# Practical Aspects of Compressed Suffix Arrays and FM-index in Searching DNA Sequences

Wing-Kai Hon* Tak-Wah Lam* Wing-Kin Sung†
Wai-Leuk Tse* Chi-Kwong Wong* Siu-Ming Yiu*

**Abstract**

Searching patterns in the DNA sequence is an important step in biological research. To speed up the search process, one can index the DNA sequence. However, classical indexing data structures like suffix trees and suffix arrays are not feasible for indexing DNA sequences due to main memory requirement, as DNA sequences can be very long.

In this paper, we evaluate the performance of two compressed data structures, Compressed Suffix Array (CSA) and FM-index, in the context of searching and indexing DNA sequences. Our results show that CSA is better than FM-index for searching long patterns. We also investigate other practical aspects of the data structures such as the memory requirement for building the indexes.

## 1 Introduction

With the availability of different DNA sequences (in particular, the human genome), many biological research activities involve searching DNA sequences for various patterns (say, genes). To speed up the search process, one would naturally try to index a DNA using classical indexing data structures like suffix trees [13] and suffix arrays [12], which require $\Theta(n)$ words, or $\Theta(n \log n)$ bits, of storage and can locate a pattern $P$ efficiently in $\mathcal{O}(|P| + occ)$ and $\mathcal{O}(|P| \log n + occ)$ time, where $n$ and $occ$ denote the length of the DNA sequence and the number of occurrence of $P$, respectively. However, such efficient schemes are not feasible due to the main memory requirement. For example, the human genome contains 2.88 Giga characters (bases) and even a single chromosome contains several hundred Mega characters. The best known implementation of suffix trees and suffix arrays for DNA sequence, however, requires $17.25n$ bytes and $4.25n$ bytes in practice [9, 12]. These figures amount to several tens of Gigabytes of main memory, which far exceed the capacity of a PC nowadays.[1]

In recent years, exciting results have been obtained in compressing the suffix arrays. Compressed Suffix Arrays (CSA) [6] and FM-index [5] are such examples. For indexing DNA sequence, the bound for the basic structure of CSA consists of only $5n$ bits, while for FM-index, it occupies $3n$ bits in practice [4]. These figures immediately imply that we can store the CSA and FM-index of DNA of length up to a few Gigabases, which covers almost every known DNA sequences nowadays. In contrast, the suffix tree and suffix arrays can only handle DNA of length up to 180 Mbases and 900 Mbases, respectively.

Despite the compactness in size, the searching performance of CSA and FM-index is asymptotically of the same order as (or even better than) suffix arrays. Given this promising theoretical bound, it is natural to ask how fast CSA and FM-index can search in practice? Moreover, which one is better?

For FM-index, previous experiments [4] have shown that its performance is comparable to suffix arrays, when searching a pattern whose length is

*Department of Computer Science and Information Systems, The University of Hong Kong, Hong Kong, {ckwong3,twlam,wkhon,wltse,smyiu}@csis.hku.hk

†School of Computing, National University of Singapore, Singapore, ksung@comp.nus.edu.sg

[1]The maximum size of main memory for a PC is 4 Gbytes.

short (8-15 bases). However, for searching long patterns (say, a few hundred or even a few thousand bases), it is not known whether the result will be consistent.

Another issue with respect to searching is that, in the literature, there are two types of searching methodology for CSA or FM-index. One of them is called *forward search*, which is the classical approach for suffix arrays. The other one is called *backward search*, which is the method tailored for CSA or FM-index and has been shown to be optimal in the worst-case. It is interesting to find out in practice, whether the optimal backward search always beat the forward search.

Finally, if CSA and FM-index are to be workable in practice, we have to consider yet another important issue: the memory requirement for their construction. This requirement can be more severe than for storage requirement, as the original construction methods are to first build the suffix arrays and then perform compression. For instance, the suffix array for the human genome (2.88 Gbases) already occupies at least 12 Gbytes, which by itself is impossible to be built on an ordinary PC, not to mention performing the compression afterwards.

Nevertheless, we can overcome the memory problem by constructing the CSA and FM-index in an alternative way. Previous results show [7, 10] that CSA can be directly constructed from the DNA sequence in $\mathcal{O}(n)$ bits space, while FM-index can be converted from the CSA in negligible space. In this case, we would like to determine the maximum length of a genome whose CSA and FM-index can be constructed, and the time needed to construct such index.

This paper evaluates the performance of the CSA and FM-index when they are used to index DNA sequences, and attempts to answer all the above questions.

For the searching performance, we have constructed the CSA and the FM-index for E.coli (4.6 Mbases), Fly (98 Mbases) and Human (2.88 Gbases). In each setting, we tested the searching times using patterns of length from 10 to 10,000. Patterns are extracted from random positions in the corresponding DNA sequence to boost the worst-case performance. From our experiments, we find that FM-index is faster than CSA for searching short patterns, while for long patterns, CSA is better.

For the comparison between forward search and backward search, we observe that using backward search, FM-index is consistently faster than CSA. However, using forward search, CSA is faster than FM-index. The most surprising result is that, for long patterns, forward search is more efficient than backward search. Roughly speaking, for patterns of length less than 2000, FM-index with backward search is most efficient; otherwise, CSA with forward search is fastest. See Figure 1 for the timing of the experiments on Fly.

For the construction limits, we have implemented programs that can successfully construct the CSA and FM-index for DNA sequences of length up to 3 Gbases. The construction times needed are 24 and 28 hours for CSA and FM-index, respectively.

*Remark:* For the storage space, we observe that the basic structure for CSA in practice occupies $4n$ bits of space.

The paper is organized as follows. Section 2 gives some background on CSA and FM-index. Section 3 investigates the searching performance of CSA and FM-index, while Section 4 comments the construction requirements. Concluding remarks are shown in Section 5.

## 2 Background

In this section, we give a brief introduction on CSA and FM-index. The first part describes their basic structures, while the second part discusses the two searching methodologies associated with them. Finally, we highlight our implementation details in the third part.

**2.1 Basic Structure** Given a text $T[1..n]$, the suffix array $SA[1..n]$ is a permutation of integers of $\{0, 1, \dots, n-1\}$ that denotes the lexicographical relation of the suffixes of $T$. Precisely, $SA[i] = j$ if and only if $T[j..n]$ is the $i$-th smallest suffix

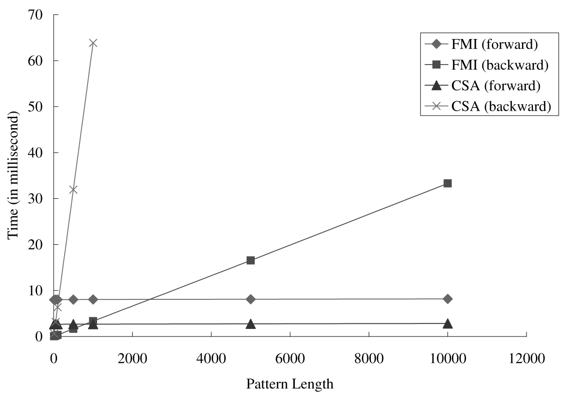

Figure 1: Searching performance of CSA and FM-index of Fly.

among all the other suffixes. The basic structure for both CSA and FM-index is a sequence of $n$ integers obtained from a transformation of $SA$. Precisely, CSA stores the function $\Psi$, where $\Psi[i] = SA^{-1}[SA[i]+1]$; for FM-index, it stores the function $\Phi$, where $\Phi[i] = SA^{-1}[SA[i]-1]$. These integers can be stored in $32n$ bits in a brute force manner, but with suitable transformation, we can show that most of the integers in the transformed sequence are small, and space reduction is achieved when we encode each integer using *prefix-free code.* The size of the basic structure for CSA is bounded by $5n$ bits, while in practice it occupies only $4n$ bits. For FM-index, the worst-case bound is $10n$ bits, while in practice it is close to $3n$ bits.

However, the use of prefix-free code has a disadvantage: to decode any region of the sequence, we must start the decoding process from the beginning of the encoded sequence. In order to speed up the decoding process, which is important to the performance of the compressed indexes, the original papers suggested to store auxiliary information in addition to the basic structure. These auxiliary information are markers that are stored at regular locations within the encodings, so that decoding can be started from the closest marker instead of from the very beginning.

As the performance of the compressed indexes is poor when the auxiliary information is stringent, because of this, the rest of the paper will consider the auxiliary information to occupy a space of $\mathcal{O}(n)$ bits.

**2.2 Searching Methodology** Let $T_i$ denote the the suffix $T[i..n]$. Given the text $T$ and the corresponding suffix array $SA$, we can search for the occurrence of a pattern $P$ in the following way [12]: (1) For $i = 1, 2, \ldots, m$, find the range $s$ and $t$ such that $T_{SA[s-1]} < P[1..i] \leq T_{SA[s]}$ and $T_{SA[t]} \leq P[1..i] < T_{SA[t]}$; (2) If $s \leq t$, report $SA[s], \ldots, SA[t]$ as the occurrences of $P$ in $T$. Otherwise, report no occurrences found.

In the above algorithm, we recursively compute the range of the suffix array that corresponds to the occurrences of $P[1..i]$, for $i = 1, 2, \ldots, m$. We refer this searching methodology as the forward search, as we are intuitively matching the characters of $P$ with $T$ in a forward manner.

On the other hand, the definition of the CSA, or the FM-index, favours another searching methodology, in which the pattern $P$ should be matched with the text $T$ starting from the last character. We refer this as the backward search.

Table 1 gives a summary of the worst-case searching performance of CSA and FM-index. Note that in the table, the bounds depend on a parameter $\epsilon$, where $0 < \epsilon \leq 1$. This $\epsilon$ is in fact a tradeoff parameter for the searching time and index space. In order to achieve the stated timing bounds, the corresponding CSA or FM-index would require $\frac{1}{\epsilon}$ times the size of the basic structure. In this paper, we consider $\epsilon$ to be 1.

*Remark:* Although the worst-case behaviour of backward search (in both indexes) is better than that of the forward search, in practice, however, forward search may outperform backward search

Table 1: Searching performance of CSA and FM-index

| Index | Forward Search | Backward Search |
|---|---|---|
| CSA | $\mathcal{O}(\|P\| \log^{1+\epsilon} n + occ \log^{\epsilon} n)$ | $\mathcal{O}(\|P\| \log n + occ \log^{\epsilon} n)$ |
| CSA ($\epsilon = 1$) | $\mathcal{O}(\|P\| \log^2 n + occ \log n)$ | $\mathcal{O}(\|P\| \log n + occ \log n)$ |
| FM-index | $\mathcal{O}(\|P\| \log^{\epsilon} n + occ \log^{\epsilon} n)$ | $\mathcal{O}(\|P\| + occ \log^{\epsilon} n)$ |
| FM-index ($\epsilon = 1$) | $\mathcal{O}(\|P\| \log n + occ \log n)$ | $\mathcal{O}(\|P\| + occ \log n)$ |

for two reasons: Firstly, based on Manber and Myers [12], the average time of forward search is expected to be $\mathcal{O}(|P| \log n + occ \log n)$ and $\mathcal{O}(|P| + occ \log n)$ for CSA and FM-index, respectively, which matches to the worst-case time of backward search. Secondly, each operation of the backward search is usually more computationally involved, so that the hidden constant in the worst-case bound could be very high.

**2.3 Implementation Details** Our implementation for CSA is based on Lam et al.[10].[2] In their paper, they observed that the basic structure, $\Psi$, can be partitioned into four increasing sequences with values between 0 and $n$. Then, for each sequence $s_1, s_2, \ldots$, we can store the difference values (that is, $s_1, s_2 - s_1, s_3 - s_2$, and so on) instead of the original values. Compression can be achieved by encoding each of these difference values separately using variable-length prefix-free codes, such as $\gamma$ code or $\delta$ code [3].

The original paper suggests to use $\delta$ code, which has better worst-case size bound for general alphabets; but for our implementation, we use $\gamma$ code instead, for it achieves smaller size than $\delta$ code when encoding the $\Psi$ function of DNA sequence in practice.

As mentioned before, we need to store auxiliary structure for the prefix-free coded sequence to speed up the decoding process. In our implementation, we store the $\Psi[i]$ explicitly whenever $i \bmod \ell = 0$ for some $\ell = \mathcal{O}(n/\log n)$. Then, to compute $\Psi[k]$, we first find $\Psi[c\ell]$ with $c\ell \le k < (c+1)\ell$ in constant time. Afterwards, we look at the encoded sequence for $\Psi[c\ell+1] - \Psi[c\ell], \Psi[c\ell+2] - \Psi[c\ell+1], \ldots, \Psi[k] - \Psi[k-1]$ to compute $\Psi[k] - \Psi[c\ell]$. The latter part can be done using constant number of standard table-lookups, so that the overall time to compute $\Psi$ takes $\mathcal{O}(1)$ time. Moreover, the space required is $\mathcal{O}(n)$ bits. Note that the more values of $\Psi$ we store explicitly, the more space we need, but the faster the decoding process.

To support computing the $SA$ values, we need to store another auxiliary data structure. Precisely, we store the $SA[i]$ explicitly whenever $i \bmod t = 0$ for some $t = \mathcal{O}(n/\log n)$. To compute $SA[i]$, we compute $\Psi[i], \Psi^2[i], \Psi^3[i]$ and so on, until $\Psi^k[i] \bmod t = 0$. Then, we obtain the value $s = SA[\Psi^k[i]]$, and it is easy to verify that $SA[k] = s - k$. Note that this implementation requires $\mathcal{O}(n)$ bits space. On the other hand, it does not guarantee worst-case time for retrieving the $SA$ values. Despite of this, it has good average case performance, where an $SA$ value can be retrieved in expected $\mathcal{O}(\log n)$ time in practice. In fact, we can modify our implementation to guarantee worst-case time for getting the $SA$, but this would require some complicated data structure that answers the `rank` and `select` queries [8] in $\mathcal{O}(1)$ time.[3] In practice, this means more space for the index *and* more time for getting the $SA$.

For the same reason, our implementation of FM-index adopts an auxiliary data structure that supports the retrieval of $SA$ in average $\mathcal{O}(\log n)$ time instead of worst-case. For the other parts, it follows closely to that in [5], which includes techniques like Burrows-Wheeler transform [2], move-

[2]We have implemented and tested three other variations of CSA. Their performance for searching DNA sequence is quite similar; the variation reported in this paper is consistently the best in our studies.

[3]Though the theoretical bound is $\mathcal{O}(1)$, the hidden constant, however, is high in practice.

to-front encoding [1] and run-length encoding. (See the original paper for more details.)

## 3 Searching a Pattern with CSA and FM-index

This section investigates the practical searching behaviour of CSA and FM-index. In the first part, we compare the two searching methodologies for CSA and FM-index, and show that forward search in practice performs better than backward search when the pattern length is long. In addition, other interesting findings are also observed. In the second part, we perform a case study on the practical performance of CSA and FM-index with the suffix tree and suffix arrays.

We ran all the experiments on a machine equipped with a 1.7 GHz Pentium IV processor with 512 Kbytes of L2 cache, and 4 Gbytes of RAM. The operating system was Solaris 9.

### 3.1 Forward Search and Backward Search

We have constructed the CSA and the FM-index for the following genomes: E.coli (4.6 Mbases), Fly (98 Mbases), Human Genome (2.88 Gbases). As mentioned, the amount of auxiliary information affects the performance of the compressed index greatly. For our experiments, we have investigated three implementations of each index, which respectively requires a total of $4.5n$, $6n$ and $8n$ bits of space, which are refer to as the small, medium and large implementations.[4] In case where we conduct forward search, an additional $2n$ bits of memory is required for storing the DNA text.

For each genome, we have tested the searching times using patterns of lengths 10, 50, 100, 500, 1000, 5000, and 10000, where patterns are selected from the corresponding genome at random positions, so as to get a more accurate account of the worst-case behaviour.[5] For each test case, it is repeated for 1000 times to obtain an average timing. Finally, only searching times for existence (that is, to determine whether the pattern $P$ occurs in the DNA sequence $T$) are reported, as the time for enumeration (that is, reporting the occurrence of $P$ in $T$) is independent of the searching methodology.

From our experiments, we observe that if the index are provided with the same amount of space (in terms of $n$), the searching performance with genomes of different length *does not vary* a lot. For instance, compare Tables 2 and 3 to see the timing difference of searching with Fly and with Human. In this section, we shall focus on the experimental result of Fly (Table 2), and use it to present the general observations that we have made.

Some interesting findings can be summarized as follows.

- Using backward search, FM-index is at least ten times faster than CSA in all testing cases. Both CSA and FM-index have searching time increasing linearly on pattern length.

- Using forward search, CSA is, however, two times faster than FM-index in the medium or the large implementation, and slightly slower than FM-index in the small implementation.

  Unlike backward search, forward search is not sensitive to pattern length. We believe that in practice, forward searching with CSA and FM-index requires $\alpha|P| + \beta \log n$ time, for some constants $\alpha \ll \beta$. In other words, the time is determined by the $\log n$ factor instead of the pattern length.

- Theoretically, backward search is better than forward search for both indexes. Most surprisingly, experiments show that for long patterns, forward search is more efficient. For CSA, forward search outperforms backward search for patterns of length 50 or more; for FM-index, this occurs for patterns of length around 3000 or more.

[4]We have also investigated another setting in which both data structures have the same amount of auxiliary storage (precisely, $2n$ bits). Note that this setting is not fair to FM-index as we allow CSA together with its auxiliary storage to use more memory; nevertheless, the finding is similar to those to be reported below.

[5]As DNA is a very biased string, a random pattern is unlikely to be found there; thus, searching for a random string is often very fast as a few comparisons could confirm the non-existence. To test the worst behavior of the indexing data structures, we use substrings or modified substrings of the DNA.

Table 2: Searching performance (in msec) of CSA and FM-index for Fly.

| Index | Index Size | Searching Method | Pattern Length | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 |
| CSA | small | forward | 28.70 | 28.70 | 28.71 | 28.72 | 28.74 | 28.80 | 28.86 |
| | | backward | 1.87 | 9.36 | 18.81 | 95.00 | 189.3 | 947.5 | 1894 |
| | medium | forward | 2.654 | 2.655 | 2.660 | 2.677 | 2.684 | 2.753 | 2.821 |
| | | backward | 0.635 | 3.160 | 6.357 | 31.95 | 63.87 | 319.4 | 638.5 |
| | large | forward | 1.001 | 1.001 | 1.021 | 1.022 | 1.031 | 1.094 | 1.160 |
| | | backward | 0.435 | 2.170 | 4.371 | 21.97 | 43.95 | 220.0 | 439.0 |
| FMI | small | forward | 20.47 | 20.48 | 20.50 | 20.50 | 20.53 | 20.54 | 20.62 |
| | | backward | 0.062 | 0.283 | 0.551 | 2.668 | 5.321 | 26.54 | 52.99 |
| | medium | forward | 7.983 | 8.021 | 8.023 | 8.029 | 8.040 | 8.101 | 8.180 |
| | | backward | 0.037 | 0.175 | 0.343 | 1.678 | 3.325 | 16.54 | 33.29 |
| | large | forward | 2.699 | 2.700 | 2.707 | 2.711 | 2.732 | 2.796 | 2.877 |
| | | backward | 0.027 | 0.130 | 0.253 | 1.231 | 2.446 | 12.21 | 24.41 |

Table 3: Searching performance (in msec) of CSA and FM-index for Human.

| Index | Index Size | Searching Method | Pattern Length | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 |
| CSA | small | forward | 38.96 | 38.96 | 38.98 | 38.98 | 39.01 | 39.10 | 39.20 |
| | | backward | 2.51 | 12.37 | 24.74 | 123.7 | 247.6 | 1235 | 2474 |
| | medium | forward | 3.603 | 3.604 | 3.607 | 3.618 | 3.656 | 3.658 | 3.757 |
| | | backward | 0.852 | 4.200 | 8.443 | 42.66 | 85.15 | 425.2 | 851.2 |
| | large | forward | 1.359 | 1.359 | 1.360 | 1.362 | 1.370 | 1.454 | 1.545 |
| | | backward | 0.584 | 2.910 | 5.802 | 29.17 | 58.45 | 291.5 | 583.2 |
| FMI | small | forward | 20.01 | 20.02 | 20.04 | 20.05 | 20.08 | 20.13 | 20.22 |
| | | backward | 0.074 | 0.352 | 0.710 | 3.554 | 7.120 | 35.62 | 71.31 |
| | medium | forward | 7.805 | 7.805 | 7.806 | 7.808 | 7.811 | 7.819 | 7.829 |
| | | backward | 0.044 | 0.208 | 0.412 | 2.051 | 4.108 | 20.55 | 41.15 |
| | large | forward | 2.638 | 2.639 | 2.640 | 2.644 | 2.672 | 2.735 | 2.821 |
| | | backward | 0.032 | 0.155 | 0.312 | 1.536 | 3.084 | 15.37 | 30.72 |

Roughly speaking, for patterns of length less than 1500, FM-index with backward search is the best; otherwise, CSA with forward search is fastest.

**3.2 Comparison with Suffix Trees and Suffix Arrays** As one can expect, the searching performance for suffix tree or suffix arrays should be better than CSA and FM-index, as there are no compression involved in the former indexes. In this section, we try to give a quantitative comparison between these four indexes in practice.

We have constructed the four index for the E.coli (4.6 Mbases) genome. For CSA and FM-index, we just consider the medium implementations, which each occupies $6n$ bits. We have tested the searching times using patterns of lengths 10, 50, 100, 500, 1000, 5000, and 10000. Forward search are conducted for all the indexes, and backward search are conducted for CSA and FM-index. In case where we conduct forward search, an additional $2n$ bits of memory is required for storing the DNA text.

Patterns are selected from the E.coli genome at random positions. For each test case, it is repeated for 1000 times to obtain an average timing. The searching times are separated into two parts: the time for reporting whether the pattern exists in the text, and the time for enumerating the location of each occurrence. Tables 4(a) and 4(b) show the best time obtained by each index.

From Table 4, we observe that both CSA

Table 4: Searching performance of different indexes. (a) Average time (in msec) for one existential query for different pattern length. (b) Average time (in $\mu$sec) for reporting the location of one occurrence

| Index | Pattern Length | | | | | | |
|---|---|---|---|---|---|---|---|
| | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 |
| Suffix Tree | 0.003 | 0.003 | 0.004 | 0.006 | 0.010 | 0.035 | 0.067 |
| Suffix Array | 0.010 | 0.010 | 0.011 | 0.015 | 0.021 | 0.060 | 0.111 |
| CSA | 0.512 | 2.145 | 2.150 | 2.166 | 2.173 | 2.232 | 2.318 |
| FM-index | 0.035 | 0.162 | 0.320 | 1.573 | 3.152 | 5.225 | 5.309 |

(a)

| Index | Time |
|---|---|
| Suffix Tree | 11.5 |
| Suffix Array | 0.5 |
| CSA | 49.0 |
| FM-index | 114.0 |

(b)

and FM-index are much slower than suffix trees and suffix arrays, in terms of both existential query or enumerating occurrences. Nevertheless, in terms of absolute time, each existential query using CSA or FM-index can be answered within a few milliseconds, and the enumeration of one occurrence is in the order of *microseconds*, which is acceptable in most applications.

## 4 Construction Requirements

When considering the memory requirement for CSA and FM-index, one often focuses on the size of the resulting data structures. In fact, another important concern is the memory requirement for constructing these data structures. Original methods for constructing CSA and FM-index requires building the suffix array first and then perform compression, which is infeasible for genome whose length is of some Gigabases.

Nevertheless, we can overcome the memory problem by constructing the CSA and FM-index in an alternative way. Previous results show [7, 10] that CSA can be directly constructed from the DNA sequence in $(5+\epsilon)n$ bits space, for any $\epsilon > 0$, while FM-index can be converted from the CSA in negligible extra space. Note that the larger the $\epsilon$, the faster the algorithm is. Based on this, we have implemented space-efficient programs for constructing CSA and FM-index on a PC, and tested it with E.coli, Fly and Human as the input genomes.

For the actual construction, we set $\epsilon = 5$ to make the maximal use of the main memory. Table 5 shows the corresponding construction space and time for each genome.

To conclude this section, Table 6 shows the limitations on the index that can be constructed in an ordinary PC nowadays, assuming a RAM of size 4 Gbytes.[6]

## 5 Concluding Remarks

We have demonstrated that CSA and FM-index can be constructed for a genome of length up to a few Gigabases. This allows most of the genomes nowadays to become indexable, which was previously infeasible if we use suffix trees or suffix arrays.

For the searching performance, we observe that CSA is better than FM-index for searching long patterns. Moreover, we have compared backward search with the forward search of both indexes, and find that forward search is faster than backward search when the input pattern is long. This is counter-intuitive, since in theory, backward search is always better than forward search.

## References

[1] J. Bentley, D. Sleator, R. Tarjan, and V. Wei. A locally adaptive compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[6]Technically speaking, we cannot make full use of all the RAM as some space must be reserved for the operating system. In our case, only a maximum of 3.6 Gbytes out of 4 Gbytes are available.

Table 5: Construction time and space for CSA and FM-index

| DNA | Construction Space | Construction Time | |
|---|---|---|---|
| | | CSA | FM-index |
| E.coli | 5.8M byte (10n bits) | 60 sec | 72 sec |
| Fly | 125M byte (10n bits) | 30 min | 36 min |
| Human | 3.6G byte (10n bits) | 24 hour | 28 hour |

Table 6: Limitations on the genome to be constructed

| Index | Construction Algorithm | Maximum Genome Constructed | Maximum Genome Able To Reside |
|---|---|---|---|
| Suffix Tree | Kurtz [9] | 180Mb | 180Mb |
| Suffix Array | Larsson-Sadakane [11] | 450 Mb | 900 Mb |
| CSA | Lam et al. [10] | 5000 Mb | 7200 Mb |
| FM-index | Hon et al. [7] | 5000 Mb | 9600 Mb |

[2] M. Burrows and D. J. Wheeler. A Block-sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Equipment Corporation, Paolo Alto, California, 1994.

[3] P. Elias. Universal codeword sets and representation of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

[4] P. Ferragina and G. Manzini. An experimental study of an opportunistic index. In *Proceedings of Symposium on Discrete Algorithms*, pages 269–278, 2001.

[5] P. Ferragine and G. Manzini. Opportunistic Data Structures with Applications. In *Proceedings of Symposium on Foundations of Computer Science*, pages 390–398, 2000.

[6] R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. In *Proceedings of Symposium on Theory of Computing*, pages 397–406, 2000.

[7] W. K. Hon, T. W. Lam, K. Sadakane, and W. K. Sung. Constructing Compressed Suffix Arrays with Large Alphabets. In *Proceedings of International Conference on Algorithms and Computation*, 2003. To appear.

[8] G. Jacobson. Space-efficient Static Trees and Graphs. In *Proceedings of Symposium on Foundations of Computer Science*, pages 549–554, 1989.

[9] S. Kurtz. Reducing the Space Requirement of Suffix Trees. *Software Practice and Experiences*, 29:1149–1171, 1999.

[10] T. W. Lam, K. Sadakane, W. K. Sung, and S. M. Yiu. A Space and Time Efficient Algorithm for Constructing Compressed Suffix Arrays. In *Proceedings of International Conference on Computing and Combinatorics*, pages 401–410, 2002.

[11] J. Larsson and K. Sadakane. Faster Suffix Sorting. Technical Report Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1-43/(1999), Lund University, 1999.

[12] U. Manber and G. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[13] E. M. McCreight. A Space-economical Suffix Tree Construction Algorithm. *Journal of the ACM*, 23(2):262–272, 1976.