A Method of Fast Update of Absolute Central Sample Moments of Time Series*

D. Z. Rybalko [I,III], K. B. Bulatov [II,III], D. V. Polevoy [II,III]

[I] National University of Science and Technology MISIS, Moscow, Russia
[II] Federal Research Centre "Informatics and control" of Russian Academy of Science, Moscow Russia
[III] Smart Engines Service LLC, Moscow Russia

Abstract. The methods of updating central moments are often covered in various works where the large updating samples are present. However, absolute central moments of odd orders still remain unaddressed. In online systems and systems that are highly dependent on data transfer speed the issue of updating the absolute central moment of a time series on a certain constantly updating sample often comes up. In this paper we will propose a method for fast update of absolute central moments of time series and its programmatic implementation based on the treap data structure.

Keywords: absolute central moments, treap, online systems, central moments.

DOI 10.14357/20718632210101

Introduction

When processing large quantities of numerical data, building predictive models and solving image processing- and computer vision-related problems, the moments of time series and samples are applied as relevant features of objects or phenomena [1]. When developing the algorithms that analyze changes of the characteristics of the objects over time, the challenge of updating the values of the moments of samples arises, i.e., the problem of recalculation of the values of the moments with the increase (or decrease) of the sample size. While there are effective algorithms for updating the initial and central moments of samples, the problem of updating the absolute moments turns out to be more complex. The objective of this work is to develop an effective method for updating the absolute central moments of the increasing samples and to study it experimentally.

We will define a sample as a set of values generated during a certain number of observations of the time series. Suppose we are given a sample $A = \{a_1, ..., a_n\}$. **A sample moment** of order k is its numerical characteristic that:

1. When it's **initial**, is calculated using the formula $\nu_k(A) = \frac{1}{n}\sum_{i=1}^{n} a_i^k$,

2. When it's **central**, is calculated using the formula $\mu_k(A) = \frac{1}{n}\sum_{i=1}^{n}\left(a_i - \nu_1(A)\right)^k$.

The following characteristics are called the absolute and the central absolute sample moments, respectively:

* This work was partially financially supported by RFBR, projects №№ 19-29-09055, 19-29-09092.

3. $\tilde{\nu}_k(A) = \frac{1}{n}\sum_{i=1}^{n}|a_i|^k$,

4. $\tilde{\mu}_k(A) = \frac{1}{n}\sum_{i=1}^{n}|a_i - \nu_1(A)|^k$.

It is also apparent that 1-st initial moment is equal to sample mean of A, and 2-nd central moment is equal to a sample variance of A. Also, absolute central moments of even orders and moments of same even orders are same, that being $\nu_k(A) = \tilde{\nu}_k(A)$, $\mu_k(A) = \tilde{\mu}_k(A)$ for even k.

Let's assume that at any point in time a new value of some time series is observed. This means that at any point in time there is a sample A_n of such a size n, where n is the number of previous observations and it generates a sequence of samples of increasing size.

Let's say that the process that generates a number a_{n+1} at each new stage is being considered. This process produces a sequence of increasing samples $A_1, A_2, \ldots, A_{n+1}$, for each of which it is necessary to get an absolute central moment of some order.

When we get a new element of a sample, the updated sample mean can be calculated using the following formula:

$$\nu_1(A_{n+1}) = \frac{n \cdot \nu_1(A_n) + a_{n+1}}{n+1}. \quad (1)$$

On the other hand, there is no general form of such formulas for the third-order and higher-order moments due to the nature of the formulas for calculating the moments of time series, in which each component changes (since each component is the equation $(a_i - \nu_1(A_{n+1}))^k$, while previously this component $(a_i - \nu_1(A_n))^k$ was equal), whereas with the sample mean there is just one component added. And this raises the problem of fast update of the absolute moment of a sample with the regards to the procedure of recalculating each of the $n+1$ components.

Existing solutions

The obvious way to update the absolute moments of time series is to update the mean of a sample, first, by applying the formula (1) to it, and then calculate the moment itself:

$$\tilde{\mu}_k(A_{n+1}) = \sum_{i=1}^{n+1}|a_i - \bar{a}_{n+1}|^k. \quad (2)$$

If exponentiation k is performed using the binary exponentiation algorithm [2], which complexity is equal to $O(\log k)$, then the complexity of the algorithm for updating the value of the absolute central moment at an n+1 stage of the process will be $O(n \cdot \log k)$.

The effectiveness and accuracy results of a few increment formulas that are applied within the systems with high demands for accuracy and performance speed were produced in this work [3]. The formulas and methods reviewed in this article were generated due to the idea of dividing the entire sample into two sets, and as a result, the increment formulas themselves were generated through a special case of the derived formulas, when the first set is the entire previous sample, and the second one contains just one new element.

Also the formula presented in the work [3]

$$\mu_k(A) = \sum_{i=1}^{k} C_k^i \cdot \left(\frac{1}{n}\sum_{i=1}^{n} a^{p-k}\right) \cdot (-\bar{a})^k, \quad (3)$$

which is a result of a simple expansion of the central moment formula according to which the binomial theorem was not modified by dividing the entire sample into two sets. As part of the experiment, the accuracy and effectiveness of such a formula for absolute moments will be examined.

The results produced in the work [3] are important since they offer stable (i.e. slightly deviating in terms of accuracy from the classic method of going through the entire sample) methods of updating the central moments, as well as keep continuing and expand into weighted samples (i.e. those samples that have a weight w_i associated with each value a_i).

In this work [4] a problem was considered of finding a stopping rule during the process of text field recognition in a video stream which is essential when developing the systems of document recognition with the use of mobile digital cameras [5]. The recognition process in a video stream entails combining frame-by-frame recognition results using the method described in the work [6]. In the work [4] the authors suggest a modified method based on the modeling the subsequent result of inter-frame combination [7] taking into account a set

of approximations, and in this simplified combined scheme there is a need for an effective update of a variable that is similar to the first-order absolute central moment in its structure. It was proposed to divide the sample into two parts and use balanced search trees in order to eliminate the absolute values in the formula for calculating the absolute central moment. Within the framework of this article this method will be expanded for a more general purpose of a fast update of an absolute central moment of any order.

Computational scheme

This scheme is based on the concept of dividing an entire sample into two subsets which will make it possible to eliminate absolute values. Let a_i be values from a sample A and $\overline{a}$ an arithmetic mean of all the values in this sample, i.e. the mean of the reviewed sample.

Let's review two subsets of the sample, $L = \{a \in A \mid a < \overline{a}\}$ and $R = \{a \in A \mid a \geq \overline{a}\}$. And it is obvious that $A = L \cup R, L \cap R = \varnothing$. Then, by definition:

$$\tilde{\mu}_k(A) = \frac{1}{n}\left(\sum_{i=1}^{n} |\overline{a} - a_i|^k\right). \quad (4)$$

It can be replaced by the following formula:

$$\tilde{\mu}_k(A) = \frac{1}{n}\left(\sum_{l \in L} |\overline{a} - l|^k + \sum_{r \in R} |\overline{a} - r|^k\right). \quad (5)$$

Then, since $\forall r \in R : \overline{a} - r \leq 0$ and $\forall l \in L : \overline{a} - l > 0$, the absolute value bars can be eliminated:

$$\tilde{\mu}_k(A) = \frac{1}{n}\left(\sum_{l \in L} (\overline{a} - l)^k + \sum_{r \in R} (r - \overline{a})^k\right). \quad (6)$$

Then after expanding the brackets using the formula of the binomial theorem, we can get

$$\tilde{\mu}_k(A) = \frac{1}{n}\left(\sum_{l \in L}\sum_{j=0}^{k} \overline{a}^{k-j} \cdot (-1)^j \cdot l^j \cdot C_k^j + \sum_{r \in R}\sum_{j=0}^{k} r^{k-j} \cdot (-1)^j \cdot \overline{a}^j \cdot C_k^j\right), \quad (7)$$

where $C_k^j = \frac{k!}{(k-j)! \cdot j!}$ is a binomial coefficient. Let's note that the inner sum of j in both double sums doesn't depend on the subset based on which we sum up L or R. Moreover, the order of summation for the finite sums can be altered. It means

$$\sum_{l \in L}\left(\sum_{j=0}^{k} \overline{a}^{k-j} \cdot (-1)^j \cdot l^j \cdot C_k^j\right) = \sum_{j=0}^{k}\left(\sum_{l \in L} \overline{a}^{k-j} \cdot (-1)^j \cdot l^j \cdot C_k^j\right),$$

$$\sum_{r \in R}\left(\sum_{j=0}^{k} r^{k-j} \cdot (-1)^j \cdot \overline{a}^j \cdot C_k^j\right) = \sum_{j=0}^{k}\left(\sum_{r \in R} r^{k-j} \cdot (-1)^j \cdot \overline{a}^j \cdot C_k^j\right).$$

Then if the summation $(-1)^j \cdot C_k^j$ is put outside the brackets, we get:

$$\tilde{\mu}_k(A) = \frac{1}{n}\left(\sum_{j=0}^{k} (-1)^j \cdot C_k^j \cdot \left(\sum_{r \in R} r^{k-j} \cdot \overline{a}^j + \sum_{l \in L} l^j \cdot \overline{a}^{k-j}\right)\right). \quad (8)$$

Now we can see that the equations $\overline{a}^j$ and $\overline{a}^{k-j}$ do not depend on the components or r or l, which means they can be put outside the summation signs, and as a result we'll get the final version of the formula:

$$\tilde{\mu}_k(A) = \frac{1}{n}\left(\sum_{j=0}^{k} (-1)^j \cdot C_k^j \cdot \left(\overline{a}^j \cdot \sum_{r \in R} r^{k-j} + \overline{a}^{k-j} \cdot \sum_{l \in L} l^j\right)\right). \quad (9)$$

As it can be seen from the formula (9), in order to get the values of the moment, it is sufficient to know the sums of the n-degrees of the elements of the subsets L and R. That's why the data structure needed for the implementation of the decision built using this formula has to solve the problem of a quick component insertion and subsequent generation of the required sums of the n-degrees of the elements of the sets L and R.

The process of calculating even-order moments is the same as calculating non-absolute central moments of the same order, however the computational scheme described further can be applied to them as well.

Data structure

We propose to use balanced search trees as the main data structure in this work.

A binary search tree is a data structure presented as **a binary tree** that meets the following requirements:

1. Both sub-trees are binary search trees as well.
2. All the keys of the left subtree nodes of arbitrary vertex X are less or equal to the key X. A node key is a value stored in this node. Fig. 1 shows an example of a binary search tree.

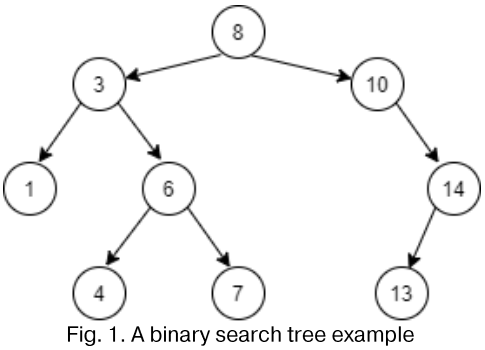

Fig. 1. A binary search tree example

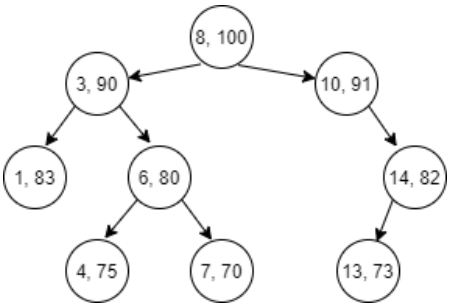

Fig. 2. A cartesian tree example

A **balanced binary search tree** is a tree which height does not exceed $O(\log n)$, where n is a number of nodes. The examples of balanced binary trees would be AVL trees [8] and red-black trees [9].

Out of all the existing balanced binary trees it is worth paying closer attention to a Cartesian tree [10].

A **binary heap** or a **pyramid** is a binary tree with the values of any of its nodes being less than the values of its descendants. A **cartesian tree** [11] or a **treap** is a binary tree with two keys, and it's such a tree that according to one of the keys x it's a binary search tree, and according to the second key y, it's a binary heap. We'll be referring to x as a **key** and to y as a **priority**. Fig. 2 demonstrates an example of a Cartesian tree. The values in the edges are recorded in the "key, priority" form.

In order to update the moment using the formula (9) we generated earlier, a vector $\left(\sum a_i^0, \sum a_i^1, \ldots, \sum a_i^k\right)$ will be stored in the nodes where its each component $\sum_{i=1}^{m} a_i^j$ is a sum of the nth-degrees of all the components in the subtree with a root node in a given node (the first component in the given data set is equal to the number of edges in this subtree). In fact, these sums are calculated "recursively": the sum of each node is a sum of all the sums from its child nodes, the sum of its child nodes is a sum of its child nodes' child nodes, etc. That's why when we introduce a new element into a tree, we have to update the sums only for those nodes which subtrees are going to be on the way to the edge during the insertion process.

When updating the absolute central moments according to the procedure described in the Algorithm description section, it is necessary to get the sums of the j-th degrees of the components of the subsets L and R and the original sample A for $j \in \{0, \ldots, k\}$ to calculate the formula (9). And the set L is defined as a subset of the original sample with the components that are less than the average value of a component in the sample A. In order to calculate such degrees using a balanced search tree, let's introduce an additional operation $\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$. Originally, the vector $(0, 0, \ldots, 0)$ consisting out of k zeroes is expressed as ACC_L.

$\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$ finds all the components of the subset L, starting with the root node of the tree. It checks the key value of the root node, then, if the key value is lesser, we can definitely state that the entire left subtree (including the root) belongs to L. Then the vector stored in the node that is the root of this tree is added to the vector ACC_L. If the node key is higher than the value which is transferred to the operation, then $\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$ is run from the left descendant of the tree root node until (i.e. $\mathrm{root} = \mathrm{root}_l$, the new root node transferred to the operation is a left descendant) the transferred node has the key lesser than value.

As the next step, we need to determine which components of the right subtree belong to L as well. In order to do that, after we add the values to the data set ACC_L, the operation $\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$ is run for the right descendant by the earlier determined edge with a key lesser than value, and it's run until the transferred node doesn't have any descendants left.

The **complexity** of the operation $\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$ is equal to $O(h \cdot k)$,

where h is a tree height since this operation will be initiated not more than h times, and each time it will be performed in k (because we need to stack two vectors of the size k).

Therefore, as a result of the performance of the function $\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$, we'll be able to find out both the sums of the j-th degrees of the subset L and the sums of the j-th degrees of the subset R through the substraction of the sums of the subset L out of the sums of the root node of the tree, i.e.

$$\mathrm{ACC}_R = \mathrm{ACC} - \mathrm{ACC}_L, \qquad (10)$$

where ACC is a vector of the j-th degrees of all the tree nodes that is stored in the root node.

The algorithm description

The process described in the observation process section moves to a k-th stage, and it is vital to update the k-th absolute moment.

1. To add a new element a_{n+1} to the data structure.

2. To get the values of the sums of the k-th degrees for L_{n+1} and R_{n+1}. The sums for L_{n+1} are generated by the operation $\mathrm{SUMM}_{L_{n+1}}(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_{L_{n+1}})$ described in section 7, the sums for R_{n+1} can be recovered out of the total sums for the sample A stored in the root node of the tree and calculated sums of L_{n+1}.

3. To perform an addition with the help of the formula (9).

In order to solve these problems, a new edge with the key a_{n+1} and with the priority y_{n+1} generated for it is inserted into a Cartesian tree, and after that the sums of the k-th degrees in the affected by this insertion edges are updated.

Then after the operation $\mathrm{SUMM}_L(\mathrm{root}, \mathrm{value}, \mathrm{ACC}_L)$ is applied to the tree, the data set ACC_L is calculated and then the data set ACC_R is calculated using the formula (10).

As the next step, the formula (9) is applied with the substitution

$$\sum_{r\in R} r^{k-j} = \mathrm{ACC}_R^{k-j}, \sum_{l\in L} l^j = \mathrm{ACC}_L^j. \qquad (11)$$

The updated absolute moment is the value of the formula (9) with the substitute values.

The **complexity** of this algorithm is $O(k \cdot \log n)$. There is a block-scheme of the algorithm demonstrated in Fig. 3. It shows the process starting from the moment of the process transition into the next step and ending with the moment update.

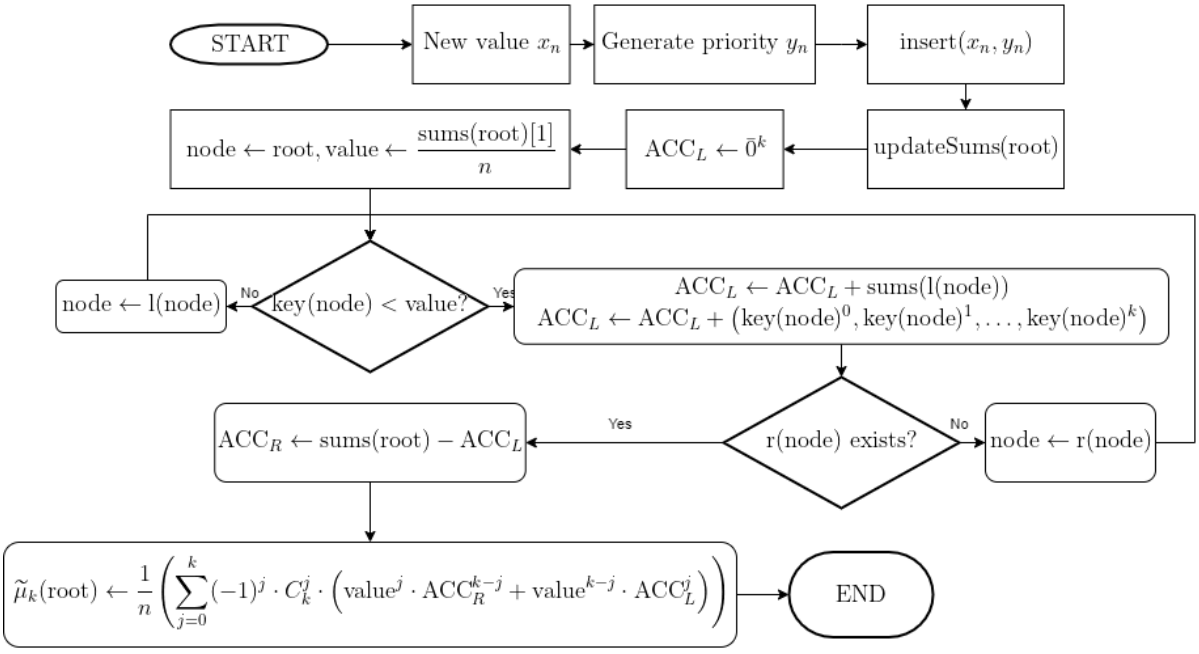

Fig. 3. A block diagram of the proposed algorithm

The experiment and the results

The experiment is carried out by performing the actions described in the previous solutions section (hereinafter referred to as basic) and running the algorithm proposed in the article and described in the Algorithm description section (hereinafter referred to as an algorithm for treaps). For each cross-section of the sample as big as 100000 units (i.e, from the first element to the first one, from the first one to the second one, from the first one to the third one, etc.) the time performance for each of the algorithms is measured 21 times. After that the mid-points for each measurement are compared. For each of the 21 runthroughs the identical numbers taken from the even distribution on the interval $[0,1]$ are generated. The time in the tables is presented in milliseconds, if not specified otherwise.

Fig. 4 shows the results of time measurements of the 7th degree moment update. The Table 1 exhibits the results of time measurements of the 1st, 3rd, 5th and 7th degree moment updates. In order to reflect the differences, the logarithmic scale of time was used.

Fig. 5 and Table 2 show the results of time measurement for the 7th order moment update with a small-sized sample (1-100).

Fig. 6 and Table 3 represent the results of the measurements of the total performance time of the algorithms for updating the moments of the 1st to the 50th order.

Table 4 demonstrates the treap algorithm deviation compared to the basic algorithm using double-precision floating-point arithmetic. $|\Delta|$ reflects the absolute difference between produced values.

Experimental time measurements were performed on a PC (Intel Core i5 6600 processor, Microsoft Visual C++ Compiler v14.25). The original code needed for the reproduction of the experiments is available here: https://github.com/devourers/absolute-moment-computation.

The experimental data demonstrates the treap algorithm superiority over the classic algorithm when it comes to performance time, and the largest deviation is compared to 10^{-13} (as we can see from Table 4).

In addition, Figure 6 shows that the treap algorithm performance time is slowly increasing relative to the moment order which is updating. The time of the 9th order moment measurement is compared to the time of the 1st order moment measurement for the treap algorithm, while the basic algorithm time performance for the 8th order is 4 times higher than for the 1st order.

When working with smaller-sized samples (up to 80 elements), the classic algorithm performs better time-wise due to the fact that the treap algorithm spends additional time for insertion which is commensurable for the samples of this size.

Conclusion

In this work we reviewed the existing results of the fast update of the central moments in samples and developed an effective method for updating the absolute moments in a sample. The method described in this article demonstrates the accuracy of the 10^{-13} order and a low logarithmic increase relative to the number of elements in a sample and a low logarithmic increase relative to the moment order.

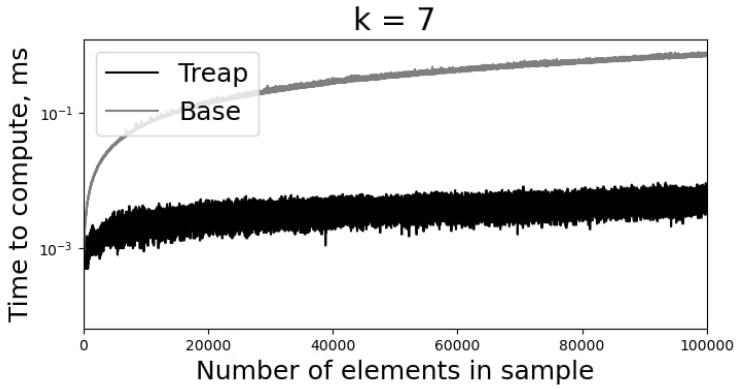

Fig. 4. The graphs representing the relationship between the time of moment update and the size of the sample (1 – 100000), the black graph shows the time of the algorithms for the treaps, the grey one – the time of the basic algorithm

Table 1. The update time of the absolute central moment for the 1st, 3rd, 5th and 7th order moments

Moment Order	Number of Elements	Treap algorithm time	Basic algorithm time
1	100	0.0003	0.0004
	1000	0.0004	0.0037
	10000	0.0015	0.0366
	50000	0.0022	0.1844
	75000	0.0026	0.2809
	100000	0.0029	0.3717
3	100	0.0005	0.0006
	1000	0.0005	0.0053
	10000	0.0019	0.0509
	50000	0.0027	0.2704
	75000	0.0034	0.4156
	100000	0.0034	0.5166
5	100	0.0006	0.0008
	1000	0.0007	0.0068
	10000	0.0024	0.0672
	50000	0.0029	0.3497
	75000	0.0039	0.5674
	100000	0.0041	0.7086
7	100	0.0007	0.0008
	1000	0.0009	0.0068
	10000	0.0028	0.0693
	50000	0.0036	0.3426
	75000	0.0046	0.5327
	100000	0.0047	0.7254

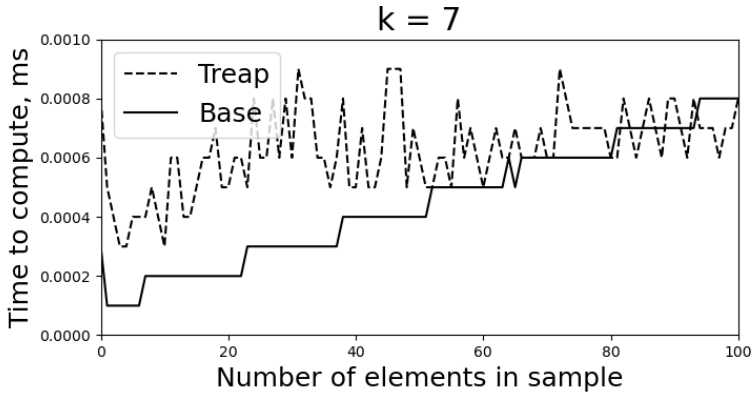

Fig. 5. Dependency graph showing the relationship between time measurements of the 7th order moments for the first 100 elements

Table 2. Update time of the absolute moment with a smaller-size sample

Number of elements	Treap algorithm time	Basic algorithm time
20	0.00043	0.00019
40	0.00045	0.00024
60	0.00057	0.00053
80	0.00063	0.00057

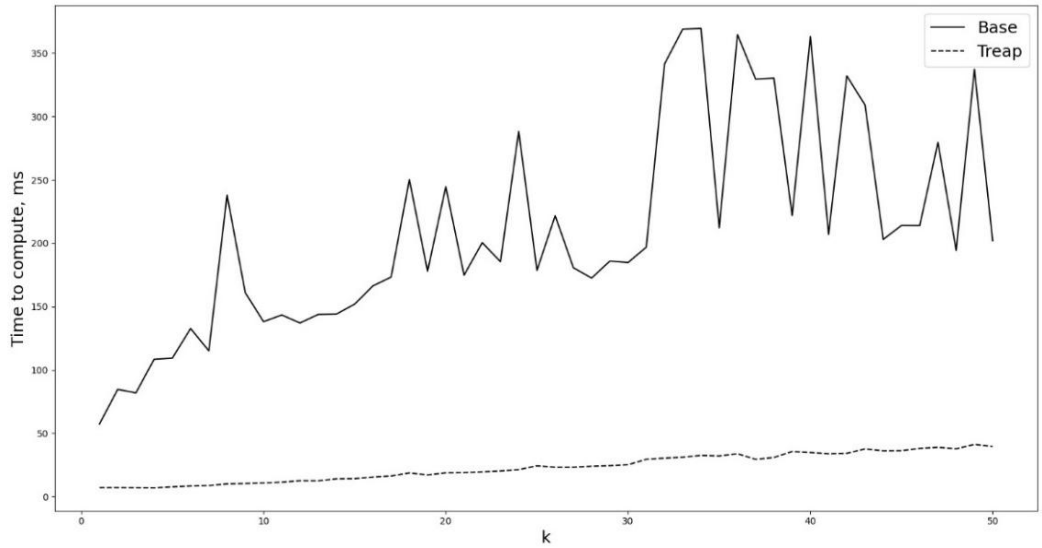

Fig. 6. The dependency graph showing the relationship between the total time measurement for the entire sample using the treap method and the order of the absolute moment. The sample size is 10000

Table 3. Total time for updating all the cross-sections (i.e. samples $A_1, A_2, A_3, \ldots A_n$) of the treap algorithm and the basic algorithm

Moment order	Treap algorithm time	Basic algorithm time
1	10.0366	62.5009
2	8.0027	87.2331
3	9.9450	85.6776
4	10.3507	114.3370
5	9.9535	113.5313
6	9.5546	129.8519
7	10.9674	118.8513
8	11.2111	244.9523
9	12.4585	169.0804

Table 4. The table of deviations of the results of the absolute central moment update for the treap algorithm and the basic algorithm

Order	Number of elements	Basic algorithm Treap algorithm	$\lvert \Delta \rvert$
1	1000	241.9220 241.9220	$8.5265 \cdot 10^{-14}$
3	10000	306.7266 306.7266	$1.1368 \cdot 10^{-13}$
5	8	0.0372 0.0372	0
7	100	0.0935 0.0935	$5.2305 \cdot 10^{-14}$
9	3	0.0000 0.0000	$1 \cdot 10^{-16}$
9	1	0.0000 0.0000	$4 \cdot 10^{-16}$

References

1. G. Kramer. Matematicheskie metodi statistiki. — 2-nd publ. — M.: Mir, 1975. — P. 196-197, 284. — 648 p.
2. Gueron S. Efficient software implementations of modular exponentiation //Journal of Cryptographic Engineering. – 2012. – Т. 2. – №. 1. – С. 31-43.
3. Pébay P. et al. Numerically stable, scalable formulas for parallel and online computation of higher-order multivariate central moments with arbitrary weights //Computational Statistics. – 2016. – Т. 31. – №. 4. – С.1305-1325.
4. K. Bulatov, N. Fedotova, V. V. Arlazarov, Fast Approximate Modelling of the Next Combination Result for Stopping the Text Recognition in a Video //In Proc. 2020 25th International Conference on Pattern Recognition (ICPR) Milan, Italy, Jan 10-15, 2021, p. 239-246.
5. V. V. Arlazarov, A. E. Zhukovskiy, V. E. Krivtsov, D. P. Nikolaev and D. V. Polevoy, “Analiz osobennostey ispolzovaniya statsionarnykh i mobilnykh malorazmernykh tsifrovykh video kamer dlya raspoznavaniya dokumentov,” ITiVS, no 3, pp. 71-81, 2014.
6. Bulatov K. B. A Method to Reduce Errors of String Recognition Based on Combination of Several Recognition Results with Per-Character Alternatives // Вестник ЮУрГУ ММП. — 2019. — Т. 12. — № 3. — С. 74-88. — DOI: 10.14529/mmp190307.
7. Bulatov K., Savelyev B., Arlazarov V. V. Next integrated result modelling for stopping the text field recognition process in a video using a result model with per-character alternatives // ICMV 2019 / Wolfgang Osten, Dmitry Nikolaev, Jianhong Zhou. — SPIE. — янв. 2020. — Т. 11433. — ISSN 0277-786X. — ISBN 978-15-10636-44-6. — 2020. — Т. 11433. — 11433 2M. — С. 1-7. — DOI: 10.1117/12.2559447.
8. Adelson-Velsky, Georgy; Landis, Evgenii (1962). "An algorithm for the organization of information". Proceedings of the USSR Academy of Sciences (in Russian). 146: 263–266.
9. Storer J. A. An introduction to data structures and algorithms. – Springer Science & Business Media, 2012.
10. Blelloch G. E., Reid-Miller M. Fast set operations using treaps //Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures. – 1998. – С. 16-26.
11. Seidel R., Aragon C. R. Randomized search trees //Algorithmica. – 1996. – Т. 16. – №. 4-5. – С. 464-497.

Rybalko D. Z. Student of NUST “MISiS”. Fields of interest: computer vision, object recognition, topological data analysis. E-mail: d.rybalko@smartengines.com

Bulatov K. B. PhD**,** senior researcher at Federal Research Center "Computer Science and Control" of Russian Academy of Sciences. Graduated from NUST "MISIS" in 2013. Fields of interest: pattern recognition, machine learning, information systems. E-mail: kbulatov@smartengines.com

Polevoy D. V. PhD, senior researcher at Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, graduated from MIPT in 2004. Fields of interest: computer vision, image processing, information systems. E-mail: polevoy@smartengines.com