Detecting Academic Plagiarism with Graphs

Bin-Hui Chou, Einoshin Suzuki

Dept. Informatics, Kyushu University, Japan
chou@i.kyushu-u.ac.jp, suzuki@inf.kyushu-u.ac.jp

Abstract. In this paper, we tackle the problem of detecting academic plagiarism, which is considered as a severe problem owing to the convenience of online publishing. Typical information retrieval methods, stopword-based methods and fingerprinting methods, are commonly used to detect plagiarism by using the sequence of words as they appear in the article. As such, they fail to detect plagiarism when an author reconstructs a source article by re-ordering and recombining phrases. Because graph structure fits for representing relationships between entities, we propose a novel plagiarism detection method, in which we use graphs to represent documents by modeling grammatical relationships between words. Experimental results show that our proposed method outperforms two n-gram methods and increases recall values by 10 to 20%.

1 Introduction

Online publishing provides a platform for researchers to share their research results while it also brings a severe side effect, the academic plagiarism problem. That is, students or researchers copy all the content or a part of passages from others' papers without appropriate citation (Howard, 1995). It is difficult for editors of proceedings and journals to discover all the plagiarism behaviors due to the time limitation and the quantity of publications. An automatic detection method can be used to help editors' jobs and to mitigate the problem.

Existing methods of plagiarism detection evaluate document similarities by using content words (Gustafson et al., 2008; Hoad and Zobel, 2003), stopwords (Stamatatos, 2011) or document fingerprints (Seo and Croft, 2008; Schleimer, 2003). As common in information retrieval (IR), methods (Grman and Ravas, 2011; Gustafson et al., 2008; Hoad and Zobel, 2003) discard stopwords, e.g., "the", "is", and regard the remaining content words as meaningful words. This kind of methods use sequences of the content words to represent a document. Stamatatos (2011) considers that a plagiarist may replace content words to avoid detection, and proposed to represent documents by removing content words but retaining stopwords. Seo and Croft (2008); Schleimer (2003) use hashes of fixed-length chunks as document fingerprints.

Both directly copying and paraphrasing passages without citation are considered as academic plagiarism (Howard, 1995; Rosamond, 2002). Here, we aim to detect plagiarized documents where one paraphrases text from other documents by re-ordering phrases or altering modifiers. Most of the existing approaches use sequences of words as they appear in the document to represent the document so they fail to detect this kind of plagiarism.

To detect this kind of plagiarism, we consider representing a document by modeling relationships between pairs of words in the document. By capturing relationships among words, we are still able to detect plagiarism even if a plagiarist largely alters the order of phrases. In this paper, we propose a novel plagiarism detection method by representing documents with graphs. In our method, each document is transformed to graph structure according to syntactical relationships between words. We detect a plagiarism if the two graphs contain similar subgraphs. Experimental results show that our method is more effective than existing methods in detecting paraphrasing plagiarism.

2 Motivation and Problem Definition

2.1 Motivation

Consider an example shown in Table 1, where text A is an excerpt from (Shi and Malik, 2000) and text B is a text that we re-wrote from Text A by re-ordering the phrases in A and adding/removing words without altering its content. Texts A and B depict the same concept but have different expressions in their sentences and constructions. We regard texts A and B as a source document and a plagiarized document, respectively.

TAB. 1 – *Example of a source and plagiarized documents.*

(a) Text A (Source)	(b) Text B (Plagiarism)
We propose a novel approach for solving the perceptual grouping problem in vision. Our approach aims at extracting the global impression of an image. We treat image segmentation as a graph partitioning problem and propose a novel global criterion, the normalized cut, for segmenting the graph.	In this paper, we treat image segmentation, i.e., the perceptual grouping problem in vision, as a graph partitioning problem and aim to extract the global impression of an image. We propose a novel global criterion for segmenting the graph, called the normalized cut.

We categorize the existing methods into typical IR methods (Grman and Ravas, 2011; Gustafson et al., 2008; Hoad and Zobel, 2003), fingerprinting methods (Schleimer, 2003; Seo and Croft, 2008) and a stopword-based method (Stamatatos, 2011). The typical IR methods often evaluate the similarity of two documents by splitting text into sequences of words of a specified length and comparing the number of common words. Since the order of phrases in text B in Table 1 is largely changed, which decreases the number of common words between word sequences, the typical IR methods fail to detect such a kind of plagiarism.

The existing fingerprinting methods (Schleimer, 2003; Seo and Croft, 2008) represent a document by using hashes of fixed length chunks. Both characters and words can be used for chunks but most of them (Schleimer, 2003; Seo and Croft, 2008) consider word n-grams. Thus, adding or removing a small number of words alters the hashing result, making fingerprinting methods less effective in detecting plagiarism where a plagiarist largely modifies the order of words or phrasing (Seo and Croft, 2008). Stamatatos (2011) discards all the content words in a document and proposed the stopword n-grams method (SWNG). Since the order of stopwords is changed as well in text B in Table 1, SWNG is also not effective in detecting the plagiarism.

From the example, we are motivated to invent a plagiarism detection method, where the representation of a document is based on relationships among words instead of their occurrence positions. Precisely speaking, we use graphs to model grammatical relationships between pairs of words. Our method represents syntactical structures of documents such that our method is able to detect the plagiarism even if words in a document are re-arranged. As shown in Figure 1, texts in Table 1 are different but their graph structures are similar in our transformation.

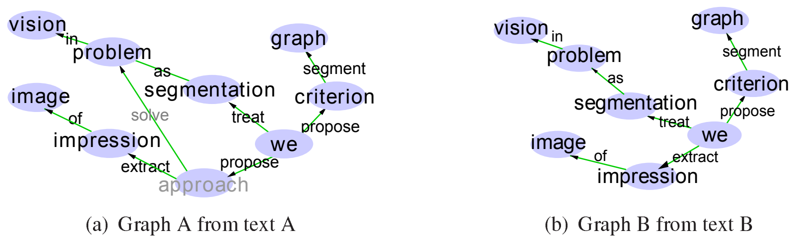


FIG. 1 – *An example of modeling documents in graphs.*

2.2 Definition of Plagiarism Detection

Here we formalize the problem. We consider monolingual plagiarism detection in this work. The input is a set D_{src} of source documents and a set D_{susp} of suspicious documents, the minimum number k of common nodes in a candidate subgraph, and the maximum length δ of a path, which we will discuss in the following section. Given a source and a suspicious document, our task is to decide whether there exist any plagiarized passages in the suspicious document, and discover their corresponding source passages if plagiarized passages exist.

We adopt the definition in (Potthast et al., 2010). Let a plagiarism $s = \langle s_{\mathrm{plg}}, d_{\mathrm{plg}}, s_{\mathrm{src}}, d_{\mathrm{src}} \rangle$ as a 4-tuple that contains a passage s_{plg} in a document d_{plg} that is the plagiarized version of a certain source passage s_{src} in document d_{src}. Given d_{plg}, the task of a plagiarism detector is to detect s by reporting a plagiarism detection $r = \langle r_{\mathrm{plg}}, d_{\mathrm{plg}}, r_{\mathrm{src}}, d'_{\mathrm{src}} \rangle$ that consists of an allegedly plagiarized passage r_{plg} in document d_{plg} and its source r_{src} in d'_{src}. A detection r is defined as: r detects $s \iff s_{\mathrm{plg}} \cap r_{\mathrm{plg}} \neq \emptyset$, $s_{\mathrm{src}} \cap r_{\mathrm{src}} \neq \emptyset$, and $d_{\mathrm{src}} = d'_{\mathrm{src}}$.

3 Proposed Approach

3.1 Overview

We transform each document to a graph, in which syntactical relationships among words are preserved. After the transformation, similar texts are expected to have similar structures so we propose a similarity measure and a graph matching algorithm.

Our approach includes four procedures in each run of detection: the pre-processing, transformation, matching and post-processing steps. In the pre-processing step, we obtain lemmas, offsets and lengths of words and resolve coreference relations of pronouns by the Stanford

Algorithm 1: Transforming a document to a graph

```
Input  : A document $d$, which contains $s_1, s_2, \ldots, s_n$ sentences
Output: A graph $G = \langle V, E \rangle$
1  $G \leftarrow \emptyset$;
2  for $i \leftarrow 1$ to $n$ do
3      $\Gamma \leftarrow$ all the dependency relations in $s_i$;
4      foreach $r \in \Gamma$ do
5          switch $r.\psi$ do
6              case nsubj nsubj($r, \Gamma, G$); break;
7              case xsubj xsubj($r, \Gamma, G$); break;
8              case iobj iobj($r, \Gamma, G$); break;
9              case agent agent($r, \Gamma, G$); break;
10             case prep prep($r, \Gamma, G$); break;
11             case prepc prepc($r, \Gamma, G$); break;
12             case partmod parmod($r, \Gamma, G$); break;
13 MergeNodes($G$);
14 return $G$;
```

CoreNLP (Klein and Manning, 2003; Raghunathan et al., 2010). We will discuss the transformation and matching steps in sections 3.2 and 3.3, respectively. In the post-processing step, we transform the discovered pairs of similar subgraphs to their corresponding passages. We obtain a plagiarized passage by including words located in the range of the minimal and maximal offsets of words among nodes in the subgraph.

We use the case of one suspicious document and one source document to simplify discussions. We can obtain results of D_{susp} and D_{src} by performing $n_{\text{susp}} n_{\text{src}}$ runs, where n_{susp} and n_{src} represent the numbers of documents in D_{susp} and D_{src}, respectively. Let a suspicious document and a source document be d_{susp} and d_{src}, respectively. Let graphs generated from d_{susp} and d_{src} in the transformation step be G and H, respectively. Both G and H are directed graphs, and nodes and edges have labels representing words. $V(G)$ and $E(G)$ denote the set of nodes and the set of edges in G, respectively. $L_G(v)$ and $L_G(x, y)$ denote labels of node v and edge (x, y) in G, respectively. $L_G(v)$ is abbreviated to $L(v)$ when there is no ambiguity.

3.2 Transformation to Graphs

We think that nouns are essential elements in a sentence and verbs or prepositions are usually related to nouns. Thus, we intuitively regard nouns as nodes and verbs or prepositions as edges in the transformation. Intuitively speaking, if we are able to find a verb or a preposition "relating" two nouns, we create a directed link between them. Instead of deriving directly from word positions, we derive the graph representation of text by modeling grammatical relationships between words. Algorithm 1 shows an overview of transforming sentences in document d to graph G. We invent procedures in lines to 6 to 12 to generate nodes and edges

```
Function prep($r, \Gamma, G$)
  Input  : A prep relation $r = \langle \psi, w_{\text{gov}}, w_{\text{dep}} \rangle$, where $\psi$ denotes a string of combining
           “prep” with a preposition by an underscore (e.g., “prep_into”), the set $\Gamma$ of
           relations of the sentence that $r$ belongs to, and graph $G = \langle V, E \rangle$
  Output: Updated graph $G$
1  if $r.w_{\text{gov}}$ is a verb then
2      $R \leftarrow$ rFindDobj($r, r.w_{\text{gov}}, \Gamma$);
3      $R \leftarrow R \cup$ rFindNsubj($r, r.w_{\text{gov}}, \Gamma$);
4      $R \leftarrow R \cup$ rFindNsubjpass($r, r.w_{\text{gov}}, \Gamma$);
5      while $R \neq \emptyset$ do
6          $L(v) \leftarrow$ pop($R$).$w_{\text{dep}}$; $L(u) \leftarrow r.w_{\text{dep}}$;
7          $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\text{gov}}$;
8          Insert $v$, $u$ and $e$ to $G$ ;   /* When inserting a node, $v$, we check if
           there exists $L(v)$ among existing node labels in $G$ */
9  else if $r.w_{\text{gov}}$ is a noun then
10     $L(v) \leftarrow r.w_{\text{gov}}$; $L(u) \leftarrow r.w_{\text{dep}}$;
11     $e \leftarrow (v, u)$; $L(e) \leftarrow$ the preposition after the underscore in $\psi$;
12     Insert $v$, $u$ and $e$ to $G$;
```

based on the grammatical relationships, i.e., dependency relations (explained later). After a graph is generated, we further merge nodes according to their coreference relationships.[1]

To identify grammatical relations between nouns, we use the Stanford parser (version 2.0.1) (Klein and Manning, 2003), which provides the Stanford typed dependency relations (de Marneffe and Manning, 2008). A dependency relation is a simple description of the grammatical relationship between words in the format of triples, $\langle \psi, w_{\text{gov}}, w_{\text{dep}} \rangle$, which represents a grammatical relation ψ between a governor word w_{gov} and a dependent word w_{dep}. A sentence is composed of an ordered set of dependency relations. For instance, given sentence “Community detection is the problem of clustering nodes in a graph into communities”, one of the obtained relations, relation $\langle$dobj, clustering, nodes$\rangle$ indicates that word nodes is a direct object of word clustering. Relation $\langle$prep_into, clustering, communities$\rangle$ indicates that word communities accompanied with into is a prepositional modifier of verb clustering.

Because dependency relations do not necessarily contain just one pair of nouns and a verb/preposition representing the relationship between the nouns, we refer to multiple relations to obtain related nouns and their corresponding verb/preposition in most cases. The intuition behind the heuristic rules is that we wish to capture as many nouns as possible from subjects, objects or complements of sentences. Specifically speaking, we find pairs of nodes, where each pair is associated with a verb/preposition, by searching for relations nsubj, nsubjpass, or dobj. We consider that dependency relations involved in the part of the subject, object and complement of a sentence include relations nsubj, xsubj, iobj, agent, prep, prepc, and

[1]Suppose v is merged to its counterpart u. We copy all adjacent edges of v to u and then we neglect v in the graph.

partmod.[2] Each of the relations has a corresponding searching rule as shown in Algorithm 1.

Function prep shows how we find related words when given a relation where ψ is prep (prepositional modifier). In function prep, $r.w_{\mathrm{gov}}$ denotes governor word w_{gov} of relation r. rFindDobj$(r, r.w_{\mathrm{gov}}, \Gamma)$ returns all the dobj relations located in front of r in Γ each of whose w_{gov} is $r.w_{\mathrm{gov}}$.[3] Similarly, we can define rFindSubj$(\cdot)$ and rFindNsubjpass$(\cdot)$. pop(R) pops out a relation from the set R of relations. With function prep, we generate a subgraph of the sentence, node $\xrightarrow{\text{into}}$ community, when given relations $\langle$dobj, clustering, nodes$\rangle$ and $\langle$prep_into, clustering, communities$\rangle$. The rest of heuristic rules are shown in Figure 3.

3.3 Graph Matching

If two texts are similar, their transformed graph structures are expected to be similar. Thus, we interpret a plagiarism detection as the discovery of a pair of similar subgraphs in this paper. Below we will define the similarity of two subgraphs and propose a discovery algorithm.

3.3.1 Similarity Definition

Due to the diversity in natural language expressions, a definition of exact match limits its ability of discovering similar subgraphs in a real application. To detect similar texts that have different graphs, we consider inexact matching in our definition of similar subgraphs.

Consider sentences, "TextRunner is an OIE system which extracts relational tuples" and "TextRunner extracts relational tuples". We notice that even if there exist additional words (e.g., OIE and system) in the first sentence, the two sentences still have the same meaning because the additional words are between common words (e.g., TextRunner, tuples) such that the additional words function as modifiers. In the graph level, we call common words as *common nodes* and additional words as *uncommon nodes*. We define similar subgraphs by allowing uncommon nodes to exist in the path between common nodes.

Let a subgraph of G and a subgraph H be g and h, respectively. Consider the case of two common nodes for the time being. Subgraphs g and h are similar subgraphs if they satisfy:

1. (**Node Similarity**) $L_g(\alpha) = L_h(a) \wedge L_g(\beta) = L_h(b)$ [4]
2. (**Path Similarity**) $Path(\alpha, \beta)$ is similar to $Path(a, b)$,

where a and b are common nodes in g, and α and β are common nodes in h. The four nodes are common nodes. $Path(x, y)$ represents the shortest path between x and y. Since there exist uncommon nodes, we consider the similarity of paths instead of edges.

Definition. (***Path Similarity***) *Let path $Path(v_1, v_n)$ be the shortest path from node v_1 to v_n in G, and path $Path(u_1, u_m)$ be the shortest path from node u_1 to u_m in H.[5] If $|Path(v_1, v_n)| \leq \delta$, $|Path(u_1, u_m)| \leq \delta$, and $L(v_{n-1}, v_n) = L(u_{m-1}, u_m)$, we say that paths $Path(v_1, v_n)$ and $Path(u_1, u_m)$ are similar.*

[2] nsubj, nsubjpass, dobj, xsubj, iobj, prepc and partmod represent nominal subject, passive nominal subject, direct object, controlling subject, indirect object, prepositional clausal modifier, and participial modifier, respectively.

[3] FindDobj$(r, r.w_{\mathrm{gov}}, \Gamma)$ returns all the dobj relations located in back of r, used in other rules.

[4] To simplify the explanation, we ignore the problem of synonyms here. In fact, we consider synonyms of words in our approach by checking the synonym set of a word with WordNet (Miller, 1995). In other words, $L(x) = L(y)$ if x is a synonym of y and so are labels of edges.

[5] If there are two shortest paths, we check the two.

Algorithm 2: Overview of our discovery algorithm

```
Input  : A suspicious graph $G$, a source graph $H$, $k$, $\delta$
Output: Pairs $(g, h)$ of similar subgraphs between $G$ and $H$
1 $(V_s, U_s) \leftarrow \{v \in V(G), u \in V(H) \mid$
  $v$ and $u$ are common nodes; there is a 1-to-1 mapping relationship between $v$ and $u\}$;
2 while PopSeeds($V_s, U_s$) $\neq \emptyset$ do
3     $(v_s, u_s) \leftarrow$ PopSeeds($V_s, U_s$);
4     if $v_s$ and $u_s$ are not included in any pair of $(g, h)$ then
5         $(g, h) \leftarrow$ match($G, H, v_s, v_u, \delta$);
6         if SimNodesNum($g, h$) $\geq k$ then
7             output($g, h$); $g \leftarrow \emptyset$; $h \leftarrow \emptyset$;
```

Let path p_1 denotes $\alpha \rightarrow x_1 \rightarrow \cdots \rightarrow x_n \rightarrow \beta$, where α and β are common nodes while each x_i represents an uncommon node. Similarly, we define path p_2 as $a \rightarrow y_1 \rightarrow \cdots \rightarrow y_m \rightarrow b$. To define the similarity between p_1 and p_2, we need to consider: the maximum number of uncommon nodes that we allow in the paths (i.e., the maximum length of paths), and labels of edges. The more the uncommon nodes are allowed, the less dissimilar the subgraphs are. Therefore, we set a threshold, δ, to determine the length of a path between common nodes.

We define $L(v_{n-1}, v_n) = L(u_{m-1}, u_m)$ due to two reasons. The numbers of edges from v_1 to v_n and those from u_1 to u_m may be different so comparing each pair of them is difficult. Furthermore, as shown in the sentences, consider that a plagiarist breaks a sentence apart by inserting modifier phrases or clauses. The last verb or preposition in the inserted phrases is the edge connecting a common node so we believe that it is more important than others.

3.3.2 Discovery Algorithm

Our goal is to discover pairs (g, h) of maximal similar subgraphs between a suspicious graph G and a source graph H in our algorithm. With the definitions in the previous section, we can determine whether g with two common nodes α and β is similar to h with two common nodes a and b. If g and h are similar, we can further use α and β, and a and b as seeds and search in their proximity to enlarge g and h by concatenating newly found subgraphs g' and h', which also have two common nodes, with g and h, respectively.

Algorithm 2 shows an overview of our discovery algorithm. We first find all the pairs of common nodes. We do not match v to other nodes if v is matched to node u so each pair of nodes has a one-to-one mapping relationship. Given a pair of common nodes, we search for their maximal similar subgraphs in match($\cdot$). Since we compare common words from the beginning to the end of texts, PopSeeds($\cdot$) returns common nodes in their order. If a returned subgraph has few common nodes, i.e., fewer than threshold k, we consider it as a trivial discovery so we discard it, checked in function SimNodesNum. SimNodesNum($\cdot$) returns the number of common nodes between a pair of subgraphs.

Function match shows how we search for a pair of maximal similar subgraphs. Our strategy is to check whether we can expand a subgraph of one common node to a subgraph of two common nodes and so on, until no more common nodes can be included in the graph.

```
Function match($G, H, v_s, u_s, \delta$)
   Input  : $G, H$, seed nodes $v_s$ and $u_s$, length of a path $\delta$
   Output: Nodes $V(g), V(h)$ of subgraphs $g, h$
   /* We estimate the passage of a candidate subgraph from nodes,
      which have information of character offsets and lengths so we
      only generate nodes in $g$ and $h$ in our implementation.      */
1  $g \leftarrow \emptyset$; $h \leftarrow \emptyset$; $S_1 \leftarrow \emptyset$; $S_2 \leftarrow \emptyset$;
2  $V(g) \leftarrow V(g) \cup \{v_s\}$; $V(h) \leftarrow V(h) \cup \{u_s\}$; $S_1 \leftarrow S_1 \cup \{v_s\}$; $S_2 \leftarrow S_2 \cup \{u_s\}$;
3  repeat
4      $v \leftarrow$ pop($S_1$); $u \leftarrow$ pop($S_2$); $(V, U) \leftarrow$ FindCandiNodes($G, H, v, u, \delta$);
5      foreach $(x, y) \in (V, U)$ do
6          if there exists a path between $v$ and $x$ && there exists a path between $u$ and $y$
           then
7              if $x$ and $y$ are not included in any pair of $(g, h)$ && PathSim($v, x, u, y$)
               then
8                  $V(g) \leftarrow V(g) \cup \{x\}$; $V(h) \leftarrow V(h) \cup \{y\}$;
9                  $S_1 \leftarrow S_1 \cup \{x\}$; $S_2 \leftarrow S_2 \cup \{y\}$;
10 until $S_1 = \emptyset$, $S_2 = \emptyset$;
11 return $V(g), V(h)$;
```

FindCandiNodes(G, H, v, u, δ) finds common nodes that are near within length δ of v or u in the underlying graphs obtained by replacing all directed edges with undirected edges. For example, FindCandiNodes($\cdot$) returns adjacent nodes of node v that are common nodes in the undirected version of G when $\delta = 1$. In PathSim(v, x, u, y), we check both similarity σ_1 between $Path(v, x)$ and $Path(u, y)$ and similarity σ_2 between $Path(x, v)$ and $Path(y, u)$. PathSim(v, x, u, y) returns true when either σ_1 or σ_2 is true and returns false, otherwise.

4 Experiments

We compare our method with a naive n-grams method, which uses content words as the representation of a document, and a state-of-the-art method called SWNG (stopword n-grams method) (Stamatatos, 2011). In SWNG, θ, which is an upper threshold of gap-length allowed in a passage, is set to be 100 as the author suggests. We examine many values of n used in both the naive n-grams method and SWNG in our experiments and we show the results when $n = 5, 6$ that we consider their best performances. In our method, k, which is a threshold deciding the minimal number of common nodes required in an output subgraph, is set to be 3. A large value for δ implies that many uncommon words are allowed to be included in pairs of similar subgraphs. To avoid discovery of dissimilar passages, we set δ to a small value, $\delta = 2$.

4.1 Data Sets and Evaluation Metrics

We use the DBLP-citation network data set (Tang et al., 2008) in our experiments, where each record, i.e., an article, is generally assigned information of authors, venue, abstract, references, and publication year.From the citation network data set, we prepare two data sets, DBLP1 and DBLP2, used to examine performances of methods when the whole content of a document is plagiarized and only a part of content of a document is plagiarized, respectively.

In DBLP1 data set, we select a set of papers D, which are published in the proceedings of top conferences such as KDD, and use their abstracts D_{src} as test documents in the experiments. For each document d in D, we prepare a corresponding plagiarized document d_{plg} by splitting sentences, re-organizing and re-ordering phrases without altering the content, adding minor words and randomly replacing nouns and verbs with words from their synonym sets by WordNet (Miller, 1995). We also use abstracts D_{ref} of the papers cited in papers of D for obfuscation. To sum up, suspicious documents D_{susp} include all the plagiarized documents D_{plg} and D_{ref}. $|D_{\mathrm{src}}| = 20$. $|D_{\mathrm{susp}}| = 89$, where $|D_{\mathrm{ref}}| = 69$, which are non-plagiarized documents, and $|D_{\mathrm{plg}}| = 20$, which are plagiarized documents.

In DBLP2 data set, source documents are the same as in DBLP1. There are 100 suspicious documents, among which 31 and 69 documents are partially plagiarized and non-plagiarized documents, respectively. Half sentences in each of the partially plagiarized documents are extracted from a document in D_{plg} and half from one of the corresponding document in D_{ref}.

We use the measures of precision, recall, granularity and an overall metric combining precision, recall and granularity, *PlagDet*, which are proposed for plagiarism detection (Potthast et al., 2010), to evaluate experimental results. Let S denote the set of plagiarisms in the suspicious documents of a corpus, and let R denote the set of plagiarism detections that a detector reports. To simplify the notation, a plagiarism s, $s \in S$, is represented as a set $\mathbf{s}$ of references to the characters of d_{plg} and d_{src} that form the passages s_{plg} and s_{src}. Similarly, a plagiarism detection r, $r \in R$, is represented as $\mathbf{r}$.

$$prec(S,R) = \frac{1}{|R|}\sum_{r\in R}\frac{|\bigcup_{s\in S}(\mathbf{s}\sqcap\mathbf{r})|}{|\mathbf{r}|} \qquad rec(S,R) = \frac{1}{|S|}\sum_{s\in S}\frac{|\bigcup_{r\in R}(\mathbf{s}\sqcap\mathbf{r})|}{|\mathbf{s}|}$$

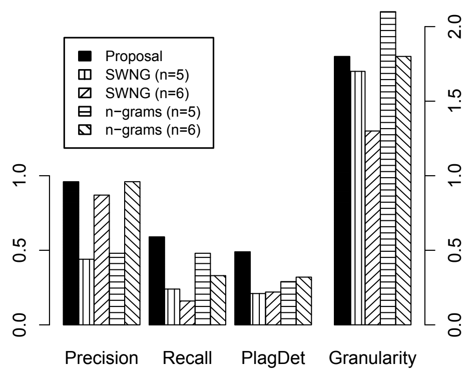
where $\mathbf{s}\sqcap\mathbf{r} = \mathbf{s}\cap\mathbf{r}$ if r detects s, and $\mathbf{s}\sqcap\mathbf{r} = \emptyset$ otherwise. A high score of precision represents that many of the detected passages are plagiarized passages. A high score of recall represents that the detector identifies many plagiarized passages.

Because plagiarism detectors may report overlapping or multiple detections for a single plagiarism, we use the granularity measure besides precision and recall. The granularity (Potthast et al., 2010) is defined as, $gran(S,R) = \frac{1}{|S_R|}\sum_{s\in S_R}|R_s|$, where $S_R \subseteq S$ are cases detected by detections in R, and $R_s \subseteq R$ are detections of s. A high score of granularity represents that many segments are reported as detections of the same plagiarized passage.

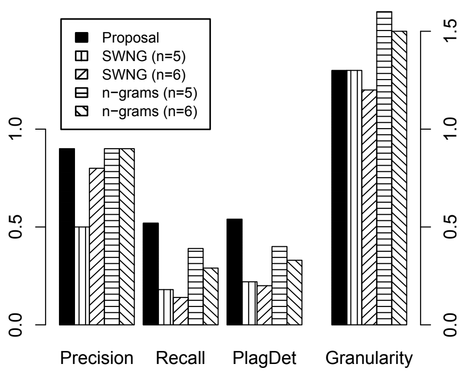
Combining precision, recall and granularity, PlagDet is defined as $PlagDet(S,R) = \frac{F_1}{\log_2(1+gran(S,R))}$, where $F_1 = 2 \cdot \frac{prec(S,R)\cdot rec(S,R)}{prec(S,R)+rec(S,R)}$. Values of precision, recall and PlagDet except granularity range from 0 to 1. The minimum and ideal value for granularity is 1.

4.2 Experimental Results

Figures 2(a) and 2(b) show experimental results on the DBLP1 and DBLP2 data sets, respectively. From the figures, we see that our method outperforms SWNG and the naive n-grams method in the experiments of detecting both wholly and partially plagiarized passages.



(a) DBLP1 results



(b) DBLP2 results

FIG. 2 – *Experimental results on the DBLP1 and DBLP2 data sets.*

Although our method has a larger tendency to obtain overlapping results than SWNG, which is observed from the values of granularity, our method achieves higher scores on precision, recall and PlagDet than SWNG. That is, our method shows its competitiveness in detecting plagiarized passages, which are largely modified by re-ordering words and phrases.

By comparing the two figures, we observe that all the methods obtain more favorable results in the DBLP1 experiments than in the DBLP2 experiments. This is because the problem of detecting documents that have partially plagiarized contents is intuitively more challenging than the problem of detecting documents that all of their contents are plagiarized from other documents. Despite of the difficulty of the problem, results of our method are still superior to those of SWNG and the n-grams method.

We examine the graphs that we transform from text. A generated graph is occasionally not a connected graph due to the diversity in natural language expressions. Since our method detects plagiarized passages by searching for common nodes along paths in the graph, disconnected graphs may cause unsuccessful detections of plagiarized passages. This is probably why our method does not achieve higher scores on recall than 0.6. However, our method is still effective in detecting plagiarism of re-ordered or re-constructed text compared to the existing methods, showing the advantage of modeling relationships between words with graph structure.

5 Conclusions and Future Work

We proposed a method of detecting plagiarism by representing documents with graphs. We transform documents to graphs according to grammatical relationships between words, and discover pairs of similar subgraphs, each of which is a detection of plagiarism. Experimental results show that our method largely improves the recall values and is more effective in detecting paraphrasing plagiarism than the existing methods. In our future work, we consider comparing our method with other graph-based methods used to detect software plagiarism such as GPLAG (Liu et al., 2006) and analyzing how our proposal scales with respect to the number and size of documents. We also consider the extension of our method to patent infringement.

Function agent(r, Γ, G)

```
$R \leftarrow$ rFindNsubjpass($r, r.w_{\mathrm{gov}}, \Gamma$);
while $R \neq \emptyset$ do
    $L(v) \leftarrow$ pop($R$)$.w_{\mathrm{dep}}$;
    $L(u) \leftarrow r.w_{\mathrm{dep}}$;
    $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{gov}}$;
    Insert $v$, $u$ and $e$ to $G$;
```

Function nsubj(r, Γ, G)

```
if $r.w_{\mathrm{gov}}$ is a verb then
    $R \leftarrow$ FindDobj($r, r.w_{\mathrm{gov}}, \Gamma$);
    while $R \neq \emptyset$ do
        $\gamma \leftarrow$ pop($R$);
        $L(v) \leftarrow r.w_{\mathrm{dep}}$; $L(u) \leftarrow \gamma.w_{\mathrm{dep}}$;
        $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{gov}}$;
        $V \leftarrow V \cup \{v\}$; $V \leftarrow V \cup \{u\}$;
        $E \leftarrow E \cup \{e\}$;
else if $r.w_{\mathrm{gov}}$ is a noun then
    $L(v) \leftarrow r.w_{\mathrm{dep}}$; $L(u) \leftarrow r.w_{\mathrm{gov}}$;
    $e \leftarrow (v, u)$; $L(e) \leftarrow$ "be";
    $V \leftarrow V \cup \{v\}$; $V \leftarrow V \cup \{u\}$;
    $E \leftarrow E \cup \{e\}$;
```

Function xsubj(r, Γ, G)

```
$R_1 \leftarrow$ FindPrep($r, r.w_{\mathrm{gov}}, \Gamma$);
while $R_1 \neq \emptyset$ do
    $L(v) \leftarrow r.w_{\mathrm{dep}}$;
    $L(u) \leftarrow$ pop($R_1$)$.w_{\mathrm{dep}}$;
    $e \leftarrow (v, u)$; $L(e) \leftarrow$ pop($R_1$)$.w_{\mathrm{gov}}$;
    Insert $v$, $u$ and $e$ to $G$;
$R_2 \leftarrow$ FindDobj($r, r.w_{\mathrm{gov}}, \Gamma$);
while $R_2 \neq \emptyset$ do
    $L(v) \leftarrow r.w_{\mathrm{dep}}$;
    $L(u) \leftarrow$ pop($R_2$)$.w_{\mathrm{dep}}$;
    $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{gov}}$;
    Insert $v$, $u$ and $e$ to $G$;
```

Function partmod(r, Γ, G)

```
$R \leftarrow$ all relations in back of $r$ each of whose $w_{\mathrm{gov}}$ is $r.w_{\mathrm{dep}}$ ;
while $R \neq \emptyset$ do
    $L(v) \leftarrow r.w_{\mathrm{gov}}$;
    $L(u) \leftarrow$ pop($R$)$.w_{\mathrm{dep}}$;
    $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{dep}}$;
    Insert $v$, $u$ and $e$ to $G$;
```

Function iobj(r, Γ, G)

```
$R \leftarrow$ rFindNsubj($r, r.w_{\mathrm{gov}}, \Gamma$);
while $R \neq \emptyset$ do
    $L(v) \leftarrow$ pop($R$)$.w_{\mathrm{dep}}$;
    $L(u) \leftarrow r.w_{\mathrm{dep}}$;
    $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{gov}}$;
    Insert $v$, $u$ and $e$ to $G$;
```

Function prepc(r, Γ, G)

```
if $r.w_{\mathrm{gov}}$ is a verb then
    $R \leftarrow$ rFindDobj($r, r.w_{\mathrm{gov}}, \Gamma$);
    $R \leftarrow R \cup$ rFindNsubj($r, r.w_{\mathrm{gov}}, \Gamma$);
    $R \leftarrow R \cup$
    rFindNsubjpass($r, r.w_{\mathrm{gov}}, \Gamma$);
    $R' \leftarrow$ FindDobj($r, r.w_{\mathrm{dep}}, \Gamma$);
    while $R \neq \emptyset$ do
        $\gamma_1 \leftarrow$ pop($R$);
        while $R' \neq \emptyset$ do
            $\gamma_2 \leftarrow$ pop($R'$) ;
            $L(v) \leftarrow \gamma_1.w_{\mathrm{dep}}$;
            $L(u) \leftarrow \gamma_2.w_{\mathrm{dep}}$;
            $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{dep}}$;
            Insert $v$, $u$ and $e$ to $G$;
else if $r.w_{\mathrm{gov}}$ is a noun then
    $R \leftarrow$ FindDobj($r, r.w_{\mathrm{dep}}, \Gamma$);
    while $R \neq \emptyset$ do
        $L(v) \leftarrow r.w_{\mathrm{gov}}$;
        $L(u) \leftarrow$ pop($R$)$.w_{\mathrm{dep}}$;
        $e \leftarrow (v, u)$; $L(e) \leftarrow r.w_{\mathrm{dep}}$;
```

FIG. 3 – *Rules for functions agent, partmod, nsubj, iobj, xsubj, prepc*

Acknowledgments.

This work was supported by JSPS KAKENHI Grant Number 24800049, 21300053.

References

de Marneffe, M.-C. and C. D. Manning (2008). The Stanford Typed Dependencies Representation. In *Proc. CrossParser*, pp. 1–8.

Grman, J. and R. Ravas (2011). Improved Implementation for Finding Text Similarities in Large Sets of Data - Notebook for PAN at CLEF 2011. In *Proc. PAN*.

Gustafson, N., M. S. Pera, and Y.-K. Ng (2008). Nowhere to Hide: Finding Plagiarized Documents Based on Sentence Similarity. In *Proc. WI-IAT*, pp. 690–696.

Hoad, T. C. and J. Zobel (2003). Methods for Identifying Versioned and Plagiarized Documents. *J. Am. Soc. Inf. Sci. Technol. 54*(3), 203–215.

Howard, R. M. (1995). Plagiarisms, Authorships, and the Academic Death Penalty. *College English 57*(7), 788–806.

Klein, D. and C. D. Manning (2003). Accurate Unlexicalized Parsing. In *Proc. ACL*, pp. 423–430.

Liu, C., C. Chen, J. Han, and P. S. Yu (2006). GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis. In *Proc. KDD*, pp. 872–881.

Miller, G. A. (1995). WordNet: A Lexical Database for English. *Commun. ACM 38*, 39–41.

Potthast, M., B. Stein, A. Barrón-Cedeño, and P. Rosso (2010). An Evaluation Framework for Plagiarism Detection. In *Proc. COLING*, pp. 997–1005.

Raghunathan, K., H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning (2010). A Multi-Pass Sieve for Coreference Resolution. In *Proc. EMNLP*, pp. 492–501.

Rosamond, B. (2002). Plagiarism, Academic Norms and the Governance of the Profession. *Politics 22*(3), 167–174.

Schleimer, S. (2003). Winnowing: Local Algorithms for Document Fingerprinting. In *Proc. SIGMOD*, pp. 76–85.

Seo, J. and B. Croft (2008). Local Text Reuse Detection. In *Proc. SIGIR*, pp. 571–578.

Shi, J. and J. Malik (2000). Normalized Cuts and Image Segmentation. *IEEE Trans. PAMI*, 888 – 905.

Stamatatos, E. (2011). Plagiarism Detection Based on Structural Information. In *Proc. CIKM*, pp. 1221–1230.

Tang, J., J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su (2008). ArnetMiner: Extraction and Mining of Academic Social Networks. In *Proc. KDD*, pp. 990–998.

Résumé

Dans cet article, nous nous intéressons au probléme de la détection des plagiats dans le monde académique, qui est un véritable fléau notamment en raison de la facilité d'accès aux publications sur Internet. Les méthodes classiques de recherche d'information couramment utilisées en détection de plagiat se basent sur les mots vides et l'identification de signatures, et utilisent les séquences des mots tels qu'ils se présentent dans les articles. Ces approches ne détectent par conséquent pas les situations de plagiat lorsqu'un auteur reconstruit un article en réordonnant et réorganisant les phrases. Dans ce contexte, une structure de graphe est plus adaptée pour représenter les relations entre les entités. Nous proposons ainsi une nouvelle méthode de détection de plagiat dans laquelle nous utilisons des graphes pour représenter des documents en modélisant les relations grammaticales entre les mots. Les résultats expérimentaux montrent que la méthode que nous proposons dépasse deux méthodes de n-grammes et augmente le rappel par des valeurs allant de 10 à 20%.