Computers and Mathematics with Applications 62 (2011) 2162–2168

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

A logarithmic time complexity algorithm for pattern searching using product–sum property

Khalid Mahmood Aamir [a], Mujahid Abbas [b], Stojan Radenović [c,*]

[a] Department of Computer Sciences, Rachna College of Engineering and Technology, Gujranwala, Pakistan
[b] Center for Advanced Studies in Mathematics, Lahore University of Management Sciences, Lahore, Pakistan
[c] Faculty of Mechanical Engineering, University of Belgrade, Kraljice Marije 16, 11 120 Beograd, Serbia

ARTICLE INFO

Article history:
Received 30 November 2010
Accepted 5 July 2011

Keywords:
Product–sum property
Database search
Logarithmic complexity algorithm

ABSTRACT

Product–sum property states that an ordered pair (s_n, p_n) is unique for any ordered set $a_1, a_2, \ldots, a_n$ where $a_i, n \in \mathbf{N}$, and s_n and p_n are the sum and product of the elements of the set, respectively. This fact has been exploited to develop an $O(\log(M))$ time complexity algorithm for pattern searching in a large dataset, where M is the number of records in the dataset. Two potential applications (from databases and computational biology) of this property have been demonstrated to show the effectiveness and working of the proposed algorithm. The space complexity of the algorithm rises to the quadratic order.

1. Introduction

Deriving useful knowledge from huge data and its further processing is an active research area in the field of computer science. It is always desired to discover specific patterns, in large databases, relevant to the user's need [1]. Some statistical algorithms, for pattern searching, may generate some irrelevant patterns that may give some misleading information in information processing systems. Therefore, the quality of knowledge thus developed depends on the closeness of such patterns to the specific patterns. To extract effective information from a huge amount of data in databases, algorithms must be efficient. A lot of research activity has been carried out in this direction; see for example [2–11].

In the existing literature in this area, Grover's algorithm [12] is regarded as the most efficient one. It works on an unsorted database of M records out of which only one item satisfies a given condition and only one record is retrieved. Grover's algorithm scans at an average of $M/2$ records before finding the desired record, and its computational complexity is $O(\sqrt{M})$.

Given that a database contains M records, a problem of identification of records of which one or more satisfy a particular property, is considered. An algorithm presented for this purpose is faster than Grover's algorithm. Our algorithm is based on the product–sum property of sets. Grover's algorithm identifies only unique records whereas our algorithm searches for queries demanding more than one record, in logarithmic computational complexity in terms of the number of records in the database. A utility of our algorithm, in search of a continuous string of characters appearing in genomic data, is also established.

The product–sum property is defined and some relationships are derived. A logarithmic complexity algorithm, to scan databases using the product–sum property, is presented. Then, two examples, one from databases and the other from computational genetics, are demonstrated to show the effectiveness and working of the algorithm.

* Corresponding author.
E-mail addresses: kmaamir@lums.edu.pk (K.M. Aamir), mujahid@lums.edu.pk (M. Abbas), radens@beotel.rs, sradenovic@mas.bg.ac.rs (S. Radenović).

0898-1221/$ – see front matter
doi:10.1016/j.camwa.2011.07.001

2. Product–sum property

Suppose that we have some data, call it D_1, arranged in the form of m rows and n columns. So, the size of data will be $m \times n$. By an n-dimensional ordered data, we mean a set of n natural numbers, $\{a_1, a_2, a_3, \dots, a_n\}$ (say) such that $a_1 \le a_2 \le a_3 \le \cdots \le a_n$. Such a set may be viewed as a row in D_1, we call it a record in D_1. An ordered pair of natural numbers (n_1, n_2) (say) will be termed as two dimensional data. The main idea is to transform D_1 to two dimensional data D_2 whose size is $m \times 2$. The purpose of this paper is to construct an algorithm whose search space is D_2 instead of D_1 to check the presence of some specific record in D_1. For example, we consider two rows $A = \{2, 4, 512\}$ (say S_3^A) and $B = \{8, 16, 32\}$ (say S_3^B) from the given data of size 5×3. We map S_3^A and S_3^B to $(4096, 520)$ and $(4096, 56)$, respectively. Thus corresponding to each row in the data, we find an ordered pair.

Definition. Two subsets A and B of set of all positive real numbers are said to have product–sum property if the followings hold:

$$\sum_{a\in A} a = \sum_{b\in B} b \tag{1}$$

and

$$\prod_{a\in A} a = \prod_{b\in B} b. \tag{2}$$

For example, $A_4 = \{1, 2, 3, 4\}$ and $B_2 = \{4, 6\}$ satisfy product–sum property. Note that cardinalities of these two sets are not same. On the other hands, note that if two rows $A_n = \{a_1, a_2, a_3, \dots, a_n\}$ and $B_n = \{b_1, b_2, \dots, b_n\}$ in D_1 have product–sum property then $a_i = b_i$ for each $i \in \{1, 2, \dots, n\}$ (we denote product and sum of n elements from a set A_n by p_n^A and s_n^A, respectively). This assertion is proved using mathematical induction as follows: For $n = 1$, proof is trivial.
For $n = 2$, consider

$$A_2 = \{a_1, a_2\} \quad \text{and} \quad B_2 = \{b_1, b_2\}. \tag{3}$$

Given that $p_2^A = p_2^B$ and $s_2^A = s_2^B$, where

$$p_2^A = a_1 \cdot a_2 \quad \text{and} \quad p_2^B = b_1 \cdot b_2$$
$$s_2^A = a_1 + a_2 \quad \text{and} \quad s_2^B = b_1 + b_2.$$

Now $a_1 + a_2 = b_1 + b_2$ implies that

$$a_1 = b_1 + b_2 - a_2. \tag{4}$$

Using (4) and $p_2^A = p_2^B$ we obtain

$$(b_1 + b_2 - a_2)a_2 = b_1 \cdot b_2$$

which further implies that

$$a_2^2 - (b_1 + b_2)a_2 + b_1 b_2 = 0. \tag{5}$$

Keeping b_1 and b_2 arbitrarily fixed, (5) becomes quadratic in a_2. Solving (5) we arrive at either $a_2 = b_1$ or $a_2 = b_2$. Since A_2 and B_2 are two rows therefore $a_1 = b_1$ and $a_2 = b_2$ and hence the result follows. Suppose that $A_k = \{a_1, a_2, \dots, a_k\}$ and $B_k = \{b_1, b_2, \dots, b_k\}$ with

$$s_k^A = \sum_{i=1}^{k} a_i = \sum_{i=1}^{k} b_i = s_k^B \quad \text{and} \quad p_k^A = \prod_{i=1}^{k} a_i = \prod_{i=1}^{k} b_i = p_k^B \tag{6}$$

always give $a_i = b_i$ for $1 \le i \le k$.

Consider $A_{k+1} = \{a_1, a_2, \dots, a_k, a_{k+1}\}$ and $B_{k+1} = \{b_1, b_2, \dots, b_k, b_{k+1}\}$ then

$$p_{k+1}^A = a_{k+1} \prod_{i=1}^{k} a_i = b_{k+1} \prod_{i=1}^{k} b_i = p_{k+1}^B \quad \text{and}$$

$$s_{k+1}^A = a_{k+1} + \sum_{i=1}^{k} a_i = b_{k+1} + \sum_{i=1}^{k} b_i = s_{k+1}^B$$

immediately implies $a_{k+1} = b_{k+1}$. This proves the assertion.

Geometric interpretation: Consider a hyperbola C given by $xy = p$. Let $P(x_1, y_1)$ be a point on C such that

$$x_1 < y_1. \tag{7}$$

Since equation of a line passing through $P(x_1, y_1)$, having a slope of -1, is

$$y - y_1 = (-1)(x - x_1)$$

which implies that:

$$x + y = x_1 + y_1 = s \quad \text{(say)}. \tag{8}$$

Also:

$$x_1 y_1 = p. \tag{9}$$

Eqs. (8) and (9) can also be satisfied by a point (y_1, x_1). Since $x_1 < y_1$, therefore selection of such a point $P(x_1, y_1)$ is unique.

For N-dimensional hyperboloid bisected by an N-dimensional plane, there are infinity many possible points satisfying Eqs. (1) and (2). Due to extension of condition (7) to N-dimensions, there is only one point satisfying Eqs. (1) and (2).

3. Algorithm

Before processing of algorithm, database is preprocessed as outlined below.

3.1. Preprocessing

Preprocessing includes the following steps.

(1) Construction of code table.
(2) Coding of database.
(3) Construction of product–sum table.
(4) Sorting the product–sum table.

Detailed algorithms of the above preprocessing steps are as follows.

3.1.1. Algorithm for construction of code table

An algorithm for construction codes for the database is as follows.

Input *database*.
Output List of codes of unique items in the database, i.e., *unique_items_codes*.
Algorithm Algorithm is as follows:

(1) Define list of unique items in the database as *unique_items*[].
(2) For each column of the database table:
  - Search all unique items in the column and append them to the list *unique_items*.
(3) Find the length of *unique_items*, say K. Hence, size of *unique_items_codes*[] is also K.
(4) For $i = 1$ to K,
  - *unique_items_codes* [i] $= c$, where c is an integer and $c > 1$ (for example, $c = i + 1$).
(5) Sort *unique_items_codes* (for efficient use only).

3.1.2. Algorithm for coding of database

Input *unique_items_codes*, *database*.
Output A matrix of $M \times N$ size, *coded_data* = [][] having the codes of each field.
Algorithm For all Fields of all the records of *database*[][], i.e., $i = 1$ to M and $j = 1$ to N,

- Take a field of d of *database* [i][j] and find the code c of d in *unique_items_codes*.
- *coded_data* [i][j] $= c$.

3.1.3. Construction of product–sum table

Input Two matrices of $M \times N$ and $M \times 2$ sizes, i.e., *coded_data* = [][] having the codes of each field and an empty matrix B[][] having products and sums, respectively.
Output Matrix B[][] having products and sums.
Algorithm For $i = 1$ to M.

- $B[i][1]$ = product of all the numbers in the ith row of *coded_data*.
- $B[i][2]$ = sum of all the numbers in the ith row of *coded_data*.

3.1.4. Sorting the product–sum table

Input Matrix B[][] having products and sums.
Output An $M \times 3$ matrix *ps*[][] having products and sums, sorted and preserving index of occurrence in B.
Algorithm Sort the matrix B (with respect to B[][1] and then B[][2]) and copy values to *ps* such that *ps*[i][1], *ps*[i][2] and *ps*[i][3] have product, sum and index, respectively.

Table 1
An example database.

Day	Outlook	Temperature	Humidity	Wind	Tennis Play
D_1	Sunny	Hot	High	Weak	No
D_2	Sunny	Hot	High	Strong	No
D_3	Overcast	Hot	High	Weak	Yes
D_4	Rain	Mild	High	Weak	Yes
D_5	Rain	Cool	Normal	Weak	Yes
D_6	Rain	Cool	Normal	Strong	No
D_7	Overcast	Cool	Normal	Strong	Yes
D_8	Sunny	Mild	High	Weak	No
D_9	Sunny	Cool	Normal	Weak	Yes
D_{10}	Rain	Mild	Normal	Weak	Yes
D_{11}	Sunny	Mild	Normal	Strong	Yes
D_{12}	Overcast	Mild	High	Strong	Yes
D_{13}	Overcast	Hot	Normal	Weak	Yes
D_{14}	Rain	Mild	High	Strong	No

Table 2
Code table for database shown in Table 1.

Term	Numeric code	Term	Numeric code
Sunny	2	High	17
Overcast	3	Normal	19
Rain	5	Weak	23
Hot	7	Strong	29
Mild	11	Yes	31
Cool	13	No	37

Table 3
Sorted code table.

Term	Numeric code	Term	Numeric code
Cool	13	Overcast	3
High	17	Rain	5
Hot	7	Strong	29
Mild	11	Sunny	2
No	37	Weak	23
Normal	19	Yes	31

3.2. Query searching algorithm

Searching a record in the database is carried out as follows.

Input Query q, *database* and *ps*.
Output I as a set of indices of occurrence of query record in the database.
Algorithm Search of the input query q is processed as follows.

(1) Assign numeric codes to q using *unique_items* and *unique_items_codes*, in *coded_q*.
(2) Calculate $a =$ product of all numbers in *coded_q*.
(3) Calculate $b =$ sum of all numbers in *coded_q*.
(4) Using binary search, find a and b in *ps* and assign all corresponding *ps*[][3] to I.

3.3. Computational complexity of query searching algorithm

Assigning the numeric codes to the query takes computational time of the order $O(MN)$ in worst case. A step of sorting the *unique_items_codes* using binary search algorithm, reduces it to $O(\log(MN)) = O(\log(M) + \log(M)) = O(\log(\max(M, N)))$, and usually $M \gg N$. Calculation of a and b requires $O(N)$. Furthermore, binary search requires a computational time of $O(\log(M))$. Therefore, overall computational complexity turns out to be $O(\log(M))$.

4. Applications

Databases. Consider an example database shown in Table 1 (taken from [13]). Note that number of rows M and columns N are 14 and 5, respectively. Then computational complexity turns out to be $O(MN)$ in worst case, if each row is matched with a query.

Preprocessing.

Let us assign the numerics to the terms in Table 1 as shown in Tables 2 and 3.

Generating coded database and its products–sums, we obtain Eq. (8).

Query search.

$$\underset{\text{Coded data table}}{\begin{pmatrix} 2 & 7 & 17 & 23 & 37 \\ 2 & 7 & 17 & 29 & 37 \\ 3 & 7 & 17 & 23 & 31 \\ 5 & 11 & 17 & 23 & 31 \\ 5 & 13 & 19 & 23 & 31 \\ 5 & 13 & 19 & 29 & 37 \\ 3 & 13 & 19 & 29 & 31 \\ 2 & 11 & 17 & 23 & 37 \\ 2 & 13 & 19 & 23 & 31 \\ 5 & 11 & 19 & 23 & 31 \\ 2 & 11 & 19 & 29 & 31 \\ 3 & 11 & 17 & 29 & 31 \\ 3 & 7 & 19 & 23 & 31 \\ 5 & 11 & 17 & 29 & 37 \end{pmatrix}} \Longrightarrow \underset{\text{Product–sum table}}{\begin{pmatrix} 202\,538 & 86 \\ 255\,374 & 92 \\ 254\,541 & 81 \\ 666\,655 & 87 \\ 880\,555 & 91 \\ 1\,325\,155 & 103 \\ 666\,159 & 95 \\ 318\,274 & 90 \\ 352\,222 & 88 \\ 745\,085 & 89 \\ 375\,782 & 92 \\ 504\,339 & 91 \\ 284\,487 & 83 \\ 1\,003\,255 & 99 \end{pmatrix}} \Longrightarrow \underset{\text{Sorted product–sum table.}}{\begin{pmatrix} 202\,538 & 86 & 1 \\ 254\,541 & 81 & 3 \\ 255\,374 & 92 & 2 \\ 284\,487 & 83 & 13 \\ 318\,274 & 90 & 8 \\ 352\,222 & 88 & 9 \\ 375\,782 & 92 & 11 \\ 504\,339 & 91 & 12 \\ 666\,159 & 95 & 7 \\ 666\,655 & 87 & 4 \\ 745\,085 & 89 & 10 \\ 880\,555 & 91 & 5 \\ 1\,003\,255 & 99 & 14 \\ 1\,325\,155 & 103 & 6 \end{pmatrix}} \tag{10}$$

Take a query, for example, $q_1 = [\textit{Overcast Mild High Strong Yes}]$. We convert the fields of query to numeric form $[3 \quad 11 \quad 17 \quad 29 \quad 31]$.Product and sum become (504 339, 91). Using binary search, search 504 339 in the *ps* array is found in the row 8. The index given in row 8 is found to be 12. This implies that record D_{12} in the database has exact match with q_1.

For such a process, Grover's algorithm requires $O(\sqrt{M})$ computations, whereas proposed algorithm requires $O(\log(M))$ computations. In the rest of this section, we explore some potentials of the proposed algorithm which Grover's algorithm does not exhibit.

Consider a query q_2 which asks for all the records with *Play Tennis*= '*yes*'. Code for '*yes*' in the code table is 31. We divide all values in the first column of sorted product–sum table by 31. We output the indices corresponding to which division process produces a zero remainder, i.e., $[3 \quad 13 \quad 11 \quad 12 \quad 7 \quad 4 \quad 10 \quad 5]$,which are $[3 \quad 4 \quad 7 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13]$when sorted. Therefore, records D_3, D_4, D_7, D_9, D_{10}, D_{11}, D_{12} and D_{13} are the days when tennis was played.

In case, we want to replace a record [*Overcast Cool Normal Strong Yes*] by a new record [*Sunny Cool Normal Strong No*], whose numeric form is $[2 \quad 13 \quad 19 \quad 29 \quad 37]$. Firstly, we find the location of the record in the database using the scheme as mentioned above. This is record number 7. Replace the record at row number 7 with the new record in the database. Now we identify which row in the last column of the *ps* table bas an index value 7. This is the 10th row of *ps* table. Now, we replace the 10th row of *ps* table $[24\,948 \quad 42 \quad 7]$by $[530\,062 \quad 100 \quad 7]$.Note that a replacement is a composition of a deletion and then an insertion.

4.1. *Computational biology*

Consider two DNA Sequences as follows:

DNA1: **aattatcatatcctgtaattgtttgatattgatttgcaaaat**.
DNA2: **tttgat**.
Length of DNA 1: M.
Length of DNA 2: N.

Matching technique is similar to convolution and its overall computational complexity is $O(NM)$.

Consider all possible sequences of DNA 1 and DNA 2 in the form of codons:

Possible sequences of DNA 1:

(1) **aat tat cat atc ctg taa ttg ttt gat att gat ttg caa aat**.
(2) a **att atc ata tcc tgt aat tgt ttg ata ttg att tgc aaa** at.
(3) aa **tta tca tat cct gta att gtt tga tat tga ttt gca aaa** t.

Possible sequences of DNA 2:

- **ttt gat**.

Preprocessing.

Generate a code table as shown in Table 4. Now, we generate product–sum tables for all possible codon sequences of DNA 1 (see Tables 5–7).

Comparing the DNAs.

Assign codes to codons of DNA 2, i.e., (ttt, gat) $\rightarrow$ (23, 10). This requires a complexity of $O(N)$. Now, we calculate (product, sum) with a complexity of $O(N)$, i.e., (product, sum) = (230, 33). Search this pair in the table product–sum for

Table 4
Code table for DNA sequences.

Codon	Numeric code	Codon	Numeric code
aaa	1	gtt	13
aat	2	taa	14
ata	3	tat	15
atc	4	tca	16
att	5	tcc	17
caa	6	tga	18
cat	7	tgc	19
cct	8	tgt	20
ctg	9	tta	21
gat	10	ttg	22
gca	11	ttt	23
gta	12		

Table 5
Code assignment and product–sum for DNA sequence 1: **aat tat cat atc ctg taa ttg ttt gat att gat ttg caa aat**.

Codon	Code	Codon	Code	Product	Sum
aat	2	tat	15	30	17
tat	15	cat	7	105	22
cat	7	atc	4	28	11
atc	4	ctg	9	36	13
ctg	9	taa	14	126	23
taa	14	ttg	22	308	36
ttg	22	ttt	23	506	45
ttt	23	gat	10	230	33
gat	10	att	5	50	15
att	5	gat	10	50	15
gat	10	ttg	22	220	32
ttg	22	caa	6	132	28
caa	6	aat	2	12	8

Table 6
Code assignment and product–sum for DNA sequence 2: a **att atc ata tcc tgt aat tgt ttg ata ttg att tgc aaa** at.

Codon	Code	Codon	Code	Product	Sum
att	5	atc	4	20	9
atc	4	ata	3	12	7
ata	3	tcc	17	51	20
tcc	17	tgt	20	340	37
tgt	20	aat	2	40	22
aat	2	tgt	20	40	22
tgt	20	ttg	22	440	42
ttg	22	ata	3	66	25
ata	3	ttg	22	66	25
ttg	22	att	5	110	27
att	5	tgc	19	95	24
tgc	19	aaa	1	19	20

Table 7
Code assignment and product–sum for DNA sequence 3: aa **tta tca tat cct gta att gtt tga tat tga ttt gca aaa** t.

Codon	Code	Codon	Code	Product	Sum
tta	21	tca	16	336	37
tca	16	tat	15	240	31
tat	15	cct	8	120	23
cct	8	gta	12	96	20
gta	12	att	5	60	17
att	5	gtt	13	65	18
gtt	13	tga	18	234	31
tga	18	tat	15	270	33
tat	15	tga	18	270	33
tga	18	ttt	23	414	41
ttt	23	gca	11	253	34
gca	11	aaa	1	11	12

DNA 1 with a complexity of $O(M)$ and for sorted product–sum tables, searching costs $O(\log(M))$. Therefore, the overall computational complexity turns out to be $O(\log(M))$.

Discussions and conclusions.

The proposed algorithm is based on the product–sum property and it is demonstrated that using the product–sum property searching a unique record in the database requires logarithmic computational complexity time. This reduction in computational time poses a burden in the form of increased space complexity. Space complexity to save the code table and (product, sum) values are $O(MN)$ and $O(M)$, respectively. Therefore, space complexity rises from linear to quadratic order. Note that insertions to and deletions from the database can be performed with the logarithmic computational time complexity.

We compare the computational complexity of the proposed algorithm with the Grover's algorithm [12] whose computational complexity is $O(\sqrt{M})$. Consider that the number of records is 2^p where p is an even integer. Complexity of Grover's algorithm becomes $O(2^{p/2})$ and complexity of our algorithm becomes $O(p)$. We know that $p < 2^{p/2}$, for $p > 4$. Therefore, our algorithm's complexity is lesser than that of Grover's algorithm if number of records is more than $2^4 = 16$.

If numeric codes are natural numbers, then unique queries can be retrieved only. However, if the numeric codes are relatively prime numbers, then any query asking a particular or some specific characteristics can also be retrieved. A problem appears with relatively prime numbers is that sizes of numbers grow rapidly. A solution to this problem may be taking the logarithms of the numbers rather than using the numbers themselves.

Acknowledgments

This work was done while the first author was visiting the Center for Advanced Studies in Mathematics (CASM), Lahore University of Management Sciences (LUMS), Lahore, Pakistan as a post-doctoral fellow. The third author is grateful to the Ministry of Science and Technological Development of Serbia.

References

[1] S. Piramuthu, H.M. Chung, Data mining and information retrieval, 2003, p. 67.
[2] S.S. Anand, D.A. Bell, J.G. Hughes, EDM—a general framework for data mining based on evidence theory, Data & Knowledge Engineering 18 (1996) 189–223.
[3] J.F. Baldwin, Knowledge from data using fuzzy methods, Pattern Recognition Letters 17 (1996) 593–600.
[4] C. Faloutsos, K.-I. Lin, Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets, SIGMOD Record 24 (1995) 163–174.
[5] E. Keogh, P. Smyth, A probabilistic approach to fast pattern matching in time series databases, 1997, p. 24.
[6] R.P. Gopalan, Y.G. Sucahyo, Improving the efficiency of frequent pattern mining by compact data structure design, 2003, pp. 576–583.
[7] V.S. Ananthanarayana, M.N. Murty, D.K. Subramanian, Tree structure for efficient data mining using rough sets, Pattern Recognition Letters 24 (2003) 851–862.
[8] T. Mielikäinen, Summarization techniques for pattern collections in data mining (nonpeerreviewed), Department of Computer Science, University of Helsinki, Finland, 2005.
[9] F.C. Gey, H.M. Chung, Data mining, knowledge discovery, and information retrieval—minitrack introduction, 2001.
[10] S.Y. Chen, X. Liu, Data mining from 1994 to 2004: an application-orientated review, International Journal of Business Intelligence and Data Mining 1 (2005) 4–21.
[11] Papadimitriou, Novel computational approaches to information retrieval and data mining, 1999.
[12] L.K. Grover, A Fast Quantum Mechanical Algorithm for Database Search, ACM, New York, NY, USA, 1996, pp. 212–219.
[13] T.M. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.