w \leftarrow v$ 
12:    for  $j$  von 1 bis  $l$  do
13:       $sprungTyp \leftarrow$  RANDOM(0, 1)
14:      if  $sprungTyp \leq p_s$  then
15:         $w \leftarrow$  STURKTURELLERSPRUNG( $w$ )
16:      else
17:         $w \leftarrow$  INHALTLICHERMEHRFACHSPRUNG( $w$ )
18:         $w \leftarrow v.GETLABEL()$  ▷ Zähle das Label
19:         $d[w] \leftarrow d[w] + 1$ 
20:      if  $d$  ist leer then ▷ Es wurde kein gelabelter Knoten gesehen
21:         $M_H \leftarrow$  HÄUFIGSTELABELIMGRAPH()
22:      else
23:         $M_H \leftarrow$  MAX( $d$ )
24:
25:        ▷ Wähle aus der Menge der häufigsten Label  $M_H$  zufällig eines aus
26:         $label \leftarrow$  RANDOM( $M_H$ )
27:         $v.ADDLABEL(label)$  ▷ und weise dieses  $v$  zu
28: return Labels für  $\mathcal{N}_t \setminus \mathcal{T}_t$ 

```

---

### 3 Schwächen und Verbesserungsvorschläge

Der in [AgLi11] vorgestellte Algorithmus hat einige Probleme, die im Folgenden erläutert werden. Außerdem werden Verbesserungen vorgeschlagen, die es allerdings noch zu untersuchen gilt.

#### 3.1 Schwächen von DYCOS

**3.1.1 Anzahl der Labels** So, wie er vorgestellt wurde, können nur Graphen bearbeitet werden, deren Knoten höchstens ein Label haben. In vielen Fällen, wie z. B. Wikipedia mit Kategorien als Labels haben Knoten jedoch viele Labels.

Auf einen ersten Blick ist diese Schwäche einfach zu beheben, indem man beim zählen der Labels für jeden Knoten jedes Label zählt. Dann wäre noch die Frage zu klären, mit wie vielen Labels der betrachtete Knoten gelabelt werden soll.

Jedoch ist z. B. bei Wikipedia-Artikeln auf den Knoten eine Hierarchie definiert. So ist die Kategorie „Klassifikationsverfahren“ eine Unterkategorie von „Klassifikation“. Bei dem Kategorisieren von Artikeln sind möglichst spezifische Kategorien vorzuziehen, also kann man nicht einfach bei dem Auftreten der Kategorie „Klassifikationsverfahren“ sowohl für diese Kategorie als auch für die Kategorie „Klassifikation“ zählen.

**3.1.2 Überanpassung und Reklassifizierung** Aggarwal und Li beschreiben in [AgLi11] nicht, auf welche Knoten der Klassifizierungsalgorithmus angewandt werden soll. Jedoch ist die Reihenfolge der Klassifizierung relevant. Dazu folgendes minimale Beispiel:

Gegeben sei ein dynamischer Graph ohne textuelle Inhalte. Zum Zeitpunkt  $t = 1$  habe dieser Graph genau einen Knoten  $v_1$  und  $v_1$  sei mit dem Label  $A$  beschriftet. Zum Zeitpunkt  $t = 2$  komme ein nicht-gelabelter Knoten  $v_2$  sowie die Kante  $(v_1, v_2)$  hinzu.

Nun wird der DYCOS-Algorithmus auf diesen Knoten angewendet und  $v_2$  mit  $A$  gelabelt.

Zum Zeitpunkt  $t = 3$  komme ein Knoten  $v_3$ , der mit  $B$  gelabelt ist, und die Kante  $(v_3, v_2)$  hinzu.

Würde man nun den DYCOS-Algorithmus erst jetzt, also anstelle von Zeitpunkt  $t = 2$  zum Zeitpunkt  $t = 3$  auf den Knoten  $v_2$  anwenden, so würde eine 50%-Wahrscheinlichkeit bestehen, dass dieser mit  $B$  gelabelt wird. Aber in diesem Beispiel wurde der Knoten schon zum Zeitpunkt  $t = 2$  gelabelt. Obwohl es in diesem kleinem Beispiel noch keine Rolle spielt, wird das Problem klar, wenn man weitere Knoten einfügt:

Wird zum Zeitpunkt  $t = 4$  ein ungelabelter Knoten  $v_4$  und die Kanten  $(v_1, v_4)$ ,  $(v_2, v_4)$ ,  $(v_3, v_4)$  hinzugefügt, so ist die Wahrscheinlichkeit, dass  $v_4$  mit  $A$  gelabelt wird bei 75%. Werden die als ungelabelten Knoten jedoch erst jetzt und alle gemeinsam gelabelt, so ist die Wahrscheinlichkeit für  $A$  als Label bei nur 50%. Bei dem DYCOS-Algorithmus findet also eine Überanpassung an vergangene Labels statt.

Das Reklassifizieren von Knoten könnte eine mögliche Lösung für dieses Problem sein. Knoten, die durch den DYCOS-Algorithmus gelabelt wurden könnten eine Lebenszeit bekommen (TTL, Time to Live). Ist diese abgelaufen, wird der DYCOS-Algorithmus erneut auf den Knoten angewendet.

### 3.2 Schwächen des Papers

Die Ergebnisse der experimentelle Analyse können aus folgenden Gründen nicht überprüft werden:

- DYCOS verwendet als Vokabular die Top- $m$ -Wörter mit dem höchsten Gini-Index aus einer Sample-Menge von Texten, die wie in [Vitt85] beschrieben erzeugt wird. Allerdings wird niemals erklärt, wie  $m \in \mathbb{N}$  bestimmt wird. Es ist nicht einmal klar, ob  $m$  für den Algorithmus als konstant anzusehen ist oder ob  $m$  sich bei der Vokabularbestimmung ändern könnte.
- DYCOS beschränkt sich bei inhaltlichen Mehrfachsprüngen auf die Top- $q$ -Wortknoten, also die  $q$  ähnlichsten Knoten gemessen mit der Aggregatanalyse. Auch hier wird nicht erklärt wie  $q \in \mathbb{N}$  bestimmt oder nach welchen Überlegungen  $q$  gesetzt wurde. Allerdings ist hier wenigstens klar, dass  $q$  für den DYCOS-Algorithmus konstant ist. Für die Experimentelle Analyse wurde zwar erwähnt, dass  $q$  ein Parameter des Algorithmus ist [AgLi11, S. 362], aber nicht welcher Wert in der Analyse des DBLP-Datensatzes genutzt wurde. Für den CORA-Datensatz wurde  $q = 10$  gewählt [AgLi11, S. 364].
- Für die Analyse der CORA-Datensatzes<sup>1</sup> analysiert. Dieser beinhaltet Forschungsarbeiten, wobei die Forschungsgebiete die in einen Baum mit 73 Blättern eingeordnet wurden. Aus diesen 73 Blättern wurden 5 Klassen extrahiert und der Graph, der keine Zeitpunkte beinhaltet, künstlich in 10 Graphen mit Zeitpunkten unterteilt. Wie jedoch die TODO

## 4 Ausblick

Den sehr einfach aufgebauten DYCOS-Algorithmus kann man noch an vielen Punkten verbessern. So könnte man vor der Auswahl des Vokabulars jedes Wort auf den Wortstamm zurückführen. Dafür könnte zum Beispiel der Porter-Stemming-Algorithmus verwendet werden [Port97]. Durch diese Maßnahme wird das Vokabular kleiner gehalten, mehr Artikel können mit einander durch Vokabular verbunden werden und der Gini-Koeffizient wird ein besseres Maß für die Gleichheit von Texten.

Eine weitere Verbesserungsmöglichkeit besteht in der Textanalyse. Momentan ist diese noch sehr einfach gestrickt und ignoriert die Reihenfolge von Wörtern beziehungsweise Wertungen davon. So könnte man den DYCOS-Algorithmus in einem sozialem Netzwerk verwenden wollen, in dem politische Parteiaffinität von einigen Mitgliedern angegeben wird um die Parteiaffinität der restlichen Mitglieder zu bestimmen. In diesem Fall macht es jedoch einen wichtigen Unterschied, ob jemand über eine Partei gutes oder schlechtes schreibt.

Eine einfache Erweiterung des DYCOS-Algorithmus wäre der Umgang mit mehreren Labels.

<sup>1</sup> inzwischen unter <http://people.cs.umass.edu/mccallum/data/cora-classify.tar.gz>

## Literatur

- AgLi11. Charu C. Aggarwal und Nan Li. On Node Classification in Dynamic Content-based Networks. In *SDM* [AgLi11], S. 355–366.
- BhCM11. Smriti Bhagat, Graham Cormode und S. Muthukrishnan. Node Classification in Social Networks, 2011.
- Port97. M. F. Porter. Readings in Information Retrieval. Kapitel An Algorithm for Suffix Stripping, S. 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
- Vitt85. Jeffrey S. Vitter. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.*, 11(1), 1985, S. 37–57.