Efficient Text Searching for Read-Only Optical Disks

Ricardo Baeza-Yates

Departamento de Ciencias de la Computación
Universidad de Chile
Santiago, Chile

Eduardo Fernandes Barbosa

Departamento de Engenharia Eletrônica
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil

Nivio Ziviani

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil

Abstract

The objective of this paper is to present an efficient implementation of a recently known index for text databases presented in the literature, when the database is stored on read-only optical disks, known as CD-ROMs. The implementation is built on top of a new and simple index for texts called PAT array (also called suffix array). Considering the limitations of read-only optical disks, which are 10 to 20 times slower than magnetic disks, we propose an additional data structure and searching algorithm for PAT arrays that reduces the number of disk accesses and the distance traveled by the optical head.

KEY–WORDS: Text searching, PAT arrays, suffix arrays, read-only CLV optical disks, CD-ROM.

1 Introduction

The preponderant use of the computer for typesetting in the publishing industry and the continued decline in the cost of mass storage devices make textual database one of the fastest growing category of databases. For static databases it is worthwhile to preprocess the database and build an index to speed up the searching time. A new and simple type of index is the PAT array (Gonnet and Baeza-Yates, 1991). A PAT array is a compact representation of a digital tree called PAT tree, because it stores only the external nodes of the tree. A PAT tree is a Patricia tree (Morrison, 1968) built on all indexing points (called semi-infinite strings) of a text database. The PAT tree was originally described by Gonnet (1983) and later used in conjunction with the computerization of the Oxford English Dictionary (Gonnet, 1987).

The most important complexity measures for preprocessed text database are (i) the time required to build the index, (ii) the time required to search for a particular query and (iii) the extra space used by the index. Building a PAT array is similar to sorting variable length records, at a cost of $O(n \log n) \times time$, where n indicates the size of the text database, either characters or number of indexing points. Searching in a PAT array takes at most $2m \log_2 n$ character comparisons and at most $4 \log_2 n$ disk accesses, where m is the length of a given query. The extra space used by a PAT array is only one pointer per indexing point (Gonnet and Baeza-Yates, 1991).

The choice of optical media, specifically CD-ROM disks, as storage device for textual database is due to its large storage capacity (up to 600 megabytes), low cost to the end user (it can be as low as two dollars per unit for large scale production) and good physical and logical integrity.

The objective of this paper is to present an efficient implementation of PAT arrays for CD-ROM disks. As CD-ROM disks are 10 to 20 times slower than magnetic disks and searching in a PAT array can cost up to $4 \log_2 n$ disk accesses, a naive implementation of PAT arrays for CD-ROM disks might take too long to answer a query from a practical point of view. We propose an additional index and searching algorithm that improves searching time by reducing the number of disk accesses and the distance traveled by the optical head. We know that similar index structures have been used in some text search softwares, like *PAT* (Snider, 1993) and *Search City* (Ars Innovandi, 1992), but no publication about them are known so far.

In the next Section we briefly describe the CD-ROM medium and its retrieval performance as compared with magnetic disks. In Section 3 we present the structure of the PAT array and exemplify the indexing and retrieval operations for a very simple text database. In Section 4 we present an indexing method called *Short* PAT *array*. In Section 5 we analyze the method presented and show the feasible performance gain that can be achieved by using the Short PAT array model for retrieving text stored in CD-ROM disks.

2 CD-ROM Disks

The main differences between CD-ROM and magnetic disks are: (i) CD-ROM is a read-only medium and, as such, the structure of the information is static; (ii) the efficiency of data access is affected by its location on the disk and by the sequence in which it is retrieved and (iii) due to constant linear velocity (CLV) the tracks have variable capacity and rotational latency depends on disk position.

The CD-ROM track has a spiral pattern and an access to distant tracks requires more time due to the need of optical head displacement and changes in disk rotation. The capability of accessing nearby tracks with no displacement of the reading mechanism is called *span* and the number of tracks which can be accessed in this way is called *span size*. In actual CD-ROM drives span size is up to 60 tracks. The access of data located within span boundaries is efficient and seek time is reduced to about 1 millisecond per additional track. In this situation, seek time is negligible compared to rotational latency. When accessing data located outside span boundaries, seek time requires from 200 to 600 milliseconds due to optical head displacement, against approximately 30 milliseconds for a typical magnetic disk.

Indexed sequential structures can be efficiently implemented in CD-ROM, by considering the static nature of the information and the span capabilities of the reading device. The set of tracks covered by a span in a CD-ROM might be compared to the set of tracks belonging to a cylinder in a magnetic disk. We consider each set of tracks bounded by a span as an *optical cylinder*. Two differences between the two media are: the optical cylinders have different capacities (depending on disk position) and a subset of tracks might belong to two or more optical cylinders (span overlapping).

Experimental results on different file structures and file sizes show that rotational latency, file size, file structure, and file allocation strongly affects the retrieval performance in this medium. The influence of the skewed track capacity distribution and rotational latency variations leads to tradeoffs between the number of seeks and rotational delays which are the main components for the total retrieval costs in CD-ROM disks (Barbosa and Ziviani, 1992).

Although CD-ROM disks present many economical advantages, the high retrieval costs inherent to this technology reinforces the need for designing algorithms and data structures that compensates its low performance. As a comparative measure, let we consider a text database with $n = 50$ million index points (which corresponds to a 300 megabytes text file, with an index of approximately 200 megabytes). The number of disk accesses needed to perform the search in the worst case is $A_n = 4 \times \log\ n = 102$ accesses. If a magnetic medium is used, seek time (t_s) has an average value of 0.030 seconds. So, this search could be completed in (at most) $102 \times t_s = 3.06$ seconds. If the medium is a CD-ROM disk, where t_s has an average value of 0.60 seconds the same search would take about 102×0.60 seconds $= 61.20$ seconds, which is 20 times greater. This example shows that any improvement in the retrieval costs will result in great benefit to the end user.

Some common solutions for reducing disk accesses in CD-ROM applications are: (i) store the main index structure in main memory; (ii) use of proximal storage technique, by storing correlated data in near place on the disk and (iii) employ a redundancy technique, duplicating data on the disk region where they are more frequently required for retrieval.

When the Pat-Array model is associated to a large text data base, none of these solutions are of practical application. The reasons are: (i) if the main memory is used for full index storage, the main database size will be bounded by very narrow values of text size. In a Pat-Array structure, the index size is about 60% of text size (Gonnet et al., 1992); (ii) in a random search scheme, there is no deterministic way of finding out what is the correlated data when a random query is processed and (iii) duplication of data to obtain cost reduction in all queries is a difficult planning task and it clearly restrains the maximum text file size that could be stored on the disk.

3 PAT Arrays

The traditional model of text divides a textual database in a *set of documents*. A list of significant words, called *keywords*, are extracted from the text and assigned to each document. Consequently, queries are restricted to pre-selected keywords.

A new approach for textual databases sees the text as one long string (Gonnet and Baeza-Yates, 1991; Manber and Myers, 1990). Each position in the text is called a *semi-infinite string* (or *sistring* for short). A sistring is defined by a starting position and extends to the right as far as needed or to the end of the text. The database may be viewed as having an infinite number of null characters at its right end. Sistrings are compared lexicographically character by character, and so any two strings in different positions do not compare equal.

Depending on the character sequences that are likely to be sistrings for searching, the user must define which positions of the text will be indexed. An *index point* is a position in the text that must be indexed. In large databases, not all character sequences are indexed, just the alphabetical symbols that are preceded by a blank symbol, that is, beginning of words.

Figure 1 illustrates an example of a text database with nine index points. Each index point corresponds to the address of a sistring. Figure 2 shows the nine sistrings in lexicographical ascending order.

This	text	is	an	example	of	a	textual	database
↑	↑	↑	↑	↑	↑	↑	↑	↑
1	6	11	14	17	25	28	30	38

Figure 1: A simple text database with nine index points

28	a textual database
14	an example of a textual database
38	database
17	example of a textual database
11	is an example of a textual database
25	of a database
6	text is an example of a textual database
30	textual database
1	This text is an example of a textual database

Figure 2: Database ordered by sistrings

Any search structure that allows for range searches can be used to search on the set of sistrings. However, the most suitable structures are digital trees, in particular Patricia trees (Morrison, 1968). A Patricia tree built on all the sistrings of a text database was suggested by Gonnet (1983), and later on called PAT trees (Gonnet and Baeza-Yates, 1991).

In the PAT tree structure (1) the search is done over the tree while scanning the bits of the search string; (2) the whole set of sistrings answering a query is contained in a single subtree and hence the searching time is independent of the size of the answer. Every subtree of the PAT tree contains, by construction, all the sistrings with a given prefix. Prefix searching in a PAT tree ends when the prefix is exhausted. At that point all the answer is available in a single subtree.

PAT trees are very efficient because only $O(m)$ bit inspections are necessary to obtain the whole set of sistrings answering a query. However, the PAT tree needs at least two pointers for every internal node and one pointer to every external node, which means $12n$ bytes for n index points.

A simple, efficient and elegant solution to this problem is to store only the external nodes of the PAT tree. Thus, the whole index turns into a single array of external nodes ordered lexicographically by sistrings. This single array was called PAT *array* by Gonnet during the computerization of the Oxford English Dictionary. This idea was independently discovered by Manber and Myers (1990), where it is called *suffix array*.

Figure 3 shows the PAT array for the sample text database presented in the Figure 1. The size of the index is now reduced to $4n$ bytes, where n is the number of index points. This represents a significant economy in space at the cost of a factor of $\log_2 n$ (Gonnet, Baeza-Yates and Snider, 1992). In this case prefix searching can be implemented by doing two indirect binary searches over the PAT array with the results of the comparison being less than, equal to or greater than. To search for the prefix *tex* in the PAT array presented in Figure 3 we perform an indirect binary search over the array and obtain [7, 8] as answer (position 7 of the PAT array points to the sistring begining at the position 6 in the text, and position 8 of the PAT array points to the sistring begining at position 30 in the text). The size of the answer in this case is 2, which is the size of the array interval.

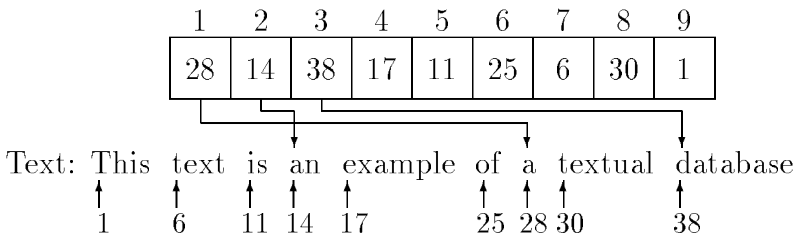

Figure 3: PAT array

4 The Short-Pat-Array Model

The solution we present consists in building one more index level to be stored in main memory during the searching phase, holding additional information about the PAT array and the data in the text file, so that the number of disk accesses is reduced. We call this index structure *Short-PAT-Array* or SPAT array for short. The SPAT array structure is a sorted array of strings where each string contains the first l_s characters of a sistring. The PAT array is considered as a set of equal size blocks: for each block we store in the SPAT array the first l_s characters of the sistring pointed to by the last entry in each PAT array block.

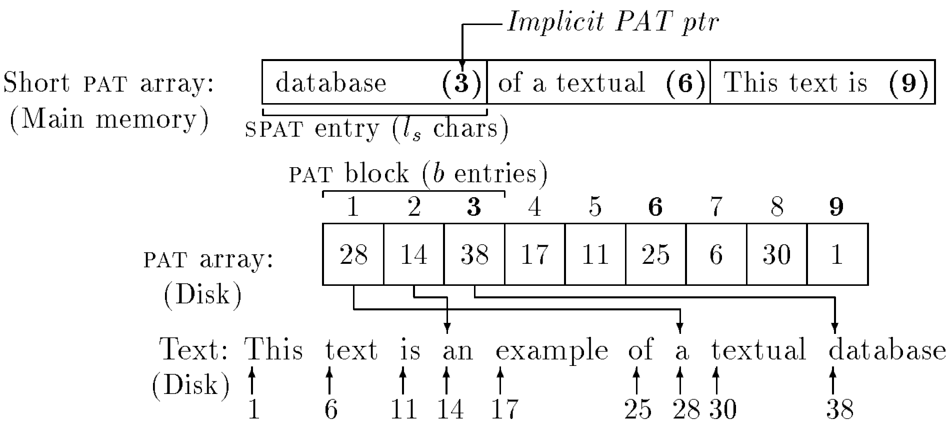

Figure 4: A simple model of the *Short* PAT *array* (SPAT)

Figure 4 illustrates a SPAT array index for the PAT array of Figure 3. We divide the PAT array in 3 blocks, each one with 3 entries. For each PAT array block we transfer to the SPAT array l_s characters from the sistring pointed by the last PAT array entry in this block ($l_s = 12$ in this example). If each entry in the SPAT array corresponds to a block in PAT array with b elements, then $SPAT[i]$ corresponds to the $PAT[b \times i]$ position in the PAT array. Therefore, the pointer from a SPAT array entry to a PAT array block is always an *implicit address*.

We now define certain measures:

1. Let $Msize$ indicate the main memory available to store the SPAT array index, in bytes.
2. Let l_s indicate the number of characters of a SPAT array entry (l_s is selected such that no SPAT entries are equal).
3. Let n indicate the number of index points in text.
4. Let r indicate the number of elements of the SPAT array ($r = Msize/l_s$).
5. Let b indicate the number of elements of each PAT array block. ($b = n/r$)
6. Let B_p indicate the size of each PAT array block, in bytes ($B_p = 4 \times b$).
7. Let g_i indicate the *index granularity* of the SPAT array ($g_i = 1/b = r/n$).

More formally, we can define this additional index as:

Definition 4.1: The SPAT array structure is a sorted array of r strings $[s_1, s_2, ..., s_r]$, where s_i ($1 \leq i \leq r$) is an entry formed by the first l_s characters from sistrings in the text database. Each SPAT array entry corresponds to a PAT array block with $b = n/r$ elements in which the i^{th} block has the b^{th} entry pointing to the sistring corresponding to the entry s_i in SPAT array index. The entries in SPAT array are sorted lexicographically so that $s_i < s_j$ for any $i < j$ and $1 \leq [i, j] \leq r$. The value of l_s is selected such as no two consecutive SPAT array entries are equals. The sorting in the lexicographical order of the entries in the SPAT array is performed upon the comparison of symbols α that forms each entry.

Definition 4.2: The *Index granularity* (g_i) of the SPAT array is a measure of PAT array representation in the main memory, given by: $g_i = 1/b = r/n$, and $1/n \leq g_i \leq 1$. Making $b = 1$ means that each element of the PAT is also in the SPAT array ($r = n$), and we have the maximum index granularity ($g_i = 1$). Conversely, if we make $b = n$, means than only 1 element of the main index is in the SPAT array and the PAT array file is treated as a single block ($r = 1$), and we have the minimum index granularity ($g_i = 1/n$). Due to practical limitations, the maximum granularity can only be achieved when the text and index file are very small and the complete PAT array index can be read into main memory to make $g_i = 1$.

It will become clear in the next section that the bigger the index granularity the better is the retrieval performance, since the index (and text) image in main memory becomes more representative as $g_i \rightarrow 1$. However, there are practical bounds for g_i, since it depends on $1/b$ (or $[n/(Msize/l_s)]^{-1}$). So, for a given value of text size (or n index points), g_i is bounded by $Msize$ and l_s values. If $Msize$ size is chosen for a given application then l_s is the factor which determines the maximum index granularity.

The steps of the algorithm to build the SPAT array structure from an existing PAT array index are:

```
1.  Define main memory size (Msize) to be used by SPAT array;
2.  Select the number of characters for each SPAT array entry (l_s);
3.  Compute the number of elements of SPAT array:  r = Msize/l_s;
4.  Divide the PAT array in r blocks of equal size:  B_p = Sizeof(PAT)/r;
5.  For i = 1 to r do
      For each i block of the PAT array
        Copy the first l_s characters of the sistring corresponding to the
        (b×i)^th entry of the PAT array to the i^th entry of the SPAT array;
```

The search is done in two phases, as follows:

```
1.  Read the SPAT array file to main memory; {Done only once}
    {First phase of search begins}
2.  Repeat Until {End of queries}
     2.1.  Read the search key;
     2.2.  Perform binary search in memory; {Cost ≈ 0}
     2.3.  If a match is found in the SPAT array then
           2.3.1.  Find left and right bounds in SPAT array ; {Cost ≈ 0}
                   {Second phase of search begins}
           2.3.2.  Transfer left and right blocks of PAT array
                   from disk to memory;
                   {Cost = 2 × t_s + 2 × t_x ,    t_x = transfer time}
           2.3.3.  With the left block do binary search between memory
                   and text file; {find exact left bound}
                   {Cost = (2 × log  b − 2) × t_s}
           2.3.4.  With the right block do binary search between memory
                   and text file; {find exact right bound}
                   {Cost = (2 × log  b − 2) × t_s}
                   { At this point we have the number of occurrences and
                   their locations on disk
           2.3.5.  Read the occurrences from text file as desired;
```

Ideally, the amount of main memory should be as large as possible, but this is not only an expensive solution for very large text files but also will be reflected in low disk space utilization, because the SPAT array file is first stored in some disk area, before being transferred to main memory. If we divide the space in memory ($Msize$) by the PAT array size ($n \times 4$) we have a parameter for memory overhead. As an example, if we allocate a 4 megabytes memory space to accommodate a SPAT array for a text with $n = 50$ million indexing points and each PAT array entry requires 4 bytes, we have an overhead of only $0.02 = 2.0\%$. This overhead is still of less significance if we compare the SPAT array index size with the text file size associated to it. Therefore, the space requirement for the SPAT array is very small if compared with the PAT array index requirement.

Performance improvement is achieved with this additional structure due to the fact that: (i) a first search phase to find the left and right bounds in PAT blocks is performed in main memory with negligible cost; (ii) the second search phase is performed over a much smaller index structure, since it will be done in a small fraction of the PAT array index (a small PAT block); and (iii) the exact left and right bounds of occurrences in the text file is performed with half of the number of seeks. This is because the small PAT array block found during the first phase is transferred to main memory and the exact left and right bounds in text are found through a binary search between *memory and disk* and not between a *disk region* (PAT array) and *another disk region* (text file).

5 Analysis of the SPAT Model

5.1 Number of Blocks to Qualify an Answer

A hit is said to occur in the SPAT array when the result of the comparison between the search key and the SPAT array entry is *less than* or *equal to*. When a search is done with the SPAT array model, we may have as result: (i) there is no hit of the search key with the SPAT array entries and, consequently, there is no need to access the disk to certify this or (ii) one or more hits occurs and we have to access the disk to verify the occurrence(s) and find out the exact (left and right) bounds of it. We shall now demonstrate that we never need to read (and transfer to main memory) more than 2 PAT array blocks, independently of the size of the answer.

We now define certain complexity measures:

1. Let q indicate the search key, of length l_q.

2. Let s_{pa} indicate an entry in the SPAT array, of length l_s.

3. Let N_{Bp} indicate the number of PAT array blocks to be read during the searching phase.

4. Let A indicate the size of the answer obtained in the searching phase.

Suppose that we are given a query string q of length l_q. If $l_q \geq l_s$, then the first phase of search will return only one entry of the SPAT array structure. The binary search compares the first $Min(l_q, l_s)$ symbols of the SPAT array entry (s_{pa}) with the query string (q). In this case, the whole length of the string s_{pa} will be compared with q, since $l_s \leq l_q$. When a hit occurs, it can only be the last comparison between q and s_{pa} that results in *less than* or *equal to* a given entry in the SPAT array. That is, the search ends when we find the last SPAT array entry that satisfies $q \leq s_{pa}$. If there where others entries before or beyond this point satisfying the condition for ending the search, then we would be violating the definition of the SPAT array. Note that *all* the symbols of the strings in the SPAT array are used for the comparison. This fact reinforces the conclusion that there cannot be more than one SPAT array entry containing the (possible) answer(s), because $s_i < s_j$ for any $i < j$, $1 \leq [i, j] \leq r$. Then, we conclude that, in this case, $N_{Bp} < 2$ blocks and the number of occurrences is $A \leq b$ answers.

Now assume that we are given a query string q of length l_q, with $l_q < l_s$. Because the binary search compares the first $Min(l_q, l_s)$ symbols of strings s_{p_a} and q, only the prefix symbols of the string s_{pa} will be compared with q. Suppose that a hit occurs in a given SPAT array entry, $s_{pa} = \alpha_1\alpha_2, ..., \alpha_{l_q}, ..., \alpha_{l_s}$, in which the first l_q symbols are equal to the symbols of q. The last $(l_s - l_q)$ symbols are obviously different and produce *don't care* matches. While the strings in the next entries beyond this point have the same prefix, others hits will occur and more than one PAT block may contain the answers to the query. Due to the rule for string formation and the lexicographical ordering of SPAT array entries, it will become clear that we do not need to read the text to verify if the intermediate blocks between the first and the last hits have the prefix of the given query. The fact that the last $(l_s - l_q)$ symbols differ in the first hit does not restrain the existence of other different strings with prefix equal to q in the next k entries in the SPAT array $(1 < k < r)$. Then, in this case, $N_{Bp} \leq 2$, and we conclude that we never have to read more than 2 blocks in the PAT array.

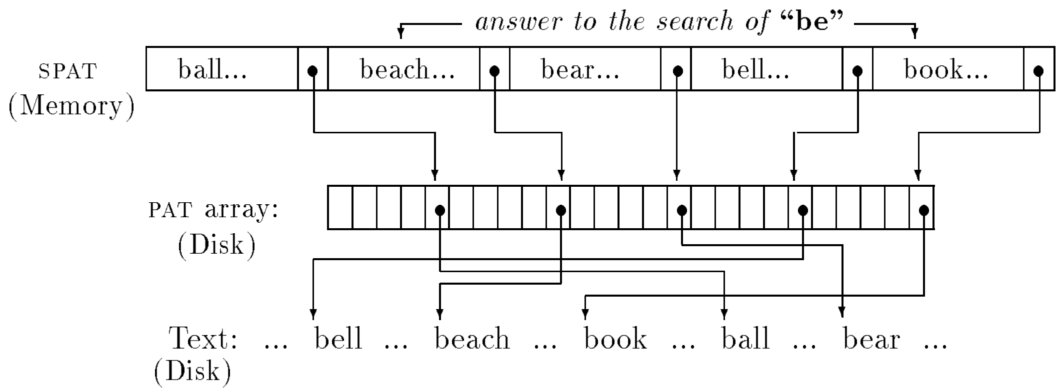

Figure 5: Example of answers bounded by one or more PAT blocks

Figure 5.1 illustrates the above conclusions: suppose that we are searching for the occurrence of $q =$ "**be**", in the SPAT array shown in this figure. By comparing the first two symbols of the s_{pa} strings with "**be**", we found that the answer must be between the blocks of PAT pointed by "**be**ach" and "**bo**ok". The last entry that ended the search contains the entry "**bo**ok...". The PAT block corresponding to that entry may contain entries starting with "bet...", "bit...", "blue...", etc. Similarly, the first block that produced a hit contains the entry "beach..." and we may find in the corresponding PAT block prefixes such as "bank...", "bar...", etc. The exact bounds of the answer are then found in some point in the middle of the first and the last PAT array blocks that produced matches. Clearly, in that case, $A \geq b$.

To illustrate the situation in which only one hit occurs, suppose that we are now searching for the string $q =$"**best...**" in the SPAT array structure of Figure 5.1. Clearly, this occurrence(s) (if exists) can only be in the block pointed by "**book...**" in Figure 5.1.

5.2 Cost Analysis for the SPAT array Index

In this section we give a general cost expression for the worst case search using the SPAT array model. The following definitions will be used in the SPAT array cost analysis:

1. Let t_s indicate the seek time of the CD-ROM reading device, in seconds ($0.4 \leq t_s \leq 0.8$).
2. $L_s = 2048$ bytes (Length of one CD-ROM sector).
3. Let k indicate the number of sectors transferred to main memory ($k = \frac{B_p}{L_s} = \frac{4b}{L_s}$).
4. Let b_{tt} indicate the block transfer time ($b_{tt} = 0.01333$ seconds for one CD-ROM sector).
5. Let t_x indicate the transfer time from disk to main memory, in seconds ($t_x = kb_{tt}$).
6. Let C_{1spat} indicate the retrieval cost in t_s units when 1 PAT block is read.
7. Let C_{2spat} indicate the retrieval cost in t_s units when 2 PAT blocks are read.
8. Let C_{1pat} indicate the retrieval cost in t_s units for PAT array model, which corresponds to the situation that produced C_{1spat}.
9. Let C_{2pat} indicate the retrieval cost in t_s units for PAT array model, which corresponds to the situation that produced C_{2spat}.
10. Let G_1, G_2 indicate the feasible performance gain when 1 or 2 PAT blocks are read, respectively.

The transfer time, t_x, is computed as a function of the number of CD-ROM sectors. One CD-ROM sector is the minimum data block that can be transferred from disk to main memory in any case, and contains $L_s/4$ PAT entries. All cost computations will be done in terms of t_s units, so that any retrieval cost, given in seconds, will be divided by t_s. When only one block is read to find the answer ($A \leq b$), we have the following expression for cost retrieval in the SPAT array:

$$C_{1spat} = [2\log(b/2) + 1]t_s + t_x \tag{1}$$

Expressing C_{1spat} as t_s units we have:

$$C_{1spat} = 2\log(b/2) + 1 + \frac{t_x}{t_s} \tag{2}$$

$$\frac{t_x}{t_s} = \frac{k \times b_{tt}}{t_s} = \frac{4 \times b \times b_{tt}}{t_s \times L_s} \tag{3}$$

But $\frac{b_{tt}}{L_s} = 0.01333/2048 = 6.51 \times 10^{-6}$. Let D_x be that constant. Then, C_{1spat} is:

$$C_{1spat} = 2\log(b) + \frac{4bD_x}{t_s} - 1 \tag{4}$$

Similarly, using the same procedure as above, when there are two blocks to be retrieved from disk, the cost expression using the SPAT array is:

$$C_{2spat} = 2\log(b) + \frac{8bD_x}{t_s} + 2 \tag{5}$$

Considering an analogous situation for the PAT array structure, and taking the parameters for CD-ROM disks (such as sector size), we have to consider two cases. First, $A \leq b$ and $b \geq L_s/4$ (a corresponding PAT block would fill at least one sector) in which case we have:

Cost for index search: $[\log(n/b) + 2\log(4b/L_s)](t_s + b_{tt})/ts$

Cost for text search: $[\log(n/b) + 2\log(4b/L_s) + 2\log(L_s/4)](t_s + b_{tt})/t_s$

With no loss of precision, if we consider typical CD-ROM standards, we can make $(t_s + b_{tt})/t_s = 1$, because $b_{tt} \approx 0.015t_s$. So that the total cost, in this case, is:

$$C_{1pat} = 2\log(n) + 2\log(b) - 18 \tag{6}$$

If $A \leq b$, and an equivalent PAT array block is less than one CD-ROM, that is, $b < L_s/4$, then we have the following equivalent costs in the PAT array:

$$C_{1pat'} = [2\log(4n/L_s) + 2\log(L_s/4)](t_s + b_{tt})/t_s \tag{7}$$

Which can be simplified to:

$$C_{1pat'} = 2\log(n) \tag{8}$$

If two blocks are read from disk to find the answer ($A \gg b$), the corresponding situation for the PAT array is given by:

Cost for index search: $[2\log\frac{n/2}{L_s/4} + 1](t_s + b_{tt})/t_s$

Cost for text search: $[2\log\frac{n/2}{L_s/4} + 2\log L_s/4 + 1](t_s + b_{tt})/t_s$

And the total cost will be:

$$C_{2pat} = [4\log(2n/L_s) + 2\log(L_s/4) + 2](t_s + b_{tt})/t_s \tag{9}$$

This expression may be simplified to:

$$C_{2pat} = 4\log(n) - 20 \tag{10}$$

Comparing the cost expression (6) with (4), (8) with (4), and (10) with (5), we can express the improvement obtained with the SPAT array structure in terms of the number of times that the cost is reduced, which shall have the meaning of a performance gain factor (G). The feasible gain for small answers ($A \leq b$) and having PAT array block size larger than one CD-ROM sector ($b \geq L_s/4$), using $b = (n \times l_s)/M_{size}$, is given by:

$$G_1 = f(n, M_{size}, l_s, t_s) = \frac{C_{1pat}}{C_{1_{spat}}} = \frac{2\log(n) + \log(l_s) - \log(M_{size}) - 9}{\log(n) + \log(l_s) - \log(M_{size}) + \frac{2nD_x l_s}{M_{size} t_s} - \frac{1}{2}} \tag{11}$$

If $b < L_s/4$ the feasible gain is expressed by:

$$G_{1'} = f(n, M_{size}, l_s, t_s) = \frac{C_{1pat'}}{C_{1_{spat}}} = \frac{\log(n)}{\log(n) + \log(l_s) - \log(M_{size}) + \frac{2nD_x l_s}{M_{size} t_s} - \frac{1}{2}} \tag{12}$$

And the feasible gain for very large answers ($A \gg b$) is given by:

$$G_2 = f(n, M_{size}, l_s, t_s) = \frac{C_{2pat}}{C_{2spat}} = \frac{2\log(n) - 10}{\log(n) + \log(l_s) - \log(M_{size}) + \frac{4nl_s D_x}{M_{size} t_s} + 1} \tag{13}$$

Analyzing expressions (11) and (13) for a given text database with n index points and given values of M_{size}, l_s, and t_s, we can see that if the transfer time is neglected, the limit of G_1 and G_2 when $n \to \infty$ is 2. This is an expected result because in the SPAT model we reduce the number of comparisons by a factor at least equal to 2. But, if we consider the linear growth of the transfer time, there will be values of n that makes G_1 and G_2 equal to 1, for given parameters of M_{size}, l_s, and t_s. For example, if we consider $M_{size} = 8$ megabytes, $l_s = 30$, and $t_s = 0.6$ seconds, we found that G_1 will become 1 for $n = 3.77 \times 10^{11}$, which corresponds to a text size of about 2.2 terabytes. For the same parameters, G_2 will become 1 for $n = 2.84 \times 10^{11}$, which corresponds to a text size of about 1.7 terabytes. We conclude that a performance gain will always be obtained with the SPAT array model for any practical value of text size that can be actually stored in a single CD-ROM disk.

We can also see that G_1 and G_2 may be maximized if M_{size} is maximized or l_s is minimized. Both actions implies increasing the index granularity g_i, as pointed out in section 4. The following examples may illustrate the performance improvement when the SPAT array model is used:

Example 1: Consider a text database with 50×10^6 index points (file of about 300 megabytes), $Msize = 4$ megabytes, $l_s = 40$ characters, and $t_s = 0.5$. This gives $r = 1 \times 10^5$, $b = 500$ and $B_p = 2$ kilobytes (one CD-ROM sector). Observe that this size of text file plus the index file, nearly fills up the entire storage capacity of a CD-ROM disk. Using (11) and (13) we have the feasible gains for the worst case:

- $A \leq b$: $G_1 = 3.03$ (67% improvement in retrieval time)
- $A > b$: $G_2 = 4.13$ (76% improvement in retrieval time)

Example 2: Consider a text database with 1×10^6 indexing points (file of about 6 megabytes), $Msize = 4$ megabytes, $l_s = 20$ characters, and $t_s = 0.5$ seconds. This gives $r = 2 \times 10^5$, $b = 5$ and $B_p = 20$ bytes. Using (11) and (13) we have the feasible gains for the worst case:

- $A \le b$: $G_1 = 11.5$ (91% improvement in retrieval time)
- $A > b$: $G_2 = 9.1$ (89% improvement in retrieval time)

Computing G_1 and G_2 for n ranging from 1.6×10^6 to 51.2×10^6 index points (text files of about 10 to 300 megabytes), and $Msize$ ranging from 2 to 8 megabytes, we have the results shown in Table 1.

$Msize = 2$ megabytes, $r = 100 \times 10^3$, $l_s = 20$ bytes, $t_s = 0.5$

n Index points	$b = n/r$ Elements in PAT block	$B_p = b \times 4$ PAT array block size (bytes)	G_1	G_2
1.6×10^6	16	64	5.95	6.29
3.2×10^6	32	128	4.84	5.57
6.4×10^6	64	256	4.14	5.05
12.8×10^6	128	512	3.65	4.67
25.6×10^6	256	1024	3.29	4.37
51.2×10^6	512	2048	3.02	4.13

$Msize = 4$ megabytes, $r = 200 \times 10^3$, $l_s = 20$ bytes, $t_s = 0.5$

n Index points	$b = n/r$ Elements in PAT block	$B_p = b \times 4$ PAT array block size (bytes)	G_1	G_2
1.6×10^6	8	32	8.36	7.87
3.2×10^6	16	64	6.23	6.69
6.4×10^6	32	128	5.06	5.90
12.8×10^6	64	256	4.32	5.34
25.6×10^6	128	512	3.80	4.92
51.2×10^6	256	1024	3.43	4.59

$Msize = 8$ megabytes, $r = 400 \times 10^3$, $l_s = 20$ bytes, $t_s = 0.5$

n Index points	$b = n/r$ Elements in PAT block	$B_p = b \times 4$ PAT array block size (bytes)	G_1	G_2
1.6×10^6	4	16	14.06	10.53
3.2×10^6	8	32	8.76	8.38
6.4×10^6	16	64	6.52	7.09
12.8×10^6	32	128	5.29	6.24
25.6×10^6	64	256	4.50	5.63
51.2×10^6	128	512	3.96	5.17

Table 1: Performance gain with SPAT for various values of n and M_{size}

6 Conclusions

In this paper we presented an index structure and an algorithm called *Short* PAT *array*– SPAT that reduces disks access when PAT array indices are used to retrieve information from free text files. Due to the low retrieval performance of CD-ROM optical disks, we conclude that the implementation of the SPAT array model in the design of text retrieval systems in CD-ROM medium represents a great benefit to the end user. The cost analysis derivated for the SPAT array model shows the feasible performance gain that can be achieved when practical value of text sizes and actual CD-ROM devices are used. By observing that consecutive entries in the SPAT array index may have the same prefix, a better performance could still be obtained by storing in the index only the differential part of the entries. Further research on this and other ideas is being done.

Acknowledgements

The authors wish to acknowledge the financial support from the Brazilian CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, IBM do Brasil, and Programa de Cooperación Científica Chile-Brasil de Fundación Andes.

References

Ars Innovandi (1992) *Search City 1.1 - Text Retrieval For Windows™ Powers Users*, Reference Guide, Santiago, Chile.

Barbosa, E. F., e Ziviani, N. (1992) "Data Structures and Access Methods for Read-Only Optical Disks", in R. Baeza-Yates and U. Manber (eds) *Computer Science: Research and Applications*, Plenum Publishing Corp., pp. 189-207.

Gonnet, G. H. (1983) "Unstructured Data Base or Very Efficient Text Searching", *Proceedings of the 2nd ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, Atlanta, Georgia, pp. 117-124.

Gonnet, G. H., Baeza-Yates, R. and Snider, T. (1992) PAT *Arrays: An Alternative to Inverted Files*, chapter 5, Prentice-Hall.

Gonnet, G. H. and Baeza-Yates, R. (1991) *Handbook of Algorithms and Data Structures*, Addison-Wesley.

Gonnet, G. H. (1987) "PAT 3.1: An Efficient Text Searching System", User's Manual, Center for the New OED, University of Waterloo.

Manber, U. and Myers, G. (1990) "Suffix Arrays: A New Method for On-Line String Searches", *ACM-SIAM Symposium on Discrete Algorithms*, January 1990, pp. 319-327.

Morrison, D. R. (1968) "PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric", *Journal of the ACM 15(4)*, pp. 514-534.

Snider, T., (1993) *Private communication*, Waterloo, Canada.