Experiments on Block Indexing

Outi Lehtinen, Erkki Sutinen, Jorma Tarhio

Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland.
E-mail:{olehtine,sutinen,tarhio}@cs.helsinki.fi
The work was supported by the Academy of Finland.

Abstract. We present a block indexing scheme based on q-grams for personal data archives. The scheme marks for each continuous or gapped q-gram u those text blocks where u occurs. The index is applied in a similar way as in the widely-used Glimpse system to searching for given keywords. Experiments show that our approach is a compromise between the size of the index and the speed of retrieval. Our scheme produces small indexes especially for archives containing texts of a highly inflected language or of many languages.

1 Introduction

Managing a personal data archive, consisting of several megabytes of text, is getting a more and more important task. The task has a close connection with various methods applied for CD-ROM indexing and retrieval [2]. In practice, an efficient information retrieval mechanism is the most crucial factor for a successful data management.

Manber and Wu [5] present a tool called *Glimpse*, which is a search engine working at the file system level. Glimpse allows also approximate searches, i.e. it is able to find approximate matches of the given keyword. Glimpse applies *block indexing*. First, the file system to be indexed is divided into at most 256 blocks. Second, all words are included in the index; an index entry for a word is a profile indicating the blocks where the word occurs. Third, to locate the occurrences, Glimpse applies *agrep* [9], a general-purpose search program, in two phases: first, agrep finds the index words which contain the approximate matches of the keyword; second, it checks the corresponding blocks, according to the profile of the found index words. In a typical case, the index consumes not more than 2–4% space of the size of the text, according to Manber and Wu [5].

In highly inflected languages, like Finnish, a word can have even thousands of different forms. For such languages the word indexing scheme of Glimpse produces much larger indexes — typically over 20% of the text — than for English. This is because each occurring form of a word is stored separately in the index. For example, a Finnish word 'yhdistelmättömyydestänsäkään', meaning 'even though of its uncombinational character', is formed adding suffixes '-lmä', '-tön', '-yys', '-stä', '-nsä', and '-kään' to root 'yhdiste', which means 'a combination'.

The heterogeneousness of data is another reason for larger indexes. Suppose, for example, that there are files in different languages which is a common situation in a non-English speaking country. Then, each additional language increases substantially the size of the word index, because of its distinctive vocabulary.

For the reasons stated above, there are cases when the two-level approach of Glimpse can benefit from refinements in its indexing mechanism. We have experimented with different indexing schemes and also enlargened the generalization by Baeza-Yates [1], which has been applied, for instance, by Barbosa and Ziviani [3].

In the generalization of Baeza-Yates [1], the index stores all the *q-grams* [8], i.e. strings of length q, of the text. An entry of the index, corresponding to q-gram d, points to the blocks containing an occurrence of d. Depending on the situation, the index can be restricted to q-grams starting a word. In addition to length q, the other parameter of the approach is block size b of the indexed text.

Barbosa and Ziviani [3] combine the suffix array to the two-level approach of Manber & Wu and Baeza-Yates. The novelty of their contribution [3] lies in its distributed search mechanism: the suffix array resides on disk, and each entry of the index of sequences of length k has only one pointer to the suffix array.

In this paper, we introduce a variation of Glimpse, called *Grampse* for GRAM based Pattern SEarch. Instead of indexing words, Grampse indexes q-grams. For example, a phrase 'lämmin mämmi', meaning a Finnish Easter dish, consists of *continuous* 4-grams 'lämm', 'ämmi', ..., 'mämm'. In addition, it includes several *gapped* 4-grams [6], like 'ämnm'. Parameters length q and gapwidth δ fix the q-grams to be included in the index. For example, the index of text T = 'This paper...' for $q = 3$ and $\delta = 1$ would include 1-gapped 3-grams 'Ti ', 'hsp' and so forth.[1] The reason for using gapped q-grams is obvious: they include more information of a word than continuous q-grams, which more or less reflect the language than a specific word, because of the strong correlation between the subsequent characters.

The q-gram index is used for searching for a keyword P as follows. First, the algorithm constructs set S of δ-gapped q-grams of P. Then, depending on variation v of the algorithm, it looks up the blocks of text containing an occurrence of each of the q-grams in subset $v(S)$. For example, if v is the 'most-infrequent' heuristic, $v(S)$ is defined as the most infrequent q-gram of S in the text. The found blocks are checked using agrep.

Besides applying q-grams, the implementation of Grampse is based on two other ideas. First, to cut down the q-gram index size or to be able to apply a larger value for q, we reduce the alphabet by encoding a set of characters to one code. This clustering of characters is based on their frequencies. Second, the algorithm can adjust the width of a block. The details are described in Section 2.

We analyze the method in Section 3, using the i.i.d. model where the characters of the text and the pattern are independently and identically distributed. For a natural language, the assumption is, however, not appropriate. There-

[1] Pevzner and Waterman [6] use the term gapsize which is gapwidth plus one.

fore, we have tested the efficiency of Grampse in several experiments, which are reported in Section 4.

It turns out that Grampse falls in between agrep and Glimpse, as far as search speed is concerned. The main benefit of Grampse, compared to Glimpse, is its lighter space consumption, making it a promising choice also for personal data archives of size of order 20 MB.

2 Implementation

In this section, we will introduce the basic ideas behind the implementation of Grampse.

2.1 Definitions

Let the text be $T = T[1 \ldots n]$ and the pattern $P = P[1 \ldots m]$. The characters of T and P belong to alphabet Σ of size c. By a q-gram of another string we mean a string of length q which can be either *continuous* or *gapped*. String $u = a_1 a_{2+\delta} a_{3+2\delta} \ldots a_{q'}$ is called a *δ-gapped q-gram*, denoted *(q,δ)-gram*, of string v, if $w = a_1 a_2 \ldots a_{q'}$ is a continuous q'-gram of v where $q' = (q-1)(\delta+1)+1$. In other words, (q,δ)-gram u is formed by taking every $(\delta+1)$th character of w. Note that a continuous q-gram is a $(q,0)$-gram. When there is no danger of confusion, a q-gram denotes a $(q,0)$-gram.

2.2 The q-Gram Index

In order to index the q-grams of text T, we divide T into N blocks of length $b = n/N$. At the implementation level, one has to take care of an overlap of $(q-1)(\delta+1)$ characters between subsequent blocks. Then, the (q,δ)-grams of T are scanned. For each q-gram u of text T, the algorithm builds a list of blocks where u occurs. The list can be implemented either as an array or as a bit vector.

2.3 Search Heuristics

Let set S consist of all the (q,δ)-grams of pattern P using the same length q and gapwidth δ as in the index. The search itself applies a subset of S, and there are several heuristics of selecting this subset. Let $v(S)$ be the subset of S according to heuristic v.

The search consists of two phases. First, Grampse marks the blocks containing every (q,δ)-gram of $v(S)$. Then, it checks the marked blocks. The heuristics described below differ from each other mainly in the balance of the execution times of the two phases.

- The 'total' heuristic selects the entire set and looks for blocks with an occurrence of each (q,δ)-gram in S. The advantage of restricting the number of blocks with the AND relation is shadowed by the time consumed for marking the blocks.

- The 'non-overlapping' heuristic selects only consecutive, non-overlapping (q, δ)-grams of the pattern. Compared to the 'total' heuristic, it marks more blocks, but faster.

- The 'most-infrequent' heuristic selects only one (q, δ)-gram: among all pattern (q, δ)-grams, it is the most infrequent in the text. The heuristic results in a fast mark phase.

2.4 Clustering the Alphabet

One way to reduce the size of the index is to cluster the alphabet. The basic method is similar to the Huffman compression [4]. Assume that the wanted size of the alphabet is c'. Two of the most infrequent characters are identified, with a frequency equal to the sum of their original frequencies. This procedure is repeated until the size of the clustered alphabet is c'.

The Huffman clustering described above produces an almost uniform distribution for the target alphabet. We consider also skew distributions. Let p, $0 < p < 1$, denote the skewness of the distribution. Let us consider a simple way to build a *p-skew* alphabet of c' characters. Let $f_i, i = 1, \ldots, c'$ be the relative frequency of the ith character in the skew distribution. Then $f_i = p(1-p)^{i-1}$, when $i < c'$, and $f_{c'} = 1 - \sum_{i=1}^{c'-1} f_i$.

Clustering the alphabet makes it possible to use larger values for q.

3 Analysis

We will analyze the efficiency of Grampse in a so called i.i.d. model, where the characters of T and P are distributed identically and independently. Let c denote the size of the alphabet and let b denote the length of a block. In an i.i.d. model, the probability p for an occurrence of any (q, δ)-gram is constant, i.e. $p = 1/c^q$.

Let the searched (q, δ)-gram be u. We denote by $E(u)$ the expected number of occurrences of u in a block. According to Régnier and Szpankowski [7][2],

$$E(u) = \frac{1}{c^q} \cdot b.$$

The minimal requirement for an efficient indexing is that the expected number of the searched (q, δ)-gram is less than 1. Since $E(u)$ remains the same for each u, we denote $E = E(u)$. We get the efficiency condition: $E < 1$, which leads to the inequality:

$$b < c^q.$$

To get a flavor of how to choose b, consider case $c = 30$ and $q = 3$. Using the previous formula, we get $b < 27.000$. Assuming that the number of blocks

[2] The formula for $E(u)$ is not trivial to derive since the probabilities for an occurrence of u in subsequent positions of a block are not independent of each other.

equals to 256, this leads to a maximum applicable text size of $27.000 \cdot 256$, which is about 6.9 MB. If $q = 4$, Grampse can handle a text archive of size 207 MB.

The problem of using the i.i.d. model in the analysis comes from its unsuitability for natural language text. In a natural language, only a fraction of all possible (q, δ)-grams are used. In addition, the distribution is far from uniform. For example, a q-gram 'hhh' is extremely rare in an English text; at least, it is much more infrequent than 'the'. However, the idea of using gapped q-grams increases the number of (q, δ)-grams and, at the same time, makes the distribution more balanced.

4 Experimental Results

Our test data consisted of about 7.6 MB of Finnish text. Searched patterns were either Finnish words or their prefixes or phrases. For each pattern length we searched for 50 different patterns and calculated the average search times. All experiments were executed on a Linux workstation with 32 MB RAM and a 133 MHz Pentium processor.

The implementation of Grampse is based on the Glimpse version 3.0 which was also used for comparison in all the tests. The implementation language is C.

Our experiments consisted of three parts. We examined the effect of block size and alphabet clustering to the size of an index. We tested how effectively Grampse filtrates blocks to be checked. And last we compared search times for Glimpse and Grampse, using both words and phrases as patterns.

4.1 Index Size

Figure 1 shows how the block size affected the index size when the block list was implemented as an array. When the block size got smaller the index grew, because the number of blocks increased.

We made experiments with two kinds of alphabet clustering: an almost uniform distribution and a skew distribution. Table 1 shows the effect of the clustering methods on the index size. In this test the target alphabet had 10 characters and q had values 3 and 4. Clustering the alphabet to 10 characters so that the distribution became almost uniform made the index smaller than the unclustered q-gram index. This is because there are only at most 10^q different q-grams. The usage of a skew distribution resulted in considerably smaller indexes. In the case of the skew distribution there are only a few frequent characters and infrequent characters are very rare. In this case 4-gram indexes using the skew distribution were even smaller than the 3-gram index without clustering.

4.2 Filtration Efficiency

The block size is an important parameter of Grampse. Looking for (q, δ)-grams instead of words leads to checking extra blocks that may not contain accurate

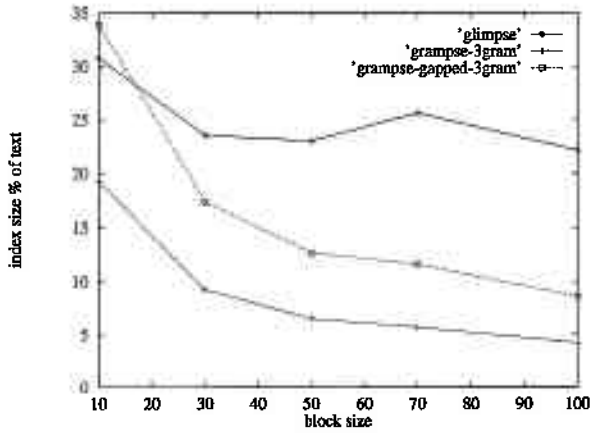

Fig. 1. Index sizes for different block sizes (kbytes). Grampse uses the value $q = 3$.

Table 1. Index sizes for different target alphabets, when the block size is 50 kB.

the index type	index size (kB)
Glimpse	1 764
3-gram	378
3-gram, $c = 10$, uniform	132
3-gram, $c = 10$, 0.4-skew	80
3-gram, $c = 10$, 0.5-skew	74
3-gram, $c = 10$, 0.6-skew	57
4-gram	965
4-gram, $c = 10$, uniform	796
4-gram, $c = 10$, 0.4-skew	368
4-gram, $c = 10$, 0.5-skew	303
4-gram, $c = 10$, 0.6-skew	202

matches of the query. When the block size was over 50 kB and $q = 3$, Grampse had to check almost every block. This happened because 3-grams are so common that almost all of them were included in every block. Table 2 shows that for short patterns Grampse had to check over 80% of all blocks when the block size was 50 kB. When the block size was cut down to 10 kB, the percentage of the checked blocks fell below 60%. Using the 'most-infrequent' heuristic or gapped q-grams the checking percentage became smaller when the pattern got longer, because then the probability to find a rare q-gram in the pattern increased.

Table 2. Percentage of the checked blocks using $(3, \delta)$-grams when the block size is 50 kB and 10 kB. Pattern length varies between 4 and 8. Used search heuristics: (a) total and non-overlapping, (b) most-infrequent, (c) total with gapped q-grams.

block size 50 kB				block size 10 kB			
m	(a)	(b)	(c)	m	(a)	(b)	(c)
4	85	80	-	4	52	58	-
5	84	83	99	5	62	58	99
6	82	78	89	6	59	49	68
7	81	65	77	7	57	31	41
8	81	74	72	8	58	46	37

Filtering with 3-grams does not seem to be very efficient for short patterns. Most of the 3-grams are very common in a Finnish text. Better results can be achieved by using 4-grams at the expense of larger indexes.

4.3 Search Times

We implemented a version of Grampse which uses hashing for retrieving continuous q-grams from the index. We saved hash values instead of q-grams in the index file. Searching was done by first reading the entire index into a hash table in memory and then getting the block profiles of the searched q-grams. With this approach we used the 'most-infrequent' heuristic, because it turned out to be the most efficient heuristic. Only the most infrequent q-gram of the pattern was chosen and the corresponding blocks were checked. Figure 2 presents search times of Grampse using hashing and a preloaded index. In this case the index size affected heavily search times. The search time was shorter for smaller indexes, because the time for loading the index was short.

Figure 3 shows search times of Grampse when the index is already in memory. Search times no longer depended on the size of an index. Grampse in a case of 4-grams and a 15-character alphabet was the slowest one in Figure 2, because the index was also the biggest. In Figure 3 it was the fastest, because the index contained more information than in other cases.

In Figure 4 the search times of Grampse with different clustering methods are compared using 4-grams of a 10-character alphabet. Search times for the both

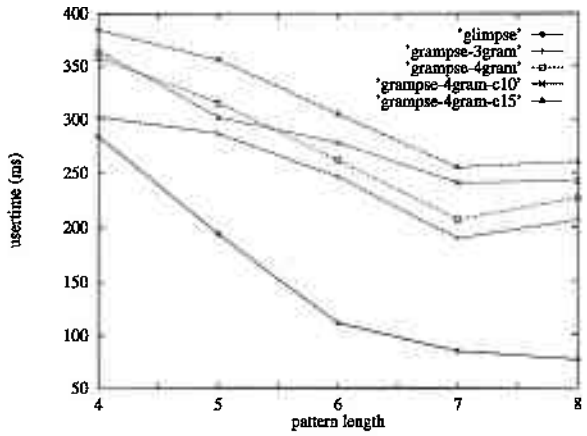

Fig. 2. Search times of Grampse (the index is not stored in the memory) and Glimpse. The block size is 50 kB.

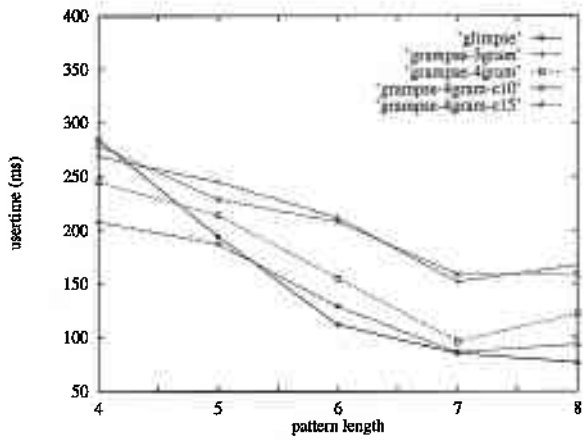

Fig. 3. Search times of Grampse (the index is stored in the memory) and Glimpse. The block size is 50 kB.

clustering methods were a little slower than the search time when no clustering was used. Using the uniform distribution appeared to be faster than using the skew distributions.

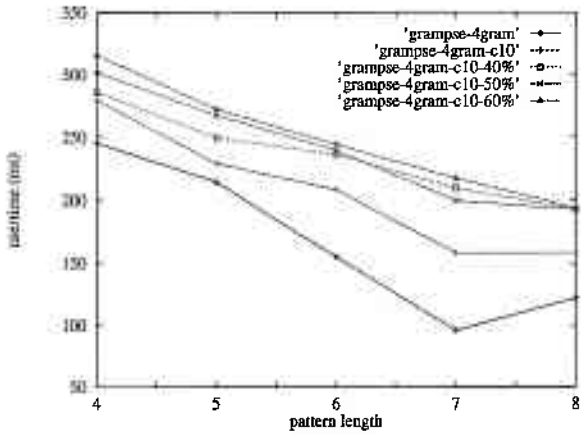

Fig. 4. The effect of different clustering methods on search times. The block size is 50 kB and $q = 4$.

Searching for phrases is slow with Glimpse especially when the searched phrase contains common words. Glimpse has to search for every word of a phrase separately and then combine their block profiles to get the answer. Common words are likely to appear in almost every block and that results in checking of extra blocks. Grampse chooses the most infrequent q-gram from the search phrase and checks only the blocks containing it. Though the words are common, the phrase may contain a rare q-gram, which helps to cut down the number of blocks.

Table 3 shows the search times of Glimpse and Grampse when the search phrases contained three words of which at least one was two characters long and one was three characters long. In this task Glimpse was considerably slower than Grampse. Glimpse needed on the average over a second to answer this kind of query when Grampse needed only about 0.2 seconds.

The block size has influence on the search times in Glimpse. In our experiments it became clear that the search gets faster when the block size decreases. Intuitively, the search gets more accurate and fewer blocks have to be checked. The same applies also to Grampse. The careful selection of the block size is important, because it affects to the filtration efficiency and that way also to search times.

Glimpse was faster than Grampse when searching for words or parts of word. But when we kept the index in the memory and used 4-grams with the 'most-infrequent' heuristic, the search times for Glimpse and Grampse were close to each other. Clustering the alphabet provides the possibility to choose a smaller

Table 3. Search times for phrases in milliseconds, when (a) the index is not stored in the memory, and (b) the index is stored in the memory. The block size is 50 kB.

the index type	(a)	(b)
glimpse	1110	-
3-gram	210	170
4-gram	240	140
4-gram, $c = 10$	230	150
4-gram, $c = 10$, 0.4-skew	220	180
4-gram, $c = 10$, 0.5-skew	210	170
4-gram, $c = 10$, 0.6-skew	210	180

index with a little longer search times. The best case for Grampse was searching for phrases containing common words, when it was clearly faster than Glimpse.

5 Concluding Remarks

We have presented a new tool, called Grampse, for searching a pattern in a personal file archive. The tool is a modification of the widely used Glimpse system, differing from it mostly in the index structure. The indexing is based on (q, δ)-grams and is thus a natural choice for highly inflected languages like Finnish. Therefore, we ran our tests with a Finnish text of 7.6 MB, exemplifying a personal file collection.

Compared to Glimpse, Grampse produced smaller indexes. While the index size of Glimpse was around 25% of the text, Grampse needed about 5%. Of course, the index size for Grampse depends on the block size: the longer the blocks, the smaller the index.

Since the indexing of Grampse is not based on words but (q, δ)-grams, it is possible to decrease the index size by clustering the alphabet. In this way we were able to get indexes smaller than 1% of the text.

Since the probability of an occurrence of a (q, δ)-gram in a block is typically larger than that of a word, long blocks encourage using Glimpse instead of Grampse. This fact encouraged us to experiment on shorter blocks. The decrease in the number of checked blocks was most obvious with the 'most-infrequent' heuristic and the 'total' heuristic with gapped q-grams. Thus, gapped q-grams improve the filtration of Grampse when compared to continuous q-grams. This is also a new feature of Grampse when compared to Baeza-Yates' generalization [1].

Although slower than Glimpse, Grampse searches faster than agrep; however, agrep is remarkably fast. The difference between Glimpse and Grampse is not significant if a hashed index of Grampse is stored in the memory. In addition, finding phrases with common words is clearly faster with Grampse than with Glimpse. Decreasing the size of the index with the clustering techniques had an

increasing but tolerable effect on the search time; the increase was larger with a skew distribution than with a uniform one.

Up till now, we have concentrated on reducing the number of blocks to be checked by agrep. Our preliminary experiments indicate that there is a combination of problem parameters n, m, q, δ, b (block size), and c' (size of the clustered alphabet) where Grampse is a balanced choice for personal information retrieval.

Glimpse uses agrep to find approximate matches of a keyword in the index. Though this approach could be applied also in Grampse, it is not practical. This is because the set of (q, δ)-grams of a searched keyword P and its approximate match P' are different. If the difference between P and P' is small enough, the intersection is, however, not empty. This observation can be employed when revising Grampse for approximate retrieval.

References

1. R. Baeza-Yates: Space-time trade-offs in text retrieval. In: *Proc. First South American Workshop on String Processing* (ed. R. Baeza-Yates and N. Ziviani), Universidade Federal de Minas Gerais, 1993, 15–21.

2. R. Baeza-Yates, E. Barbosa, and N. Ziviani: Hierarchies of indexes for text searching in read-only optical disks. In: *Proc. First South American Workshop on String Processing* (ed. R. Baeza-Yates and N. Ziviani), Universidade Federal de Minas Gerais, 1993, 25–41.

3. E. Barbosa and N. Ziviani: From partial to full inverted lists for text searching. In: *Proc. Second South American Workshop on String Processing* (ed. R. Baeza-Yates and U. Manber), Universidad de Chile, 1995, 1–10.

4. T. Bell, J. Cleary, and I. Witten: *Text Compression*, Prentice Hall, Englewood Cliffs, 1990.

5. U. Manber and S. Wu: GLIMPSE: A tool to search through entire file systems. Report TR 93-34, Department of Computer Science, University of Arizona, 1993.

6. P. Pevzner and M. Waterman: Multiple filtration and approximate pattern matching. *Algorithmica* **13** (1995), 135–154.

7. M. Régnier and W. Szpankowski: Frequency of pattern occurrences in a (DNA) sequence. Draft, 1995.

8. I. Witten, A. Moffat, and T. Bell: *Managing Gigabytes*, Van Nostrand Reinhold, New York, 1994.

9. S. Wu and U. Manber: Fast text searching allowing errors. *Communications of ACM* **35**, 10 (1992), 83–91.

This article was processed using the LaTeX 2_ε macro package with CUP_CS class