Web Search Results Clustering Based on a Novel Suffix Tree Structure

Junze Wang, Yijun Mo, Benxiong Huang, Jie Wen, and Li He

Institude of Communication Software and Switch Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
wangjunze@smail.hust.edu.cn

Abstract. Web search results clustering are navigator for users to search needed results. With suffix tree clustering (STC), search results can be clustered fast, automatically, and each cluster is labeled with a common phrase. Due to the large memory requirement of suffix tree, some other approaches have been proposed, with lower memory requirement. But unlike other algorithms, STC is an incremental algorithm and a promising approach to work on a long list of snippets returned by search engines. In this paper we proposed an approach for web search results clustering and labeling, based on a new suffix tree data structure. The approach is an incremental and linear time algorithm, with significantly lower memory requirements. This approach also labels every final cluster a common phrase, thus it is suitable for quickly browsing by users. Experimental results show that the new approach has better performance than that of conventional web search result clustering.

Keywords: Search results organization, document clustering, incremental clustering, a new suffix tree, lower memory requirement.

1 Background and Related Work

Existing search engines often return a long list of search results, so users have to go through the list to identify their required results. The goal of a clustering algorithm on our domain is to group each document with others sharing a common topic, thus helps users to find relevant results.

Most traditional clustering algorithms cannot be directly used for web search results clustering because of some practical issues. For example, the clustering algorithm should be fast enough for online calculation; and the generated clusters should have readable descriptions for quick browsing by users, etc. Zamir and Etzioni [1][2] gave a detailed analysis on these issues. They also proposed an algorithm named STC, which finds clusters based on the common phrases shared by snippets.

In recent years several web search results clustering algorithms have been proposed [3-8]. But they are not incremental clustering algorithm. Unlike them, the STC algorithm is an incremental algorithm, so web search result clustering based on this algorithm is a promising approach to work on a long list of snippets returned by search

C. Rong et al. (Eds.): ATC 2008, LNCS 5060, pp. 540–554, 2008.

engines. However the original STC algorithm often constructs a long path of suffix tree and suffers from large memory requirements.

Hau-Jun Zeng and etc. [9] introduced an improved suffix tree with N-gram to deal with the problem of the original suffix tree. However, the suffix tree with N-gram can discover only partial common phrases when the length of N-gram is shorter than the length of true phrases. For example, given a true phrase "suffix tree clustering algorithm", a suffix tree with 3-gram can discover partial phrases: "suffix tree clustering" and "tree clustering algorithm". In this case, STC with N-gram labels a cluster with a partial phrase (probably unreadable), and gives too many candidate clusters. Thus, it may hurt the final cluster quality.

In response to this situation, Jongkol Janruang [10] proposed a new partial phrase join operation, which can join the partial phrases, combine the candidate clusters, and generate more readable labels.

Additionally, STC algorithm extracts all right-complete substrings of the true phrase (including the true phrase itself). Take the phrase "suffix tree clustering algorithm" for example, all the right-complete substrings of it, such as "tree clustering algorithm", "clustering algorithm" and "algorithm", will be discovered, and all these partial phrases are regarded as candidate phrases. In this condition STC algorithm gives too many candidate clusters and may hurt the final cluster quality too. On this issue [11] has given a good analysis on this issue.

We will analyze the shortcomings of STC with N-gram further more in section 2; In section 3, we will propose a new suffix tree data structure, named suffix tree with X-gram, along with a improved STC algorithm which overcomes the shortcomings of conventional STC algorithms, and still maintains the advantages; At last the experimental results show that our new approach has better performance than that of conventional STC algorithms.

2 Suffix Tree with N-Gram

Original suffix tree is a very efficient way to identify true common phrases in snippets, but suffers from large memory requirements. Suffix tree with N-gram performs the similar function, and has lower memory requirements. "With N-gram" means the suffixes fed to the suffix tree is limited no more than N. If a suffix is longer than N, only the first N chars will be fed to the tree and the chars after the Nth char will be discarded. As an example, an original suffix is shown in Figure 1, and a suffix tree with N-gram (N=3) is shown in Figure 2, given the snippet [suffix, tree, clustering, algorithm, x1, x2, x3] for building the two suffix trees. The expression $\{x_1, x_2, \ldots, x_i\}$ (m, n) means the word sequences "$x_1, x_2, \ldots, x_i$" present in snippet m, at position n. So in Figure 2, {suffix, tree, clustering} (1, 1) means "suffix tree clustering" presents in snippet 1, at position 1; {tree, clustering, algorithm} (1, 2) means "tree clustering algorithm" presents in snippet 1, at position 2.

Obviously suffix tree with N-gram maintains fewer words than original suffix tree. In this way it has lower memory requirements. Maintaining fewer words also implies it spends less time in building the tree.

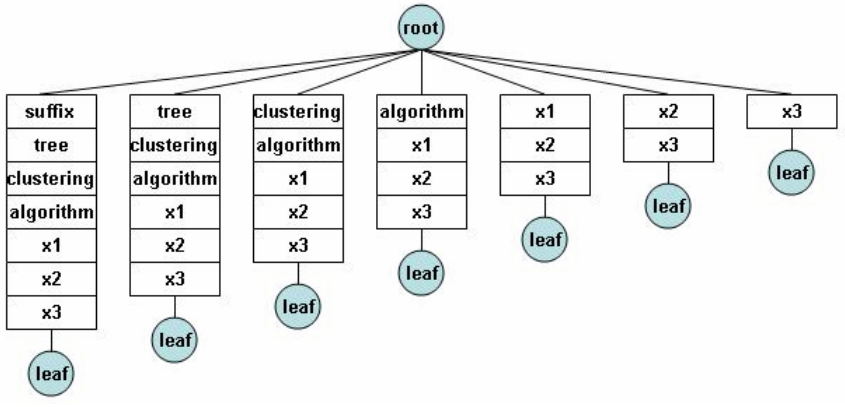


Fig. 1. The original suffix tree building depends on the snippet [suffix, tree, clustering, algorithm, x1, x2, x3]

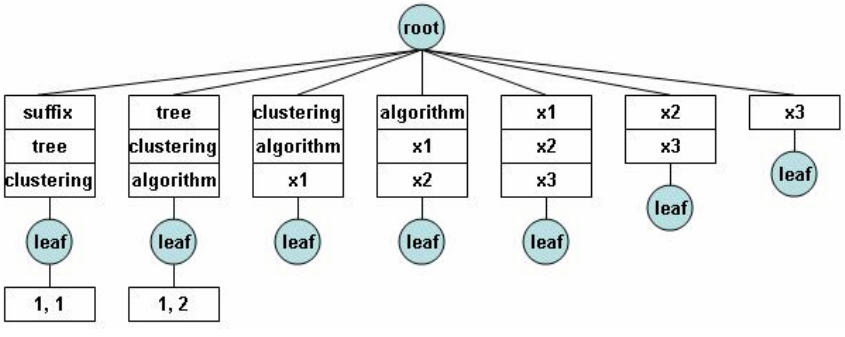


Fig. 2. The suffix tree with 3-gram building depends on the snippet [suffix, tree, clustering, algorithm, x1, x2, x3]. The maximum depth of this suffix tree is 3.

However, suffix tree with N-gram discovers only partial common phrases when the length of N-gram is shorter than the length of true phrases. For example, give a true phrase "suffix tree clustering algorithm", a suffix tree with 3-gram can discover partial phrases: "suffix tree clustering" and "tree clustering algorithm".

A new partial phrase join operation is proposed in [10]. The candidate cluster combining technique uses the join operation to define a new common phrase of a new cluster when merging a pair of similar candidate clusters. For example, the candidate cluster A = {suffix, tree, clustering} (1, 1) and B = {tree, clustering, algorithm} (1, 2) shown in Figure 2, a new common phrase is defined as

$$A + B = \{\text{suffix, tree, clustering, algorithm}\}\ (1, 1)$$

This new phrase can be discovered in snippet [suffix, tree, clustering, algorithm, x1, x2, x3].

The partial phrase join operation generates the true common phrases which are more readable, but still can not overcome the shortcomings of "right-complete" [12]. Given the true phrase "suffix tree clustering algorithm", STC discovers all its right-complete substrings, such as "tree clustering algorithm", "clustering algorithm", "algorithm" and the phrase itself, and considers all of them as candidate clusters. But in all of them, only the phrase itself is useful for clustering! Given a phrase with length L, there are at most L right-complete substrings. Generate so many useless candidate clusters will increase the consumption of STC algorithm and hurt the final cluster quality.

In addition, there is still some redundant data in the suffix tree with N-gram. In the next section we will introduce a new data structure named suffix tree with X-gram, which can be constructed with less memory space; and we will introduce a complement operation to eliminate the useless "right-complete" substrings of the true phrases.

3 The Clustering Algorithm Based on Suffix Tree with X-Gram

In order to lower the memory requirement, the maximum length of suffixes fed to the suffix tree should be limited. In STC with N-gram, it is no more than a constant variable N. But in fact, we want the true phrases fed to the tree as a whole one, even it is longer than N; and the noisy word sequences fed to the suffix tree as short as possible, even it is already shorter than N.

In the clustering algorithm based on suffix tree with X-gram, we use X to denote the maximum length of suffixes fed to the suffix tree. We make X an adaptive variable. The suffix tree with X-gram also limits the length of the suffixes which fed to the tree, but is more reasonable than with N-gram. With this data structure, the word sequences which are presented more frequently are considered more likely to be true common phrases, and will be fed to the tree as a whole one; but the noise word sequences fed to tree will be limited, even it is not so long. In this way suffix tree with X-gram discovers the true common phrases, and maintains fewer words than suffix tree with N-gram. Now we show the construction process of suffix tree with X-gram.

A suffix tree with X-gram for the word sequences S[1…m] can be built like this: first enters the first word S[1] into the tree as a leaf node. Then it successively enters the suffix S[i…j] into the growing tree, for i increasing from 2 to m, and S[i…j] is the longest prefix of suffix S[i…m] matched the conditions. The details of this construction method are presented as follow:

1. Initialize a tree only has a root node. Add the first word S[1] to the tree and then generate the suffix tree T_1, which only has a leaf node denotes the word S[1].

2. Tree T_{i+1} is constructed from T_i. The steps are as follows:

2.1. Starting at the root of Ti the algorithm finds the longest path from root whose label matches a prefix of S[i+1…m]. This path is found by successively comparing and matching words in suffix S[i+1…m] to words along a unique path from the root, until no further matches are possible.

2.2. When no further matches are possible, the algorithm must arrive at a node, say N_{cur}. Now the match part between the path in the tree and the suffix S is S[i+1, j]. The algorithm creates a new leaf node labeled S[j+1], which is a child node of N_{cur}.

For instance, given the snippet [suffix, tree, used, in, suffix, tree, clustering], this algorithm first feeds the word "suffix" to the tree (step 1 in the algorithm), then feeds the words "tree", 'used', "in" to the suffix tree ordinal. Then it is turn to the word "suffix", and "suffix" already exists in the tree as a path from root to a leaf node. So add word "tree" to tree, as a child node of "suffix" (step 2.2 in the algorithm). The suffix tree generated at last is shown in Figure 3.

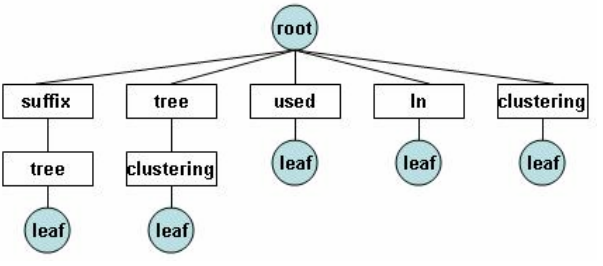


Fig. 3. An Example of Suffix Tree with X-gram. Building depends on the snippet [suffix, tree, used, in, suffix, tree, clustering]. Step 1: feed "suffix" to tree; step 2: feed "tree" to tree; step 3: feed "used" to tree; step 4: feed "in" to tree; **step 5: feed "suffix tree" to tree; step 6: feed "tree clustering" to tree;** step 7: feed "clustering" to tree.

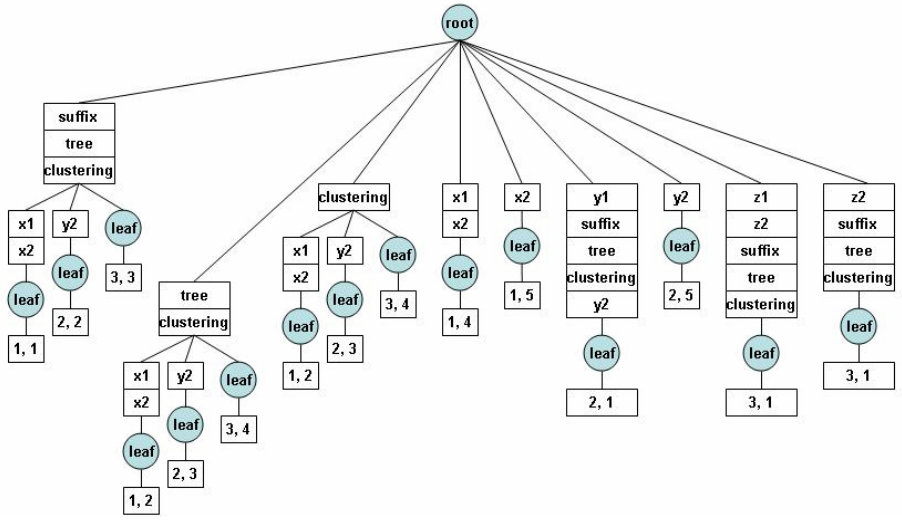
The suffix tree with N-gram limited the depth of suffix tree with a constant variable N; but with X-gram, the depth of suffix tree will be limited with a variable X, which is not constant but an adaptive variable. So if a word sequences present several times in snippets, it will be fed to the tree wholly, even it is longer than N; and if a word sequences present little times, it will be feed to the tree partial, even it is shorter than N.

Given the snippets set show in Table 1:

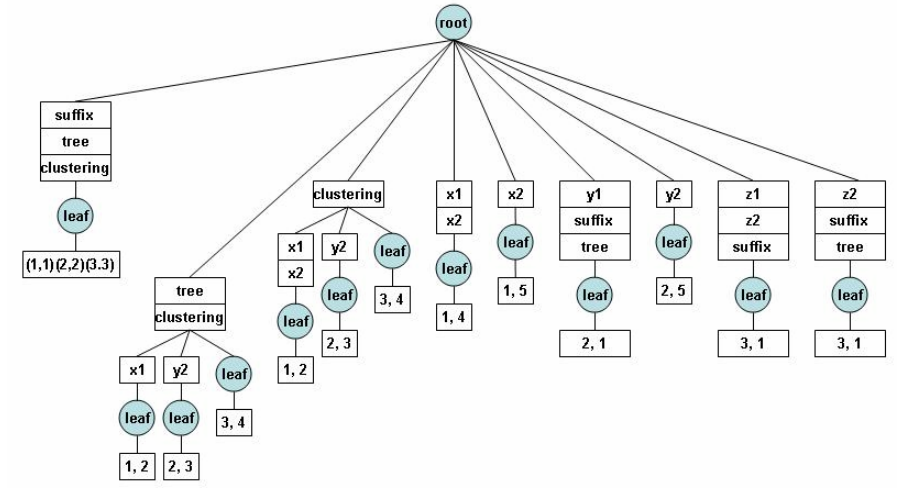
Table 1. Snippets set

D1	suffix, tree, clustering, x1, x2
D2	y1, suffix, tree, clustering, y2
D3	z1, z2, suffix, tree, clustering

Build the original suffix tree, suffix tree with 3-gram and suffix tree with X-gram, which are shown in Figure 4(a), Figure 4(b) and Figure 4(c). The number of nodes maintained by respective suffix tree is show in Table 2. We use Ukkonen's algorithm [13] to construct the suffix tree and every node denotes a word.

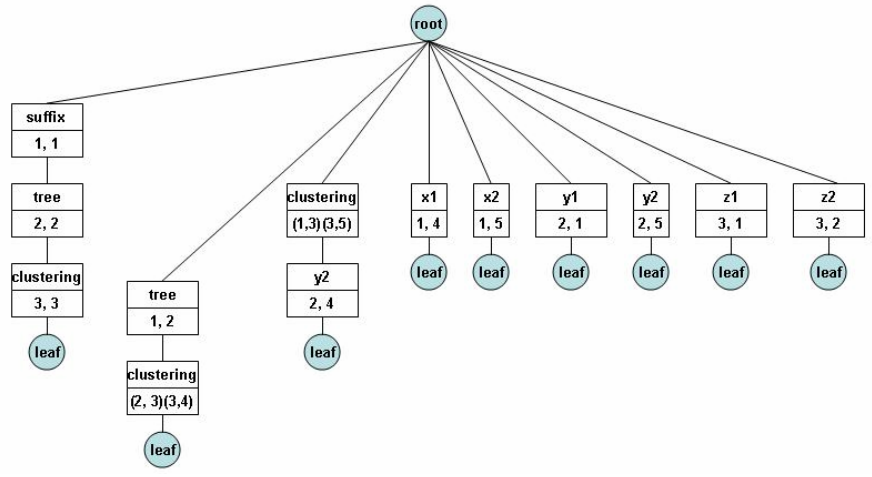


(a)



(b)

Fig. 4. (a) Example of original suffix tree. Building with the snippets set shown in Table 1. (b) Example of suffix tree with 3-gram. Building with the snippets set shown in Table 1. (c) Example of suffix tree with X-gram. Building with the snippets set shown in Table 1.



(c)

Fig. 4. (*continued*)

From Table 2 we see the suffix tree with X-gram maintains fewer nodes, and the true common phrase "suffix tree clustering" has been feed to the tree wholly. It is because of many word sequences in the snippet sets are noise, and noisy word sequences feed to the suffix tree with X-gram are shorter than the other two types of suffix tree. So, with our approach the algorithm maintains fewer nodes, and lowers the memory requirements.

Table 2. The total number of nodes maintained by three different STC algorithms

	Original STC	STC with 3-gram	**STC with X-gram**
number of nodes be maintained	33	24	**13**

But there are still some problems needed to be dealt with in this new approach.

1. Some phrases can not feed to the tree wholly.

Review the process of building a suffix tree with X-gram. The true phrase can not be fed to the tree wholly when it first presents in the snippets set. For a true phrase with length L, it will be fed to the tree wholly after it presented L times at most. Take the example of the phrase "suffix tree clustering" in snippets set shown in Table 1. When it presents at the first time in D1, the substring "suffix" was fed to the tree; when it presents at the second time in D2, the substring "suffix tree" was fed to the tree; when it presents at the third time in D3, the whole phrase "suffix tree clustering" was fed to the tree.

It need to be noticed that in application, most true common phrases no longer than 4 (almost 80% [14]). It means most true common phrases can be fed to the tree as a whole phrase, after they present 4 times at most.

On the other hand, the word sequences with frequency no greater than threshold T are considered as noise and should be filtered out in clustering process. Suppose T is 4, then the word sequences with frequency less than 4 will be considered as noise and should be filtered out.

In a word, if a word sequence with frequency no less than threshold T (suppose T is 4), it will be considered as a true phrase, and it probably can be fed to the tree as a whole one; and if a word sequence with frequency less than the threshold T, it should be a noisy word sequences and need not to be feed to the tree as a whole one.

Review the snippets set shown in Table 1, the word sequence "suffix tree clustering" presents 3 times and fed to the tree wholly at last, so it should be a true phrase; and the word sequence "x1 x2" can not be fed to the tree as a whole one, and it obviously a noise.

In addition, the vast majority of the true common phrases with length no more than 6 (more than 90% [14]), so we limited the most depth of suffix tree with X-gram no more than 6. In this case, the vast majority of the true common phrases can still be found wholly.

Undoubtedly there are still some true phrases be broken up and can not fed to the tree wholly. In this condition, we adopt the partial phrases join operation, and then the true common phrase can be discovered.

Although join operation still needed in STC with X-gram, due to more true phrases can be fed to the suffix tree wholly, so fewer partial phrases are needed to be joined, than that in STC with N-gram.

2. A phrase can be fed to the tree wholly, but not all snippets contain this phrase can be discovered in the suffix tree with X-gram.

The snippets set are given in Table 3, we build a suffix tree with X-gram which is shown in Figure 5. We give only 3 branches (branch A, B, and C) and omit other branches. Here we use Ukkonen's algorithm. The dotted lines denote the suffix links.

Table 3. Snippets set

D1	suffix, tree, clustering, x1, x2
D2	y1, suffix, tree, clustering, y2
D3	z1, z2, suffix, tree, clustering
D4	suffix, tree, clustering, v1, v2
D5	w1, suffix, tree, clustering, w2

The true phrase "suffix tree clustering" presented in D1, D2, D3, D4 and D5, but in the suffix tree we can only find it presented in D3, D4 and D5. In the suffix tree with X-gram, we merely discover D1 contains "suffix", D2 contains "suffix tree", but we can not discover D1 or D2 contains the phrase "suffix tree clustering". It means that not all the snippets contain this phrase can be discovered in the suffix tree with X-gram.

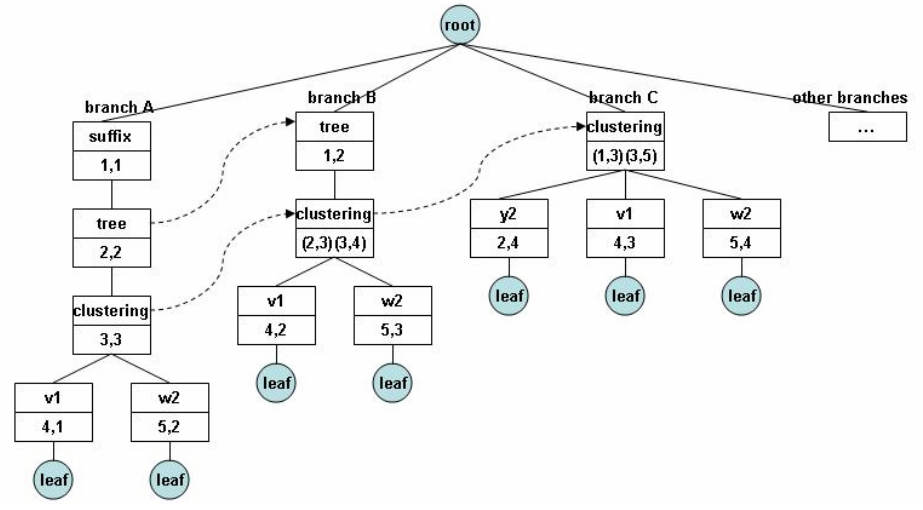


Fig. 5. The suffix tree with X-gram building with the snippets set shown in Table 3. The dotted lines denote the suffix links.

In this condition, we can not select the candidate clusters depending on the frequency of a word sequences in the suffix tree directly. We should complement this branch first. In this step, the successive word sequences followed "suffix" in D1 and successive word sequences followed "tree" in D2, will be fed to the tree, in order to complement the branch. The length of these word sequences is limited; ensure the depth of the tree no more than 6. Because we used the Ukkonen's algorithm to construct the suffix tree, so the successive word sequences can be located with suffix links.

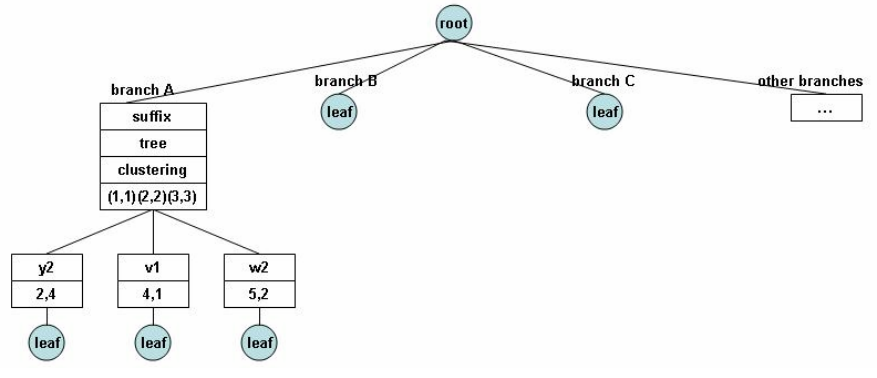


Fig. 6. The suffix tree with X-gram after complement

After complement operation, the suffix tree is shown in Figure 6.

Now we can get a candidate cluster {suffix, tree, clustering} (1, 1)(2, 2)(3, 3)(4,1)(5, 2) from the tree. And after complement operation, all the merely "right-complete" substring of true phrase "suffix tree clustering", such as "tree clustering" and "clustering" are all filtered out.

In the branch A shown in Figure 5, the word sequences followed "suffix" in D1, the word sequences followed "suffix tree" in D2, the word sequences followed "suffix tree clustering" in D3, are all uncertainty. These parts may form "suffix tree clustering v1" 3 times, or "suffix tree clustering w2" 3 times, or "suffix tree clustering" 3 times. Suppose the threshold T of the frequency is 3, this branch may contain candidate clusters, so it needs to be complemented.

It should be emphasized that not all the branches need to be complemented. Take the branch C shown in Figure 7 for example, it can not contain phrase with frequency greater than 2. Suppose T is 3, then this branch can not contain candidate clusters. This branch needs not to be complemented.

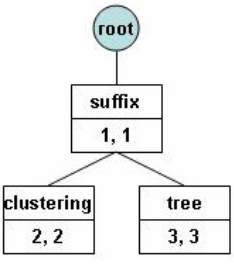


Fig. 7. A branch which need not to be complemented

Suppose a branch contains such a phrase, which with frequency C in the suffix tree, and with length L. If C+L is greater than T, this branch should be complemented.

In fact, the phrase of length L and with frequency C may presents L-1 times before it can be fed to tree as a whole one. It means this phrase may present C+L-1 times in whole snippets set. When C+L > T, this phrase may be a candidate cluster. So the branch which contains this phrase should be complemented.

Take the example of the phrase "suffix tree clustering" in snippets set shown in Table 3. In Figure 5 we discover this phrase presents 3 times in branch A ([suffix, tree, clustering] (3, 3)(4, 1)(5, 2)), and its length is 3. C is 3, L is 3, so this phrase should presents 5 times in the snippets set (C+L-1=5). Actually this phrase presents 5 times (in D1, D2, D3, D4, and D5).

The steps of search results clustering algorithm based on suffix tree with X-gram are as follows:

1. The document cleaning stage. For most text-based document clustering algorithms, this stage is very similar. The HTML tags, punctuation and other similar non-informative text are removed; a set of stop words removed; stemming is applied to reduce words to their root form.

2. In the second stage, a suffix tree with X-gram is created using the word sequences in the snippets set. Then complement the braches which may contain the candidate clusters. In this process we filter out all the merely "right-complete" substrings of true common phrases, and get the candidate phrases.

3. The candidate clusters are merged, scored and sorted. Then generate the final clusters. To keep the cost of this last step constant, we do not check all the candidate clusters, but only with the k highest scoring ones (we take k to be 100 in our experiments).

4 Experiments

4.1 Experiment Setup

We now show a few experimental results to give the user a feel for the cluster performance of STC with X-gram.

We first compare three different clustering algorithms (original STC, STC based on suffix tree with 3-gram, and STC based on suffix tree with X-gram) for space complexity, then time complexity, and clustering quality at last.

For this purpose we defined 10 queries, which are listed in Table 4. For every query we collect search result snippets form google [15], and feed the snippets to three different algorithms.

Table 4. 10 queries used for experiment

type	queries
ambiguous queries	apple, jaguar, java, matrix
entity names	data mining, information retrieval,
general terms	salsa, resume, music, yellow page

It should be noticed that in STC based on suffix tree with N-gram, N valued 2, 3, and 4 respectively, to evaluate this algorithm more comprehensive. If N valued 1, this algorithm is just like "bag of word", can not generate more readable labels for final clusters, or taking $O(n^2)$ time complexity to generate the frequent sets; and if N valued too big, this algorithm becomes to original STC, and will suffer from large memory requirement.

4.2 Space Complexity

The original STC algorithm can often construct a long path of suffix tree, and suffers from large memory requirements. The improved suffix tree with N-gram performs the same function but has lower memory requirements. Our approach has also significantly reduced memory requirements. We use the total number of nodes maintain by three different algorithms to compare the space complexity.

Table 5. The total number of words maintained by each algorithm

	total number of nods be maintained
Original STC	10767
STC with 4-gram	8178
STC with 3-gram	5765
STC with 2-gram	3315
STC with X-gram	**2803**

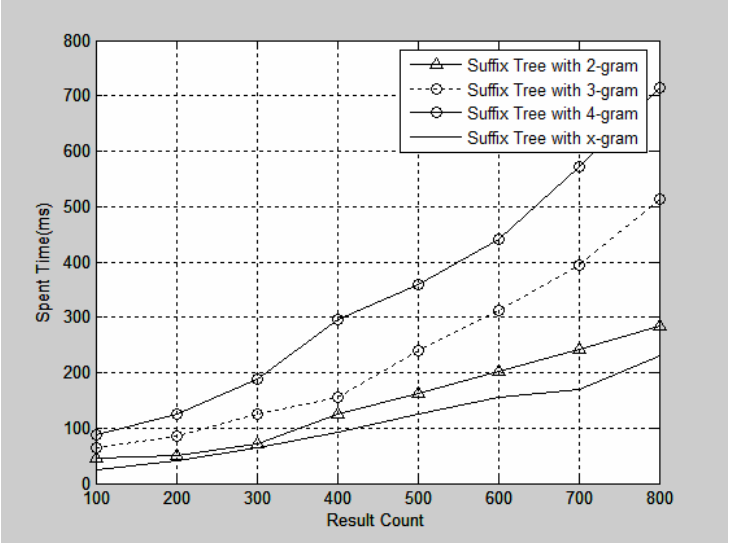


Fig. 8. Time complexity analysis

Table 6. Example cluster labels for "data mining" that is query word

A New STC	STC+N-gram	Original STC
Data Mining and Knowledge Discovery	Data Mining and Knowledge	Data Mining and Knowledge Discovery
Principles of Data Mining and Knowledge Discovery	Principles of Data Mining	Principles of Data Mining and Knowledge Discovery
data mining concepts and techniques	data mining concepts	data mining concepts and techniques
data mining with sql server 2005	data mining with sql	data mining with sql server 2005

For each query listed in Table 4, every algorithm collects first 200 search results form google and building a suffix tree. The total number of nodes maintained by three approaches is shown in table 5.

Table 7. Average precision of all clusters

query	STC with 3-gram	STC with X-gram	Original STC
apple	0.81	0.80	0.81
jaguar	0.89	0.88	0.87
java	077	0.76	0.80
Salsa	0.80	0.78	0.80
data mining	0.73	0.76	0.75
information retrieval	0.71	0.79	0.76
matrix	0.79	0.84	0.81
music	0.74	0.81	0.81
yellow page	0.71	0.77	0.77
Resume	0.73	0.81	0.80
AVG	**0.77**	**0.80**	**0.80**

The noisy word sequences feed to the suffix tree with X-gram are shorter than the other two types of suffix tree. So, with our approach the algorithm maintains fewer nodes.

4.3 Execution Time

In this section we measured the execution time of the various STC algorithms while clustering snippets collection of various sizes (100 to 1000 snippets, collected form google).

We select a query from table 4 for analyzing the time complexity of these three algorithms. Each reported time is averaged over 10 snippet sets. The results are shown in Figure 8, in which the X-axis stands for the number of results returned form original search engine, and the Y-axis is the time spent in the whole algorithm. It should be emphasized that the times values are not total processing time, excluding snippets downloading, parsing time.

In this experiment we search only for maximal frequent sets, not for all frequent sets. And we use a 3% threshold, start at 3 (100 * 3%) and end with 30 (1000 * 3%).

The result of the execution time is shown in Fgure 8.

It is plain that the time complexity of all three approaches are approximately linear, but our approach, based on suffix tree with X-gram, is the fastest. It is because suffix tree with X-gram maintains fewer words than that of suffix tree with N-gram or original STC algorithm, so fewer words are feed to the tree and fewer nodes to be pruned. And STC with X-gram can fed more true phrases to the tree wholly than that of with N-gram, so there are less partial phrases need to be joined. In this way suffix tree with X-gram can be faster than other two.

4.4 More Readable Description

Our cluster labels are true common phrases and more readable than conventional STC technique. As shown in Table 6, that is example of “data mining” is query words.

4.5 Clustering Precision

Due to lacks of standard dataset for testing web search result clustering, we have to build a small test dataset. For this purpose, we have defined a set of queries for which search results were collected from Dmoz.com [16]. The average of precision of the three approaches is shown in Table 7.

Form the Table we can see the average difference in precision performance using the STC+X-gram is litter better than using STC+N-gram.

5 Conclusion

STC based on suffix tree with X-gram significantly lower the memory requirements than the original STC and it generate more readable label than STC+N-gram, nearly the same precision as original STC, and still an incremental and a linear time algorithm, and faster than other types of STC algorithm.

We also proposed a complement operation to complement the suffix tree with X-gram, which filter out all the merely right-complete substrings of the true phrase, and can generate more reasonable candidate clusters.

So, our web result clustering algorithm based on the two novel approaches get better performance than conventional web result clustering algorithms so it is very suitable for online application.

References

1. Zamir, O., Etzioni, O.: Grouper: A Dynamic Clustering Interface to Web Search Results. In: Proceedings of the Eighth International World Wide Web Conference (WWW 8), Toronto, Canada (May 1999)
2. Zamir, O., Etzioni, O.: Web Document Clustering: A Feasibility Demonstration. In: Proceedings of the 19th International ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR 1998), pp. 46–54 (1998)
3. Osinski, S., Weiss, D.: A Concept-Driven Algorithm for Clustering Search Results. IEEE Intelligent Systems 20(3), 48–54 (2005)
4. Weiss, D.: Carrot2: Design of a Flexible and Efficient Web Information Retrieval Framework. In: Szczepaniak, P.S., Kacprzyk, J., Niewiadomski, A. (eds.) AWIC 2005. LNCS (LNAI), vol. 3528, pp. 439–444. Springer, Heidelberg (2005)
5. Weiss, D., Stefanowski, J.: Web search results clustering in Polish: Experimental evaluation of Carrot. In: Proceedings of the New Trends in Intelligent Information Processing and Web Mining Conference, Zakopane, Poland (2003)
6. Osinski, S., Weiss, D.: Conceptual Clustering Using Lingo Algorithm: Evaluation on Open Directory Project Data. Institute of Computing Science, Poznan University of Technology (2004)
7. Osinski, S., Stefanowski, J., Weiss, D.: Lingo: Search Results Clustering Algorithm Based on Singular Value Decomposition. Institute of Computing Science, Poznan University of Technology (2003)

8. Dell, Z., Yisheng, D.: Semantic, Hierarchical, Online Clustering of Web Search Results. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, Springer, Heidelberg (2004)
9. Hua-Jun, Z., et al.: Learning to Cluster Web Search Results. In: SIGIR 2004, Peking University (2004)
10. Janruang, J., Kreesuradej, W.: A New Web Search Result Clustering based on True Common Phrase Label Discovery. In: Computational Intelligence for Modeling, International Conference on Computational for Modeling Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, p. 242 (November 2006)
11. Dong, Z.: Towards Web Information Clustering, doctoral dissertation. Southeast Univ., Nanjing (2002)
12. Chang, C.H., Lui, S.C.: IEPAD: Information Extraction based on Pattern Discovery. In: Proceedings of the tenth International Conference on World Wide Web, Hong Kong, May 2-6 (2001)
13. Ukkonen, E.: On-line construction of suffix trees. Algorithmic 14(3), 249–260 (1995)
14. Zamir, O.: Clustering Web Document: A Phrase-Based Method for Grouping Search Engine Results. Doctoral Dissertation, University of Washington (1999)
15. Google search engine (2008), `http://www.google.com`
16. Open Directory Project (2008), `http://dmoz.org`