Semantic Suffix Tree Clustering

Jongkol Janruang, Sumanta Guha
Computer Science & Information Management Program
Asian Institute of Technology
P.O. Box 4, Klong Luang
Pathumthani 12120
Thailand
Email: [jongkol.janruang, guha]@ait.asia

***Abstract*—This paper proposes a new algorithm, called Semantic Suffix Tree Clustering (SSTC), to cluster web search results containing semantic similarities. The distinctive methodology of the SSTC algorithm is that it simultaneously constructs the semantic suffix tree through an on-depth and on-breadth pass by using semantic similarity and string matching. The semantic similarity is derived from the WordNet lexical database for the English language. SSTC uses only subject-verb-object classification to generate clusters and readable labels. The algorithm also implements directed pruning to reduce the sub-tree sizes and to separate semantic clusters. Experimental results show that the proposed algorithm has better performance than conventional Suffix Tree Clustering (STC).**

***Index Terms*—senantic search results clustering, semantic clustering, semantic suffix tree clustering, text clustering.**

I. Introduction

Search engines are an invaluable tool to retrieve information from the internet. On the other hand they tend to return an enormous amount of search results and this causes a time consuming task to find the relevant ones. Moreover, if the relevant results do not occur in the first part of the returned results, then the user may fail to find them. A possible solution to this problem is use of the search results clustering techniques that works on snippets – a short text summarizing the context of search results. Search results clustering engines aim to cluster search results into different groups of relevant data and create a navigator to easily access the relevant search results for users. Nowadays, the development of a search results clustering system involves semantic search results clustering, which in turn uses the semantic meaning of words to cluster. This idea considers semantically related words such as synonyms or hyponyms for increasing the quality of clusters. In 2010, Ahmed Sameh and Amar Kadray [1] proposed the Semantic Lingo algorithm which uses synonyms to extract phrase terms to use as discovery cluster labels. The Semantic Lingo algorithm extends the Lingo techniques by adding semantic recognition, particularly using the WordNet database to achieve semantic recognition. Semantic recognition can improve the quality of the clusters generated. However, incremental processing significantly improves the efficiency of search results clustering [2][3], but the Semantic Lingo algorithm is not incremental. This paper proposes a new algorithm for semantic search results clustering called Semantic Suffix Tree Clustering (SSTC). The proposed system has two significant contributions to semantic result clustering. Firstly, a new data structure called Semantic Suffix Tree is introduced. This data structure is a method to simultaneously construct the semantic suffix tree through an on-depth and on-breadth pass suffix link by using semantic similarity and string matching. The semantic similarity equation is achieved by using a lexical database for the English language, the so-called WordNet [4]. Secondly, a new algorithm for search results clustering is presenting, called Semantic Suffix Tree Clustering (SSTC). According to evaluations, the SSTC can generate specific clusters and more readable labels than conventional STC. This paper is organized as follows. The related work and motivation are introduced in Section II. The Semantic Suffix Tree Clustering (SSTC) algorithm is proposed in Section III. In Section IV, the experimental are presented. Finally, the conclusions are concluded in Section V.

II. Related Work

Diverse approaches to search results clustering have been presented. Interestingly, the SHOC approach [5] was presented by Dell and Yisheng which is an extension of Zamir and Etzioni [6]. The SHOC algorithm uses suffix arrays to extract frequent phrases and singular value decomposition (SVD) techniques to discover the cluster content. The modified version of SHOC algorithm is the Lingo algorithm. This was presented by Stanislaw and Dawid [7] and combines common phrase discovery and latent semantic indexing techniques to group search results into meaningful groups. Lingo can create semantic descriptions by applying the cosine similarity formula and computing the similarity between frequent phrases and abstract concepts. In 2010, the latest version of Lingo techniques was proposed, called the Semantic Lingo techniques [1]. These techniques carry out semantic search results clustering which integrates semantic recognition to enable the application of synonyms in snippets and thus increasing the quality of the cluster generated. Unfortunately, due to the vector space model, it does not support incremental processing and is time consuming when applied to large numbers of snippets. According to the survey of web clustering engines [2] [3], it is clear that incremental processing is an important technique that can be used to improve the efficiency of search results clustering. An important algorithm to facilitate search results clustering is Suffix Tree Clustering

(STC) [6], which is an incremental and linear-time clustering and labeling algorithm – linear in the size of document set – using string matching on the suffix tree structure for finding shared common phrases of documents. The STC is a highly efficient but also has drawbacks. Various approaches have been proposed to solve the problems of STC. For example, the NSTC algorithm [8] was developed by using the vector space model to calculate the similarity of document pairs to solve the problem of large-sized clusters with poor quality returned by STC. The STC with n-gram technique [9] is used to solve the problem of the long path of a suffix tree created by the original suffix tree clustering. However, a difficulty is that the n-gram technique generates interrupted cluster labels if the common phrase size is greater than the defined n-gram size. Janruang and Kreesuradej [10] proposed a new partial phrase join operator to solve the problem of the interrupted cluster label arising from using the STC with n-gram techniques. To reduce the memory space, the STC with x-gram algorithm [11] combines the on-line construction of the suffix tree [12] with an x-gram technique to actually build the suffix tree structure. However, the STC with x-gram algorithm does not modify the STC to do semantic search results clustering. Even though the STC algorithm gives significant search performance it does not consider semantically related words to construct suffix tree and, therefore, eventually generates a huge tree with a complexity hard to manage. For instance, *doctor* and *physician* often imply a similar meaning though using different terms. However, if semantic similarity is combined with string matching to construct the suffix tree then the number of nodes decreases; consequently, the tree is attenuated. The significant contribution of this paper is a new approach to semantic search results clustering by using synonyms from the WordNet database to improve the similarity-based approach of the Semantic Lingo algorithm [1]. The logic is that text clustering algorithms should use meaning of the words for the purpose of clustering. In fact, humans directly use the concept or meaning of the words to group them. For example, in case of biomedical textual information clustering, the CSUGAR approach [13] is proposed but Medical Subject Heading (MeSH) ontology is used to create semantic clustering. This confirms the importance of semantic clustering in practical situations.

III. Semantic Suffix Tree Clustering

A. Semantic Suffix Tree

Semantic suffix tree (SST) is a new data structure that extends the suffix tree [14] since we require to use suffix trees on the meaning of word strings not characters. Words have meaning as an important property that relates them to other words, although there may not be a match of text string *per se*. Based on this idea we add the semantic similarity condition to the suffix tree. The semantic similarity equation is derived from the synonym set in the WordNet database by calculating the similarity measure of word pairs. Therefore, a semantic suffix tree will use both semantic similarity and string matching as conditions to create the suffix tree.

Semantic similarity is a measure that derives the synonyms set from WorldNet database. The semantic similarity equation is shown as Equation (1).

$$\begin{aligned} SemSim\,(w_a, w_b) &= 1 if\, |synset(w_a) \cap synset(w_b)| \geq 1 \\ SemSim\,(w_a, w_b) &= 0 \quad otherwise \end{aligned} \tag{1}$$

This equation uses the association of synonyms set of w_a and w_b to calculate the similarity. For example, a synonyms set of *eat* = {*feed, consume, corrode*} and a synonyms set of *ate* = {*eat, feed, consume*}, both synonyms sets of *eat* and *ate* contain *feed* and *consume*, which means more than one word. Therefore, a $SemSim(eat, ate)$ measure is equal 1. The pseudo-code for construction of the semantic suffix tree is shown in Algorithm 1.

```
Algorithm1:Construct Semantic Suffix Tree
1: input <-- set of String
2: output <-- semantic suffix tree
3: for each word (txt)
4:   if root is empty then{
5:     create a new node
6:     and update position
7:   }else{
8:     add a new node into cn
9:     do until pn = root {
10:      if SemSim = 0 && no match then{
11:        add a new node into pn
12:        and update suffix link
13:        update position
14:      }else{update suffix link
15:            and update position }
16:    }
17:  }
18: }
```

Figure 1 is a diagram which illustrates the working of Algorithm 1. Given a string in the order 1, 2, 3, and 4 as S = $\{w_1, w_2, w_3, w_4\}$, create a first node as cn (current node) by using the first word in the order that is directed via the root node, shown in Figure 1-A. Subsequently, a second word is created and connected to the cn node (on-depth traversal), as well assigned as a cn node. Then is created a new sub-tree (on-breadth traversal) because the second word does not match either the semantics or string child of the pn node as shown in Figure 1-B.

Similarly, a third and fourth word is generated, creating the sub-tree until the pn node is a root node that is like a stair ($currentlevel - 1$ to 0), shown in Figure 1-C and Figure 1-D. Finally, the position of the words is updated at the leaf node as shown in Figure 1-E. This example shows the characteristic of semantic suffix trees that makes it depend on the order of the words. This characteristic is limited by the length of the string.

Certainly, our semantic suffix tree algorithm simultaneously constructs the semantic suffix tree through an on-depth and on-breadth process via a suffix link. This algorithm can supply all

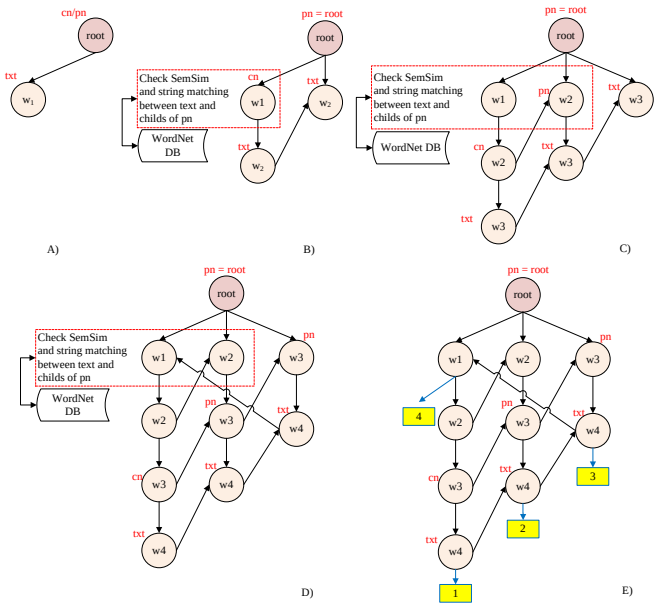

Fig. 1. An example of the construction of the semantic suffix tree based on Algorithm 1 and the string w_1, w_2, w_3 and w_4. Figure 1-A depicts an initial step where root is empty. Figures 1-B and 1-A show the creation of the new sub-tree when semantic similarity is 0 so there is no matching. Figures 1-D shows how to update the suffix link when semantic similarity = 1 so there is matching. Figures 1-E is shows how to update the position of a word when it is a last word.

phrases to the tree wholly. Unlike this algorithm, the on-line construction of suffix trees of Ukkonen's algorithm [12] used only an on-line path. It needs lower memory space but some phrases cannot be fed to the tree wholly [11].

B. Semantic Suffix Tree Clustering

The Suffix tree clustering (SSTC) algorithm is incremental and automatically clusters and labels. Appropriately, SSTC creates specific concept clusters that are of higher precision. Therefore, the clusters in this paper mean concept or meaning clusters. The concept cluster is a cluster that collects semantically similar documents into the same cluster. For instance, if a document A contains *cat ate cheese* and a document B contains *cheese was eaten by cat*, then both A and B should group to the same cluster as they share the same concept. The SSTC is done in three logical phases as follows.

```
Algorithm 2: Main algorithm of SSTC
1: input<--set of snippets
2: output<--set of final cluster
3://Phase1:preprocessing & construct tree
4: for each snippets
5:    extract the sentences
6:    for each sentence
7:      detected Subject,Verb, and Object
8:      construction of SSTC(Algorithm1)
9:    }
10: }
11://Phase2:Tree pruning(algorithm3)
12:   2.1 tree pruning step
13:   2.2 compact tree step
14://Phase3:Identify cluster(algorithm4)
15:   3.1 keep base cluster
16:   3.2 finding final cluster
```

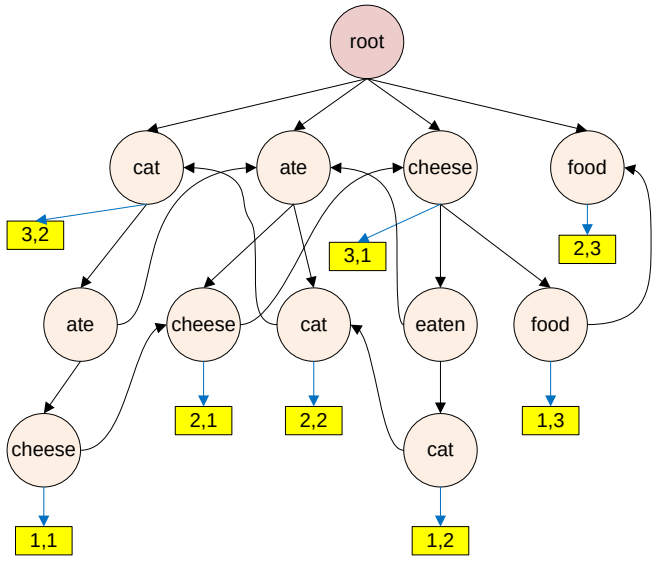

Fig. 2. This illustration it shows the completely semantic suffix tree of three documents: D_1 = {cat, ate, cheese} , D_2 = {cheese, eaten, cat} and D_3 = {cheese, food}.

Phase 1: Preprocessing and constructing semantic suffix tree

This step uses OpenNLP [15] techniques to extract the sentences and then uses Stanford typed dependencies techniques [16] to detect the Subject, Verb, and Object of each sentence. In a preprocessing step, it can find the maximum length of suffixes fed to the semantic suffix tree. SSTC uses only Subject, Verb, and Object to create a semantic suffix tree as explained in detail in Algorithm 1. For example, given the three documents/snippets D_1 = {*cat, ate, cheese*}, D_2 = {*cheese, was, eaten, by, cat*} and D_3 = {*cheese, is, food*}, after natural language processing, these are changed to be D_1 = {*cat, ate, cheese*}, D_2 = {*cheese, eaten, cat*} and D_3 = {*cheese, food*}. Natural language processing can reduce the number of nodes in the tree but the the phrase does not lose meaning. The resulting semantic suffix tree is show in Figure 2.

Phase 2: Tree pruning

A semantic suffix tree returns a huge tree in respect of both on-depth and on-breadth. It makes it difficult for execution. Tree pruning is the process to reduce the number of nodes on-depth and sub-trees on-breadth while still retaining the concept or meaning of each node and sub-tree.

The tree pruning step uses preorder traversal that traverses from left to right using the depth-first traversal method. The details of this step include a move to update or to delete a branch (on lines 3-24) and then compactify the tree (on lines

27-28) when the tree pruning step is finished. To summarize, the pruning tree algorithm is shown in Algorithm 3.

```
Algorithm 3: Tree pruning algorithm
1: assign gn=root
2: for each Ti
3:  if Ti is subset Tj then Ti is deleted
4:  else{
5:     assign pn node
6:     for each pn's child node{
7:       assign cn
8:      for each cn's child node{
9:       if cn has suffix link then {
10:       assign sn
11:       case gn!=sn {
12:        if sn's childs related pn||ccn
13:        then move sn to replace cn
14:        else{move sn's documents to cn
15:             and move csn into cn}
16:        }
17:        case gn==sn {
18:         move cn's documents into pn
19:         and delete cn branch }
20:        }
21:       gn=pn, pn=cn, cn = ccn
22:       }
23:     }
24:  }
25:}//end of tree pruning step
26://compact tree
27: if pn's childs==1&documents==null
28: then pn and cn are combined
```

The conditions on lines 11-20 check the grandparent node (gn) and the suffix node (sn). The suffix node is a suffix of pn because the suffix link of a child of pn is directed to this node. We assume that if gn is equal to sn then a loop of nodes is generated. This loop is deleted (line 17-20 and the example as shown in Figures 5-6). Conversely, if gn is not equal to sn, we move a sub-tree of sn to update the branch along the cn node because sn is subset of pn. The association between a child node of sn (csn) with pn and a child of cn (ccn) are the conditions for directing the subset of sn and pn (lines 11-16 and the example as shown in Figures 3-4.).

Additionally, compactification is a technique used to reduce the length of tree. The node pairs between pn and cn after pruning the tree are merged if pn does not have document members and suffix link, or if it has a child node (line 27-28 and the example as shown in Figure 7).

For instance, the diagram of some steps and its results are shown in Figures 3-7). In Figure 3 a sub-tree that starts at sn is a sub-tree of pn because $cn = sn$. It means cn and sn represent the same concept.

The possibility of deleting sub-tree T_i when T_i is a sub-tree of T_j on line 3 when $j = 0, 1, 2, ..., i-1$. Consequently, If all nodes of T_i have suffix link as the nodes of a T_j, T_i is subset of T_j, and this is a meaning of the subset between T_i and T_j.

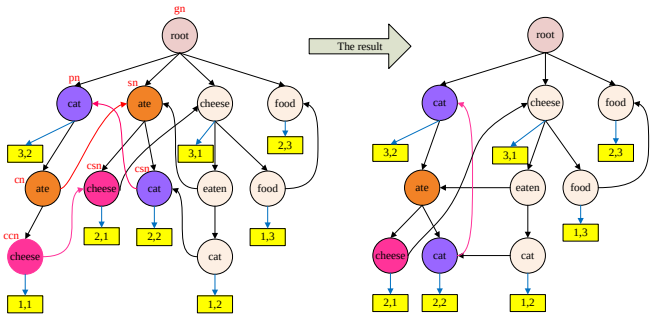

Fig. 3. An example is shown to prune a sub-tree that has the same concept as pn. The number of sub-trees is reduced because sn updates to cn.

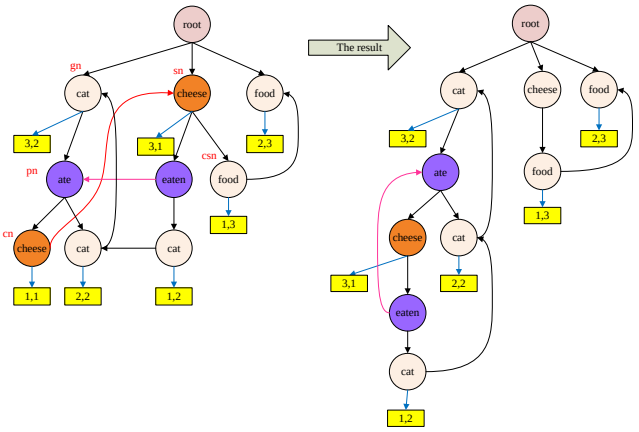

Fig. 4. Only some parts of the sn sub-tree that are related with pn are transferred to update cn node since $sn \neq cn$.

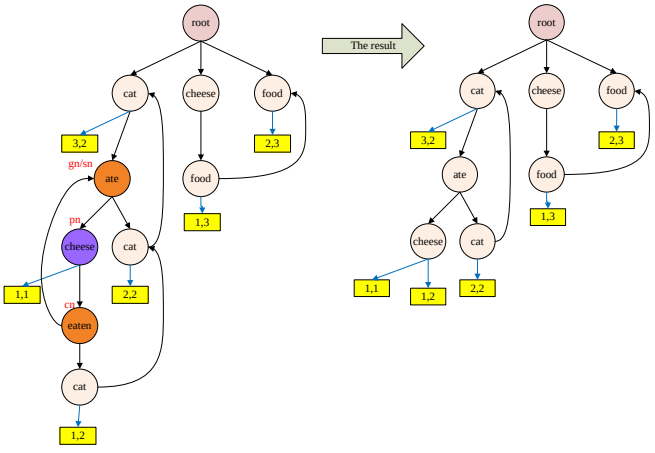

Fig. 5. The same meaning is shared by sentences of passive voice and active voice causes the loop of a string on a semantic suffix tree. In this example, the loop is deleted.

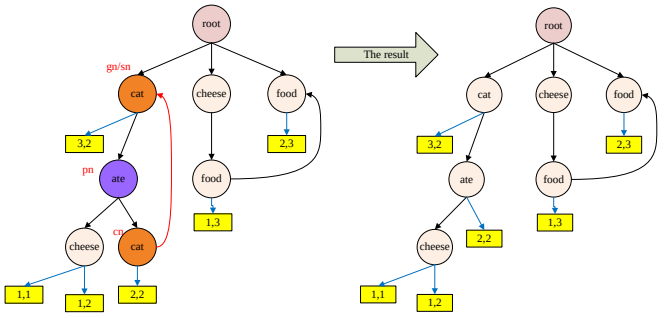

Fig. 6. This figure is similar to Figure 7 in deleting the loop of a string. It shows the right node when the left is nearing finish. For implementation we use a stack to collect the right node and then process the stack until it is empty.

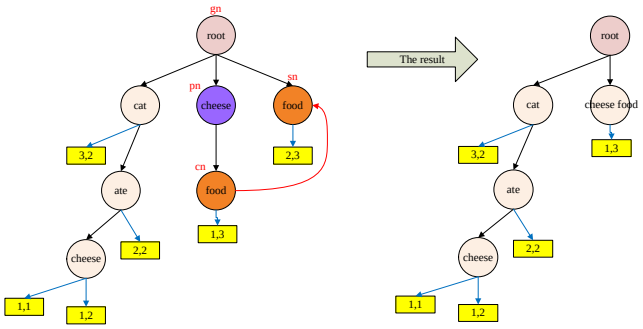

Fig. 7. After tree pruning between cn and sn, the next step is to compactify the tree. In this case, pn does not have document member and suffix link but it has a child node. Therefore, pn and cn combine to a node *cheese food*.

The tree pruning step can separate the sub-tree that has a similar concept with help of a useful structure related via a suffix link of semantic suffix tree. In addition, this step can reduce the number of nodes on-depth and sub-trees on-breadth. It causes a large reduction in size of a semantic suffix tree.

Phase 3: Identify clusters

In this paper, we assume each sub-tree (T_0, T_1, ... , T_i) is a concept cluster and each node a cluster that has a set documents be member. We use the postorder traversal techniques to calculate cluster and label as shown in Figure 8. Consequently, the structure of a semantic suffix tree is created node by node from left to right because this clustering will direct to a depth to keep the prefix phrases of set documents in a manner to avoid useless clusters. It makes for readable labels because phrase labels are more readable than a single word.

```
Algorithm 4: Identify cluster
1: assign gn=root;
2: for each sub-tree{
3:   assign pn node
4:   collect into CanCluster
5:   if pn has child node {
6:     //use stack to keep right node
7:     //and work until stack is empty
```

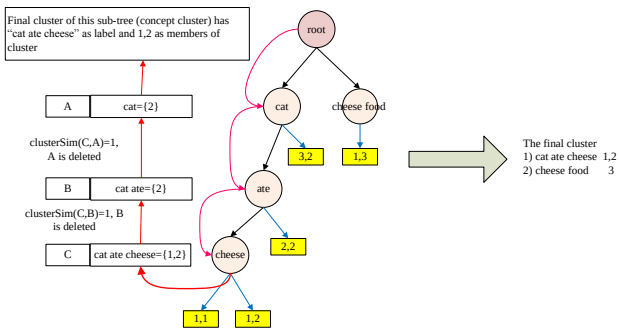

Fig. 8. The diagram to illustrate the cluster identification step that uses postorder traversal to calculate cluster similarity.

```
8:      for each child of pn node
9:       keep cluster&label to CanCluster
10:       if cn does not child node{
11:            compute ClusSim
12:            filter the useless cluster
13:       }
14:      }
15:    }
16:  transfer CanCluster to final_cluster
17:  and clear CanCluster
18: }
19:return final_cluster
```

The cluster similarity measure is to calculate by using ClusSim equations as shown in Equation (2).

$$\begin{aligned} ClusSim\,(C_a, C_b) &= 1 if\, |C_a \cap C_b| = |C_a|, C_a \text{ is deleted} \\ &\quad or\, |C_a \cap C_b| = |C_b|, C_b \text{ is deleted} \\ ClusSim\,(C_a, C_b) &= 0\ otherwise \end{aligned} \tag{2}$$

For example, given three clusters *cat* = {2}, *cat ate* = {2}, and *cat ate cheese* = {1,2}. After checking for the similarity cluster and deleting the useless cluster, the resulting final cluster is *cat ate cheese* = {1,2}. Correctly used, this equation can reduce the number of useless clusters and return a specific readable cluster. Although, the document number 2 does not have the phase *cat ate cheese*, its meaning is *cat ate cheese*.

The final results of this example : D_1 = {*cat, ate, cheese*}, D_2 = {*cheese, eaten, cat*} and D_3 = {*cheese, food*} shown in Figure 8 are *cat ate cheese* = {1,2} and *cheese food* = {3}. The labeled cluster is *cat ate cheese* and the members of this cluster are D_1 and D_2. From the final results, the SSTC algorithm can cluster the documents that have a semantic similarity because it can cluster D_1 and D_2 to a cluster and then separate D_3 to another cluster.

IV. Experimental Results

In order to evaluate the SSTC algorithm, we compare with the STC algorithm for precision, coverage, overlap, number of node, and number of branch,. The results of comparing are as shown in Table 1. The dataset for testing the search

TABLE I
THE AVERAGE OF PRECISION, COVERAGE, OVERLAP, NUMBER OF NODE, AND NUMBER OF BRANCHS COMPARED BETWEEN STC AND SSTC.

Algorithm	precision	coverage	overlap	Num. of node	Num. of branch
STC	0.68	1	12.28	330,064	47,073
SSTC	0.81	0.99	5.70	128,657	29,552

results clustering engine is created from a small testing dataset by using 26,890 search results from 10 queries words on Dmoz.com [17].

From Table 1 the precision of SSTC is higher than the STC algorithm because SSTC tries to get specific clusters. Users can examine specific clusters to access the relevant document. However, the coverage and overlap measure is less than the STC algorithm. The SSTC can reduce the number of nodes and branches of the original STC more than 39.98 and 37.22 percent, respectively.

From Table 2 the SSTC algorithm can generate specific clusters and readable cluster labels more efficiently than Semantic Lingo. The specific clustering of SSTC makes possible higher precision and faster access to the relevant result. For instance, SSTC returned four specific clusters, *doctor*, *doctor guide*, *doctor directory* and *doctor directory guide*, while Semantic Lingo returned the big cluster *doctor directory*.

V. CONCLUSION

This paper proposes a new algorithm, called Semantic Suffix Tree Clustering (SSTC), that uses the meaning of the words to cluster. SSTC can cluster documents that share a semantic similarity. Specific cluster are returned in a readable form. Additionally, the SSTC can improve the performance of approaches that use the original STC algorithm because it can cluster semantically similar documents, reduce the number of nodes and reach higher precision.

For future work we plan to extend the SSTC algorithm to hierarchical clustering.

TABLE II
DOCUMENT TITLES USED FOR COMPARISON BETWEEN SSTC AND SEMANTIC LINGO [1] AND THE RESULTS CLUSTER GENERATED BY SSTC VS. SEMANTIC LINGO.

Document title	Comparison of clusters	
	SSTC	Semantic Lingo
1) doctor's guide 2) think like a doctor 3) physician directory 4) emergency physician directory group, pc 5) ama physician select 6) doctor directory 7) large scale singular value computation	**Cluster: doctor** 2) think like a doctor **Cluster: doctor guide** 1) doctor's guide 5) ama physician select **Cluster: doctor directory** 3) physician directory 6) doctor directory **Cluster: doctor directory group** 4) emergency physician directory group, pc	**Cluster:Doctor directory** 2) think like a doctor 3) physician directory 4) emergency physician directory group, pc 5) ama physician select 6) doctor directory

REFERENCES

[1] Ahmed, M.S., Amar, M.K.: Semantic Web Search Results Clustering Using Lingo And Wordnet. In: IJRRCS: Kohat University of Science and Technology (KUST), Vol. 1, No 2, pp. 71–76. , Pakistan (2010)

[2] Stanislaw, O., Jerzy, S.,: An algorithm for clustering of web search results. Master Thesis, Poznan University of Technology, Poland, June 2003

[3] Carpineto, C., Osinski, S., Romano, G., and Weiss, D.: A survey of Web clustering engines. In: ACM Computing Surveys, Volume 41 , Issue 3, pp. 1–38. ACM, New York, USA (2009)

[4] a large lexical database of English, http://wordnet.princeton.edu/

[5] Dell, Z., Yisheng, D.: Semantic, Hierarchical, Online Clustering of Web Search Results. In: APWeb 2004, pp.69-78 (2004)

[6] Zamir, O., Etzioni, O.: Web document clustering: a feasibility demonstration. In: SIGIR 1998, pp. 46-54 (1998)

[7] Stanislaw, O., Jerzy, S.,:A concept-driven algorithm for clustering search results. In: Intelligent Systems, IEEE , Vol. 20, No. 3. , pp. 48-54 (2005)

[8] Chim, H., Deng, X.: A new suffix tree similarity measure for document clustering. In: Proceedings of the 16th international conference on World Wide Web (WWW 2007), pp. 121-130., ACM, New York, NY, USA (2007)

[9] Zeng, H., He, Q., Chen, Z., Ma, W., Ma, J.: Learning to cluster web search results. In: SIGIR '04, pp. 210-217., ACM, New York, NY, USA (2004)

[10] Janruang, J., Kreesuradej, W.: A New Web Search Result Clustering based on True Common Phrase Label Discovery. In: CIMCA '06, pp. 242, IEEE Computer Society, Washington, DC, USA (2006)

[11] Wang, J., Mo, Y., Huang, B., Wen, J., He, L.: Web Search Results Clustering Based on a Novel Suffix Tree Structure. In: ATC '08, pp. 540-554, Springer-Verlag, Berlin, Heidelberg (2008)

[12] Esko, U.: On-Line Construction of Suffix Trees. In: Algorithmica, Vol. 14, No. 3., pp. 249-260 (1995)

[13] Illhoi, Y., Xiaohua, H., Il-Yeol, S.: A coherent graph-based semantic clustering and summarization approach for biomedical literature and a new summarization evaluation method. In: BMC Bioinformatics 2007, 8(Suppl 9):S4 (2007)

[14] Gusfield, D.: Algorithms on String, Trees, and Sequences computer science and computational biology. Cambridge University Press (1997)

[15] OpenNLP projects, http://opennlp.sourceforge.net/

[16] Marie-Catherine de, M., Christopher, D.: Stanford typed dependencies manual. http://nlp.stanford.edu

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