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 based on epsm 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 big hash value to decrease byte-by-byte comparisons. Experimental results show that the algorithm is more efficient than epsm especially when the pattern length is short.

Key words: Exact string matching, SIMD, biological sequence, epsm algorithm

### Introduction

String matching is an important issue 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 outstanding 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. Epsm 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, especially for biological sequences. In present study, we improve epsm by getting more shift distances and less byte-by-byte comparison calls.

### Material and methods

##### Algorithm

EPSM算法将模式串的所有8字节长的子串及其位置保存在hash表中。匹配时通过hash表检查当前窗口的最后8个字符，也就是块子符是否出现在模式中。并对所有这些可能的位置进行逐字节的匹配。由于采用了SIMD来计算字符串的hash值，所以块子符的长度固定为8字符。当前窗口处理完成后，跳跃m-8字符，继续下一次匹配。

Epsm algorithm record all the sub-string of fixed-length 8 bytes and their position in the hash table. In the searching phase, it checks last 8 bytes in the current searching window using hash table, i.e. to check whether the packed string exist in the pattern string and then compare all the possible position byte-by-byte. Due to the SIMD function which used to calculate hash value of string, apparently we must set length of packed string as 8 bytes. When finishing deal the current window, it jump forward m-8 bytes and prepare next preparation.

The basic idea to improve the epsm algorithm is to use longer jump distance and less comparisons byte-by-byte. The specific improvement is as following.

1. Optimize the size of packed string on biological sequences. In fact, jumping distance of the epsm is m-α, here α means the size of packed string. If taking smaller α, we will achieve bigger jumping distance. However, taking smaller α will result in more hash conflict and possible comparisons which will cause more checking. So we will take some experiments to find the optimization length of packed string in biological sequences.
2. Reducing byte-by-byte comparison between the pattern string in current window and matching string. The epsm uses a 32bit number mod the size of hash table to get the hash value. And using this number as the entrance of hash table and the trigger condition of byte-by-byte comparison. In general, taking bigger number as the trigger condition will trigger less byte-by-byte comparison. So we choose 64bit number as the trigger condition. It means to compare 64bit fingerprint of the current window with the matching string at the corresponding position. With this way, it will reduce the number of function calls memcmp. The specific implementation is adding a new field to record the 64bit number in Node and initialize it according to the pattern string in the pre-processing phase. Due to the SIMD function which used to calculate hash value of string, so we take 8 bytes to generate the fingerprint. The data structure of hash table is as follows.

Typedef struct node

{

struct node \*next;

intpos;

int64 fingerprint;

} NODE;

In the node, the fingerprint is introducing by the improved algorithm, which mark the fingerprint of sub-string of length 8 bytes on current position. It is not exist in the epsm algorithm.

对EPSM的改进的基本思想在于实施更大的跳跃距离和更少的逐字节比较调用。具体的改进有如下两点。

（1）对生物信息匹配的优化的块字符大小。事实上，EPSM的跳跃长度为m-B，此处B为块字符的长度。如果采用较小的块字符，即能获取较大的跳跃距离。但是采用较小块字符会造成hash更多的冲突，以及更多的可能匹配，这会导致更多的进一步的检查。我们将通过实验的方法去找到最优化的块字符长度。

（2）减少当前窗口中模式串和正文之间的逐字节比较操作。EPSM采用32bit的hash值再mod hash表大小作为hash表入口和逐字节比较调用的触发条件。理论上，采用更大的数值作为逐字节比较调用的触发条件，就会触发更少的逐字节比较调用。所以我们采用64bit的触发条件。在逐字节比较调用前比较当前窗口的64bit的指纹与模式串相应位置子串的64bit指纹值，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，在预处理阶段根据模式串将其初始化。因为采用SIMD指令高速的计算指纹，我们采用8个字符来生成改指纹。Hash表的项目数据结构如下图所示：

{

Typedef struct node

{

struct node \*next;

intpos;

int64 fingerprint;

} NODE;

在图中，fingerprint为改进算法引入的，标识当前位置长度为

8的子串的指纹。该项在EPSM中不存在。

The algorithm is shown in Fig.1.

##### 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 at3.30GHz, 4G memory, running Linux Mint 13. The size of hash table in the improved algorithm equals to which in the epsm algorithm, i.e. 2048.

改进算法的hash表大小与EPSM保持一致，即2048.

##### 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. Although the dictionary size of biological sequences is 4 or 20 and it performance as random string, we can take the packed string. However length of 8 bytes which epsm take will still remain result in big hash conflict when we choose 2048 as our array size. Thus, unlike epsm 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.Wecanseethe influence of the size of packed string. And from the results, we did find out that length 6 was a suitable choice.

##### 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 difference of performance is much larger when come with short string, beat all the others in all cases.

This results were as expected. Because of the SIMD instruction, we could obtain great performance when sampling the string. So we could beat the other four algorithms which was not using the SIMD instructions.

And our algorithm could jump the distance of m-α. So the algorithm beat the epsm especially in short string.

### Conclusions

We improve the epsm algorithm in biological sequences. And the experiments show the algorithm obtain the best performance.

### ~~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 an 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 | **57.05** | **56.85** | **31.7** | **25.3** | **21.25** | **19.15** | **18.05** | **17.55** |
| 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 | **16.9** | **17.05** | **16.15** | **15.1** | **15.4** | **8.85** | **5.65** | **3.7** | **2.85** |
| 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 | **62.9** | **60** | **31.45** | **24.6** | **21.15** | **19.75** | **18.3** | **17.55** |
| 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 | **16.55** | **16.3** | **15.75** | **14.9** | **15** | **8.8** | **5.55** | **3.75** | **3.05** |
| 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 | **61.35** | **57.4** | **31.45** | **25.65** | **21.55** | **19.7** | **18.65** | **17.5** |
| 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 | **16.9** | **16.55** | **15.85** | **15.6** | **15.45** | **8.95** | **5.9** | **3.95** | **3.05** |
| 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 | **27.85** | **28.35** | **30.4** | **32.2** | **20.2** | **21.5** | **18.1** | **19.25** |
| 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 | **16.8** | **17.1** | **16.9** | **15.5** | **14.25** | **10.05** | **7.3** | **6.3** | **6.7** |
| 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 an 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 | 62.85 | 60.2 | 35.45 | 30 | 27.65 | 25.95 | 23.85 | 23.7 |
| 6 | **57.05** | **56.85** | **31.7** | **25.3** | **21.25** | **19.15** | **18.05** | **17.55** |
| 7 | 60.95 | 58.7 | 33.2 | 25.3 | 21.45 | 20.55 | 18.85 | 18.35 |
| 8 | 62.65 | 59.1 | 35.15 | 25.5 | 22.7 | 20.95 | 19.8 | 18.65 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 5 | 22.85 | 21.8 | 21.1 | 20.1 | 19.15 | 16.5 | 16.95 | 18 | 14.95 |
| 6 | 17.9 | 17.35 | 17.05 | 16.3 | 15.95 | 10.6 | 8.3 | 7.35 | 6.6 |
| 7 | 17.15 | 16.45 | 16.2 | 15.55 | 14.5 | 9.7 | 6.1 | 4.7 | 3.7 |
| 8 | **16.9** | **17.05** | **16.15** | **15.1** | **15.4** | **8.85** | **5.65** | **3.7** | **2.85** |

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 | 66.95 | 64.05 | 35.2 | 29.2 | 26.85 | 25.5 | 23.55 | 23.15 |
| 6 | **62.9** | **60** | **31.45** | **24.6** | **21.15** | **19.75** | **18.3** | **17.55** |
| 7 | 64.95 | 62.7 | 32.9 | 25.2 | 21.15 | 19.75 | 18.6 | 18.4 |
| 8 | 65.4 | 63.7 | 35.4 | 25.7 | 22.05 | 21.15 | 19.45 | 18.4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 5 | 22.25 | 21.15 | 20.85 | 19.9 | 18.75 | 16.35 | 15.25 | 17.85 | 14.65 |
| 6 | 18.3 | 18.2 | 17.7 | 16.1 | 15.2 | 11.25 | 7.65 | 6.7 | 7.45 |
| 7 | 17.35 | 16.6 | 16.25 | 15.35 | 14.6 | 10.25 | 6.15 | 4.6 | 3.85 |
| 8 | **16.55** | **16.3** | **15.75** | **14.9** | **15** | **8.8** | **5.55** | **3.75** | **3.05** |

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 | 65.45 | 61.2 | 36.2 | 30.5 | 27.75 | 26.2 | 24.6 | 23.6 |
| 6 | **61.35** | **57.4** | **31.45** | **25.65** | **21.55** | **19.7** | **18.65** | **17.5** |
| 7 | 63.65 | 59.6 | 32.8 | 25.65 | 21.55 | 19.7 | 18.65 | 18.6 |
| 8 | 64.75 | 60.05 | 32.8 | 25.65 | 22.4 | 20.7 | 19.65 | 19.4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 40 | 44 | 48 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 5 | 22.6 | 22.05 | 21.7 | 20.2 | 18.85 | 17.5 | 17.35 | 18 | 15.65 |
| 6 | 18.6 | 17.65 | 17.55 | 16.65 | 15.35 | 11.1 | 8.25 | 8.55 | 7.7 |
| 7 | 18 | 17 | 16.75 | 15.25 | 14.75 | 10.6 | 6.4 | 4.85 | 4.1 |
| 8 | **16.9** | **16.55** | **15.85** | **15.6** | **15.45** | **8.95** | **5.9** | **3.95** | **3.05** |

### Figure legends

Fig. 1 The algorithm.

### Figures

Fig. 1

hash(p)

1. crc = \_\_mm\_crc32\_u64(seed,p)
2. h = crc&mask

pre(P,m,T,n)

1. for i ← 0 to m-α do
2. h = hash(P+i)
3. shift[h].pos = i
4. shift[h].fingerprint = (ulong)P+i

search(P,m,T,n)

1. For i ← 0 to n-α do
2. h = hash(T+i)
3. f = (ulong)T+i
4. p = shift[h]
5. while(p!=null)
6. If f==p.fingerprint
7. memcmp(P,T)
8. p = p.next