Managing the Google Web 1T 5-gram Data Set

Aminul ISLAM
Department of Computer Science, SITE
University of Ottawa
Ottawa, ON, Canada
mdislam@site.uottawa.ca

Diana INKPEN
Department of Computer Science, SITE
University of Ottawa
Ottawa, ON, Canada
diana@site.uottawa.ca

Abstract:

This paper describes how the Google Web 1T 5-gram data set, contributed by Google Inc., can be stored so that it can be used efficiently with respect to time. We present an efficient way of accessing all the 5-grams for a specific word of interest from the stored files. We measure the maximum access and processing efficiency achievable for any word of interest. We also compare results (access time and memory requirements) on the task of accessing all the 5-grams for a list of words, on both the processed and the original organization of the data set.

Keywords:

Google web 1T; n-gram; 5-grams

1. Introduction

The Google Web 1T data set [1], contributed by Google Inc., contains English word n-grams (from unigrams to 5-grams) and their observed frequency counts calculated over 1 trillion words from web page text collected by Google in January 2006[1]. It is expected that this data will be useful for a variety of Research and Development projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, as well as for other uses. It contains so much data that many machines with average amounts of memory are unable to even load it. We propose a method for splitting the enormous list of 5-grams, released in 118 large files, down to a more manageable size based on user requirements.

We use the 5-grams from the Web 1T corpus such that the middle token is the term and the two tokens on either side form the context. Several researchers tried to find out the advantages of this context definition by initiating the practical question of what minimum value of window size would, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word [2]. A well-known early experiment by Kaplan [3] attempted to answer this question at least in part, by presenting ambiguous words in their original context and in a variant context providing one or two words on either side to seven translators. Kaplan [3] observed that sense resolution given two words on either side of the word was not significantly better or worse than when given the entire sentence [4]. The same phenomenon has been reported by several researchers: e.g., [5] on Russian, and [6] on French. Whenever we talk about window size, we consider the word of interest in the middle of the window and context words on either side of the word of interest. But we can not use the Web 1T 5-grams as a window of 5 as all the 5-gram files (total 118 files) are sorted based on the first word in the 5-grams and then the second word in the 5-grams and so on.

In this article, we wish to find out what maximum n-gram access and processing efficiency can be achieved for any word of interest. Searching in the original Web 1T dataset is dependent on the total number of n-grams of all the words, including the word of interest. Our goal is to make this search independent of the total number of n-grams of all the words; this ensures the maximum access and processing efficiency achievable for any word of interest.

Two of our primary assumptions (used in Section 2) are that we need to store less than or equal to n (for our experiment, $n = 100,000$) 5-grams in each single file and we want to read only a single file to process all the 5-grams for a specific word of interest. In Section 3, we omit the first assumption and add a new assumption.

This paper is organized as follow: Section 2 shows how the Web 1T 5-gram data set can be used as a balanced window and how this data can be stored, accessed or processed efficiently with respect to time and memory space and we present experimental results. In Section 3, we find out what maximum n-gram access and processing efficiency can be achieved. We summarize contributions in Section 4.

2. Processing the Web 1T 5-grams

As the Web 1T 5-grams are sorted based on the first words in the 5-grams, we need to search the middle words in 118 large files of 5-grams. That is, to find the context words of a single word, we need to search all the 5-grams (1,176,470,663), which is not time efficient at all, especially when we are in need of finding context words of a specific word of interest on the fly. So, we process all the Web 1T 5-grams as described in the next sections, in order to make the procedure of finding context words time efficient.

We replace all the times, dates, numbers and the words that start with special characters or numbers with blank spaces in all the 5-grams. If the middle word of a 5-gram is eliminated then we eliminate the whole 5-gram. We also transform all the 5-grams into lower case; this creates many duplicate 5-grams (5-grams having the same words). We

[1]Details of the Google Web 1T can be found at www.ldc.upenn.edu/Catalog/docs/LDC2006T13/readme.txt

978-1-4244-4538-7/09/$25.00

Algorithm 1: calculating unique two characters and their frequencies using 5-grams

```
input : input files                    /* 118 re-sorted files */
output: output file                    /* contains unique two
        characters and their frequencies */
for each sorted file do
    for each 5-gram do
        word_1 ← first word of the 5-gram
        str ← SubStr(word_1, 2)        /* returns first two
        characters of word_1 */
        if str ∈ list then /* list contains unique two
        characters (i.e., str) and their
        frequencies (i.e., list.str) */
            increment list.str
        else
            list.str ← 1               /* initialize */
        end
    end
end
for each str ∈ list do
    write str and list.str to output file
end
```

Algorithm 2: calculating unique $p+1$ characters and their frequencies using 5-grams

```
input : p, input files and current nextInFile /* input
        files are 118 re-sorted files (p ≥ 2) */
output: output file /* output file contains unique
        p+1 characters and their frequencies */
listFN ← nextInFile      /* listFN contains all the
tokens in nextInFile */
for each sorted file do
    for each 5-gram do
        word_1 ← first word of the 5-gram
        if SubStr(word_1, p) ∈ listFN then
            str ← SubStr(word_1, p+1)
            if list.str > 0 then       /* str is in list */
                increment list.str
            else
                list.str ← 1
            end
        end
    end
end
for each str ∈ list do
    write str and list.str to output file
end
```

merge all the duplicate 5-grams by adding their frequencies. In this way we managed to reduce the number of 5-grams from 1,176,470,663 to 356,611,338 and the file size of all the 118 files from 33.68 GB to 9.48 GB. Using the same technique, we managed to reduce the number of unigrams from 13,588,391 to 2,870,385. We re-sort all the 118 preprocessed 5-gram files based on the middle words and restore the sorted 5-gram files to place the middle words in the first position.

2.1 Calculating Unique Two Characters and Their Frequencies using 5-grams

We devise an algorithm (Algorithm 1) that generates all the unique two-character tokens/words (taking the first two characters from the first word in each 5-gram) and their frequencies from all the 5-grams of 118 re-sorted files.

Now we need to make a decision of what maximum number of 5-grams[2] (say N) we will store in a single file so that we can have efficient search results to find context words for a specific word of interest. Based on the value of N, we split the output file generated by Algorithm 1 into two files. One file (say *filesName*) contains only the words (not frequencies) having frequencies less than N. Another file (say *nextInFile*) also contains only the words (not frequencies) having frequencies greater than or equal to N.

2.2 Calculating Unique $p+1$ Characters and Their Frequencies using 5-grams

We execute Algorithm 2 with *nextInFile* and $p = 2$ (p denotes the number of characters). Algorithm 2 generates all the unique $(p+1)$-character tokens and their frequencies from those 5-grams of 118 re-sorted files having the first two characters of the first words in *nextInFile*.

We split the output file generated by Algorithm 2 and add the words having frequencies less than N into the file *filesName* and we store words having frequencies greater than or equal to N by overwriting the file *nextInFile*. We carry on executing Algorithm 2 with current *nextInFile* and incremented p (i.e., this time $p = 3$).

We follow the same steps of executing Algorithm 2 until *nextInFile* is empty. For the Web 1T 5-gram data set, we get an empty *nextInFile* when $p = 12$. Finally *filesName* contains all the file names that need to be created[3].

2.3 Creating Files and Assigning 5-grams to Files

We execute Algorithm 3 which practically creates all the files needed based on *filesName* entries and assigns each 5-gram to the appropriate file (the entries in *filesName* are the name of the files) traversing all 118 re-sorted 5-gram files.

2.4 Efficient Search in the Web 1T 5-grams

To find all the 5-grams for a specific word of interest, we need to search only in a single file among those 22,743 files created using Algorithm 3 in Section 2.3. Algorithm 4 is used to locate the specific file and then to access all the 5-grams for that word to further process as per requirement[4].

2.5 Comparing Efficiency in Time and Memory Space

We use [7] data set, a well known data set used to judge different natural language processing (NLP) tasks, to analyze the access time and main memory requirements on run time and to process a specific task[5]. Though there are 30

[2]For the Web 1T 5-gram we use $N = 100{,}000$ as the state-of-the-art computing device can process these numbers of 5-grams in less than a second.

[3]*filesName* contains 22,743 entries after running Algorithm 2 on the Web 1T 5-grams.

[4]We can insert user defined code in line 16 of Algorithm 4.

[5]As a specific task for processing, we access all the 5-grams of a word of interest and then from those 5-grams we find all the unique context/neighbor words and the pair frequencies. By pair we mean the word of interest and one context/neighbor word.

Table 1: Experimental Results on Miller and Charles Data Set

Word Number	Word	Time to access all the 5-grams (in seconds)			Time to access all the 5-grams & to find all the context words and the pair frequencies (in seconds)			Number of searched 5-grams		Number of target 5-grams	Memory required to run (in bytes)
		Dual Core Machine		Think Centre (TER 0.67)	Dual Core Machine		Think Centre (TER 0.67)	TER 0.67	TER 1		
		TER 0.67	TER 1		TER 0.67	TER 1					
1	car	1.66	0.07	3.39	4.35	3.65	5.5	272,920	272,532	272,532	455,583
2	automobile	0.28	0.01	0.58	0.42	0.18	0.69	47,395	11,863	11,863	57,993
3	gem	0.19	0.00	0.38	0.30	0.14	0.47	31,891	9,342	9,342	58,488
4	jewel	0.40	0.00	0.11	0.13	0.12	0.19	8,263	7,936	7,936	53,606
5	journey	0.13	0.01	0.27	0.33	0.27	0.45	23,996	19,508	19,508	93,099
6	voyage	0.37	0.00	0.76	0.46	0.07	0.80	60,527	4,684	4,684	37,016
7	boy	0.40	0.02	0.81	1.08	0.93	1.42	67,943	67,391	67,391	209,030
8	lad	0.01	0.00	0.03	0.03	0.03	0.05	2,471	1,837	1,837	18,424
9	coast	0.51	0.02	1.08	1.12	0.82	1.55	86,362	59,030	59,030	170,044
10	shore	0.15	0.00	0.33	0.33	0.23	0.48	27,400	15,837	15,837	78,535
11	asylum	0.09	0.00	0.17	0.15	0.07	0.23	15,656	5,057	5,057	35,086
12	madhouse	0.01	0.00	0.03	0.02	0.00	0.03	2,123	179	179	3,057
13	magician	0.39	0.01	0.81	0.46	0.04	0.83	70,113	2,497	2,497	21,479
14	wizard	0.12	0.00	0.26	0.28	0.21	0.39	22,324	14,471	14,471	75,424
15	midday	0.54	0.00	1.11	0.61	0.01	1.11	91,534	992	992	10,069
16	noon	0.10	0.00	0.20	0.17	0.08	0.27	18,058	5,806	5,806	31,747
17	furnace	0.03	0.00	0.06	0.06	0.05	0.09	4,952	3,104	3,104	23,627
18	stove	0.05	0.00	0.11	0.10	0.07	0.14	8,340	4,747	4,747	28,568
19	food	0.97	0.05	2.05	2.68	2.28	3.44	171,215	170,079	170,079	305,285
20	fruit	0.32	0.01	0.66	0.63	0.39	0.91	57,135	28,403	28,403	108,827
21	bird	0.40	0.01	0.86	0.85	0.59	1.20	71,947	40,779	40,779	152,996
22	cock	0.39	0.02	0.77	0.97	0.83	1.22	60,695	60,424	60,424	92,293
23	tool	0.50	0.02	1.06	1.40	1.18	1.78	87,837	87,022	87,022	222,354
24	implement	0.14	0.01	0.28	0.41	0.34	0.53	26,026	26,022	26,022	82,762
25	brother	0.35	0.01	0.73	0.72	0.46	1.03	63,185	34,244	34,244	119,188
26	monk	0.13	0.00	0.28	0.19	0.06	0.33	24,507	3,630	3,630	32,736
27	oracle	0.12	0.01	0.27	0.32	0.27	0.44	21,346	19,624	19,624	84,541
28	rooster	0.05	0.00	0.09	0.08	0.03	0.11	8,580	1,924	1,924	18,374
29	cemetery	0.13	0.00	0.27	0.25	0.15	0.36	22,317	10,286	10,286	64,289
30	woodland	0.05	0.00	0.09	0.11	0.08	0.16	9,048	5,286	5,286	39,352
31	hill	0.44	0.02	0.91	1.22	1.06	1.56	74,961	73,998	73,998	251,043
32	slave	0.55	0.01	1.14	0.77	0.24	1.28	98,079	16,573	16,573	72,514
33	forest	0.42	0.01	0.88	0.95	0.69	1.31	73,224	48,829	48,829	174,405
34	graveyard	0.17	0.00	0.38	0.21	0.03	0.39	32,559	1,262	1,262	15,320
35	chord	0.14	0.00	0.28	0.19	0.05	0.30	24,158	3,384	3,384	23,465
36	smile	0.54	0.00	1.14	0.71	0.14	1.22	98,610	10,149	10,149	56,234
37	glass	0.49	0.03	1.03	1.36	1.15	1.72	84,160	83,244	83,244	206,989
38	string	0.47	0.02	0.98	1.16	0.90	1.56	87,937	66,204	66,204	236,570
39	crane	0.13	0.00	0.28	0.20	0.09	0.33	23,252	5,425	5,425	43,547

Algorithm 3: creating files and assigning 5-grams

```
input : input files and filesName   /* 118 re-sorted files */
output: output files /* the names of the files are the token names in filesName */
listFN ← filesName
for each sorted file do
    for each 5-gram do
        word_1 ← first word of the 5-gram
        len ← length(word_1)
        not_found ← TRUE
        while len > 1 AND not_found = TRUE do
            if SubStr(word_1, len) ∈ listFN then
                fname ← SubStr(word_1, len)
                not_found ← FALSE
                Open fname in append mode
                write 5-gram to fname
            else
                decrement len
            end
        end
    end
end
```

Algorithm 4: finding the specific file and all the 5-grams for a word

```
input : word            /* the word of interest */
output: fname /* contains all the 5-grams of word */
listFN ← filesName
len ← length(word)
not_found ← TRUE
while len > 1 AND not_found = TRUE do
    if SubStr(word, len) ∈ listFN then
        fname ← SubStr(word, len)
        not_found ← FALSE
    else
        decrement len
    end
end
Open fname in read mode
for each 5-gram in fname do
    word_1 ← first word of the 5-gram
    if word = word_1 then
        do as per requirement
    end
end
```

pairs of words in this data set, the number of distinct words are 39. We experimented on two different machines. The first machine is a dual core Intel® Xeon™ having CPU speed of 3.20 GHz and main memory of 4 GB (we refer it as Dual Core Machine). The second machine is an IBM ThinkCentre M51 machine with Intel® Pentium 4 processor having CPU speed of 3.40 GHz and main memory of 1GB (we refer it as ThinkCentre). Table 1 shows all the test results on [7] data set. Table 1 shows that 38 words out of 39 (97.4%) were accessed in less than a second. The total access time for all the 39 words on this machine is 12 seconds, an average of 0.31 seconds per word which is almost half compared to the ThinkCentre (0.64 seconds). The first word, *car*, takes longer time to be accessed or to be processed because it has more 5-grams than the other words.

Table 1 also shows a comparison between the number of 5-grams we search through (we call them searched 5-grams) versus the number of 5-grams we needed (we call them target 5-grams). The number of searched 5-grams and the number of target 5-grams are independent of the computing devices. The ratio between the number of target 5-grams and the number of searched 5-grams (we call it Time Efficiency Ratio (*TER*)) is a key issue for efficiency in terms of time. The higher the *TER* the better. Having the time efficiency ratio equal to 1 is an ideal case. The average time efficiency ratio for all the 39 words using the original 118 files is 0.001108 (this is the base case). The average time efficiency ratio for all the 39 words using 22,743 files is 0.63. To have an ideal average time efficiency ratio, we need to have a distinct 5-gram file for each unique word (unigram) possible. In Section 3, we study the feasibility and practicability of generating this enormous number of files.

3. Making the time efficiency ratio (TER) equal to 1

To have an ideal TER, we use a new assumption (instead of the first assumption used in Section 2) that we store only all the 5-grams of each unique word in a single file. We also use the second assumption used in Section 2.

3.1 Creating Files and Assigning 5-grams when TER=1

We execute Algorithm 5 which creates unique files to store all the 5-grams associated with each unique words and assigns each 5-gram to the appropriate file traversing all 118 re-sorted 5-gram files. Algorithm 5 generates 2,870,385 files from the 118 re-sorted files and all these files are stored in a subfolder sf ($sf \in \{\{a \cdots z\} \times \{0 \cdots 9, ', a \cdots z\}\}$) under the folder f ($f \in \{a \cdots z\}$). So, there will have $26 \times 37 = 962$ unique subfolders under 26 folders. For example, all the 5-grams of word *brother* are in subfolder *br* under folder *b* (the path of the file is $/home/\cdots/b/br/brother$).

Algorithm 5: creating unique files to store all the 5-grams associated with each unique words

```
input : input files          /* 118 re-sorted files */
output: output files  /* the names of the output files are the unique word names */
for each sorted file do
    for each 5-gram do
        word_1 ← first word of the 5-gram
        char_1 ← SubStr(word_1, 1)
        char_2 ← SubStr(word_1, 2)
        fname ← /home/···/char_1/char_2/word_1
        Open fname in append mode
        write 5-gram to fname
    end
end
```

3.2 Efficient Search when TER = 1

To find all the 5-grams for a specific word of interest, we need to access[6] only a single file among those 2,870,385 files,

[6]Actually we do not need to search as all the 5-grams in the

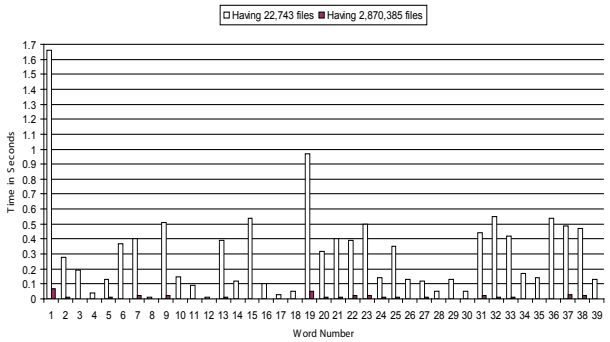

Figure 1: Comparison between time (in seconds) needed to access all the 5-grams of the 39 words on the Dual Core Machine having 22,743 files versus having 2,870,385 files.

distributed under 962 different subfolders, created using Algorithm 5 in Section 3.1. Algorithm 6 is used to locate the specific file that has all the 5-grams for a specific word and then to access all the 5-grams for that word to further process as per requirement[7].

Algorithm 6: finding the specific file and all the 5-grams for a specific word

```
   input  : word                    /* word of interest */
   output: fname /* contains the 5-grams of word */
1  char1 ← SubStr(word, 1)
2  char2 ← SubStr(word, 2)
3  fname ← /home/···/char1/char2/word
4  Open fname in read mode
5  for each 5-gram in fname do
6      do as per requirement
7  end
```

3.3 Comparing Efficiency when TER = 1

We use the same [7] data set on the Dual Core Machine. Table 1 shows all the test results on [7] data set when TER = 1. Table 1 shows that all the 39 words (100%) were accessed in less than 0.1 seconds using 2,870,385 files. The total access time for all the 39 words on the Dual Core Machine is 0.37 seconds, an average of 0.0095 seconds per word (i.e., 96.95% less average access time compared to using 22,743 files). Figure 1 shows a comparison between the time (in seconds) needed to access all the 5-grams of the 39 words on the Dual Core Machine using the 22,743 files from Section 2 versus using 2,870,385 files[8]. The single largest file with respect to the memory size is *new* having 1,310,730 5-grams and using 32,338,060 bytes (32.34 MB) of memory when we use 2,870,385 files. So, the maximum main memory required for any word is 32.34 MB. It takes 0.39 seconds on the Dual Core Machine to access all the 5-grams and 17.12 seconds[9] to access all the 5-grams and then to find all the context words and the pair frequencies for the word *new*. We can consider these values as the upper limits. Table 2 shows the actual size of the files and the size required on disk for different number of files[10].

file are target 5-grams.

[7]We can insert code for what we want to do with these 5-grams in line 6 of Algorithm 6.

[8]There are 20 words in Figure 1 having 0.00 second access time as the Benchmark module of Perl returns only 2 digits after decimal point.

[9]We need 95.44% less time to only access all the 5-grams and 7.06% less time to access all the 5-grams and then to find all the context words and the pair frequencies for the word *new* compared to using 22,743 files.

Table 2: File Size on Disk

No. of files	No. of folders	Actual size (in bytes)	Size on disk (in bytes)
118	N/A	9,483,356,102	9,483,418,624
22,743	N/A	9,483,356,102	9,495,010,304
2,870,385	988	6,865,896,655	9,514,983,424

4. Conclusions

It is expected that Google Inc. will release a larger data set than the Web 1T in the future and our proposed method is general enough to handle a larger data set than the Web 1T. Assuming that the rate of increase of the number of n-grams is much smaller than the rate of increase of the number of words (collected from web pages) over which their observed frequency are counted, our proposed approach is easily applicable to the future Web 10T, Web 100T or even Web 1Q (Q=Quadrillion). We find that if the computing device supports millions of files, then creating unique files to store all the 5-grams associated with each unique word provides the best access time efficiency and processing efficiency. Otherwise, the method described in Section 2 provides a reasonable access time efficiency and processing efficiency. It is interesting to note that though we only experimented on the 5-gram data set, the proposed methods are also applicable to Web 1T bigrams, trigrams, and 4-grams.

References

[1] T. Brants and A. Franz, "Web 1T 5-gram corpus version 1.1.," tech. rep., Google Research, 2006.

[2] W. Weaver, *Translation*. 1949. Reprinted in Locke, William N. and Booth, A. Donald (1955) (Eds.), Machine translation of languages, John Wiley & Sons, New York, 15-23.

[3] A. Kaplan, *An experimental study of ambiguity and context*. November 1950. Published as Kaplan, Abraham (1955), An experimental study of ambiguity and context, Mechanical Translation, 2(2), 39-46.

[4] N. Ide and J. Véronis, "Word sense disambiguation: The state of the art," *Computational Linguistics*, vol. 24, no. 1, pp. 1–41, 1998.

[5] A. K. Koutsoudas and R. Korfhage, "M.T. and the problem of multiple meaning," *Mechanical Translation*, vol. 2, no. 2, pp. 46–51, 1956.

[6] Y. Choueka and S. Lusignan, "Disambiguation by short contexts," *Computers and the Humanities*, vol. 19, pp. 147–158, 1985.

[7] G. A. Miller and W. G. Charles, "Contextual correlates of semantic similarity," *Language and Cognitive Processes*, vol. 6, no. 1, pp. 1–28, 1991.

[10]The actual file size using 2,870,385 files is less than the other two versions because we do not need to store the word of interest as the file name itself represents it.