

# Speeding Up Minimum Distance Randomness Tests

Krzysztof Mańk

Military University of Technology, Warsaw, Poland

<https://doi.org/10.26636/jtit.2022.159222>

**Abstract**—Randomness testing is one of the essential and easiest tools for the evaluation of the features and quality of cryptographic primitives. The faster we can test, the greater volumes of data can be checked and evaluated and, hence, more detailed analyses may be conducted. This paper presents a method that significantly reduces the number of distances calculated in the minimum distance, Bickel-Breiman, and  $m$  nearest points tests. By introducing a probabilistic approach with an arbitrarily low probability of failure, the number of calculated distances proportional to the number of required distances and independent of the number of points was achieved. In the well-known Diehard's minimum distance and 3D spheres tests, the quantity of computations achieved is reduced by the factors of 394 and 771, respectively.

**Keywords**—Bickel-Breiman test, minimum distance test,  $m$  nearest pairs test, randomness test.

## 1. Introduction

Testing of randomness is important, as a variety of analyses concerned with cryptographic primitives may be reduced to examining appropriately crafted binary sequences. In each such case, the quality of the analysis increases with the volume of data checked, but this leads to increases in time and cost as well. Any improvements in randomness testing translate directly into the quality of evaluation of RNGs, symmetric ciphers, hash functions, and other cryptographic primitives.

First proposed in Diehard as minimum distance and 3D spheres tests, then found in TestU01 as  $m$  nearest pairs test and, of course, ported to Dieharder as `diehard_2dsphere` and `diehard_3dsphere` a family of minimum distance tests raised and took its place in randomness testing. Computationally, the main part of these tests consists in calculating distances between  $n$  points randomly placed in a  $t$  dimensional hypertorus. Naive implementation leads to the calculation of  $n(n-1)/2$  distances, which, in some cases, may be unacceptable.

Dieharder and NIST's Statistical Test Suite, are the two most commonly used test suites. Both are slightly outdated, but still remain popular due to their availability.

We will distinguish three test cases:

1. minimum distance,
2.  $m$  nearest pairs,
3. Bickel-Breiman.

The first one is a special case of the second case, but because it relies on a different method to speed up the calculation process, we make a distinction between the two.

The most general formulation is the closest-pair problem, for which deterministic algorithms operating in  $O(n \log n)$  are known (described in [1]–[4]), as well as probabilistic algorithms with linear complexity due to [5]–[6].

The best-known deterministic method for performing the minimum distance test is based on Fischler's upper estimation for minimum distance. The algorithm may be found in [7]–[9]. As shown in the following section, this method is characterized by linear complexity.

The contribution of this research consists in introducing a probabilistic approach to minimum distance estimation with an arbitrarily low probability of failure. Even for extremely low probabilities of failure, it is possible to obtain significantly lower estimations of minimum distance, leading to the number of calculated distances being independent of the number of points.

Similar approaches for the  $m$  nearest pairs and the Bickel-Breiman tests are proposed in the subsequent sections. In all three tests, we obtained the number of calculated distances that was proportional to the number of the needed distances and was independent of the number of points or the number of dimensions.

In the remaining part of this paper, for the sake of clarity, we shall simply refer to a cube and a torus, instead of a  $t$  dimensional hypercube and a hypertorus.

## 2. Minimum Distance Test

In this paragraph, we will consider an algorithm deployed to identify two nearest points between  $n$  points that are randomly placed in a  $t$  dimensional hypertorus. In 2002, Fischler published a paper [7] on correcting and speeding up minimum distance tests, like the two mentioned from Diehard. Fischler's method is based on dividing the torus into equal cubes and on calculating distance only between points in the same or adjoining cubes. The smaller the cubes, the fewer distances have to be calculated. But below a certain point, no points in the same or in adjoining cubes may be identified, and an error occurs. Fischler gave the upper estimate for the minimum distance in the set of  $n$  points in a  $t$  dimensional unit hypertorus:

$$D \leq \frac{\sqrt{t}}{\sqrt[n]{n}},$$

it is equal to twice the length of the edge of the sub-cube. As it will be shown below, this is unnecessary over the secure approach.

To determine the expected number of calculated distances, let us consider to-the-right-and-down approach illustrated in Fig. 1 for the two-dimensional case.

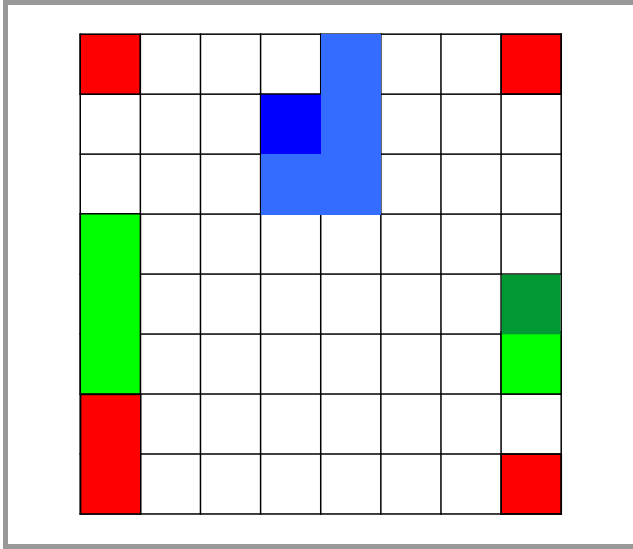


Fig. 1. To-the-right-and-down approach for calculating distances in a 2D torus.

For every cube (fully colored), we need to calculate the distance for every pair of points in that cube and from every point to all points in cubes to the right and down (highlighted by corresponding light color).

For  $t > 2$  dimensional case we can for example test all cubes in volume consisting of a series of 3-by-...-by-3  $t-1$ ,  $t-2 \dots$ , 3 cubes, "line" 3-by-1 and single cube. Note that the last two can be treated as one- and zero-dimensional cubes-of-cubes. The total number of cubes in this volume is  $(3^t - 1)/2$ . From Fischler's formula, we can calculate the number of cubes as:

$$\left\lfloor \frac{2\sqrt[t]{n}}{\sqrt{t}} \right\rfloor \leq \frac{2^t n}{t^2}.$$

Note that only for  $t$  up to 4 expected number of points in every cube will not exceed 1.

Using binomial distribution, the expected number of calculated distances per cube is:

$$\begin{aligned} E_F &= \sum_{i=2}^n \binom{i}{2} \binom{n}{i} p^i (1-p)^{n-i} \\ &+ \frac{3^t - 1}{2} \sum_{i=1}^n i \binom{n}{i} p^i (1-p)^{n-i} \sum_{j=1}^{n-i} j \binom{n-j}{j} p^j (1-p)^{n-i-j} \\ &= \frac{1}{2} n(n-1) p^2 (3^t (1-p) + p), \end{aligned}$$

where  $p = \left\lfloor \frac{2\sqrt[t]{n}}{\sqrt{t}} \right\rfloor^{-t}$ .

If we omit the floor function in  $p$  for a large  $n$ , we get the approximation:

$$E_F \approx \frac{1}{2} \left( \frac{3t}{4} \right)^t,$$

and the total number of calculated distances:

$$\frac{1}{2} \left( \frac{3}{2} \right)^t t^{\frac{1}{2}} n.$$

In Diehard's minimum distance test, we have  $n = 8000$  and  $t = 2$ , which gives us the maximum number of  $126^2$  squares and the expected number of calculated distances slightly exceeds 18,138. For the 3D spheres test, we have  $n = 4000$  and  $t = 3$ , which gives the maximum number of  $18^3$  cubes and the expected number of calculated distances 37021. For  $n = 3000$  and  $t = 5$  one would get  $4^5$  5D cubes and expected number of calculated distances equaling 1,066,477. Those numbers mean, respectively, 1,764, 216, and 4.2 times fewer computations compared to the basic naive algorithm.

From Fig. 1, we can observe that the distance is calculated if and only if two points collide in the same double-cube composed from  $(3^t - 1)/2$  base cubes. To find two nearest points, all we need is at least one such collision, which leads us to a different approach of finding the maximum number of dividing cubes.

We start from the probability of no collisions when placing  $n$  points in  $K$  boxes being small enough:

$$\frac{K!}{(K-n)!K^n} \leq \varepsilon,$$

where  $\varepsilon$  can be, for example,  $10^{-20}$ .

For efficiently solve this inequality for  $K$ , the well-known relation is used:

$$e^{-x} \geq 1 - x,$$

which leads to:

$$\begin{aligned} \frac{K!}{(K-n)!K^n} &= \prod_{i=1}^n \frac{K-i+1}{K} = \prod_{i=1}^n \left( 1 - \frac{i-1}{K} \right) \\ &\leq \prod_{i=1}^n e^{-\frac{i-1}{K}} = e^{-\frac{1}{K} \sum_{i=1}^n (i-1)} = e^{-\frac{n(n-1)}{2K}} \leq \varepsilon, \end{aligned}$$

and the upper limit for  $K$  is:

$$K \leq -\frac{n(n-1)}{2 \ln \varepsilon}.$$

However, the number of boxes  $K$  is not the number of needed cubes, which in fact is:

$$\left\lfloor -\frac{n(n-1)}{2 \ln \varepsilon} \right\rfloor 3^t.$$

In the following considerations, the following formula will be used:

$$\left\lfloor \sqrt[t]{-\frac{n(n-1)}{2 \ln \varepsilon}} \right\rfloor 3^t.$$

Similarly, we can calculate the expected number of calculated distances per cube:

$$E_C = \frac{1}{2}n(n-1)p^2 [3^t(1-p) + p] ,$$

where  $p = \left[ \sqrt[t]{\frac{n(n-1)}{2 \ln \epsilon}} \right]^{-t} 3^{-t}$ .

Assuming  $n$  is large after omitting the floor function, we get:

$$E_C \approx \frac{2 \ln^2 \epsilon}{n(n-1)3^t} ,$$

and the total number of calculated distances is:

$$- \ln \epsilon ,$$

which is independent of the number of points. According to proposed method for the three examples, we have:

- 2499<sup>2</sup> squares and 46 distances,
- 165<sup>3</sup> cubes and 48 distances,
- 27<sup>5</sup> 5D cubes and 76 distances.

Those numbers mean that the number of computations is 394, 771, and 14,033 times smaller compared to Fischler’s method, respectively.

If, instead of inequality for  $e^{-x}$ , Stirling’s approximation for factorial is used, a slightly better formula may be obtained:

$$\left( \frac{K}{K-n} \right)^{K-n+\frac{1}{2}} e^{-n} \leq \epsilon ,$$

which unfortunately cannot be efficiently solved for  $K$  and, therefore, was omitted. However, there is not so little flaw in this method.

According to the approach shown in Fig. 1, for any point put into a torus, the distances would be calculated only to the

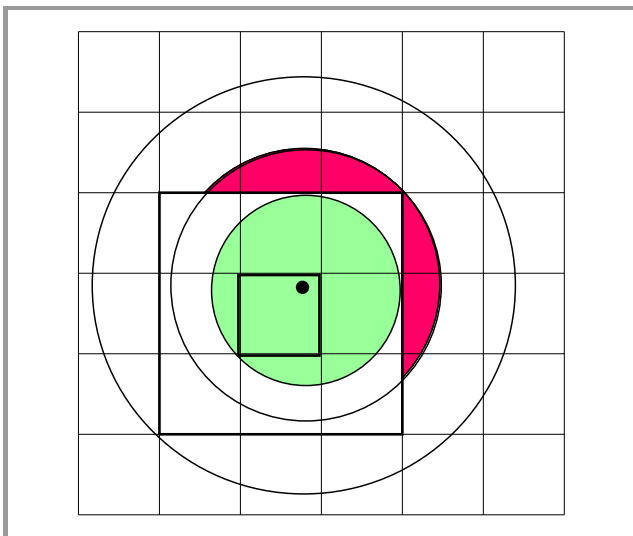


Fig. 2. A flaw in to-the-right-and-down approach.

points in the same or neighboring sub-cubes. Unfortunately, in some cases this could lead to a failure in finding the nearest point.

Let us consider a 2-dimensional case and a point which is located somewhere in the center square. The smaller green circle shown in Fig. 2 covers the entire area inside a 3-by-3 square, such that for any point therein, if there are points closer to the one that is marked, then they are also in that area. This is not true for a larger circle. In this case, for every point outside the green circle, there are points outside the 3-by-3 square which are closer to the one marked, but because they are not situated in the neighboring squares, they will not be included in the search. This error can be fully eliminated only when Fischler’s formula is used, because it guarantees that the minimum distance will be lower than the radius of the smaller circle. The largest circle corresponds to all points taken into consideration. In a 2-dimensional case, a 3-by-3 square occupies less than 36% of the maximum size circle, in 3D – less than 16% of the maximal sphere, and in 5D – less than 2.6%, which creates a significant discomfort.

We will use the cumulative distribution function for the minimum distance between  $n$  points in a  $t$  dimensional hypertorus:

$$\Pr(D \leq d) = 1 - e^{-\frac{d^t n(n-1)V_t(1)}{2}} ,$$

where

$$V_t(r) = \begin{cases} \frac{\pi^h r^t}{t!} , & t = 2h , \\ \frac{h! \pi^{h-1} r^t}{t!} , & t = 2h - 1 , \end{cases}$$

is the volume of a  $t$  dimensional hypersphere of radius  $r$  [10].

After skipping the floor function, in the minimum distance test, we get the length of the sub-cube’s edge, which is equal to the radius of the smaller circle as:

$$K = \frac{1}{3} \sqrt[t]{\frac{-2 \ln \epsilon}{n(n-1)}} ,$$

and the probability of the minimum distance being greater is:

$$\epsilon^{\frac{V_t(1)}{3^t}} ,$$

which for  $\epsilon = 10^{-20}$  equals  $10^{-7}$  for  $t = 2$ , 0.00079 for  $t = 3$ , and 0.369 for  $t = 5$ .

By inverting this argumentation, we can easily find such a sub-cube size that assures an arbitrarily low probability of that flaw affecting the computed distances. From:

$$e^{-\frac{d^t n(n-1)V_t(1)}{2}} \leq \epsilon ,$$

we get

$$d \geq \sqrt[t]{\frac{2 \ln \epsilon}{n(n-1)V_t(1)}} ,$$

which returns the number of sub-cubes

$$K = \left\lceil \sqrt[t]{\frac{n(n-1)V_t(1)}{2 \ln \varepsilon}} \right\rceil.$$

Assuming  $n$  is large and omitting the floor function, the total number of calculated distances is:

$$-\ln \varepsilon \frac{3^t}{V_t(1)}.$$

According to the proposed method, for the same three examples, we have:

- $1,515^2$  squares and 251 distances,
- $91^3$  cubes and 573 distances,
- $14^5$  5D cubes and 4,065 distances.

Those numbers are substantially worse than those obtained previously, but the expected number of computed distances is still independent of the number of points.

This flaw will not play an important role in the following two tests, because the number of points in each 3-by-... cube will be significantly greater than one.

### 3. Bickel-Breiman Test

In the Bickel-Breiman test [11], the aim is to find the nearest point for every point out of  $n$  points randomly placed in a  $t$  dimensional hypertorus. In this case, the analytic approach leads to the lack of ability of dividing into sub-cubes – for a randomly chosen point, the maximum distance to the nearest point is  $\frac{\sqrt[t]{t}}{2}$ .

To cope with this, we need a collision for every point placed, so we start with the probability of not finding any other point in the  $3^t$  cube:

$$\left(1 - \frac{3^t}{K}\right)^{n-1},$$

and then the probability that not all the points will find their neighbor is:

$$1 - \left[1 - \left(1 - \frac{3^t}{K}\right)^{n-1}\right]^n \leq \varepsilon.$$

With simple transformations we obtain:

$$K \leq \frac{3^t}{1 - \sqrt[n-1]{1 - \sqrt[n]{1 - \varepsilon}}}.$$

As before, the expected number of calculated distances per cube is:

$$E_{BB} = \frac{1}{2}n(n-1) \frac{1}{K^2} \left[3^t \left(1 - \frac{1}{K}\right) + \frac{1}{K}\right],$$

and the total number of calculated distances:

$$\frac{1}{2}n(n-1) \frac{1}{K} \left[3^t \left(1 - \frac{1}{K}\right) + \frac{1}{K}\right],$$

which is analogically in naive method, “only” with coefficient  $\frac{1}{K} \left[3^t \left(1 - \frac{1}{K}\right) + \frac{1}{K}\right]$ . This time the improvement is not as spectacular as in the previous case. It could be approximated as:

$$\frac{1}{1 - \sqrt[n-1]{1 - \sqrt[n]{1 - \varepsilon}}}$$

times faster.

For the same three examples, this time we have:

- $36^2$  squares and 144 times fewer calculated distances,
- $12^3$  cubes and 64 times fewer calculated distances,
- $6^5$  5D cubes and 32 times fewer calculated distances,

than in the naive approach, when  $\varepsilon = 10^{-20}$ .

### 4. $m$ Nearest Points Test

This test can be found in the TestU01 library [12] and is described in paper [10].

Because of the existing recommendation that  $n \geq 4m^2$ , we can assume that the number of nearest points  $m$  is much smaller than the total number of points  $n$ . In fact, it is so small that in every case from examples presented in Section 3, we get more distances calculated per every sub-cube than actually needed.

To find  $m$  nearest pairs, at least  $m$  collisions are needed, having the probability of:

$$1 - \sum_{r=0}^{m-1} \frac{K!}{(K-n+r)!K^n} \binom{n}{n-r}.$$

For given  $n$  and  $m$ , while applying Stirling’s formula for factorial and tabulating values of all needed Stirling’s numbers, the number of cubes  $K$  can be efficiently computed

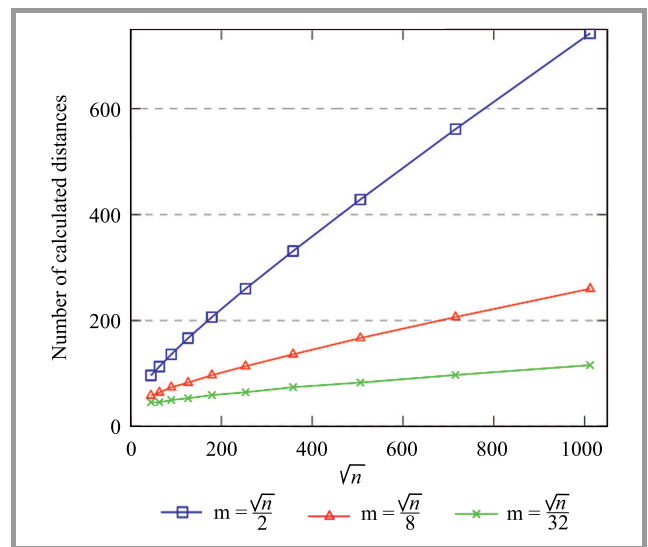


Fig. 3. Expected number of calculated distances as a function of  $\sqrt{n}$  in a 2-dimensional case.

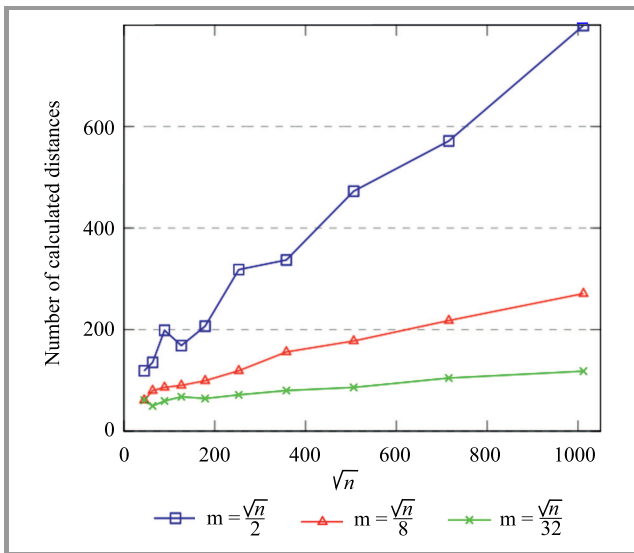


Fig. 4. Expected number of calculated distances as a function of  $\sqrt{n}$  in a 5-dimensional case.

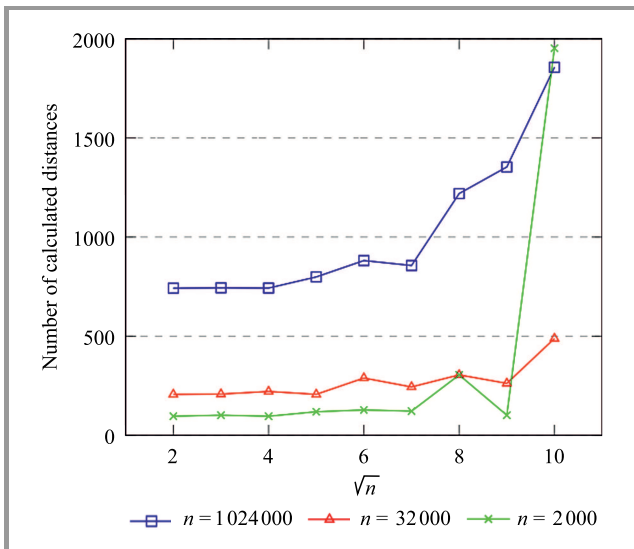


Fig. 5. Expected number of calculated distances as a function of the number of dimensions.

numerically. Like in the minimum distance test, the obtained value of  $K$  should be multiplied by  $3^t$ . On the three examples researched, we have:

- 1,455<sup>2</sup> squares and 1632 times fewer calculated distances,
- 123<sup>3</sup> cubes and 1,076 times fewer calculated distances,
- 24<sup>5</sup> 5D cubes and 1,024 times fewer calculated distances,

than using the formula for the Bickel-Breiman test, so the effort pays off.

The three graphs presented show the relation between the number of points, dimensions, and the expected number of

calculated distances. In Figs. 3 and 4, one can observe that in every case the expected number of calculated distances grows linearly according to the  $\sqrt{n}$  or, in fact, to the  $m$ . The deviations observed arise from the necessary flooring of  $\sqrt[t]{K}$ .

Figure 5 shows how the number of dimensions affects the expected number of calculated distances for a different number of points. In all cases  $m = \frac{\sqrt{n}}{2}$ . Similarly, the gradual increase in the number of calculated distances arises from the flooring of  $\sqrt[t]{K}$  – otherwise they would be constant. To understand the cause of that growth, one can calculate the actual number of sub-cubes for a different  $t$ . Let  $n = 1,024,000$ ,  $m = 505$  and  $\epsilon = 10^{-20}$ , for which we obtain  $K = 706,379,797$  and from the formula:

$$\lfloor \sqrt[t]{K} \rfloor^t$$

values from 282,475,249 ( $t = 10$ ) to 706,336,929 ( $t = 2$ ).

### 5. Implementation

At the beginning, let us consider the minimum distance test and four approaches to implementing it:

- naive implementation using a general-purpose single thread processor (CPU),
- implementation based on sub-cubes with Fischler’s result, using a general-purpose single thread processor,
- implementation based on sub-cubes with the presented method, using a general-purpose single thread processor,
- parallel implementation on a GPU engine.

None of the above methods can be compared with regard to the number of computed distances only, due to the fact that:

- naive implementation is based on calculating the distance for all pairs of points,
- naive implementation requires no additional memory,
- the amount of time consumed to review the data will depend on the number of sub-cubes,
- there is no obvious method for implementing the sub-cubes method using a GPU,
- the parallel implementation on a GPU accelerator consists in calculating the distance for all other points for every point,
- the parallel implementation works on a GPU whose compute cores differ significantly from those of CPU cores for which the sub-cubes method is suitable.

Depending on the hardware platform used, it may turn out that the following estimates are inadequate, but they should be easy to adapt and reproduce.

To perform any valid comparisons between the four cases mentioned above, two factors have to be determined:

- the ratio between naive and on sub-cubes based implementations,
- comparison of the speed of a single core of CPU and GPU.

The second factor is easy to obtain. We will consider two top computational devices:

- Tesla V100S PCIe accelerator with 5120 Nvidia CUDA cores and 16.4 TFLOPS on single-precision performance [13],
- AMD Ryzen Threadripper 3990X processor with 64 cores and 128 threads [14] and 3.7 TFLOPS on single-precision performance [15].

It may be assumed that:

- Tesla should be able to calculate 4.43 times more distances than the AMD processor,
- the single core of the AMD processor should be able to calculate 9.02 times more distances than the single core of the Tesla processor.

Based on this assumption, the parallel implementation on the Nvidia Tesla GPU may be estimated as:

- 443 times faster for  $n = 8,000$  and  $t = 2$  or  $n = 4,000$  and  $t = 3$ ,
- 332 times faster for  $n = 3000$  and  $t = 5$ ,

than naive implementation on a single core of the AMD CPU. For  $n = 8,000$  and  $t = 2$ , we assumed two runs of calculations due to lack of cores limitations.

As mentioned above, the results will vary depending on the quality of the implementation, so we have limited our two procedures to the evaluation of the square of the distance. Therefore, the only optimization was the limitation of the number of sub-cubes to the power of 2 only.

Table 1

Mean time to complete the calculations for different test cases and methods

Test case method	$n = 8000$ $t = 2$	$n = 4000$ $t = 3$	$n = 3000$ $t = 5$
Naive	180.14	67.53	63.81
Fischler's	1.13	0.95	10.23
Proposed	5.19	1.49	1.72

In Table 1 we present the mean times in milliseconds, required to complete the calculations for different test cases and methods used.

Based on these results, the following conclusions may be formulated:

- time for the naive method strictly depends on the number of calculated distances,
- the sub-cubes method is significantly faster,
- time for sub-cubes depends not only on the number of calculated distances, but also on the number of sub-cubes,
- depending on the test case, the number of sub-cubes based on the Fischler's estimate or on the proposed approach renders better results, suggesting that neither of them is optimal.

The last of the above findings has led to a series of tests, each consisting in the execution performed for a different number of cubes.

For the  $t$ -dimensional case, the number of sub-cubes is  $2^{dt}$ , where  $d = 2, 3, \dots$ , and  $2^d$  is the number of divisions for every dimension.  $d = 2$  is the smallest reasonable case, and  $d$ , based on our results, is the maximum case. In Table 2 we show the mean times (in milliseconds) required to complete the calculations for different test cases and numbers of sub-cubes.

Table 2

Mean time to complete the calculations for different test cases and numbers of sub-cubes

Test case $d$	$n = 8000$ $t = 2$	$n = 4000$ $t = 3$	$n = 3000$ $t = 5$
2	151.67	19.26	10.23
3	24.48	2.21	1.72
4	4.22	0.95	
5	1.53	0.69	
6	1.13	1.49	
7	1.00		
8	1.12		
9	1.81		
10	5.19		

This means that the tuned sub-cubes method is capable of evaluating the minimum distance 180, 98, and 37 times faster in the considered cases, compared to the naive method.

Finally, implementations performed with the use of the CPU and GPU can be compared. A simple division shows that a parallel implementation on a GPU will complete a single evaluation run 2.5 to 9 times faster than a single-core implementation using the sub-cubes method, but the AMD processor may handle up to 128 parallel runs, so the total throughput is greatly in favor of the tuned sub-cubes method.

For the  $m$  nearest pairs test and for a small  $m$ , the results are similar to those presented above. For bigger values, as well as for the Bickel-Breiman test, the GPU-based implementation will be faster.

## 6. Conclusion

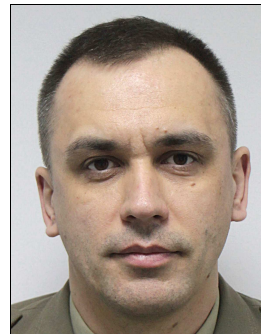
The Fischler's approach, a solution that has so far been the best known deterministic method for speeding up the minimum distance test performed on a single thread CPU, was significantly improved. Furthermore, means for similar improvements of two other tests have been presented. The methods presented, although of the probabilistic variety, offer the ability to practically mitigate error probability. Thanks to the significant reduction in the number of the computed distances, the tests could be performed on longer sequences, thus increasing their detection capabilities. After precise tuning, minimum distance and  $m$  nearest pairs tests may be run on modern multicore CPU processors and are capable of outperforming GPUs.

Because minimum distance and 3D spheres tests are only some of many tests in Dieheard's and Dieharder's portfolios, total time complexity gains seem not impressive. On the other hand, the results obtained allow to extend the existing packages by tests covering much longer sequences and thus offering a greater detection capacity.


Further work may be concerned with generalizing the presented methods in order to apply these to more general problems, e.g. the closest bichromatic pair problem.

## References

- [1] M. I. Shamos and D. Hoey, "Closest-point problems", in *Proc. of the 16th Ann. Symp. on Found. of Comp. Sci.*, Berkeley, CA, USA, 1975 pp. 151–162 (DOI: 10.1109/SFCS.1975.8).
- [2] J. L. Bentley and M. I. Shamos, "Divide and conquer in multidimensional space", in *Proc. of 8th Ann. ACM Symp. on the Theory of Comput.*, Hershey, PA, USA, 1976. pp. 220–230 (DOI: 10.1145/800113.803652).
- [3] J. L. Bentley, "Multidimensional divide-and-conquer", *Comm. of the Assoc. for Comput. Machinery*, vol. 23, no. 4, pp. 214–229, 1980 (DOI: 10.1145/358841.358850).
- [4] K. Hinrichs, J. Nievergelt, and P. Schorn, "Plane-sweep solves the closest pair problem elegantly", *Inform. Process. Lett.*, vol. 26, no. 5, pp. 255–261, 1988 (DOI: 10.1016/0020-0190(88)90150-0).
- [5] S. Khuller and Y. Matias, "A simple randomized sieve algorithm for the closest-pair problem", *Inform. and Comput.*, vol. 118, no. 1, pp. 34–37, 1995 (DOI: 10.1006/inco.1995.1049).
- [6] A. Andoni, P. Indyk, and I. Razenshteyn, "Approximate nearest neighbor search in high dimensions", 2018. [Online]. Available: <https://arxiv.org/pdf/1806.09823>
- [7] M. Fischler, "Distribution of minimum distance among  $N$  random points in  $d$  dimensions", Technical Rep. no. FERMILAB-TM-2170, Fermi National Accelerator Lab., Batavia, IL, USA, 2002 (DOI: 10.2172/794005).
- [8] M. Rütli, "A random number generator test suite for the C++ standard", Diploma Thesis, Institute for Theoretical Physics, ETH Zürich, 2004 [Online]. Available: <http://comp-phys.org/software/rngts/doc/main.pdf>
- [9] E. Luengo and L. García Villalba, "Recommendations on statistical randomness test batteries for cryptographic purposes", *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–34, Article no. 80, 2021 (DOI: 10.1145/3447773).
- [10] P. L'Ecuyer, J. F. Cordeau, and R. Simard, "Close-point spatial tests and their application to random number generators", *Operations Res.*, vol. 48, no. 2, pp. 189–350, 2000 (DOI: 10.1287/opre.48.2.308.12385).
- [11] P. J. Bickel and L. Breiman, "Sums of functions of nearest neighbor distances, moment bounds, limit theorems and a goodness of fit test", *Ann. of Probab.*, vol. 11, no. 1, pp. 185–214, 1983 (DOI: 10.1214/aop/1176993668).
- [12] P. L'Ecuyer and R. Simard, "TestU01: A C library for empirical testing of random number generators", *ACM Trans. on Math. Softw.*, vol. 33, no. 4, Article no. 33, pp. 1–44, 2007 (DOI: 10.1145/1268776.1268777). 33.
- [13] Nvidia V100 Tensor Core GPU, Nvidia Corporation V100 Datasheet, 2020 [Online]. Available: <https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf>
- [14] AMD Ryzen Threadripper 3990X Processor [Online]. Available: <https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3990x>
- [15] AMD Ryzen Threadripper PRO 3995WX GFLOPS performance [Online]. Available: <https://gadgetversus.com/processor/amd-ryzen-threadripper-pro-3995wx-gflops-performance/>



**Krzysztof Mańk** received his M.Sc. degree from the Military University of Technology in 1999. He has been working at the Institute of Mathematics and Cryptology of MUT ever since. His main research interests focus on stream ciphers and test of randomness.

 <https://orcid.org/0000-0002-5048-9049>

E-mail: [krzysztof.mank@wat.edu.pl](mailto:krzysztof.mank@wat.edu.pl)  
 Institute of Mathematics and Cryptology  
 Faculty of Cybernetics  
 Military University of Technology  
 Warsaw, Poland