A improved exact string matching method for genomic sequencing data

XXX1, Zhang Zhang1,\*

1CAS Key Laboratory of Genome Sciences and Information, Beijing Institute of Genomics, Chinese Academy of Sciences, Beijing 100029, China

2School of Computer Science and Technology, XXX

\***Corresponding authors**:

Zhang Zhang (zhangzhang@big.ac.cn): CAS Key Laboratory of Genome Sciences and Information, Beijing Institute of Genomics,Chinese Academy of Sciences, No.7Beitucheng West Road, Building G, Chaoyang District, Beijing100029, China,Tel & Fax: +86-10-82995427

Running head: Improved exact string matching algorithm using SIMD instructions.

### Abstract

String matching algorithm plays a vital role in Bioinformatics. In this paper, we proposed an improved pattern matching algorithm for biological sequences. According to the feature of biological sequences, the algorithm uses optimized word-size packed string matching instructions. Furthermore, in each test, the algorithm uses a hash table to decide shift distance, which is independent to matching result. And the Intel streaming SIMD extensions (SSE) technology is applied to compute hash values. Experimental results show that the algorithm is more efficient especially when the pattern length is shorter than 64 bytes.

Key words: Exact string matching, SIMD, biological sequence

### Introduction

String matching is an important problem that has been thoroughly studied in computer science. It is applied extensively in bioinformatics. For example, it is used to find similar sequence or locate a segment in a long sequence[10]. Currently, several string matching algorithms are used on biological sequences, such as tvsbs[5], graspm[9] etc. With the development of sequencing techniques, it has become easy to obtain sequences, i.e. the linear arrangement of residues (nucleotides or amino-acids), of DNA, RNA, or protein molecules. So it is meaningful to design more effective string matching algorithms to meet this challenge [1].

Recently, a string matching algorithm called epsm [4] has been used in bioinformatics and had obtained good results. It uses packed string matching technique [3], in which multiple characters are packed into one block-character, so that the characters can be compared in bulk rather than individually. The epsm algorithm also computes fingerprint values by a hash function using SIMD instructions, which supports parallel execution of some operations via a set of special instructions. However, the max shift distance of epsm algorithm is m-8, where m is the length of the given pattern. It was not an optimal shift distance. Also, there will be errors when the length of string is not . In present study, we improve epsm by introducing the idea of BM algorithm [2] to get more shift distances greedily.

### Material and methods

##### Algorithm

Due to the nature of biological sequences, the algorithm is designed to be fast for medium length, i.e.,between 8 and 64.

typedef struct node

{

struct node \*next;

int pos;

unsigned long long val;

} NODE;

所以容易看出，如果采样串长度i较小，即能获取较大的跳跃距离m-i，然而某些数组结点对应的链表会很长，这样每次比较都会去遍历这个链表，同时采样串的匹配概率变大，导致mencmp全匹配函数的调用次数加大，比较费时；而若采样串较长，由于shift表不能设很大，根据经验数组越大，程序会因cpu缓存原因查询越慢，合适的大小是2048，但对于基因串来说，4的8次方为65536，会造成较大的hash冲突，同样降低性能。

所以我们使用两个方法去提高算法在基因串的查询性能。

一是我们可以通过实验去找到合适的采样串长度，使得它既能获取较大的跳跃距离，同时又不造成很高的hash冲突；

二是因为采样串我们实际是读取8bytes的，但通过与上掩码来获取变长的采样串， 我们通过在调用memcmp前比较原8bytes对应的64bit数值与对应模式串位置的64bit数值，这样就能大大减少全比较函数调用次数。具体实现是在Node定义中加一字段存储原采样串的64bit数值，同时在预处理阶段根据模式串P将其初始化。

通过更改8来增大跳跃

1. 加入64bit来先进行比较，从而减少因掩码造成的memcmp的多次调用。
2. 相当于跳跃距离是m-i，粗看来i越小，跳得越远，但i越小，意味着采样字符串

Epsm算法分为两个阶段。

首先定义查询表，它是记录采样串在模式串的出现位置。程序中由一指针数组Node\* shift[2048]来实现。其中结点Node，数据结构如下：

typedef struct node

{

struct node \*next;

int pos;

unsigned long long val;

} NODE;

它存储某一采样串p1在模式串p中的位置，其中p1通过计算hash值h去定位到它的结点shift[h]。

那么在预处理阶段，对于模式串P，从串位置i起取8bytes算一个hash值h，查询表是记录采样串在模式串的出现位置，程序实现上即由一指针数组Node\* shift[2048]来存储，那么对于采样串p1，算得hash值h后，将p1在p中出现的位置存储在shift[h]中，对于hash冲突造成的重值，存储在shift[h]对应的下一链表结点中。

在搜索阶段，首先指针指向匹配串T的m-8位置，并取8bytes子串，算hash值h，若Table[h]有值i，说明该子串可能是串P在i位置的一部分，然后根据i值将串P的i位置和该子串对齐，并调用memcmp函数进行比较，若Table[h]有多个值，则需根据i值进行多次对齐和比较。完成后指针再继续前进m-8的长度，并重复刚才的搜索步骤。其具体算法描述如图1。

根据鸽笼原理，m-8是epsm不漏扫的最大距离。而我们的优化点在于用BM的思想，使指针的移动距离尽量达到m，即初始化阶段根据模式串P计算出一偏移表shift，该表数据结构为hash表，传入任意8bytes字符串即可得到指针下一步的偏移长度。我们算法的最优情况在于中等长度模式串，表shift的绝大多数偏移值为m。

并且我们利用SSE4指令可以加快hash运算速度。

The algorithm is shown in Fig.2.

HASH，SHIFT大小，用于计算shift的块大小。

The ideas of the new algorithm are:

First we define a function called hash(p,seed)(line 1-2) for calculating the CRC of a fixed-length string using the following SSE instruction, \_mm\_crc32\_u64(). It receive a string of length 8 and a seed values of 64 bits using \_mm\_crc32\_u64 instruction. Then getting a values of 32 bits with a mask 0x7FF. We use this function to calculate the fingerprint values of a string of length 8. And by changing the value of mask, we could get the fingerprint values of different length of string, such as length of 6 and length of 5.

The pre-processing phase(line 1-11):

1). First initialize the array of shift distance, make shift[i] = m.

2). Let i equals from 1 to 8, then find out all the string whose suffix of length i equals the prefix of the pattern string of length i. Then using the hash function to calculate the fingerprint values of 32 bits, say filter. Then let the shift[filter] = m – i.

3). Let i equals from 0 to m-8, then calculate the fingerprint values of the string P+i, if m-i less then shift[filter], then shift[filter] = m-i. And when calculate the string P+m-8, make the shift[filter] = 1, so we could always jump forward in the search phase.

The search phase(line 1-7):

1). Let i equals from 0 to n-8, then calculate the fingerprint values of the string T+i of length 8.

方法1：

用8位判断是否可能匹配，只使用最后一个块字符，变化位来计算shift,调整下shift表的大小。

1. 计算64bit指纹，判定是否需要memcmp
2. 遮蔽，计算shift指纹，查找shift表并跳跃。

方法2：

在epsn上引入一个shift表，用变化位来计算shift值。调整下shift表的大小，也许1-2个字符就会有很好效果。

比较两种方法。

2). If shift[filter] equals to 1, then compare the two string using memcmp function. Otherwise shifting forward the length of shift[filter].

So in most cases, we could jump the distance of m in a great probability. Unlile epsm , it could only jump the distance of m-8 [4].

##### Estimating

We conducted two experiments to estimate the performance of the algorithm. The first one was to evaluate the optimal packed string size on different pattern length. The second one was the comparison to five representative string matching algorithms.

The algorithm was implemented using C. And all experiments were conducted on a PC with Intel(R) Xeon E3-1230 V2 at 3.30GHz, 4G memory, running Linux Mint 13.

##### Data collection

We chose five state-of-art string matching algorithms to conduct comparison experiment, i.e., Tvsbs [5], Ufndmq [6], Hashq [7], Fsbndmq [8], Epsm [4]. All the algorithms executed on four test data sets, as follows [11].

|  |  |  |  |
| --- | --- | --- | --- |
| Data set | type | filename | size |
| S1 | Escherichia coli | Escherichia\_albertii\_KF1\_uid232181 | 200MB |
| S2 | Rice | OrySat\_Aug2009.fa | 200MB |
| S3 | gene sequence of the human genome on chromosome 1 | chr1.fa | 200MB |
| S4 | amino acid of the Escherichia coli K12MG1655 | NC\_000913.faa | 200MB |

### Results and discussion

##### Parameter

Experientially, to guarantee the speed of access, we set the size of the SHIFT table as 2048, which is the same as that of epsm. The application of packed string will introduce hash conflicts. The larger packed string will result in more conflicts. 在生物信息中， DNA信息的字典大小为4，远小于英文字典大小26，所以我们能采样较长的串，因为8bytes的DNA串会有种可能。并且基因串的出现概率与相同字典大小的随机串表现类似，所以hash分布比较均匀。但由于我们为读写性能将shift表大小设为2048，所以采样8bytes扔会造成较大的hash冲突。Thus, unlike epsn which uses fixed size of packed string at 8, we tested different length of packed string.

The experimental results with different packed string size on different pattern size on different data sets are shown in Table 5, Table 6 and Table 7. We can see the influence of the size of packed string. And from the results, we did find out that length 6 was a suitable choice when m<64 and 8 otherwise.

##### Efficiency

The experimental results shown in Table 1, Table 2, Table 3 and Table 4 was a comparison between our new algorithm and other five algorithms in four different types of gene data, such as ecoli, rice, human and amino acid.

From these tables, we could see our algorithm perform stably in different data. And the new algorithm obtained the best performance when m<64 and m>8, beat the other four algorithms such as tvsbs, ufndmq, hashq and fsbndmq in all cases. The only challenger was epsm algorithm, which perform better when m≥64.

This results were as expected. Because of the SIMD instruction, we could obtain great performance when sampling the string. So we could beat the none four algorithms which was not using the SIMD instructions.

And our algorithm could jump the distance of m in a high probability. So the algorithm beat the epsm in middle string, but the results show the epsm beat our algorithm when m≥64. Because the advantage of our algorithm faded when the length of pattern string grew, the probability of shifting distance of m had dropped.

### Conclusions

We presented a new exact string matching algorithm using Intel SSE instructions. Experiments show the algorithm obtained better performance for medium biological sequences.

### ~~Acknowledgments~~

~~We thank XXX for YYY. This work was supported by the “100-Talent Program” of Chinese Academy of Sciences (Y1SLXb1365; ZZ).~~

### References

[1] Eric Rivals, Leena Salmela, and Jorma Tarhio. EXACT SEARCH ALGORITHMS FOR BIOLOGICAL SEQUENCES. Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications, John Wiley & Sons, Inc. (Ed.) (2011) 91-111.

[2] D. M. Sunday. “A very fast substring search algorithm”, Communications of the ACM, 33(8):132-142, 1990.

[3] K. Fredriksson. Faster string matching with super-alphabets. String Processing and Information Retrieval, Springer (2002), pp. 207–214.

[4] S. Faro and M. O. Kulekci. Fast Packed String Matching for Short Patterns. Meeting on Algorithm Engineering and Experiments, ALENEX 2013, (2013).

[5] R. Thathoo and A. Virmani and S. S. Lakshmi and N. Balakrishnan and K. Sekar. TVSBS: A Fast Exact Pattern Matching Algorithm for Biological Sequences. J. Indian Acad. Sci., Current Sci., vol.91, n.1, pp.47--53, (2006).

[6] B. Durian and J. Holub and H. Peltola and J. Tarhio. Tuning BNDM with q-Grams. Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX 2009, pp.29--37, SIAM, New York, New York, USA, (2009).

[7] T. Lecroq. Fast exact string matching algorithms. ipl, vol.102, n.6, pp.229--235, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, (2007).

[8] H. Peltola and J. Tarhio. Variations of Forward-SBNDM. Proceedings of the Prague Stringology Conference '11, pp.3--14, Czech Technical University, Prague, Czech Republic, (2011).

[9] S. Deusdado and P. Carvalho. GRASPm. an efficient algorithm for exact pattern-matching in genomic sequences. Int. J. Bioinformatics Res. Appl., vol.5, n.4, pp.385--401, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, (2009).

[10] K.K.Senapati, Sandip Mal, G.Sahoo. RS-A Fast Pattern Matching Algorithm for Biological Sequences. International Journal of Engineering and InnovativeTechnology (IJEIT) Volume 1, Issue 3, March (2012).

[11] ftp://ftp.ncbi.nlm.nih.gov/genbank/genomes/Bacteria/

### Tables

Table 1.

Experimental results for searching different length of patterns on a ecoli genome sequence using six algorithm. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| new | **146.55** | **95.9** | **72.35** | **58.2** | **49.25** | **43.1** | **38.9** | **35.9** |  |
| epsm | 158.05 | 157.8 | 123.55 | 124.75 | 65.05 | 66.1 | 46.35 | 47.15 |  |
| tvsbs | 531.29 | 409.43 | 372.43 | 325.86 | 292 | 276.29 | 300.43 | 278.86 |  |
| ufndmq | 527.86 | 407.29 | 370 | 324.86 | 291 | 275.57 | - | 262.57 |  |
| hashq | 5022.86 | 1015.71 | 574.43 | 400.86 | 307.71 | 253.14 | 216.14 | 187 |  |
| fsbndmq | 376.57 | 296.14 | 226.14 | 184 | 152.71 | 142.14 | 126.29 | 120.57 |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| new | **33.7** | **32.25** | **30.95** | 29.2 | 26.4 | 41.25 | 32.1 | 17.65 | 11.65 |
| epsm | 37.2 | 37.5 | 31.3 | **25.9** | **21.45** | **17.95** | **13.4** | **10.95** | **9.15** |
| tvsbs | 245 | 256.71 | 244.29 | 268.71 | 277.14 | 293.86 | 270.57 | 301 | - |
| ufndmq | 225 | 274.86 | 286.71 | 243 | 254.57 | 274.86 | 280.43 | 254.71 | - |
| hashq | 167.14 | 149.57 | 138 | 103.86 | 63.71 | 45.29 | 43.71 | 40.29 | 40.29 |
| fsbndmq | 120.57 | 113.86 | 113.14 | 94.14 | 94.29 | 95.86 | 94 | 97.14 | 96.71 |

Table 2.

Experimental results for searching different length of patterns on a rice genome sequence using six algorithm. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| new | **148.05** | **96.3** | **71.95** | **58.5** | **49.8** | **44.05** | **39.6** | **36.4** |  |
| epsm | 161.5 | 164.45 | 122.6 | 124.25 | 65.15 | 65.9 | 46.75 | 47.4 |  |
| tvsbs | 502 | 385.86 | 371.86 | 369.71 | 306.14 | 283 | 281.86 | 260.71 |  |
| ufndmq | 500.29 | 385.57 | 370.29 | 368.43 | 304.14 | 283.29 | - | 293 |  |
| hashq | 5033 | 1016.14 | 570.86 | 399.71 | 308.71 | 252.71 | 213.71 | 186.43 |  |
| fsbndmq | 317 | 280.29 | 224.43 | 180 | 148.14 | 139.71 | 124.57 | 116.57 |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| new | **34.4** | **33.35** | **31.7** | 30 | 26.95 | 42.9 | 32.45 | 18.1 | 11.45 |
| epsm | 37.4 | 38.1 | 31.75 | **26.25** | **21.8** | **18.7** | **13.7** | **11.75** | **9.65** |
| tvsbs | 273.43 | 224 | 216 | 211.57 | 198.43 | 263.43 | 235.71 | 144.86 | - |
| ufndmq | 228 | 239.57 | 219.29 | 247.14 | 213 | 202 | 229.29 | 219.29 | - |
| hashq | 165.29 | 149.86 | 136.71 | 102 | 62.14 | 44.86 | 42.86 | 40 | 39.57 |
| fsbndmq | 121.43 | 117.86 | 107.43 | 93.43 | 92.14 | 94.43 | 91.71 | 91.43 | 93.71 |

Table 3.

Experimental results for searching different length of patterns on a human genome sequence using six algorithm. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| new | **146.5** | **95.9** | **71.75** | **57.85** | **49.25** | **42.9** | **39.05** | **35.9** |  |
| epsm | 163.05 | 159.2 | 123.45 | 125.6 | 65.5 | 66.8 | 47.1 | 47.35 |  |
| tvsbs | 509.71 | 426.86 | 370.86 | 302.86 | 299.71 | 281.86 | 284.29 | 293.71 |  |
| ufndmq | 505.43 | 424.29 | 369 | 301.29 | 297.29 | 280.43 | - | 281.43 |  |
| hashq | 5028.29 | 1015.71 | 571.71 | 400.57 | 308.86 | 252.29 | 214.57 | 186.57 |  |
| fsbndmq | 360.29 | 281.29 | 215.14 | 182.86 | 166.14 | 137.14 | 126.86 | 118.57 |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| new | **34** | **32.2** | **31.2** | 29.7 | 27.2 | 44.1 | 33.1 | 18.3 | 12.15 |
| epsm | 37.9 | 38.2 | 32.1 | **26.65** | **22.25** | **18.85** | **14.05** | **11.75** | **9.9** |
| tvsbs | 293.43 | 241.43 | 229.43 | 237.14 | 231.57 | 169.86 | 211 | 239.29 | - |
| ufndmq | 265.57 | 284.43 | 263.43 | 249.71 | 234.14 | 233.29 | 240.14 | 216.14 | - |
| hashq | 167 | 149 | 137.86 | 102.57 | 62.57 | 44.29 | 43 | 40.29 | 39.57 |
| fsbndmq | 120.14 | 117.14 | 113.43 | 91.14 | 96.29 | 96.14 | 95.43 | 96.29 | 93.43 |

Table 4.

Experimental results for searching different length of patterns on a protein genome sequence using six algorithm. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| new | 128.15 | **84.6** | **64.05** | **52.2** | **44.4** | **39.95** | **36.25** | **33.6** |  |
| epsm | **123** | 122.8 | 122.55 | 125.6 | 64.85 | 65.85 | 46.25 | 46.85 |  |
| tvsbs | 222.29 | 152 | 120.86 | 103.57 | 90.86 | 81.71 | 70.57 | 64.43 |  |
| ufndmq | 222.57 | 152.43 | 120.29 | 103.71 | 90.43 | 81 | - | 66.14 |  |
| hashq | 5018 | 1010.14 | 566.14 | 431.43 | 317.29 | 249.71 | 211.43 | 184.71 |  |
| fsbndmq | 126.86 | 86.86 | 68.57 | 57.86 | 52.14 | 46.57 | 43.57 | 40.29 |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| new | **32** | **30.65** | **29.9** | 28.1 | 29 | 40.4 | 29.6 | 17.05 | 10.8 |
| epsm | 37.25 | 37.6 | 31.5 | **26** | **21.8** | **18.6** | **13.4** | **10.7** | **8.9** |
| tvsbs | 59.14 | 54.71 | 50.86 | 42.14 | 31.57 | 29.29 | 31.57 | 28 | - |
| ufndmq | 60 | 56 | 50 | 43 | 31.86 | 28.71 | 31.43 | 29.14 | - |
| hashq | 163 | 147 | 133.71 | 100.14 | 59.71 | 42.71 | 41.43 | 39.43 | 39.43 |
| fsbndmq | 39.86 | 40.43 | 41.29 | 41.43 | 40.14 | 39.86 | 39.29 | 38 | 39 |

Table 5.

Experimental results for searching different length of patterns on a ecoli genome sequence using different length of packed string. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| 5 | 149.9 | 98.45 | 73.95 | 60.3 | 51.25 | 45.35 | 40.6 | 38.1 |  |
| 6 | **146.55** | **95.9** | **72.35** | **58.2** | **49.25** | **43.1** | **38.9** | **35.9** |  |
| 7 | 151.85 | 98.75 | 74.3 | 59.85 | 50.60 | 43.90 | 40.10 | 36.15 |  |
| 8 | 163.85 | 99.8 | 72.8 | 58.25 | 49.6 | 43.3 | 39.45 | 36.95 |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 5 | 35.35 | 34.6 | 32.6 | 30.15 | **26.2** | 42.35 | 44.15 | 37.35 | 32.75 |
| 6 | **33.7** | **32.25** | **30.95** | **29.2** | 27.45 | 41.05 | 35.4 | 21.15 | 15.6 |
| 7 | 33.80 | 32.90 | 31.45 | 29.7 | 26.5 | 40.65 | 32.35 | 26.40 | 18.35 |
| 8 | 34.4 | 32.9 | 31.85 | 29.3 | 26.4 | **41.25** | **32.1** | **17.65** | **11.65** |

Table 6.

Experimental results for searching different length of patterns on a rice genome sequence using different length of packed string. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| 5 | 149.7 | 98.1 | 73.3 | 59.65 | 50.9 | 44.85 | 40.85 | 37.55 |  |
| 6 | **148.05** | **96.3** | **71.95** | **58.5** | **49.8** | **44.05** | **39.6** | **36.4** |  |
| 7 | 152.15 | 99.3 | 74.3 | 60.15 | 50.65 | 44.0 | 39.64 | 37.25 |  |
| 8 | 163.35 | 99.05 | 72.5 | 58.15 | 49.4 | 43.55 | 39.75 | 36.75 |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 5 | 35.65 | 33.65 | 32.7 | 30.5 | 26.65 | 44.55 | 44.3 | 35.25 | 30.9 |
| 6 | **34.4** | **33.35** | **31.7** | 30 | 28.55 | 43.95 | 36.05 | 22.1 | 15.7 |
| 7 | 34.74 | 33.40 | 31.90 | 29.65 | **26.40** | 42.05 | 32.65 | 21.90 | 13.90 |
| 8 | 34.85 | 33.35 | 32.1 | **29.4** | 26.95 | **42.9** | **32.45** | **18.1** | **11.45** |

Table 7.

Experimental results for searching different length of patterns on a human genome sequence using different length of packed string. Running times are expressed in millisecond, best results have been boldfaced and padded.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |  |
| 5 | 149.45 | 97.35 | 73.85 | 59.8 | 51.3 | 45.15 | 40.85 | 38.3 |  |
| 6 | **146.5** | **95.9** | **71.75** | **57.85** | **49.25** | **42.9** | **39.05** | **35.9** |  |
| 7 | 153.05 | 99.15 | 73.80 | 60.05 | 50.07 | 44.00 | 39.30 | 36.15 |  |
| 8 | 164.35 | 99.45 | 73.05 | 58.7 | 49.65 | 43.75 | 40 | 37.3 |  |
|  |  |  |  |  |  |  |  |  |  |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 5 | 35.5 | 34.2 | 32.95 | 30.75 | 26.9 | 44.3 | 44.7 | 36.45 | 33 |
| 6 | **34** | **32.2** | **31.2** | **29.7** | 27.9 | **42.75** | 34.9 | 21.4 | 15.05 |
| 7 | 34.10 | 32.40 | 31.75 | 30.30 | 27.45 | 43.75 | 33.25 | 18.30 | 13.65 |
| 8 | 35.1 | 33.7 | 32.5 | 29.95 | **27.2** | 44.1 | **33.1** | **18.3** | **12.15** |

### Figure legends

Fig. 1 The algorithm.

### Figures

Fig. 1

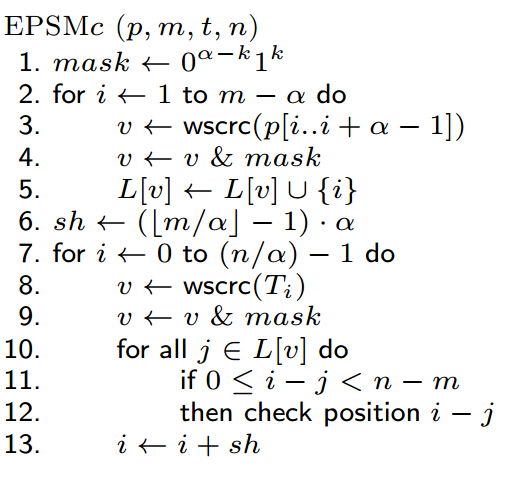


Fig. 2

