Single document Summarization based on Clustering Coefficient and Transitivity Analysis

Yanting Li† and Kai Cheng‡

† Graduate School of Information Science, Kyushu Sangyo University, 〒813-8503 Matukadai 1-3-2 Fukuoka

E-mail: † k09gjk14@ip.kyusan-u.ac.jp, ‡ chengk@is.kyusan-u.ac.jp

Abstract：

Document summarization is a technique aimed to automatically extract the main ideas from electronic documents. With the fast increase of electronic documents available on the network, techniques for making efficient use of such documents become increasingly important. In this paper, we propose a novel algorithm, called TriangleSum for single document summarization based on graph theory. The algorithm builds a dependency graph for the document based on syntactic dependency relation analysis. The nodes represent words or phrases of high frequency, and edges represent dependency relations between them. Then, a modified version of clustering coefficient is used to measure the strength of connection between nodes in a graph. By identifying triangles of nodes, a part of the dependency graph can be extracted. At last, a set of key sentences that represent the main document information can be extracted.

Keyword： summarization, dependency graph, clustering coefficient, transitivity analysis

1. Introduction

With the fast increase of electronic documents available on the network, techniques for making efficient use of such documents become increasingly important. Document summarization is a technique aimed to extract main ideas from electronic documents so that it is easy to get gist of the underlying document. Document summarization is related to the issues of keywords and key-phrases extraction, or text decomposition [2][3]. Some of the well-known approaches to extractive document summarization utilize supervised learning algorithms that are trained on collections of "ground truth" summaries built for a relatively large number of documents. However, they cannot be adapted to new languages or domains without training on each new type of data.

In this paper, we propose a graph theory based novel algorithm called TriangleSum for the task of single document summarization. Our algorithm extends the KeyGraph algorithm [2] for automatic keyword extraction in the following ways: Firstly, a dependency graph is built based on the extracted words with high frequency, and the dependency relationship between words. We introduce syntactic dependency relations among words so that key sentences instead of individual keywords can be extracted. Secondly, we extract heaviest triangles as anchor points of key sentences. A smodified version of clustering coefficient to measure the importance of each node so that partial dependency graph will be extracted. Thirdly, strongly connected components of the dependency graph will be extracted in terms of triangles based on the network's transitivity. This paper concentrates on the algorithmic aspects of computing these indices.

2. Preliminary Definitions

In this section, we begin with the introduction of some important definitions.

2.1. Word Frequency

The *word frequency* or *term frequency* of a word w in document D is the occurrence frequency of w in D, denoted by $tf(w)$.

Let *Stop* be the set of stop words. Let *HighFreq* be such a set of words in D that any $w \in HighFreq$ and $w \notin Stop$ satisfies that $tf(w) > \delta$ for some $\delta > 0$

.

2.2. Dependency Frequency

A document is defined as a set of sentences, denoted by D = $\{s_1, s_2, \ldots, s_m\}$, where s_k (k=1, 2,…, m) is called a sentence of D. A word w_i is said to be *dependent on* word w_j or simply w_i *depends on* w_j in sentence s_k if w_i is *syntactically modified by* word w_j, denoted by $w_i \rightarrow w_j$. For example, in sentence "Tom sent three letters to Jim this week" , Tom→sent, sent→letters, letters (to) →Jim.

Let $dep(w_i, w_j, s_k)$ be an indicator function of dependency relationship defined as follows:

$$dep(w_i, w_j, s_k) = \begin{cases} 1, & w_i \text{ depends on } w_j \text{in } s_k \\ 0, & \text{otherwise} \end{cases}$$

The dependency frequency between w_i and w_j in document D can be defined as below:

$$df(w_i, w_j) = \sum_{k=1}^{m} dep(w_i, w_j, s_k)$$

2.3. Dependency Graph

A *dependency graph* of a document D is a directed graph $G = (V, E)$, where V is a set of *nodes* and E is a set of *edges*,

V = HighFreq,

$E = \{(w_i, w_j) \mid df(w_i, w_j) > \lambda$, for some $\lambda > 0\}$

In other words, G is a weighted directed graph whose nodes represent high frequency words in D and edges represent the dependency relations in between a pair of words.

The *dependency weight (df)* of graph G=(V, E) can be denoted by the following equation:

$$df(G) = \sum_{(v_i, v_j) \in E} df(v_i, v_j)$$

2.4. Clustering Coefficient

Clustering coefficient (or ccf for short) is a measure of degree to which nodes in a graph tend to cluster together in graph theory. This was proposed by Watts and Strogatz in 1998[1] for analyzing the social network in real world. There are two versions of this measure: the global and the local. The global version was designed to give an overall indication of the clustering in the network, whereas the local gives an indication of the connectivity of single nodes. In this paper, we only consider local clustering coefficient.

Given an undirected graph $G = (V, E)$, where V is a set of nodes, and E is a set of directed edges between nodes. The $e_{ij} = (v_i, v_j)$ is an edge between node v_i and v_j. The neighborhood $N(v_i)$ for a node v_i is defined as its immediately connected neighbors as follows:

$$N(v_i) = \{v_j \mid e_{ij} = (v_i, v_j) \in E\}$$

Let $d(v_i)$ be the degree of node v, *i.e.* $d(v_i) = |N(v_i)|$. The degree $d(v_i)$ of node v_i is the number of nodes adjacent to v_i. The local clustering coefficient measure for undirected graphs is defined as *the probability that a random pair of its neighbors is connected by an edge*, i.e.:

$$ccf(v_i) = \frac{|\{e_{jk} \mid v_j, v_k \in N(v_i), e_{jk} \in E\}|}{\binom{|N(v_i)|}{2}}$$

A complete subgraph of three nodes of G can be considered as a *triangle*. Let $\lambda(v_i)$ be the number of triangles including node v_i. A *triple* at a node v_i is a path of length two for which v_i is the center node. Let $\tau(v_i)$ be the number of triples on $v_i \in V$. In other words, $\tau(v_i)$ is the number of subgraphs (not necessarily induced) with 2 edges and 3 nodes, one of which is v_i and such that v_i is adjacent to both edges. We can also define the clustering coefficient of node v_i as

$$ccf(v_i) = \frac{\lambda(v_i)}{\tau(v_i)}$$

The value of clustering coefficient is a real number between 0 and 1. The maximal value is 1 when every neighbor connected to v_i is also connected to every other node within the neighborhood, and the minimal ccf is 0 if none of the nodes connected to v_i connects to each other.

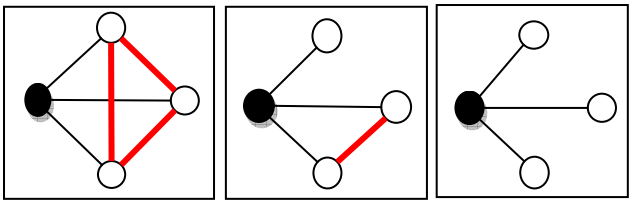

Fig. 1 Examples of clustering coefficient

Fig. 1 gives some examples for how to calculate the clustering coefficient for a single node. The degree of node v_i (dark) is 3, i.e. it has three neighbors (white). The number of edges between the 3 nodes is 3, 2, and 0 from left to tight. So the ccfs are 1, 1/3 and 0 respectively.

3. Single Document Summarization

Now we propose algorithms for single document summarization. Given a document D and a set of stop words. To summarize D, we begin by syntax analysis of D and build a dependency graph G for D. Then compute local clustering coefficient for each node of G. Delete nodes with clustering coefficient less than a threshold and obtain graph G'. Indentify all triangle in G' whose dependency weight below a threshold. Suppose these obtained triangles form a set $T = \{T_1, T_2, T_3, T_4 \cdots\cdots T_i\}$. Indentify sentences in document D where the extracted triangles are anchored.

3.1. Building a Dependency Graph

Fig. 2 describes the algorithm for building a dependency graph for a given document.

Algorithm *buildDepGraph*

Input:

- D= $\{s_1, s_2, s_3, s_4 \ldots\ldots s_m\}$, s_k : (k=1,2,...,m) is a sentence of D;
- $W(s_k)$: words contained in sentence s_k ;
- HighFreq : words of high frequency in document D;
- δ : threshold for high frequent dependency

relations
Procedure:

(1) For each sentence $s_k \in D$ Do

(2) For each $w_i, w_j \in W(s_k)$ set $df(w_i, w_j) = 0$;

(3) do dependency structure analysis and co-occurrence analysis for s_k

(4) For each $w_i, w_j \in W(s_k)$ D o

(5) If w_i is depends on w_j, or w_i co-occurs with w_j Then

(6) $dep(w_i,w_j,s_k)= 1$;

(7) $df(w_i,w_j)= df(w_i,w_j)+1$;

(8) End;

(9) End;

(10) For each (w_i, w_j) Do

(11) If $df(w_i,w_j)> \delta$, Then

(12) $E=E \cup \{(w_i,w_j)\}$

(13) End;

Fig. 2 Building dependency graph

3.2. Extracting Triangles

We now propose the algorithm for key sentence extraction from a given single document. In Fig. 3, we use the following notations.

- $N(v_i)$: neighborhood of node v_i
- $d(v_i)$: degree of node v_i
- $df(w_i, w_j)$: dependency frequency between w_i node w_j
- $tf(w_i)$: word frequency or term frequency of word w_i

Algorithm *TriangleSum*

Input:

- D: documents to be summarized,
- S: a set of stop words.
- δ: threshold for high frequent words
- λ: threshold for high frequent dependency relations
- μ: threshold of clustering coefficient
- t: number of triangles to extract

Output:

- S: a set of key sentences

Procedure:

1) . Process D and construct dependency graph $G=(V, E)$,where
   $V = getHighFreq(D, \delta)$ - S;
   $E = getDependency(D, \lambda)$;
2) . For each $e \in E$, if $\{e\}$ is a cut, E=E-$\{e\}$
3) . For each $v \in V$, if $d(v)=0$, V=V-$\{v\}$
4) . For for $v \in V$ compute $ccf(v)$
5) . If $ccf(v) < \mu$ then V=V-$\{v\}$
6) . Do breadth-first search on each connected partial graphs of G and extract all triangles T
7) . For each triangle $T_i \in T$ and compute $df(T_i)$, dependency weight of T_i
8) . Extract t triangles $\{T_1, T_2, \ldots, T_t\}$ with *largest* dependency weight
9) . Identify sentences that contain at least one triangle

Fig. 3 Algorithm of key sentence identification

With the extracted triangles, summarization of a document can be easily approached in different ways.

(1) *Entrance sentences*. Extract sentences on the entrance of the paragraphs containing more triangles. The rationale behind this approach is that bushy paths (or paths connecting highly connected paragraphs) are more likely to contain information central to the topic of the article.

(2) *Anchored sentences*. Extract sentences anchored with more triangles. In this approach, triangles are used to indicate important sentences.

For example, given the following document, we can construct a TriangleSum as shown inFig. 4. Based on this graph, we can extract two triangles. Two triangle are A_1 = {海軍, 空母, 派遣}, A_2 = {海軍, 軍事演習, 行う}. Based on these triangles, we can extract the first sentence as one summarization.

米**海軍**は昨年１０月にも韓国海軍との合同**軍事演習**を**行う**ため、黄海に**空母**「ジョージ・ワシントン」を**派遣**しているが、偶発的な軍事衝突が起きる危険性の高まる 中での派遣は初めてだ。オバマ大統領は２日、韓国関係の会合に向け、「北朝鮮によるいわれのない攻撃に対し、韓国国民に深い哀悼の意を捧げる」との声明を出した。

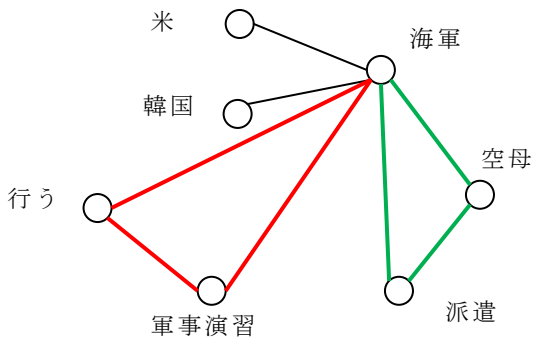

Fig. 4 Example of TriangleSum

To implement the proposed algorithm, we need efficient

computation of local clustering coefficient for each node in V and extraction of all triangles from each connected partial graphs. Both require efficient algorithm for triangle identification. Next we will describe a breadth-first search based triangle identification algorithm.

A triangle is represented as a triple <*u, v, w*> where u, *v, w* ∈ *V* and each node is adjacent to the other two. To identify a triangle, we do a breadth-first traversal for the given graph, from a starting node. Check nodes in the neighborhood to see if any of them are connected to each other.

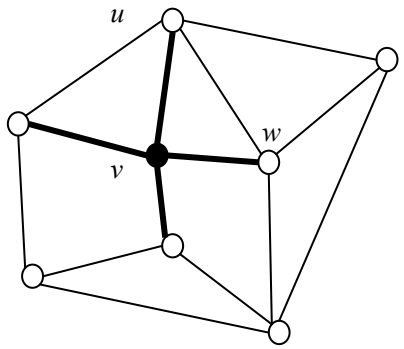

Fig. 5 Example of triangle identification

In Fig. 5, after *v* is visited, the neighborhood *N(v)* of v will enter the queue. At this time, we can check if there are any edges between them. An edge between two neighbors indicates a triangle. In this example, there is an edge between node u and w. Since both of u and w are neighbors of *v*, <*u, v, w*> is identified as a triangle.

Algorithm *TriangleCheck*

Input:

- *G=(V, E)*: an undirected connected simple graph (at most on edge between any pair of nodes) where ,
  V: a set of nodes and,
  E: a set of edges
- N(v): neighborhood of v ∈ V
- v_0 ∈ V: starting point of breadth-first traversal
- q: queue for breadth-first traversal

Output:

- T: a set of triangle in G

Procedure:

```
1) . T=∅;
2) . For each v ∈ V, v.flag = 0;
3) . q.enqueue(v0);
4) . While (not q.empty()) {
5) .    v = q.dequeue();
6) .    v.flag=1;   // mark for visited
7) .    For each u ∈ N(v) {
8) .      If (u.flag==0) {
9) .        q.enqueue(u);
10) .       For each w ∈ N(u) ∩N(v) {
11) .         T=T∪ {<u,v,w>}
12) .  Output T;
```

Fig. 6 Identifying triangles from graph

The algorithm shows that from the starting point, all child nodes obtained by expanding a node are added to a queue which works as first in, first out. It means nodes that have not yet been examined for their neighbors are placed in some container, such as a queue or linked list called "open" and then once examined, the nodes will be placed in the container called "closed" until all the nodes are visited. It's easy to show if there is no such nodes are explored, there are no triangles.

4. Experiment and Evaluation

For experiment implementation, we use a Japanese morphological parser called juman to get the high frequent words in a given document at first.

usage: juman -[b|B|m|p|P] -[f|c|e|E] [-i string] [-r rc_file]

Options: -b : show best path; -B : show best path including homographs (default); -m : show all morphemes; -p : show all paths; -P : show all paths by -B style; -f : show formatted morpheme data; -c : show coded morpheme data; -e : show entire morpheme data; -e2 : -e plus semantics data (default) ;-E : -e plus location and semantics data; -V : not search voiceless morphemes; -R : not recognize adverb automatically; -L : not search normalized lowercase; -i : ignore an input line starting with 'string'; -r : use 'rc_file' as '.jumanrc'; -v : show version.

For example, the command shown below processes a given document which is named ‘sample’ and saved as .txt formation.

「%% juman -b -e2 < sample.txt > sample.jmn」

Both ‘sample.txt’ and ‘sample.jmn’ in the command are plain text files. The ‘sample.jum’ file contains the entire morpheme data. If the content of the file ‘sample.txt’ is a sentence「私は猫です。また名前がないです。」Then, the output result is shown in Fig. 7. In this example, the symbol ‘EOS’ means “end of sentence”.

```
私 わたし 私 名詞 6 普通名詞 1 * 0 * 0 "代表表記:私/わたし
漢字読み:訓 カテゴリ: 人"
は は は 助詞 9 副助詞 2 * 0 * 0 NIL
猫 ねこ 猫 名詞 6 普通名詞 1 * 0 * 0 "代表表記:猫/ねこ 漢字
読み:訓 カテゴリ: 動物"
です です だ 判定詞 4 * 0 判定詞 25 デス列基本形 28 NIL
。 。 。 特殊 1 句点 1 * 0 * 0 NIL
また また また 副詞 8 * 0 * 0 * 0 "代表表記:又/また"
名前 なまえ 名前 名詞 6 普通名詞 1 * 0 * 0 "代表表記:名前/
なまえ カテゴリ:抽象物"
が が が 助詞 9 格助詞 1 * 0 * 0 NIL
ない ない ない 形容詞 3 * 0 イ形容詞アウオ段 18 基本形 2
"代表表記:無い/ない 反義:動詞:有る/ある"
です です です 助動詞 5 * 0 無活用型 26 基本形 2 NIL
。 。 。 特殊 1 句点 1 * 0 * 0 NIL
EOS
```

Fig. 7 Result of morphological analysis

(1) How well:
-bnst: phrasal units;
-dpnd: including dependency structure;
-case: including syntactic information;
(2) Output format:
[-tree|-bnsttree|-sexp|-tab|-bnsttab];
(3) [-normal|detail|debug] [-expand];
(4) [-C host:port] [-S|F] [-N port];
(5) [-timeout second] [-r rcfile].

```
# S-ID:1 KNP:3.01-CF1.0 DATE:2011/02/07 SCORE:-42.15779
  私は一一┐
    猫です。<P>一┐  ……①
  また一一┐      |
名前が一一┤      |  ……②
    ないです。<P>一PARA……③
EOS
```

Fig. 8 Result of dependency analysis

And then is to obtain the words which are frequently occurred in the document. A program called getHighFreq() is used to count frequency of each word in the document. The getHighFreq() program accepts the results of morphological analysis output by Juman, checks and counts the occurrences of each word one line by another. Here, there are two points should be paid attention. First, the meaningless words (stop words) should be filtered out. Second, a threshold δ for high frequency words is given to determine how many words should be selected as members of HighFreq.

The command can be used as follows:

「%% juman –b –e2 < sample.txt | perl getHighFreq.pl –」

In the example given above, the words “は”, “です”, and “が” are the stop words, so they should be filter out from the counting object. The result looks like below:

About the threshold δ, the result of getHighFreq is empty set if the input value of δ>1. The result of getHighFreq is {私,猫,また,名前,ない} if the input value of δ=1 because the occurrence frequency tf(w) of all words are 1.

The second step is to compute the frequency of dependency relationship. A Japanese language dependency relationship parsing tool called knp is used to analyze the dependency relationship among all of the words at this step. The command for getting high frequency dependency relations by knp is shown as follow:

「%%juman –e2 –B<sample.txt | knp」

There are several options for this tool:

We use the same example file which mentioned above to demonstrate the dependency relation analysis among all of the words in the given document. In order to extract dependency structure by program automatically, the output of knp should be easily handled. The parsing tool knp also provide phrasal unit based output.

There are 3 phrasal units in the output result, denoted by ①～③(annotations are added by the author.) Each phrasal unit begins with an asterisked number with a letter ‘D’. The number (double underlined) indicates the phrasal unit it is dependent on. For example, phrasal unit ① ‘私は猫。’(I am a cat.) depends on the phrasal unit ③ ‘ない’(no). It is straightforward to extract the dependency relation as:

dep(‘私は猫’, ‘ない’)=1

The performance of the proposed algorithm is determined by a number of experiment results. Firstly, performance of natural language processing is crucial. Currently precision of syntactic structure analysis for Japanese documents remains around 80%.

Secondly, the proposed algorithm uses a stop word list to eliminate meaningless words. Thus identifying stop words is also important.

Finally, we have employed a set of parameters to control the numbers of nodes and edges for triangle finding.

There are three steps contained in the setup of the experiments. Firstly, the document D is analyzed by juman and knp so that the high frequent words could be extracted

and the dependency relationship among all these words could be analyzed. This process is mentioned above, and a series of experiment results is demonstrated.

Then, we need to find out the number of triangles from the formed graph manually. What we need to pay attention in this step is the set of given parameters, because the number of triangles is determined by the set of parameters.

For example, define the values of δ, λ, and μ are 1, most of the high frequent words can be extracted and connected by edges. These procedures are done by computer automatically. With the dependency analysis result of a given document D, we could find out 62 triangles in the graph. Next, we are going to locate these triangles in each paragraph so that the sentences which represent the main points could be extracted. The result for each paragraph will be summed up in order to demonstrate the average extraction precision.

At last, the precision of experiment results are needed to be evaluated. This process is also done manually. The procedure is to compare the correctly identified key sentences with the real key sentences which are marked by human. This could be denoted as below:

$$precision = \frac{correctly\ extracted\ key\ sentences}{real\ key\ sentences\ by\ human}$$

The data of table 1 shown below demonstrates the precision of experiment results for the given document D.

Tab. 1 Experiment Results

	P.1	P.2	P.3	P.4	P.5	P.6	P.7	P.8
Triangles	5	12	11	13	4	6	6	3
Precision	100 %	50 %	60 %	40 %	80 %	60 %	50 %	90 %

In Tab. 1, the alphabet “P” in the first line stands for paragraph. The values in the second line stand for the number of triangles which contained in one paragraph. In the third line, the percentage shows the extraction precision of key sentences which represent the viewpoint of the document D.

5. Concluding Remarks

In this paper we proposed a novel algorithm called TriangleSum for automatic extraction of key sentences from electronic documents. The proposed algorithm works on single document without training data so that training cost could be removed. TriangleSum is supposed to efficiently extract heavy triangles as anchor points of key sentences from the input document. These triangles can then be used to identify sentences that central to the topic of the document.

We have described an efficient method to extract triangles from connected graph. When running on real world documents, we have to tune the some thresholds λ, δ and μ. Also the proposed algorithm relies on high quality results of morphology parsing and syntactic parsing.

References

[1] D. J. Watts and Steven Strogatz. Collective dynamics of 'small-world' networks. Nature 393(1998): pp440–442.

[2] Y. Ohsawa, N. E. Benson and M. Yachida. KeyGraph: Automatic indexing by co-occurrence graph based on building construction metaphor, IEEE ADL'98, pp.12-18. 1998

[3] G. Salton, J. Allan, C. Buckley, and A. Singhal. Automatic analysis, theme generation, and summarization of machine readable texts. Science 264(5164):1421-6, June 1994.

[4] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval* (SIGIR '98). pp. 335-336. 1998

[5] C.-Y. Lin (1999). Training a selection function for extraction. In Proceedings of CIKM ’99, pages 55–62, New York, NY, USA

[6] J. M. Conroy and D. P. O’Leary. Text summarization via hidden markov models. In Proceedings of SIGIR ’01, pages 406–407, New York, 2001