A Modified Burrows-Wheeler Transform for Highly Scalable Example-Based Translation

Ralf D. Brown

Carnegie Mellon University Language Technologies Institute
5000 Forbes Avenue
Pittsburgh, PA 15213-3890 USA
ralf+@cs.cmu.edu

Abstract. The Burrows-Wheeler Transform (BWT) was originally developed for data compression, but can also be applied to indexing text. In this paper, an adaptation of the BWT to word-based indexing of the training corpus for an example-based machine translation (EBMT) system is presented. The adapted BWT embeds the necessary information to retrieve matched training instances without requiring any additional space and can be instantiated in a compressed form which reduces disk space and memory requirements by about 40% while still remaining searchable without decompression.
Both the speed advantage from $O(log\ N)$ lookups compared to the $O(N)$ lookups in the inverted-file index which had previously been used and the structure of the index itself act as enablers for additional capabilities and run-time speed. Because the BWT groups all instances of any n-gram together, it can be used to quickly enumerate the most-frequent n-grams, for which translations can be precomputed and stored, resulting in an order-of-magnitude speedup at run time.

1 Introduction

A key component of any example-based or case-based machine translation system is a good index of the training instances. For shallow EBMT systems such as Gaijin [1], Brown's Generalized EBMT [2], EDGAR [3], or Cicekli and Güvenir's Generalized EBMT [4], this index is formed from either the original training text's source-language half, or some lightly-transformed version thereof. This paper addresses such a textual index and discusses how it impacts not only the speed of the system, but can act as an enabler for additional capabilities which may improve the quality of translations.

Until recently, our EBMT system used an index based on an inverted file – a listing, for each distinct word (type) of all its occurences (tokens) in the corpus. While such an index has several nice properties, including fast incremental updates that permit on-line training with additional examples, it scales poorly. Lookups not only take time linear in the amount of training text, they also require $O(N)$ additional working memory, as maximal matches are built by finding adjacent instances of a pair of words and then extended one word at a time. Although the code had been heavily optimized (including a five-instruction hand-coded inner scanning loop), the system's overall performance was not adequate for interactive applications with corpora exceeding about five million words.

The new index format which was selected as a replacement for the existing code is based on the Burrows-Wheeler Transform, a transformation of textual data first described in the context of data compression a decade ago [5], and now the underlying algorithm in many of the best-performing compression programs such as `bzip2` [6]. The following sections give a brief overview of the BWT, how it was adapted for use in Carnegie Mellon University's EBMT system, the performance improvements that resulted from replacing the index, and new capabilities enabled by the BWT-based index.

2 The Burrows-Wheeler Transform

The Burrows-Wheeler Transform is a block-sorting transformation which groups elements of its input lexically.

The BWT operates as follows:

1. Take the input text T (of length N) and make it row 0 of an N-by$N$ matrix M.
2. For $i = 1$ to $N - 1$, form row i of M by rotating T left by i places as in Figure 1.
3. Sort the rows of M lexically, remembering where each row of the sorted result originated, to form M' (Figure 2).
4. Due to the manner in which M was constructed, columns 1 through $N - 1$ of M' can be reconstructed from column 0 by providing a pointer from each row to the row which had been immediately below it prior to sorting. This vector of successor pointers is conventionally called V.
5. Since the first column of M' now consists entirely of runs of equal elements in lexicographic order (at most one run per member of the alphabet Σ), one can discard column 0 and represent it by an array C of size $|\Sigma|$ which contains the first row in M' containing each member of the alphabet.

In practice, neither M nor M' are ever explicitly constructed. Instead, an auxiliary array of pointers into T is sorted, with $M_{i,j}$ determined by retrieving $T_{(i+j) \ mod \ N}$. T is usually a sequence of bytes, but for EBMT, a sequence of 32-bit word IDs was used.

Together, C and V are the output of the Burrows-Wheeler Transform. These two arrays lend themselves to highly-effective data compression because the elements of C are monotonically increasing, as are the elements of V within each range C_i to $C_{i+1} - 1$ (see Figure 3. The transform is also reversible by starting at the position of V to which row 0 of M was sorted and following the successor links until all elements of V have been visited; this is the final step in the decompression phase of BWT-based compression algorithms. For this example,

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	m
s	s	i	s	s	i	p	p	i	$	m	i
s	i	s	s	i	p	p	i	$	m	i	s
i	s	s	i	p	p	i	$	m	i	s	s
s	s	i	p	p	i	$	m	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
i	p	p	i	$	m	i	s	s	i	s	s
p	p	i	$	m	i	s	s	i	s	s	i
p	i	$	m	i	s	s	i	s	s	i	p
i	$	m	i	s	s	i	s	s	i	p	p
$	m	i	s	s	i	s	s	i	p	p	i

Fig. 1. Constructing the rotated matrix

i	p	p	i	$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	$	m
i	$	m	i	s	s	i	s	s	i	p	p
m	i	s	s	i	s	s	i	p	p	i	$
p	i	$	m	i	s	s	i	s	s	i	p
p	p	i	$	m	i	s	s	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	$	m	i	s
s	s	i	p	p	i	$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	$	m	i
$	m	i	s	s	i	s	s	i	p	p	i

Fig. 2. The sorted matrix M'

$M_{i,0}$	i	i	i	i	m	p	p	s	s	s	s	$
V_i	6	9	10	11	2	3	5	0	1	7	8	4

Fig. 3. The array of successors

starting at V_4, following the pointers to V_2, V_{10}, V_8, etc. will reconstruct the original text.

The output of the BWT also lends itself to language modeling, as all occurrences of any given n-gram are adjacent in the V array – thus one can count the number of occurrences simply by subtracting the position of the first occurrence from the position of the last. Further, all $n+1$-grams beginning with the given n-gram have their occurrences as a sub-range of the range for the n-gram. For example, in Figure 2, the instances of unigram 's' occupy rows 7 through 10, while bigrams 'si' occupy rows 7 and 8 and trigram 'sis' occupies row 8. The ends of the subrange for an $n+1$-gram can be located using two binary searches within the larger range. This leads to $O(k\ log\ N)$ lookups for the number of occurrences of any particular k-gram.

To deal with ambiguous lookups, the search is split as necessary. One common operation in our EBMT system is generalization via equivalence classes (e.g. days of the week, colors, or various types of proper names). When a member of an equivalence class is encountered in the source-language text while indexing the training corpus, it is replaced by the class marker if a known translation listed in the equivalence class occurs in the target-language half of the training instance, and left unchanged otherwise. When translating, the system should match the training example in either case, so both the original word and the corresponding class marker(s)[1] form an ambiguous lookup at that position in the input.

At every word position in the input, an extension of each currently-active partial match by each alternative term is attempted; those extensions that result in a non-zero number of occurrences become active matches for the next word position. Further, a new match is started for each alternative term at that position which actually occurs in the training data, and the unextended active matches are added to the set of all phrasal matches to be processed further in the retrieval and alignment phases of translation. While this splitting of BWT matches on ambiguity could theoretically result in an exponential increase in memory and run time, in practice most alternatives peter out very quickly – usually, the bulk of the corpus matches that are found are two or three words in length. Slowdowns from ambiguous lookups have not been an issue.

3 Modifications for Use in Indexing

Burrows-Wheeler-transformed data is very good at showing relative positions, but in order to determine an absolute position in the original text without completely reconstructing it, some auxiliary information is required. Neither the C nor the V array provide any information about the *location* of an n-gram in the original input, only its surrounding context.

A considerable number of approaches to anchoring a BWT-based index to the text it indexes have been proposed [7], but these typically involve additional time and/or space proportional to the alphabet size. While this is acceptable when the text is considered to be a stream of bytes ($|\Sigma| = 256$), the alphabet size for our word-based index quickly reaches into the hundreds of thousands. We thus decided to take advantage of the line-oriented nature of the example base.

Because the EBMT system has no interest in finding phrases which span the boundaries between training examples, an end-of-line or end-of-record (EOR) marker can be inserted after each example. By reserving an entire range of values as EOR markers, the record number can be encoded within the marker itself. Given that the 32-bit IDs we used to represent both words and successor pointers in the V array provide more than 4,200 million possibilities, 1/4 of

[1] A source-language word can belong to multiple equivalence classes provided the translations are disjoint.

the possible values were reserved as EOR markers. This sets the limits on the training corpus size at some 3,200 million words in over 1,000 million examples.

Further (precisely because the EBMT system does not operate on text spanning a sentence boundary, though it allows for overlapping partial translations to be merged in generating the final translation [8]), there is no need in this application to determine what text from the next training example follows an EOR marker, and it is thus possible omit the portion of V corresponding to EOR. As a result, the index is no larger than it would have been had the entire training corpus been treated as a single example. There is also no processing overhead on finding matching n-grams resulting from the EOR markers.

In addition to inserting the EOR markers, the order of words in each training instance is reversed before adding them to the index. This allows matches to be extended from left to right even though lookups in the index can only be efficiently extended from right to left. (While the input sentence could be processed from right to left for EBMT, we also wished to use the same code to compute conditional probabilities for language modeling.)

To retrieve the training examples containing n-grams matching the input to be translated, iterate over the range of V corresponding to the n-gram. For each instance of the n-gram, follow the pointers in V until reaching an EOR, then extract the record number from the EOR and retrieve the appropriate training example from the corpus. The offset of the matched phrase within the example is determined by simply counting the number of pointers which were followed before reaching an EOR. The original source-language text is not included in the stored example since it can be reconstructed from the index if required. We opted not to store a pointer to the start position of the line in the V array, since the only use our system has for the unmatched portion of the source text is for display to the user in error messages, and we can forego display of the entire source sentence in such cases. For applications which must be able to reconstruct the entire source sentence or access adjacent sentences, separately-stored start pointers *will* be required. Even without a line-start pointer, we are still able to reconstruct the word sequence from the beginning of a training example to the end of the matched portion when we need to display an error message; this is typically the most useful part of the sentence for the user, anyway.

One drawback of the BWT-based index compared to the old inverted file is that quick incremental updates are not possible, since the entire index needs to be re-written, rather than chaining new records onto the existing occurrence lists. Because on-line incremental updates in practice are performed only a small number of times before there is an opportunity for an off-line update, we have addressed the issue by using two indices – the large main index which was built off-line when the system was trained, and a much smaller index containing only the incremental updates. Although the entire auxiliary index must be re-written on each incremental update, this can be done quickly due to its small size.

4 Compressing the Index

For applications where space is more critical than time, the index may be stored in a compressed form which is around 40% smaller than the uncompressed form (38–46% for the corpora on which this feature was tested). Because the index comprises nearly half the total disk space consumed by the processed corpus, this results in a reduction of disk usage by nearly 20% at a cost of 20% greater CPU time. This capability has already been used to good effect on a training corpus which was slightly too large to fit completely in the machine's RAM – the reduced index size eliminated disk thrashing and thereby actually resulted in faster translations.

As mentioned above, the V array consists of runs of monotonically increasing values, typically with fairly small differences between adjacent values. Thus, we opted to represent the compressed V array as an array of bytes encoding either the difference from the previous element or an index into an auxiliary table of full 32-bit values.

Out of the 256 possible 8-bit values, values 1 through 191 are used to encode the actual difference from the previous entry and values 192 through 255 to encode an index within a bucket of full addresses. Value 0 was reserved to encode EOR in applications such as language modeling which do not need to encode record numbers. The V array is split into buckets of 64 entries and a 32-bit pointer into a pool of full addresses is allocated for each bucket. Thus the value 192 represents the address pointed at by the current bucket's pool pointer, the value 193 represents the address following that one, etc. The best-case compression with this scheme would store 64 entries of V using just 68 bytes, while the worst case is 324 bytes for those same 64 entries (8.5 and 40.5 bits per entry, respectively). On average, this method achieves a size of 18–19 bits per entry and an overall size for C plus V of 20–21 bits per word indexed.

The representation just described allows random access to any element of the index without decompressing any portion of the index. One simply retrieves the ith byte of the V array and examines it. If the value corresponds to a pointer to the auxiliary array of absolute addresses, that value is retrieved; if not, scan left and accumulate the difference values until an absolute address is reached, then return the sum of that address and the accumulated difference. The number of bytes which need to be processed is typically quite small, and is bounded by deliberately forcing at least one element of each 64-entry bucket to be an absolute address (increasing the best-case compressed size to 9 bits per entry, but having negligible effect on real-world average compression). Without such deliberate breaks in the runs of difference values, common trigrams in the corpus would cause runs of difference 1 equal in length to the frequency of the trigram.

5 Pre-Computing Phrasal Translations

One of the interesting new capabilities enabled by the BWT index is the straightforward enumeration – directly from the index – of all distinct n-grams of a given length k together with their frequencies in time $O(kN)$ (more sophisticated approaches can no doubt reduce this time bound) using $O(k)$ space. Thus, we decided to precompute the most common phrasal translations in order to speed up the overall translation process. Doing so required very little code – 65 lines of C++ to enumerate all the n-grams and 150 lines more to determine the translations of the most frequent n-grams and add them back to the indexed corpus.

Precomputing the candidate translations for a common phrase improves translation speed because it becomes unnecessary to retrieve and align a large number of matches to determine translation probabilities. Instead, only a single instance of each of the best-scoring translations is retrieved and the stored frequency count is extracted from that instance.

6 Performance and Scalability

Performance was evaluated on three different corpora. The first corpus is a 100,000-sentence pair subset of the IBM Hansard corpus [9] for French-English, coupled with a 77,000-translation pair part-of-speech-tagged dictionary and some 500 context-free rewrite rules for generalization (about 2.2 million words of English). The second corpus consists of the Hindi-English parallel text collected during the June 2003 DARPA TIDES Surprise Language Exercise, less some held out sentences for testing (about 2.1 million words of Hindi). The third corpus, for Arabic-English, consists of UN proceedings and text from the Ummah newspaper (about 81 million words of Arabic).

Indexing speed when generalization is disabled is approximately equal for the old and new index formats. When generalization is enabled, lookup and retrieval of exact matches among the generalization rules account for the majority of the runtime in the old system. Thus, for the French-English test case, indexing time is cut from 58 minutes to less than 10 due to the faster lookup during generalization.

Table 1 clearly shows the $O(N)$ performance of the old index and the much better scaling of the new BWT-based index for the Hindi and Arabic test sets. The times listed are the amount of time required to locate all matching n-grams in the input sentences, without retrieving any of the matches from the corpus. The French test set also illustrates that the nature of the text influences the lookup time, since the French training text is much closer to the test sentences than is the case for the other languages, and as a result there are many more matches – and, in particular, long matches – against the corpus.

Pre-computing the translations of all n-grams occurring at least 30 times in the Hindi training corpus of 2.1 million words doubles the indexing time from 2.5 to 5 minutes, increases the total size of the indexed corpus by about 10 percent and produces a considerable speed-up in translations (see Table 2). Similarly for the Arabic corpus, training time increases from 84 minutes to about 4 hours and disk usage increases by about 18 percent when precomputing all 2- through 12-grams occurring at least 30 times each. Without the precomputation, the EBMT engine attempts to perform a word-level alignment on up to 4000 instances

Corpus	Train (words)	Test (words)	Time (Invert File)	Time (BWT Index)
Hindi	2.1M	4460	1.8s	0.5s
Arabic	81M	5003	36.0s	0.7s
French	2.2M	4628	7.2s	0.7s

Table 1. Index Lookup Performance

Corpus	Train (words)	Test (words)	Precomputed None	≥ 30
Hindi	2.1M	4460	14s	1.8s
Arabic	81M	5003	145s	6.8s
French	2.2M	4628	158s	8.7s

Table 2. Effect of Precomputation on Translation Time

of each distinct matching phrase, stopping after 1500 successful alignments, in order to accumulate reliable translation probabilities. Naturally, only a small percentage of n-grams actually have more than a small fraction of the maximum instances to be checked, but as can be seen from the run times, these account for the bulk of the processing.

The combination of new $O(log\ N)$ index and precomputation yields overall EBMT run times less than that required merely for lookups in the old index format.

7 Speed as an Enabler

In addition to the newly-added capability of precomputing translations already mentioned, pure speed can itself be an enabler for additional capabilities and applications.

The Carnegie-Mellon EBMT engine has in the past been used as part of a speech-to-speech translation system [10, 11]. In such an interactive application, it is important for translations to be completed quickly, preferably in less than half a second. Similarly, for bulk translations of large corpora such as the Chinese portion of the TDT-3 corpus [12], a system taking tens of seconds per sentence (or, worse, several minutes per sentence) would simply not be practical.

Conversely, in applications where the previous system was fast enough, the time saved by better indexing and precomputed translation hypotheses can be applied to additional, more sophisticated processing. Or one can use a larger training corpus with the same processing as before to gain additional coverage without unacceptably slowing the translation process.

8 Future Work

We plan to use the pre-computed phrasal translations to help guide word-level alignment in cases where there may be too much ambiguity to perform the alignment based purely on a bilingual dicationary and the information in that particular sentence pair. Whenever a portion of the bitext mapping between the source and target language halves of a training instance is too sparse or too ambiguous to provide a unique alignment, referring to the pre-aligned phrases to restrict the possible alignments may allow the overall alignment to succeed.

9 Acknowledgements

This work was largely supported by the TIDES program under Navy research contract N66001-00-C-8007.

The author would like to thank the reviewers for their feedback, which resulted in numerous improvements to this paper.

10 References

References

1. Veale, T., Way, A.: Gaijin: A Template-Driven Bootstrapping Approach to Example-Based Machine Translation. In: Proceedings of the NeMNLP'97, New Methods in Natural Language Processessing, Sofia, Bulgaria (1997) http://-www.compapp.dcu.ie/~tonyv/papers/gaijin.html.
2. Brown, R.D.: Adding Linguistic Knowledge to a Lexical Example-Based Translation System. In: Proceedings of the Eighth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99), Chester, England (1999) 22–32 http://www.cs.cmu.edu/~ralf/papers.html.
3. Carl, M.: Inducing Translation Templates for Example-Based Machine Translation. In: Proceedings of the Seventh Machine Translation Summit (MT-Summit VII). (1999) 250–258
4. Cicekli, I., Guvenir, H.A.: Learning Translation Templates from Bilingual Translation Examples. Applied Intelligence **15** (2001) 57–76 http://www.cs.bilkent.-edu.tr/~ilyas/pubs.html.
5. Burrows, M., Wheeler, D.: A Block-Sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Equipment Corporation (1994)
6. Seward, J.: The bzip2 and libbzip2 Home Page (1997) http://www.bzip2.com.
7. Ferragina, P., Manzini, G.: An Experimental Study of an Opportunistic Index. In: ACM-SIAM Symposium on Discrete Algorithms. (2001) 269–278 http://-citeseer.ist.psu.edu/ferragina01experimental.html.
8. Brown, R.D., Hutchinson, R., Bennett, P.N., Carbonell, J.G., Jansen, P.: Reducing Boundary Friction Using Translation-Fragment Overlap. In: Proceedings of the Ninth Machine Translation Summit. (2003) 24–31 http://www.cs.cmu.-edu/~ralf/papers.html.
9. Linguistic Data Consortium: Hansard Corpus of Parallel English and French. Linguistic Data Consortium (1997) http://www.ldc.upenn.edu/.

10. Frederking, R., Rudnicky, A., Hogan, C.: Interactive Speech Translation in the DIPLOMAT Project. In Krauwer, S., et al., eds.: Spoken Language Translation: Proceedings of a Workshop, Madrid, Spain, Association of Computational Linguistics and Eurpoean Network in Language and Speech (1997) 61–66
11. Black, A.W., Brown, R.D., Frederking, R., Singh, R., Moody, J., Steinbrecher, E.: TONGUES: Rapid Development of a Speech-to-Speech Translation System. In: Proceedings of HLT-2002: Second International Conference on Human Language Technology Research. (2002) 183–189 `http://www.cs.cmu.edu/~ralf/papers.html`.
12. Graff, D., Cieri, C., Strassel, S., Martey, N.: The TDT-3 Text and Speech Corpus (1999) `http://www.ldc.upenn.edu/Papers/TDT1999/tdt3corpus.ps`.
13. Bentley, J., Sedgewick, R.: Fast algorithms for sorting and searching strings. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms). (1997) `http://www.cs.princeton.edu/~rs/strings/`.