Semantic Suffix Net Clustering for Search Results

Jongkol Janruang
Computer Science & Information Management Program, Asian Institute of Technology
P.O.BOX 4, Klong Luang, Pathumthani 12120
Thailand

Sumanta Guha
Computer Science & Information Management Program, Asian Institute of Technology
P.O.BOX 4, Klong Luang, Pathumthani 12120
Thailand

ABSTRACT

Suffix Tree Clustering (STC) uses the suffix tree structure to find a set of snippets that share a common phrase and uses this information to propose clusters. As a result, STC is a fast incremental algorithm for automatic clustering and labeling but it cannot cluster semantically similar snippets. However, the meaning of the words is indeed an important property that relates them to other words, although there may not be a match of text strings *per se*. In this paper, we propose a new semantic search results clustering algorithm, called semantic suffix net clustering (SSNC). It is based on semantic suffix net structure (SSN). The proposed algorithm uses the net pruning technique to merge the related suffixes through their suffix links for finding base clusters. This logic causes both string matching and meaning of the words to be used as conditions for the purpose of clustering. Experimental results show that the proposed algorithm has time complexity lower than CFWMS, SSTC and STC+GSSN which are current semantic search results clustering methods. Moreover, the F-measure of the proposed algorithm is similar to that of the original STC, CFWMS, STC+GSSN, and higher than that of MSRC and SSTC.

General Terms

Algorithms, Data Mining, Text Mining, Search Results Clustering.

Keywords

search results clustering, semantic suffix net, net pruning techniques, semantic suffix net clustering, semantic clustering.

1. INTRODUCTION

Recently, information retrieval and text mining have generated a great deal of interest in the fields of business intelligence, knowledge management, and various search applications. This is because of the rapid growth of databases, belonging to government, businesses and other organizations, which contain documents in digital form [1]. Paradoxically, the volume of information available often makes finding what they need actually harder for users. Development of applications that help users retrieve the answer to their query from massive digital archives is a challenging task and one essential, therefore, to making these growing archives actually useful and not just voracious hogs of disk space.

The World Wide Web is an example of an enormous distributed database of electronic documents, or web pages as they are called [2]. Powerful and sophisticated search engines such as Google and Bing exist to help users navigate the web. However, typically, search engines return a long list of search results. The user then has to sift through these to find relevant ones, which is a time-consuming task. Moreover, if the relevant results do not occur in the first part of the returned list, then the user may even fail to find them. A possible solution to this problem is the use of the search result clustering techniques which group results in easily understood clusters, facilitating the user's search – instead of scanning linearly through a long list of items, she scans linearly through a much shorter list of groups of items, and then, once an appropriate group has been identified, through a correspondingly shorter list of items in that group. Typically, clustering is performed based on snippets - short passages of text summarizing the content of search results [3, 4, 5, 6, 7].

Search results clustering algorithm are a core component of web clustering engines [7]. Most search results clustering algorithms are a combination between search engines and text clustering, the latter grouping snippets returned from search engines (e.g., STC [3], SHOC [4], LINGO [5], SNAKET [6], CREDO [8], and NSTC [9]). Typically, they generate cluster labels as navigators which facilitate user access to relevant search results. However, the quality of search results clustering is crucial. One way to enhance the quality is to use the semantic similarity of words; in particular, one tries to relate words by finding commonality in meaning, although there may not be a match of text strings *per se.*

Semantic text clustering has drawn considerable interest in recent years. For example, the CSUGAR approach proposes biomedical textual information clustering using ontology to create semantic clusters [10]. TRSSC performs search results clustering by creating the semantic upper approximation space based on the tolerance rough set method and then organizing results into groups by relating meaning [11]. These projects confirm the importance of semantic clustering in practical situations.

In this paper, a new semantic search results clustering algorithm is proposed, called semantic suffix net clustering (SSNC). This approach is derived by combining the advantage of both semantic suffix tree clustering (SSTC) [12] and its generalization STC+GSSN [13]. The SSNC is proposed by using a generalized semantic suffix net (GSSN) as a structure to represent snippets instead of the semantic suffix tree. Subsequently, a net pruning technique is used to replace tree pruning in SSTC since net pruning is a modification of the tree pruning technique in SSTC. According to our evaluations, semantic suffix net can be used instead of the semantic suffix tree in SSTC and the time execution of SSNC is faster than that of SSTC. Additionally, the precision of SSNC results is higher than STC+GSSN**.**

This paper is organized as follows. Related works are described in Section 2. The semantic suffix net clustering is proposed in Section 3. In Section 4 experiments are presented. Finally, we conclude in Section 5.

2. RELATED WORKS

STC is a fast incremental algorithm for search results clustering but it suffers from the typically large size of the suffix tree used as a structure to represent snippets. Therefore, the MSRC algorithm [14] was introduced to manage the huge tree returned by STC but the difficulty with this approach is that the *n*-gram technique generates interrupted cluster labels if the common phrase size is longer than the defined *n*-gram size. Based on the problem of MSRC, the STC with a partial phrase join operator [15] was presented to solve the problem of the interrupted cluster labels that are generated when using the STC with *n*-gram techniques. In another approach, the STC with *x*-gram [16] was created based on the on-line construction of suffix trees to decrease the memory space used by the original suffix trees. However, these various approaches do not allow one to modify the original STC to perform semantic search results clustering. Even though, the STC algorithm gives significant search performance it does not consider semantically related words to construct suffix trees and, therefore, eventually generates a huge tree with a complexity which is hard to manage.

In addition, the original STC cannot group semantically similar snippets into the same clusters. For example, the two snippets: S_1 = *Doctor likes children* and S_2 = *Physician cares for child* have similar meanings but STC cannot group them into the same cluster. Moreover, the two snippets: S_1 = *John wrote the paper in the morning* and S_2 = *the paper was written by John in the morning* have similar meanings but STC returns three base clusters: *john*={1,2}, *paper*={1,2}, and *morn*={1,2}. That means STC cannot generate phrase cluster labels when the two snippets in the form of the passive voice and the active voice are similar in meaning. To deal with these problems of STC, SSTC [12] is created using a semantic suffix tree as a structure to represent snippets and then tree pruning to combine related suffixes. However, SSTC lacks the logic to explain the disappearance of semantically similar nodes. Thus, STC+GSSN [13] are introduced in order to address the problem of a lack of logic to explain the disappearance of semantically similar nodes of SSTC.

The significant contribution of this paper is a new approach to semantic search results clustering by using the semantic suffix net as a structure to represent snippets. The semantic suffix net is a new semantic search structure for search results. It can be used to alleviate the problem of the size of the suffix tree and to improve semantic clustering when STC ignores meaning of the words. The logic is that search results clustering which works on textual information should use meanings of words for the purpose of clustering. In fact, humans use the concept or meaning of the words directly to group them. For example, in the case of biomedical textual information clustering, the CSUGAR approach [10] is proposed based on ontology techniques. In the field of document and search results clustering, CFWMS [2] are created synsets and hypernyms from the WordNet database to generate the meaning of the union *MU* and then the apiori concept is applied to find the frequent meaning union *FMU* and this information is used to identify final clusters. These approaches confirm the importance of semantic clustering in practical situations.

3. SEMANTIC SUFFIX NET CLUSTERING

Semantic search structure is an important structure to use for grouping the semantically similar snippets since words have meaning as an important property that relates them to other words, although there may not be a match of text strings *per se*. Therefore, Semantic Suffix Net Clustering (SSNC) is proposed to present a new semantic search structure for search results clustering called Semantic Suffix Net (SSN). Also, net pruning technique is proposed to exploit the usefulness of suffix links in SSN structure. The architecture of SSNC is shown in Fig.1. The details of four processes are described as follows.

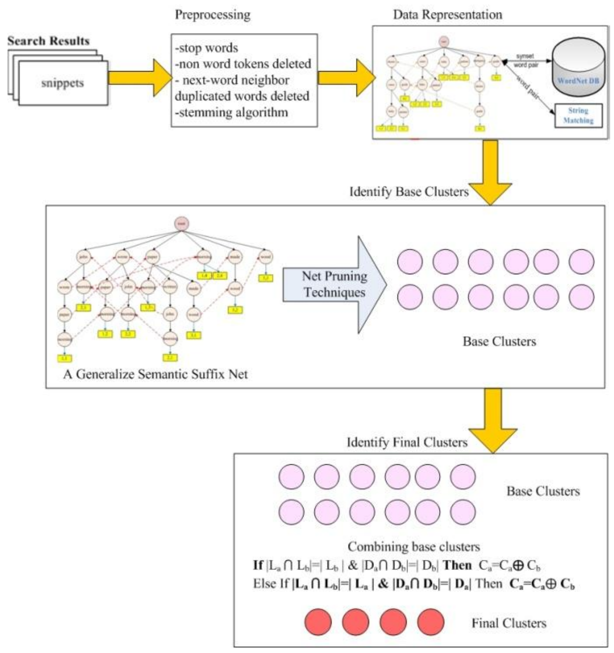

Fig 1: The architecture of SSNC which modifies search results clustering can be used for semantic search results clustering by using SSN structure and net pruning technique.

3.1 Step 1- Preprocessing

In this step, the two common text pre-processing methods (e.g. ignoring stop words and non-words, stemming algorithms) are used as tools to clean snippets. In addition, the next-word is deleted, if it is duplicated to the previous word. For example, there is a string: *He is a taxi driver and he drives a taxi in the city*, after the two common text pre-processing methods are completed, *taxi drive drive taxi city* is returned. Then, the *drive* word in the third position is deleted to reduce the space complexity and time execution of the next processes.
As example in this paper, there are three snippets: s_1 =*john wrote the paper in the morning*, s_2 =*the paper was written by John in the morning*, and s_3 =*the paper is made from wood.* The results of this process are s_1 =*john wrote paper morn*, s_2 =*paper written John morn*, and s_3 =*paper made wood*.

3.2 Step 2- Data Representation

In this step, the SSN of snippet pairs are combined to GSSN for representing a set of snippets. The detail of GSSN construction is explained in the STC+GSSN algorithm [13].

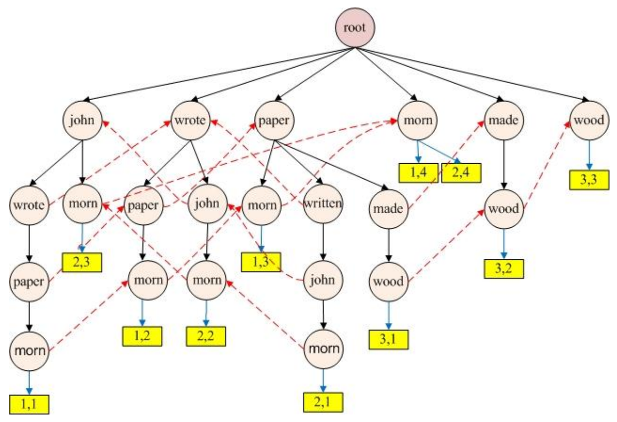

Fig 2: This is a GSSN which is created depending on the three strings: s_1 =*john wrote paper morn*, s_2 =*paper written John morn*, and s_3 =*paper made wood*.

In Fig. 2, each node is drawn in a circle, and is labeled by inputting a string word, which is connected to a suffix node with a direct link which is represented as a black line. Each branch of the root is defined as a suffix, which contains one or more sub-suffixes (e.g., the first branch of a root is a suffix which has *john* as a heading node and this suffix contains two sub-suffixes which are *wrote→paper→morn* {1,1} and *norm* {2,3}). Each suffix is connected to the next suffix with suffix links which are represented as dotted red lines. For each of the boxes at the leaf node, the first number designates the string of origin (our example is starting to 1-3) and the second number is a starting position of the suffix in that string.

3.3 Step 3 – Identifying Base clusters

The identification of base clusters can be done efficiently using suffix links and direct links since suffixes are related through their suffix links and direct links. The details of this step include a move to update or to delete branches and nodes in order to find base clusters. This step is called the net pruning technique.

The process of the net pruning technique is divided into two steps. The first step is called merging suffixes, which is shown on line numbers 4-22 in Algorithm 1 below. Then the second step is shown on line numbers 23-24 which is called compact nodes.

The process of merging suffixes uses suffix links as a path to travel in order to join the two related suffixes. Based on Algorithm 1, β_p is defined as a suffix which is a prefix of β_s whereby β_p and β_s are connected through suffix links. The details of merge suffixes in the three cases are explained as follows:

Case 1: $p = s$ when p is the ordinal number of β_p and s is the ordinal number of β_s. This case means that β_p and β_s are the same suffix. This shows that duplicated words appear on the same suffix and results in erroneous merging of related suffixes. To reduce the occurrence of duplicated words, the current node *N[k]* is merged to update at its parent node which is called *p[N[k]]*. An example of this case is shown in Fig. 5 and the equation is shown in line number 16.

Case 2: $|\beta_p \cap \beta_s| = |\beta_s|$ means β_s is a subset of β_p. To delete the duplicate suffix, all nodes of β_s are moved to update at the related nodes on β_p through their suffix links. The illustration of this case is shown in Fig. 3 and the equation is shown in line number 18.

Case 3: $|\beta_p \cap \beta_s| < |\beta_s|$ means β_s is not a subset of β_p. Only related sub-suffixes of β_p are moved from β_s to update at the related sub-suffixes on β_p through their suffix links. In contrast, sub-suffixes which are not related to β_p will remain as a suffix on the root node. The illustration of this case is shown in Fig. 4 and the equation is shown in line number 19.

```
1 Input: GSSN structure
2 Output: a tree structure
3 Initialization:
4 j←0
5 n ← the number of child's root node
6 While p ≤ n Do
7  $\beta_p$ ← j^th suffix path
8  $\beta_s$ ← suffix path of $\beta_p$
9  For all nodes on $\beta_p$ {
10    N(k) ← a child of $\beta_p$
11    s[N(k)] ← a suffix node of N(k)
12     p ← the ordinal number of $\beta_p$
13     s ← the ordinal number of $\beta_s$
14     If (p>s)  Then  root ← root - $\beta_p$
15     Else If (p = s) Then
16           d[p[N(k)]] ← d[p[N(k)]] ⊕ d[N(k)] ►case 1
17     Else{
18       If  ($|\beta_p \cap \beta_s| = |\beta_s|$) Then $\beta_p$ ← $\beta_p \oplus \beta_s$  ►case 2
19       Else  $\beta_p$ ← $\beta_p \oplus (\beta_p \cap \beta_s)$  ►case 3
20     }
21   }//end of for statement
22 End // end of while statement
23 For i ← 1 To n do // the compact nodes process
24  If d[$N_i$] = Ø and |c[$N_i$]| =1 Then $N_i$ ← $N_i$ + $N_{i+1}$
```

Algorithm 1: The process of the net pruning technique

To understand the process of merging suffixes, the result of data representation in Fig. 2 is used as an example to explain the details of this process.

The initial step to merge suffixes is defining the path of suffix β_p and β_s. According to Fig. 3-A, the root contains six suffixes which are related through their suffix links. The first suffix of the root is defined as β_p which has *john* as a label of its node and *p* is equal to 1. A node *john* is a heading node of this suffix. According to *john* node, it contains two sub-suffixes: 1) *wrote→paper→morn* {1,1} and 2) *morn* {2,3}.

After β_p is defined, β_s will be identified by a suffix link of the first child node of β_p which is called the current node *N*(*k*). Herein, *N*(*k*) has *john* node as a parent node *p*[*N*(*k*)]. In this example, the suffix link of *N*(*k*) is used to identify β_s. As a result, the second suffix of the root: *wrote* node is defined as β_s which is called *s*[*N*(*k*)] and *s* is equal to 2.

β_s has *wrote* as a heading node and it contains two sub-suffixes: 1) *paper→morn* {1,2} and 2) *john→morn* {2,2}. In this case, β_s is a subset of β_p when all nodes of β_s are related to β_p through their suffix links. Therefore, case 2 can be applied to merge related suffixes. An example of this case is shown in Fig. 3-A and its result is shown in Fig. 3-B.

After β_p and β_s are merged based on case 2, its result is used to process the next step, which can be shown in Fig. 4. In this paper, the preorder traversal based on depth-first traversal method is applied to travel on semantic suffix net structure. Thus, the first child node of *wrote* node on β_p called *paper* node is assigned as the current node *N*(*k*) and it has *wrote* node as parent node *p*[*N*(*k*)] automatically. As a result, the second suffix of the root: *paper* node is defined as β_s and *s* is equal to 2.

In this case, case 3 can be applied since β_s is not a subset of β_p. This is because only the first sub-suffix: *norm* {1,3} and second sub-suffix: *written→john→morn* {2,1} of β_s are related to β_p through suffix links. Also, the third sub-suffix: *made→wood* {3,1} of β_s is not related to β_p through suffix links. The process of this example is shown in Fig. 4-A and then its result is shown in Fig. 4-B.

In Fig. 4-B, the result of case 3 is used to process the next step of preorder traversal. The child node of *paper* node on β_p: *morn* node is assigned as *N*(*k*). After that, the third suffix of the root is assigned as β_s thus the *morn* node on β_s is assigned as s[*N*(*k*)]. In this case, case 2 can be applied. The results of this case are shown in Fig. 5-A.

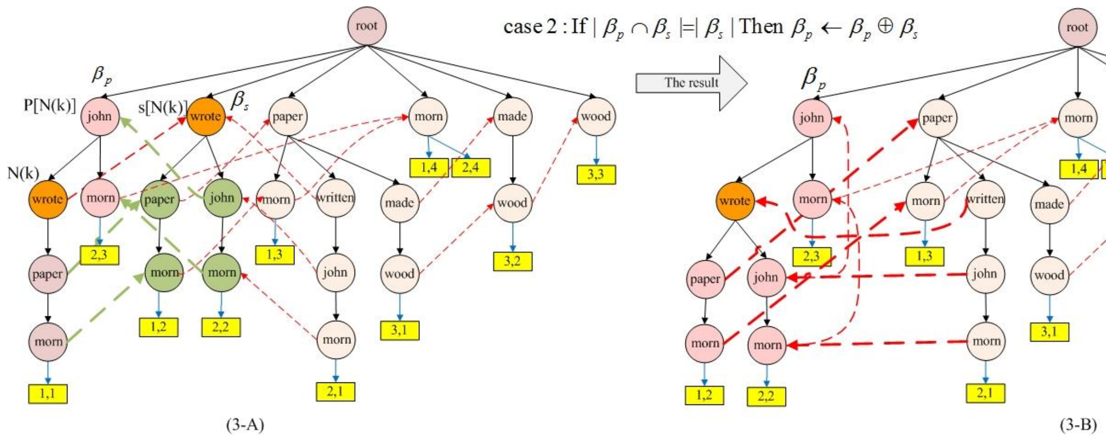

Fig 3: The initial step of net pruning is defining β_p and β_s which are related suffixes through suffix links. Fig. 3-A shows both β_p and β_s are connected through their suffix links. This figure shows an example for case 2 of net pruning and its result is shown in Fig. 3-B. Since β_s is a subset of β_p, case 2 can be applied to prune a net form through suffix links. As result, the number of suffix paths on a net form is decreased because β_s is moved to update at β_p. The equation of this case is $\beta_p = \beta_p \oplus \beta_s$.

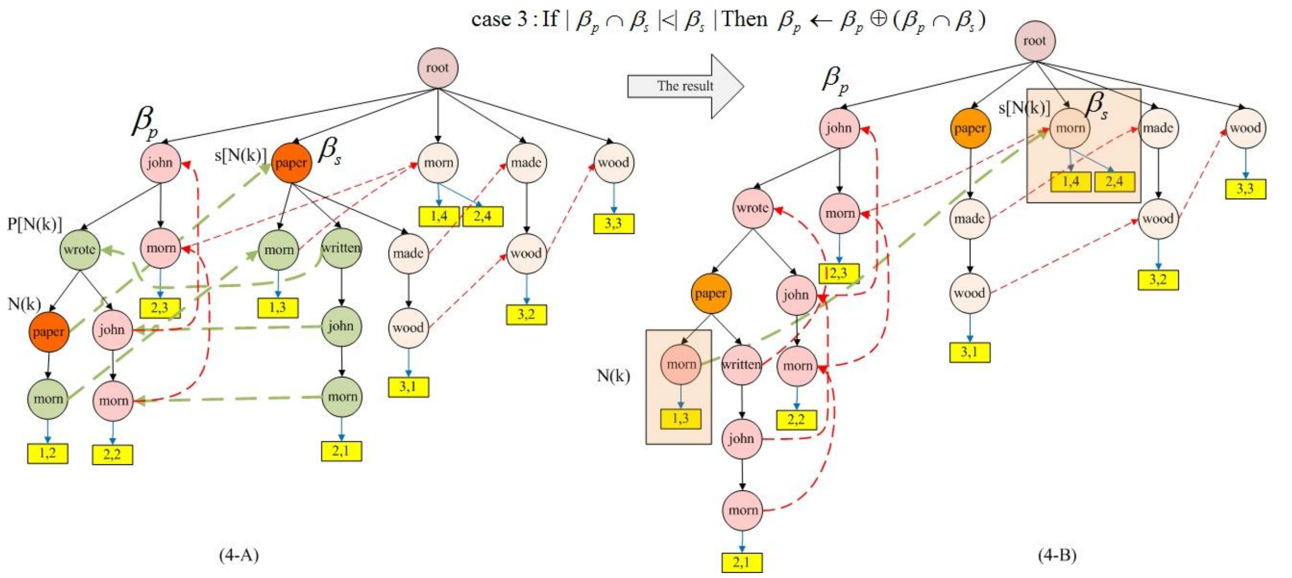

Fig 4: This figure is an example of case 3 since some sub-suffixes of β_s are not related to β_p. Therefore, only related sub-suffixes between β_s and β_p are transferred to update on β_p. Fig. 4-B shows the result of case 3 since the related sub-suffixes between β_s and β_p are moved from β_s to β_p for updating on β_p. In contrast, different sub-suffixes between β_s and β_p are not transferred to update on β_p but they will still exist on β_s (*paper→mad→wood*). The equation is $\beta_p = \beta_p \oplus (\beta_p \cap \beta_s)$.

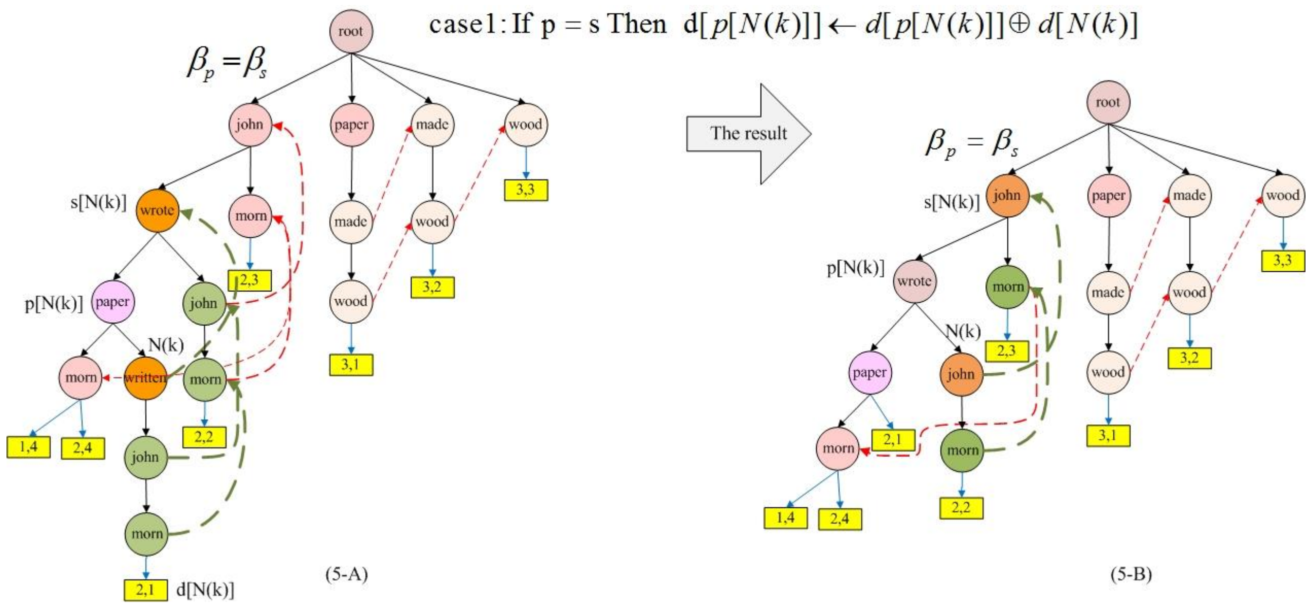

Fig 5: This figure is an example of case 1 when the same meaning is shared by sentences of both passive and active voice. It causes the loop of string on a net form to appear. To reduce the loop of string, case 1 can be applied. The equation is d[p[N(k)]] = d[p[N(k)]] ⊕ d[N(k)]. Fig. 5-B shows the results of case 1 which cause the number of the node and the height of the net to be reduced.

In Fig. 5-A, the second child node of *paper* node: *written* node is assigned as $N(k)$, that means *paper* node is $p[N(k)]$ and *wrote* node is $s[N(k)]$ automatically. In this case, case 1 is applied to avoid the mistakes of merging the related suffixes in the case 2 and 3 since β_s and β_p are the same suffix.

In case 1, $d[p[N[k]]]$ is represented to document members of node $p[N[k]]$ and $d[N[k]]$ is represented to document members of node $N[k]$ and its child nodes. Thus, $d[p[N[k]]]$ of this example is null when a node $p[N[k]]$ does not have a document member. In contrast, $d[N[k]]$ contains one document member which is document number 2. For that reason, a node *morn* {2,1} which is a child nod of $N[k]$ has a document number 2 as a document member. The illustration of this process is shown in Fig. 5-A and its results are shown in Fig. 5-B.

The process of merging suffixes is applied and will be finished since all related suffixes are merged through their suffix links based on case 1-3. The results of the merging of suffixes step are shown in Fig. 6-A.

In Fig. 6-A, the result of merging suffixes is represented in a tree form. Each node may include both label and document members. For example, a node of a root is divided into two suffixes: 1) *john→wrote→paper→morn* and 2) *paper→made→wood*. According to *john*, it has *john* as a label and number 2 is a document member. In the other case, a node of *paper* in the second suffix of the root contains only a label but the document member is empty. Thus, this node is an empty node. An empty node is a node that has a child node but does not have a document member. So, it is unable to be a cluster. It must be combined with its child node in order to reduce the number of nodes and to generate a phrase label. The definition of a compact node is defined as definition 1 and the process of this step is shown in Fig. 6-A. Its result is shown in Fig. 6-B.

Definition 1: Let N_i be an empty node and N_{i+1} be a child node of N_i. Each empty node is compacted with its child node by deleting the direct link that directs from that empty node N_i to its child node N_{i+1} in that suffix. Then, the label of N_i is concatenated to N_{i+1} as in $N_i = N_i + N_{i+1}$.

After the compact nodes step is completed, all nodes in each suffix are collected to form base clusters which label and document members. In particular, a label of a base cluster is generated by concatenating the labels of nodes from the root to that node. Similarly, a document member is generated in the same way. For example, the first suffix of the root includes four base clusters which are *john* ={2}, *john wrote* = {2}, *john wrote paper* = {2}, and *john wrote paper morn* = {1,2}. Accordingly, there is a base cluster *john wrote paper morn* = {1,2}, which has a phrase *john wrote paper morn* as a label and document number 1 and 2 as document members.

Each base cluster is assigned a score that is a function of the number of documents it contains, and the words that make up its phrase. The score $S(c)$ of base cluster B with phrase P is given by:

$$s(c) = |D| * f(|P|)$$

Let $|D|$ be the number of document members in base cluster B, and $|P|$ be the number of words in phrase label P that have a non-zero score. In the other word, $|P|$ is the length of the label. The function $f(|P|)$ means that the score of P is equal to 0 if the length of the label is less than 2 or more than 6. This is because the function f penalizes single word phrase label and longest phrase labels. Thus, the function $f(|P|)$ is equal to the length of phrase label in the range of 2-5 words which is shown in the following and the score of each base cluster is shown in Table 1.

$$f(|P|) = \begin{cases} |P| & \textit{if } 1<P<6 \\ 0 & \text{otherwise} \end{cases}$$

Table 1. This table contains the five base clusters and their scores: s(*c*)

Node	Phrase Label	document member	Score Cluster s(*c*)
john	john	2	1*0=0
wrote	john wrote	2	1*2=2
paper	john wrote paper	2	1*3=3
morn	john wrote paper morn	1,2	2*4=8
paper made wood	paper made wood	3	1*3=3

3.4 Step 4 – Identifying Final Clusters

In this paper, a string may share one or more concept clusters since the meaning of the words and matching string are used as conditions of clustering. According to Fig. 7-A, both label and document members of base clusters may overlap or may even be identical. For this reason, both label and document members should be used as conditions to combine base clusters, which are duplicated.

The logic is that the combination of base clusters should use both label and document members to combine base clusters for identifying the final clusters. This avoids duplicate clusters. The overlap criterion of label and documents is achieved by calculating the overlap ratio of the pair clusters based on the following:

```
If $|L_a \cap L_b|=|L_b|$ & $|D_a \cap D_b|=|D_b|$ Then $C_a=C_a \oplus C_b$
Else If $|L_a \cap L_b|=|L_a|$ & $|D_a \cap D_b|=|D_a|$ Then $C_a=C_a \oplus C_b$
```

There are two base clusters: C_a and C_b with size $|D_a|$ and $|D_b|$, respectively. Let $|D_a \cap D_b|$ represents the number of documents common to both C_a and C_b. Similarly, $|L_a \cap L_b|$ represents the number of words common in label to both C_a and C_b. In addition, S(c) is used to rank the final clusters.

In Fig. 7-A, there are five base clusters: *john*={2}, *john wrote*={2}, *john wrote paper*={2}, *john wrote paper morn*={1,2}, and *paper made wood*={3}. The first two base clusters are used to check overlap criterion which is shown in Fig. 7-B. Thus, *john*={2} is assigned to C_a and *john wrote*={2} is assigned to C_b. As a result, C_a is combined into C_b because they are duplicated. Moreover, the S(c) of C_b is higher than that of C_a. Therefore, C_b is selected and C_a is deleted. The results of this example are shown in Fig. 7-C.

In the other case, C_a and C_b are not duplicated. They are separated into two final clusters. For example, *john wrote paper morn*={1,2} is assigned to C_a and *paper made wood*={3} is assigned to C_b. As a result, they are separated into two final clusters which are *john wrote paper morn*={1,2} and *paper made wood*={3}. This result is shown in Fig. 7-D.

After the identifying of final clusters process is completed, the resulting final cluster is *john wrote paper morn*={1,2} and *paper made wood*={3}. Consequently, this process can reduce the number of duplicate identical clusters and return specific readable clusters.

Finally, the set of final clusters are ranked based on *S*(*c*). Therefore, *john wrote paper morn* = {1,2} is returned on the top rank of final clusters because its s(c) is higher than that of *paper made wood* ={3}. From this result, the SSNC algorithm can cluster snippets that have a semantic similarity because it can cluster s_1: *john wrote the paper in the morning* and s_2: *the paper was written by John in the morning* to a cluster and then separate s_3: *the paper is made from wood* to another cluster.

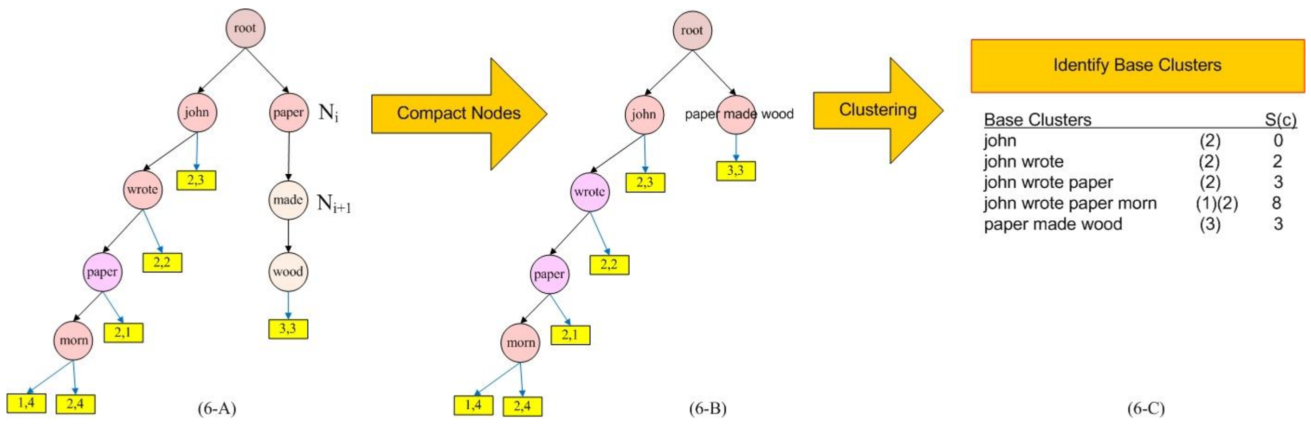

Fig 6: This illustration shows the compact nodes and results of net pruning techniques. In Fig. 6-A, the second suffix has a *paper* node as a heading node and it is an empty node. Thus, it must be compacted to its child node. As a result, there are three nodes in the second suffix, which are *paper*, *made* and *wood*. They are compressed to a node called *paper made wood* node as shown in Fig. 6-B. Finally, all nodes are constructed as base clusters as shown in Fig. 6-C. For example, a *john* {2} group has "*john*" as a cluster label and 2 as a document member.

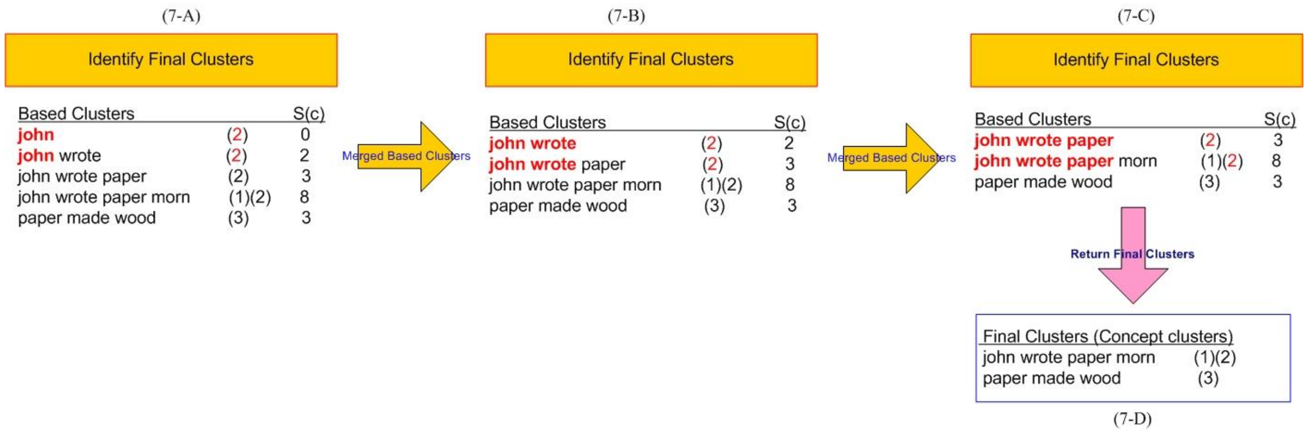

Fig 7: This diagram shows the process of identifying final clusters by using a merged base clusters equation. Fig 7-A depicts the initial step when the base clusters are the results of the examples given in Fig. 2 – 6. Fig. 7-B and 7-C are the next step after the first base clusters couple is merged into a bigger cluster. Finally, Fig. 7-D shows the final clusters, which are returned after all base clusters are merged.

4. EXPERIMENT

The results of SSNC are compared with both the original search results clustering (e.g. STC and MSRA) and semantic search results clustering algorithms (e.g. CFWMS, SSTC, and STC+GSSN).

4.1 Data Collection

The dataset is created from a testing dataset using 31,760 search results from 30 query words on Dmoz.com [17]; Dmoz.com is a human-collected directory of web pages. This utility is used in order to sample categories, and then SSNC is executed on the pre-categorized samples. Thus, the clusters produced by SSNC are compared with the Dmoz categories. In Table 2, the dataset specially selects three types of query, which are ambiguous queries, entity names, and general term queries.

Table 2. The thirty queries in three types selected from DMOZ.com

Type	Queries
Ambiguous queries	jaguar, apple, saturn, jobs, jordan, tiger, trec, ups,quotes, matrix
Entity names	susan dumais, clinton, iraq, dell, disney, world war 2, ford
General terms	health, yellow pages, maps, flower, music, chat, games, radio, jokes, graphic design, resume, timezones, travel

4.2 Experimental Results

4.2.1 Execution Time and Space Complexity in the part of data representation

To compare execution time and space complexity of three data structures (e.g. suffix tree, semantic suffix tree, and semantic suffix net), 967 search results of the “Apple” query word are selected as example of snippets for inputting to six algorithms (e.g. STC, MSRA, CFWMS, SSTC, STC+GSSN, and SSNC). After the data representation process of the six algorithms is completed, their results are used to analyze the execution time and space complexity.

The number of nodes and branches are used as a condition to compare and analyze the space complexity since they represent the space used. If the size of the data structure can be reduced, then execution time will be reduced as well.

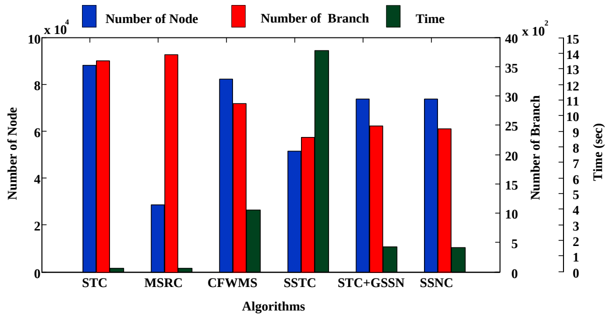

Fig 8: This figure shows the execution time and space complexity (number of nodes and branches) of the different structures in different clustering algorithms.

According to Fig. 8, both SSNC and STC+GSSN algorithms use a semantic suffix net as a structure to represent data. It causes the number of nodes and branches to decrease when they are compared with STC, MSRC, CFWMS algorithms. All three of STC, MSRC, and CFWMS algorithms use a suffix tree. Moreover, the number of nodes and the execution time of SSNC are lower than that of STC+GSSN because SSNC has a process to delete duplicated words.

The SSNC has an execution time which is lower than the original algorithms semantic search results clustering: CFWMS, SSTC and STC+GSSN, respectively. The reductions in the execution time are 60.61%, 89.00% and 1.27% compared to the CFWMS, SSTC, STC+GSSN, respectively.

Interestingly, the semantic suffix net can be used to reduce the space complexity and execution time for representing semantically similar snippets. Although SSNC spent more time than STC and MSRC to 87.18%, it could not solve the problem of STC and MSRC since they cannot be used to represent the semantically similar snippets.

4.2.2 Precision, Recall, F-Measure, and Time Execution in the part of Clustering Algorithm

To evaluate the clustering algorithms, the balanced F_1 measure (F-measure) is used as a weighted harmonic mean between precision and recall. In addition, execution time is used to compare the efficiency of clustering algorithms. In average of 30 queries, the performance of each algorithm is shown in Fig. 9.

According to Fig. 9, the effectiveness of SSNC is similar to STC, CFWMS, STC+GSSN. This is because the F-measure of SSNC is 0.48 and the F-measure of STC, CFWMS, STC+GSSN algorithms are 0.52, 0.50, and 0.52, respectively. Approximately, the F-measure of STC, CFWMS, and STC+GSSN is higher than that of SSNC by 6.46%. However, the F-measure of SSNC is higher than that of MSRC and SSTC, which can be shown as a percentage, 14.58% and 37.50%, respectively.

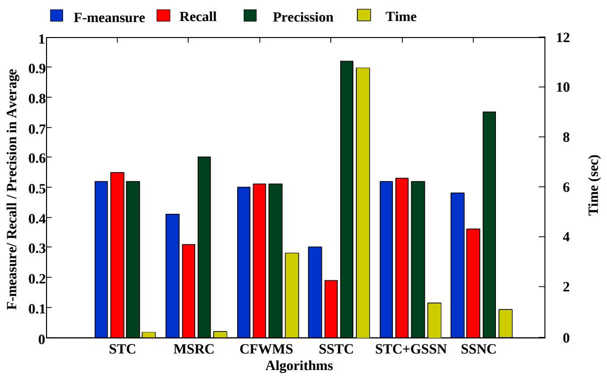

Fig 9: This illustration shows the comparison of F-measure, recall and precision in average of 30 queries for the STC, MSRC, CFWMS, SSTC, STC+GSSN and SSNC algorithms.

It can be seen that, the proposed algorithm is more effective and efficient than other algorithms. The reason for this is that SSNC can cluster the semantically similar snippets although it required a greater time execution than the STC and MSRC algorithms. In addition, the time execution of SSNC is lower than that of CFWMS, SSTC, and STC+GSSN which are current semantic clustering algorithms. Also, the F-measure of SSNC, CFWMS, STC+GSSN algorithms are similar but higher than that of MSRC and SSTC.

5. CONCLUSION

In this paper, semantic suffix net clustering (SSNC) is proposed using a semantic suffix net as a structure to represent semantically similar snippets. Additionally, a net pruning technique is used as logic to combine related suffixes. As a result, SSNC provides a better performance for clustering algorithms compared with STC, MSRC, CFWMS, SSTC, and STC+GSSN. This is because SSNC can group the snippets by using less time in execution than CFWMS, SSTC, and STC+GSSN. Moreover, SSNC can solve the problems of STC and MSRC which cannot group semantically similar snippets. Additionally, semantic suffix net (SSN) can be used as a structure to represent semantically similar snippets to instead suffix tree and semantic suffix tree. This is because the F-measure of SSNC is similar to STC and STC+GSSN and higher than that of MSRC and SSTC.

6. REFERENCES

[1] Jiawei, H. and Micheline, K. “Data Mining: Concepts and Techniques”. Morgan Kaufmann, 2006, in press.

[2] Yanjun, L., Soon, M. C., and John, D. H. 2008. Text document clustering based on frequent word meaning sequences. Journal Data & Knowledge Engineering. 64, 381-404.

[3] Oren, Z. and Oren, E. 1998. Web Document Clustering: Feasibility Demonstration. In Proceeding of SIGIR’98.

[4] Dell, Z. and Yisheng, D. 2004. Semantic, Hierarchical, Online Clustering of Web Search Results. In Proceeding of APWeb.

[5] Stanislaw, O. and Dawid, W. 2005. A Concept-Driven Algorithm for Clustering Search Results. IEEE Intelligent Systems, 20(3), 48-54.

[6] Paolo, F. and Antonio, G. 2005. A Personalized Search Engine based on Web-Snippet Hierarchical Clustering. In proceeding of WWW.

[7] Claudio, C., Stanislaw, O. and Dawid, W. 2009. A Survey of Web Clustering Engines. ACM Computing Surveys (CSUR), 41(3), 1-38.

[8] Claudio, C. and Giovanni, R. 2004. Exploiting the Potential of Concept Lattices for Information Retrieval with CREDO. Journal of Universal Computer Science, 10(8), 985-1013.

[9] Hung, C. and Xiaotie, D. 2007. A New Suffix Tree Similarity Measure for Document Clustering. In Proceeding of WWW.

[10] Illhoi, Y., Xiaohua, H. and Il-Yeol, S. 2007. A Coherent Graph-Based Semantic Clustering and Summarization Approach For Biomedical Literature and A New Summarization Evaluation Method. BMC bioinformatics.

[11] Xian-Jun, M., Qing-Cai, C. and Xiao-Long W. 2009. A Tolerance Rough Set Based Semantic Clustering Method for Web Search Results. Information Technology Journal, 8(4), 453-464.

[12] Janruang, J. and Guha, S. 2011. Semantic Suffix Tree Clustering. In Proceedings of DEIT.

[13] Janruang, J. and Guha, S. 2011. Applying Semantic Suffix Net to Suffix Tree Clustering. In Proceeding of DMO.

[14] Zeng, H., He, Q., Chen, Z., Ma, W. and Ma, J. 2004. Learning to cluster web search results. In Proceeding of SIGIR’04.

[15] Janruang, J. and Kreesuradej, W. 2006. A New Web Search Result Clustering based on True Common Phrase Label Discovery. In Proceeding of CIMCA.

[16] Wang, J., Mo, Y., Huang, B., Wen, J. and He, L. 2008. Web Search Results Clustering Based on a Novel Suffix Tree Structure. In Proceeding of ATC’08.

[17] Open Directory Project. 2012. http://www.dmoz.com