

DConvertv2.0 Working

Nitin Jeevan Motgi (nmotgi@cs.ucf.edu)

May 7, 2001

1 About this Report

This article gives working methodology of DConvertv2.0.

2 Modification History

- May 2nd 2001 : Draft created.

3 Feedback

Please send your comment, suggestion, and criticisms to the following email address : amar,nmotgi,fauzia@cs.ucf.edu

4 Copyright Information

These files are copyright(c) of M5 Research Group. It may be reproduced and distributed in whole or in part, in any medium physical or electronic, as long as this copyright notice is retained in all copies. All translations, derivative works, or aggregate works incorporating this document must be covered under this copyright notice. That is you may not produce a derivative work and impose additional restrictions on its distribution.

5 Disclaimer

No liability for the content of this document can be accepted. Use the concepts, examples and other content at your own risk. As this is a new edition of this document, there may be errors and inaccuracies, that may be of-course be damaging to your system. Proceed with caution, and although this is highly, the author(s) do not take any responsibility for that.

All copyrights are held by their respective owners, unless specially noted otherwise. Use of a term in this documents should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installations and backup at regular intervals.

6 DConvertv2.0

DConvertv2.0 is a tool used to transform English words in input file to LIPT transformed words in output file. The tool is available at website <http://vlsi.cs.ucf.edu/dconvertv2.0/>. The documentation and GNU licence code is also available at the site.

6.1 About DConvert2.0

LIPT and STAR pre-process the text to enhance the back-end compression algorithm capabilities. The process of converting a text to a more compressible form is called “transformation”. The transformation process requires two main file namely an English dictionary, which consist of around 59,991 words and corresponding transform dictionary. For every word in English dictionary there is corresponding transformed word in transform dictionary. There is one-to-one mapping from English to transform dictionary. Currently, we are creating transform dictionary offline, but we hope to incorporate online transform generation. The whole process of transformation involves “searching” a word in input file with the word in english dictionary and replacing the word with it’s transformed version in output file. If the word from the input file is not present in the dictionary it is passed “UNCHANGED” to the output file. E.g. suppose “compress” is word that is tokenized from the input file. Now, we have to find this word in the english dictionary and replace this by a corresponding transformed word (e.g. *abaaX) in the output file. Now suppose we have word “XYZ” tokenized from the input file, this word is not present in dictionary so it will go unchanged to the output file.

In both the cases considered, the cost of searching a word everything for it’s presence in English dictionary is very expensive (CPU time). In order to search dictionary using Binary Search Algorithm, which has time complexity of $O(\log n)$, we need to sort the dictionary lexicographically. If we use quick sort algorithm to do this which has time complexity $O(n \log n)$, we have total complexity of the process as $O(n \log n) + M * O(\log n)$ where M is number of words tokenized from the input file (irrespective of whether it is present in the dictionary or not, for large files $M \ll n$) and ‘n’ is number of words in dictionary. So, as M gets larger the above expression gets large too, which degrades the performance of the transformation when there are large files. **Therefore to improve the preprocessing step we carefully considered organisation of dictionary and access times in DConvertv2.0 a tool.**

6.2 Description of DConvertv2.0

This tool exploits the static nature of dictionary and the property of the word to be searched to expedite the process of word searching. We organize the dictionary into two levels. In level 1 we classify it based on the length of the word and in second level (this is in conjunction with LIPT organisation of transformed dictionary) we classify it based on first character of the word. **By classifying in two levels we break the huge dictionary into smaller dictionaries of relatively smaller sizes. By this we confine our search domain to only small set of similar words which can expedite the process.**

Whole set of 'n' words in dictionary is divided into small dictionaries $d_1, d_2, d_3, \dots, d_{max}$ each having $s_1, s_2, s_3, \dots, s_{max}$ words in it. Each block of small dictionary having s_i words is also characterized by two parameters

- 1. Length of the word (WL) and
- 2. by starting character of the word (FC). For e.g. d_1 dictionary contains s_1 words of length 2 and starting with 'a'. d_2 dictionary contains s_2 words of length 2 and starting with 'b'.

So, now we divided the whole dictionary of size 'n' into small individual dictionaries of size $s_1, s_2, s_3, \dots, s_{max}$, (Where $n = \sum_{i=1}^{max} (s_i)$).

Each of the dictionaries $d_1, d_2, d_3, \dots, d_{max}$ characterized by $(WL, FC)_i$ are sorted lexicographically using Quicksort. This process of sorting is one time process, once sorted and stored it need not be sorted every time when it is loaded into the memory. It is subjected to resorting only when there is modification to the dictionary like adding or deleting words from dictionary. In order to search a word 'W' of length 'n' and starting character as 'c', the search domain is only defined to a small block of words which have length 'n' and start with 'c' So this makes searching faster as compared to the previous method.

There are two tables that are used to support faster access to part of dictionary:

6.2.1 Length Index Information (LII)

This table consists of pointers to different Word Index Information (WII) based on length of the word in the dictionary. The length of the input word is used as 'key' into this table. This table contains pointers to each WII corresponding on length of the word. **If there are no words of length 'n', the table contains a null pointer.**

6.2.2 Word Index Information (WII)

This is basically a list of pointers pointing to lists consisting of words starting with a,b,cz. All the words defined by WII are of same length. Only difference lies in the first character of the word. Hence, for each Pointer in LII, we have 26 Pointers in WII, which ultimately point to words based on size and starting character of the word.

6.3 Working of DConvertv2.0

The tokeniser (generates tokens), generates for each word length information and also provide starting character of the word. This information is used as index into two level tables. Using length of the input word as an Index into LII we get the Word Index Information Pointer. Which points to the block of words that are of same length as the search word. Then using first character as an index into the second table we get in block of words that are of same length as search word and also start with the same character as search word. Once we get to the appropriate block of words we use binary search to find the required word in it. In case the word is not present we make less of comparison than in the previous method. If the word is not in dictionary, then it's a non-profitable search. So by this organisation the cost involved in non-profitable search will be reduced. There are different points at which you can make decisions on whether the character is present in the dictionary or no to reduce the cost of unsuccessful search.

- If the length of search word used as an index into LII does not have an entry for WII PTR (WII_PTR = null) in LII, which means there is no word of this length in the dictionary and
- When first character is used as an Index in WII and of there is no PTR to block of words of same length and words starting with same character that means "there is no word of this length and having starting character as that of the search word"

Use of DConvertv2.0 has reduced the number of comparisons that need to be made by making use of definite organisation of dictionary. DConvertv2.0 completely loads the dictionary into memory, hence has memory overhead of 1.5Kbytes for storing pointers to blocks. The results of improvement is available on website.