Advances in Science and Technology Research Journal

Advances in Science and Technology Research Journal, 2026, 20(3), 287–298
https://doi.org/10.12913/22998624/213729
ISSN 2299-8624, License CC-BY 4.0

Received: 2025.08.31
Accepted: 2025.10.29
Published: 2026.02.01

Multithreaded evolutionary detection of hyperspheres in high-dimensional point cloud data

Maciej Moryń[1*], Paweł Hoser[1], Bartłomiej Kubica[1], Ryszard Kozera[1]

[1] Institute of Information Technology, Warsaw University of Life Sciences ul. Nowoursynowska 159/34, 02-776, Warsaw, Poland
* Corresponding author's e-mail: maciej_moryn@sggw.edu.pl

ABSTRACT

Object segmentation in multidimensional data spaces is a pivotal component of modern computational analysis. Frequently, accurate segmentation hinges on the detection and localisation of object boundaries. The targeted objects often exhibit spherical symmetry. This paper introduced an algorithm for the automatic detection of n-dimensional hyperspheres embedded in $(n+1)$-dimensional Euclidean space. The algorithm utilises an evolutionary computation strategy to estimate hypersphere parameters from extensive point clouds. This method demonstrates notable advantages over traditional approaches such as the Hough transform and active surface models. Preliminary results suggest strong potential of the proposed technique as well as its adaptability to broader classes of hypersurfaces, offering a promising extension for future exploration.

Keywords: multithreaded computation, approximation, image segmentation, computer vision, evolutionary algorithm.

INTRODUCTION

Detection and localisation of various subsets in multidimensional spaces is crucial in computer data analysis [1]. An example of such action may be object recognition in the images [2–5]. The latter is essential in image segmentation tasks [6]. For instance, the problem of object detection and localisation appears during a three-dimensional analysis of medical, geological, material, astronomical, and many other types of images [7,8]. In such instances, the searched objects are usually three-dimensional. It is also quite common that the data in question may have even a higher spatial dimension. Each element of a given space can also have many properties that define its location and other characteristics. Initial object location forms valuable information for further data analysis [9].

Often, the sought objects have a spherical shape. This is due to certain dynamic processes in space. It happens when elements gravitate to a certain point in space or when individual elements spread evenly from a single point in space. If the sought object is a ball, then its edge in space is a sphere. During the analysis of multidimensional data, it is often possible to efficiently find edges of a region in space. Using edge detection operators, one can extract the points belonging to the edge of spatial objects. A set of such points creates a "cloud of points" in multi-dimensional space. Detection and localisation of an object's edge based on a cloud of points is therefore an important task. Assuming that the sought object is similar to a ball in arbitrary Euclidean space (with the corresponding boundary representing a sphere).

In this paper, it was stipulated that the research task was to automatically detect an n-dimensional hypersphere based on a cloud of points. It was assumed that studied sphere is embedded in $(n+1)$ dimensional space. For a large dimension of space and a large number of cloud points, this problem is characterised by high computational complexity [9,10]. In the adopted setting, it was assume this problem is solved with the aid of adequate evolutionary methods [11–14]. Evolutionary algorithms constitute a family of algorithms

that are inspired by the process of natural evolution and are widely studied, mainly in optimisation and approximation tasks across various disciplines [12–14]. These methods have proven to be effective in solving complex problems, where traditional methods are falling short [14]. The key idea behind using evolutionary methods for approximation problems is their ability to iteratively evolve a population of candidate solutions, with each iteration coming closer to an optimal result, using mechanisms imitating natural evolution, like selection, crossover, and mutation [10–12]. Such an iterative process benefits from combinatorial and probabilistic principles that allow for the exploration of vast solution spaces.

As evolutionary algorithms are computationally intensive, it is worthwhile to implement them in a parallel manner [15,16]. This usually results in a significant improvement in the execution time [16]. Therefore, a multithreaded implementation [16,17] was applied in this paper. More details are given in the further section of this paper.

HEURISTIC METHODS OF HYPERSPHERE APPROXIMATION

Currently, there are many different methods for detecting and locating objects in multidimensional spaces based on point clouds. Such methods can also be used to detect *n*-dimensional hyperspheres. It is possible to use purely analytical methods [18]. However, such methods usually require very complex calculations, and their use is practically impossible [9]. Moreover, such methods completely fail to handle outlier points, which should be omitted. Therefore, heuristic methods are often incorporated in such cases. The RANSAC (Random Sample Consensus) method is certainly worth mentioning here, which involves an iterative approach to the desired object [19–21]. In each iteration of this algorithm, the points that do not match the object are discarded, and the object is rematched with the remaining points in the cloud. However, the method copes very poorly with large numbers of points that do not match the object. Furthermore, the method performs well predominantly for a single object in space. Over time, numerous modifications of this method have been developed, but they are not free from numerous limitations. A completely different approach is to use the Hough transform [22–25]. Hough transform-based methods turned out to be a great idea, but they do not work at all for higher-dimensional spaces. Another very important approach is the use of active surfaces and deformable models [26–28]. However, the computational complexity of these methods also dramatically increases with the dimension of the space. Currently, with a sufficiently well-defined goal function, it is possible to use many modern heuristic methods. Population-based heuristic methods inspired by biology or physics deserve special attention. Such methods include the swarm intelligence algorithm [29], the gravity algorithm [30], the firefly algorithm [31], the cuckoo search algorithm [32], the bat algorithm [33], the gray wolf algorithm [34], the multiverse algorithm [35], and differential evolution [36]. Various evolutionary methods can also be used, such as genetic algorithms or evolutionary strategies, like in this proposed method. In the latest approach, even complex deep learning tools can be used for this purpose [37]. However, such methods are poorly scalable and easily overfit in this case. In addition, deep learning methods require high complexity in the training process. The power of evolutionary methods lies in their universality and resistance to various types of noise and interference. The proposed method has specific advantages. This method works very effectively for high spatial dimensions, and is also very stable and noise-resistant. Furthermore, this method can be easily programmed in parallel, further enhancing its benefits and capabilities.

PROBLEM SOLUTION

In the conducted research, hypersphere approximation is implemented using an evolutionary strategy. The algorithm has a population of individuals at each stage of the approximation process. Each individual is a hypersphere embedded in an $(n+1)$-dimensional real space with a specific metric. The fitness function of each individual is related to the distance of the hypersphere from the point cloud in space.

Approximation

Approximation is a process of substituting certain objects (e.g. functions, surfaces, operators) with other, usually simpler objects. This

permits to simplify further calculations. Its goal is to find an object as similar as possible to the original (possibly unknown) object, minimizing the difference between them measured according to the preselected rule.

Hypersphere

In mathematical terms, a hypersphere is a well-known concept. This is a set of points in space equidistant from a certain point in the space. This definition makes sense in any metric space, so it is a very general concept. Here, an n-dimensional hypersphere is embedded in a higher-dimensional real space. Hence, the dimension of this space must be at least (n+1). Furthermore, the hypersphere is assumed to be the boundary of a ball in this space. This implies that the dimension of the space must be exactly (n+1). In such a space, a metric, a norm, and an inner product are naturally introduced. Hence, the definitions of a ball and a hypersphere are clearly defined (Figure 1). The set of all points belonging to the hypersphere is defined as follows:

$$S = \{\vec{x} \in \mathbb{R}^{n+1} : \|\vec{x} - \vec{s}\| = r\} \tag{1}$$

where: s represents a hypersphere centre and r refers to the hypersphere radius.

Here, each n-dimensional hypersphere is represented by (n+2) parameters. These are the (n+1) coordinates of the centre of the hypersphere s_i and the length of its radius r.

Distance

In this case, it is also easy to define the distance of any point from the hypersphere. The line segment connecting any point in space and the centre of the hypersphere intersects the hypersphere surface at a right angle. Therefore, the length of the line segment connecting this point with the hypersphere surface is the distance of the point from the hypersphere. This would no longer be true for an ellipsoid. Therefore, the distance of any point from the hypersphere can be calculated as follows:

$$d(\vec{x}, S) = |d(\vec{x}, s) - r| \tag{2}$$

where: $x^{\rightarrow}$ – point, S – hypersphere, s – hypersphere centre, r – hypersphere radius.

If this value is small, we can say the point is close to the hypersphere. Calculating this value has relatively low computational complexity, and we are also sure that this value is always non-negative. The aim was to find a hypersphere as close as possible to all the points in the cloud.

Evolutionary algorithm

An evolutionary algorithm is an umbrella term used to describe population-based stochastic direct search algorithms that, in one way or another, try to mimic naturally occurring evolutionary processes [38]. Those processes include: selection, crossover, and mutation. Figure 2 illustrates the typical flow of an evolutionary algorithm with a modification of multithreading usage.

Multithreading

Evolutionary algorithms typically involve computationally intensive operations on large populations of candidate solutions. Due to this high computational demand, parallelisation is often advantageous. Parallel computations can be performed either on multiple CPUs or cores within a single machine, or on a distributed system composed of several collaborating nodes (e.g., a computing

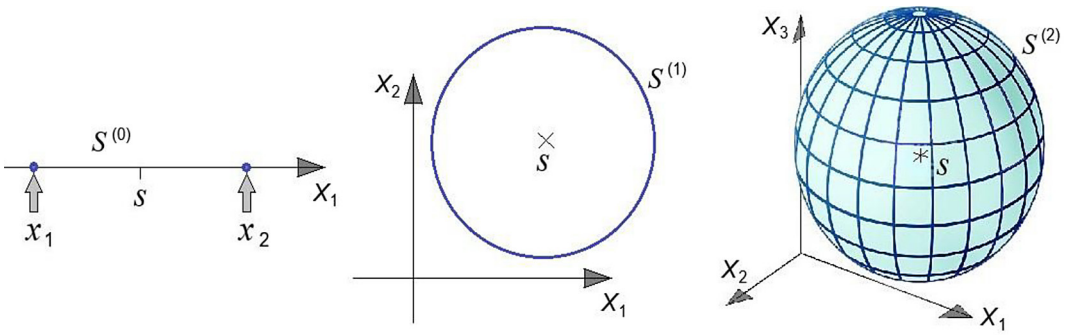

Figure 1. Three hypersphere examples: zero/one/two-dimensional hyperspheres

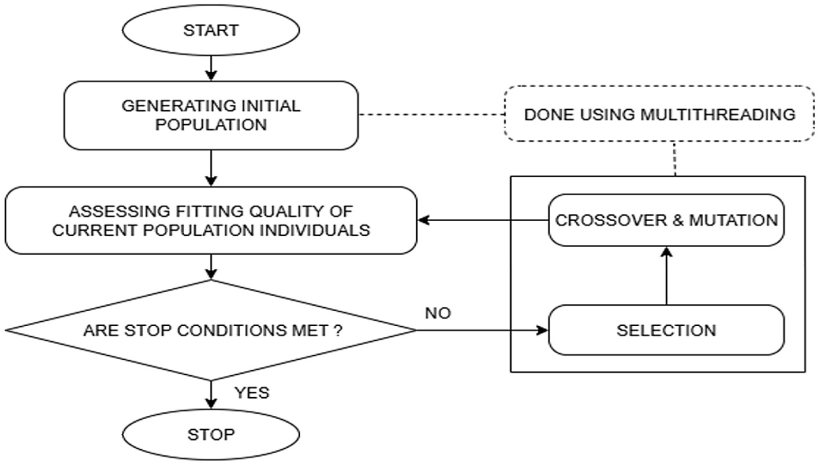

Figure 2. The evolutionary process of hypersphere approximation

cluster). In either case, the program must be decomposed into multiple components, implemented as separate processes, threads, or tasks, depending on the chosen runtime environment. Simultaneous execution of such tasks can significantly reduce total execution time. However, proper parallelisation of certain algorithms remains challenging and error-prone. In our case, a natural approach to parallelisation is the use of threads, which are more lightweight than processes and allow for efficient sharing of memory and data structures — in particular, the population of the current generation. The extent to which threads are lightweight and efficient depends on the operating system and runtime environment. Platforms such as Linux, Windows, and macOS differ in their threading models and associated performance characteristics. Nevertheless, thread-based parallelisation is supported across virtually all modern computing environments. In the presented implementation, the selection, recombination, and mutation phases are parallelised in a straightforward and efficient manner. Multiple threads independently and concurrently sample individuals from the previous generation and apply the genetic operators, each contributing a subset of new individuals to the next generation. This strategy avoids the need for critical sections, as individual solutions can be selected multiple times by different threads without causing conflicts. This parallel implementation led to a significant improvement in the overall execution speed of the algorithm. Owing to the simplicity of the multithreading model, there are no synchronisation issues. This is a certain advantage with respect to more sophisticated parallelisation models.

ALGORIHTM PARAMETERS

In the list below, a set of algorithm parameters is presented:

- mutation probability (MP): permillage chance of mutation occurring during the crossover process; checked for each gene individually
- patriarchy level (PL): permillage chance of a gene being taken from the better parent during the crossover process
- population size (PS): number of individuals (solutions) in a single population
- thread count (TC): number of threads used while creating new populations
- tournament size (TS): number of individuals that take part in the tournament

The default values for those parameters were set arbitrarily, and are shown in Table 1.

OPTIMAL PARAMETER VALUES AND METHOD LIMITATIONS

The proposed algorithm involves several parameters that can be tuned. A careful selection of their values may have a significant impact on the

Table 1. Default values for algorithm parameters

Name	Value
Mutation probability	300
Patriarchy level	600
Population size	400
Thread count	1
Tournament size	20

solver's efficiency and robustness. In the conducted experiments, an initial, non-exhaustive parameter tuning was performed, which led to the promising results presented in this paper. Due to space limitations, w full details are not provided here; however, further improvements through more extensive tuning are likely. In particular, population size may have a substantial influence on the performance of evolutionary algorithms — a topic that remains under active discussion among researchers [42]. The proposed method obviously has its limitations. It was found in this research, that with very high hypersphere dimensions, the computational complexity increases significantly. Then, the algorithm achieves the expected effect very slowly. Limitations also apply to the input data. If the examined point cloud is not at all similar to the hypersphere, the method cannot perform effectively. A possible example of such deficiency involves points distributed along an *n*-dimensional hyperplane. In such cases, the evolutionary process becomes divergent – it will constantly improve solution, never finding an optimal one. However, these are very exceptional situations.

INDIVIDUAL: STRUCTURE AND QUALITY

In evolutionary algorithms, the individual is a singular solution for the given problem. This paper describes an approximation of a hypersphere; therefore, each hypersphere can be considered an individual. The individual comprises at least one chromosome (where genes are stored), and each chromosome should contain at least one gene. For implementation purposes, the hypersphere is considered only a genotype (g), and the individual has additional properties, such as a value of the fitting function (approximation error). The hypersphere itself is described as an array of (n+2) real number values, where n is the dimension, where the solution is sought, and the last number is the hypersphere's radius. It can be mathematically as:

$$g = [g_1, g_2, \dots, g_n, g_{n+1}, g_{n+2}] \in \mathbb{R}^{n+2} \quad (3)$$

As it was stated in the second section, the formal goal of approximation is to find the minimum of an error function (which in the context of evolutionary algorithms is also an adaptation function. The input points can be defined as a $k \times n$ matrix, where k is the number of points, and n is the dimension being worked on:

$$p = \begin{bmatrix} p_{1,1} & \cdots & p_{1,n+1} \\ \vdots & \ddots & \vdots \\ p_{k,1} & \cdots & p_{k,n+1} \end{bmatrix} \in M_{k\times(n+1)} \sim \mathbb{R}^{k(n+1)} \quad (4)$$

$$s = [s_1, s_2, \dots, s_n, s_{n+1}, r] \in \mathbb{R}^{n+2} \quad (5)$$

Having defined both points and hypersphere, the approximation error for each individual can be evaluated using the given formula:

$$e = \sum_{j=1}^{k} \left(\left| \sqrt{\sum_{i=1}^{n+1} (p_{i,j} - s_i)^2} - r \right| \right) \quad (6)$$

SELECTION, CROSSOVER AND MUTATION

The most important features of the evolutionary method are three processes: selection, crossover, and mutation. In the proposed hypersphere approximation algorithm, the selection process considers the protection of the best individual, the crossover process is typical of an evolutionary strategy, and the mutation process occurs with variable intensity (relative to the generation number).

Selection

Selection, as the name suggests, is a process in which a certain number of individuals are selected to undergo crossover and mutation processes on them. The authors decided to use one of the most popular and simplest fashion for selection in their work, namely a tournament. First, two groups of n individuals are selected randomly from the population. Then, from each group, the best individual is found. Having two individuals, they became parents to a single offspring. The selection process (and then crossover and mutation) is repeated until the new population has as many individuals as the old population. The user sets the value of n, which cannot be greater than half the size of the entire population.

Crossover

Crossover is a method that allows combining certain individuals (parents) into one or more new individuals (offspring). The crossover process should be constructed so that the next generation is better suited to the environment than the previous generation. In this work, crossover was performed on two individuals (selected in the

selection process) and produced a single offspring individual. From the parents, the better one (the more suited to the environment) is called father, and the worse one – mother. Implemented crossover method is relatively simple: for each gene offspring should have, a specific random integer value from a range of [0, 1000) is generated, and if it is less than value of *patriarchy level* (set by user at the start), the offspring receives gene from its father, and if not – from the mother. The crossover operator mathematically reformulates into the following formula:

$$g_{o,i} = \begin{cases} g_{f,i} & \text{– for } (rv < PL) \\ g_{m,i} & \text{– for } (rv \geq PL) \end{cases} \quad (7)$$

where: $g_{o,i}$ – i-th offspring gene, $g_{f,i}$ – i-th father gene, $g_{m,i}$ – i-th mother gene, rv – random value from range [0, 1000), PL – patriarchy level.

Mutation

The mutation is a process that follows the crossover, but it does not always happen. Its main goal is to introduce more randomness into the evolution process, so the algorithm would not get stuck in a local optimum. The mutation probability set by user is separate for each gene. That means several genes can be mutated at the same time. The implementation is straightforward - we decrease or increase value of a gene by a certain amount. This amount is dependent on both time passed (or strictly - number of generations created) and given input data (point cloud). Maximum absolute value of this change follows a pattern of cosine function, which means it can decrease or increase over time. This approach enables both approaching local optimum, and if it turns to be bad - withdrawing from it.

RESULTS

In this work two applications are created: an approximator and a generator. As the name indicates, the generator yields a point cloud (based on user input) and the approximator searches for the best hypersphere based on the point cloud (without knowledge of how points were generated). Both applications are written in C#, using the Windows Forms library and the.NET 9.0 framework. A total of 10 hyperspheres are generated, differing in their dimension. Each hypersphere had its centre at point $[x_1, x_2 ..., x_n]$ and a radius of r, where the number of centre coordinates depends on the dimension. Two main research tasks are posed:

- finding whether the implemented algorithm works regardless of hypersphere dimension, and if it does, how less effective it is for higher-dimensional hyperspheres;
- determining how the number of threads, population size, and point cloud size affect the speed of finding optimal solutions. An "optimal solution" is a solution for which the approximation error, rounded to 5 decimal places, is equal to 0.

All tests were performed on the same workstation, with an AMD Ryzen 7 9800X3D processor, comprising eight cores and 16 threads. Most of the

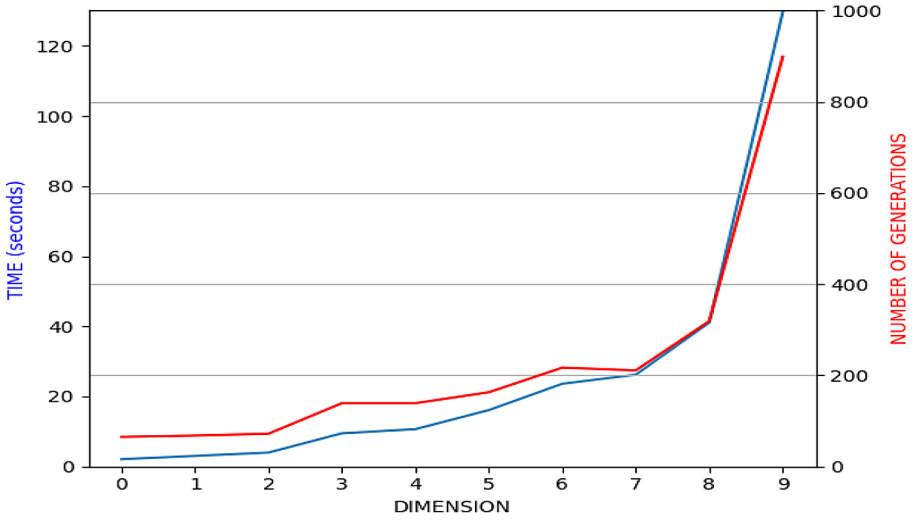

Figure 3. Dependence of time and number of generations on hypersphere dimension

tests performed used default approximation parameters (from Table 1, with the exception of the parameter currently tested) and their results are shown in tables and plots below. All time is given in seconds (with three decimal places for milliseconds).

Figure 3 presents the results of the test, how hypersphere dimension impacts time and the number of generations needed to find an optimal solution. In this test, and the three next tests, time is shown as a blue line, and its scale is on the left, and the number of generations is shown as a red line, with scale on the right side. These results show that for dimensions lesser than or equal to 7, the time needed to find a solution is relatively short, and the number of generations is relatively small. Beyond that dimension, both time and generations rapidly increase. Figure 4 shows how the execution time and generations are dependent on population size. This test was performed for a 3-dimensional hypersphere, and its results show that an increase in population size increases the time needed to find a solution and decreases the number of generations decreases. Additionally, as Figure 4 shows, an optimum for population size is visible. At 30 individuals per population, the time needed to find the optimal result is the shortest.

Another test was performed to check how the point cloud size affects the time needed to find a solution. The plot shown in Figure 5 shows the results of this test. On the basis of these results, the authors gained two pieces of information: firstly, the time needed to find a solution increases almost linearly with the size of the point cloud, and secondly, the number of generations needed to find a solution stays more or less the same.

Test for which results are shown on Figure 6 is performed to verify how the number of threads

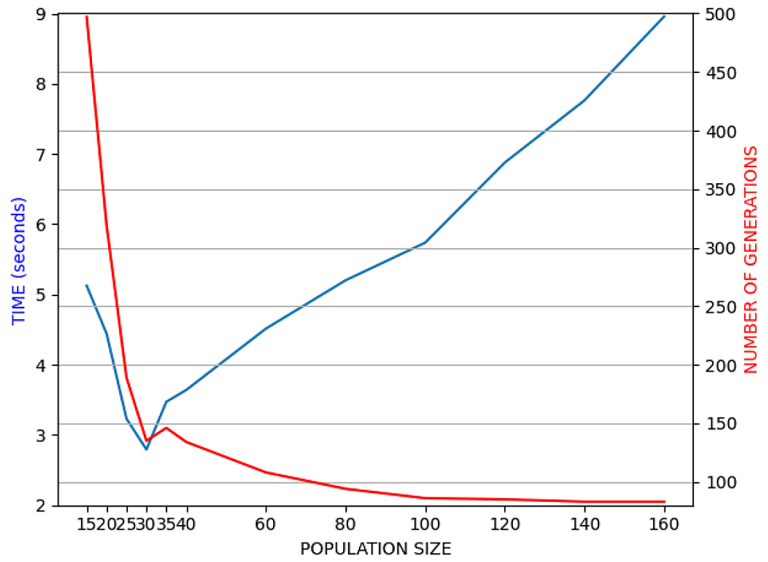

Figure 4. The dependence of time and number of generations on population size

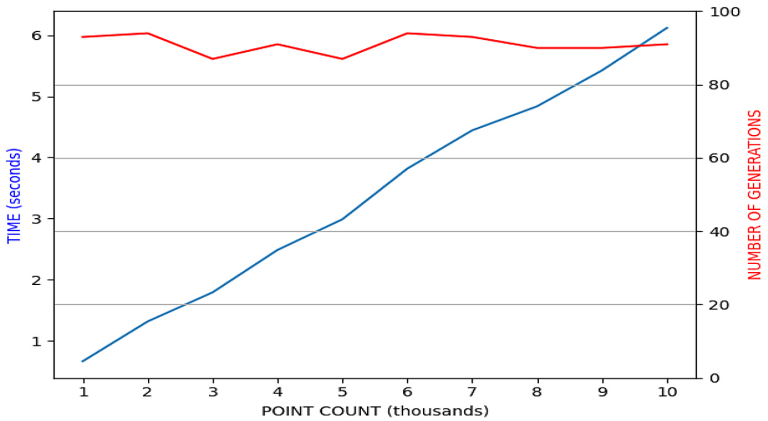

Figure 5. The dependence of time and number of generations on point cloud size

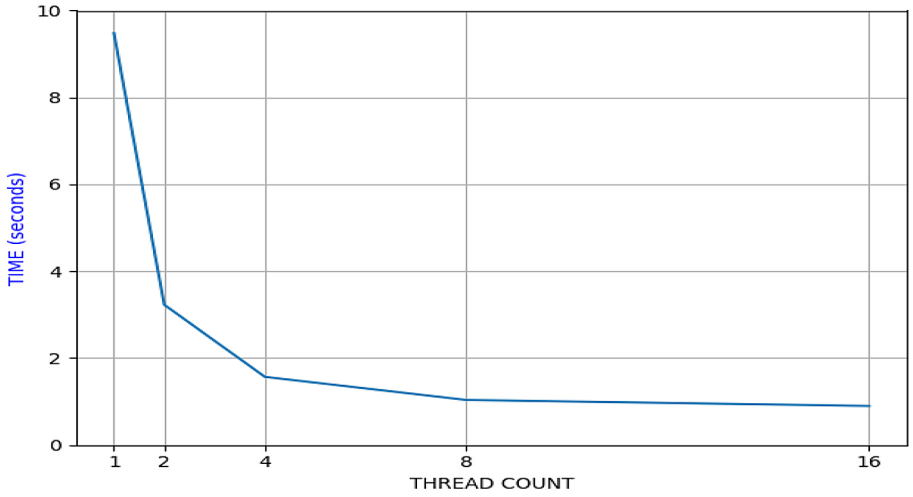

Figure 6. The dependence of time and number of generations on thread count

impacts on time of finding optimal solution. The presented values are indicative and show two things: first, using twice as much threads results in finding a solution about twice as fast, and second – while workstation has installed processor with 8 cores, using 16 physical threads was impossible – therefore, logical threads were used instead. It is also transparent that while their usage decreased time needed to find result, this improvement was not as substantial as when using physical threads.

A key feature of this type of method is its robustness to input data noise. This work involved a thorough analysis of the impact of noise on the algorithm performance. For this purpose, multiple test series were conducted. For each noise level, point clouds corresponding to the noisy hypersphere Sn were repeatedly generated, and then the hypersphere was approximated. It was assumed that all hyperspheres (of radius one) were embedded in $(n+1)$ dimensional space. Adding noise involved randomly moving points in space. This shift size was generated by a uniform distribution with the maximum magnitude of the shift represented by the parameter d. Thus, the δ parameter controlled the noise level. The approximation errors were statistically examined relative to the level of noise. The distance between the cloud and the hypersphere was also examined to confirm the accuracy of the simulations. For each noise level δ, a hundred different clouds, consisting of a thousand points in space, were generated. All distances were calculated using the L^2 metric. On the basis of the tests, it is clear that the proposed method is very resistant to noise. Only very high noise values δ cause significant approximation errors. For noise values associated with the parameter $\delta = 0.35$, the approximation errors were still very small, almost non-existent relative to the hypersphere radius. The cloud-to-hypersphere distance values confirmed the accuracy of the simulations. An interesting observation is that the dependence of error magnitude on noise is not a linear function. Additionally, a repeatability and stability analysis of the proposed method was performed. The hypersphere approximation was run multiple times for the same point cloud. The obtained parameter values were statistically analysed. These tests were repeated for various noise levels. Among other things, the error sizes and the dispersion of the obtained results were examined.

Figure 7 presents the results of the algorithm's stability test. For each point cloud, the hypersphere approximation was run one hundred times, and the standard deviation of the obtained results, averaged over all approximations, was calculated. Figure 7 shows the dependence of the standard deviation on the noise level δ. Similarly, here, an interesting observation is that the size of the standard deviation (of the approximation results) with respect to the noise level is not a linear function. On the basis of the obtained analyses, it can be concluded that the proposed method is very stable, repeatable, and highly resistant to input data noise.

Figure 8 represents a plot of dependence of time on thread count and hypersphere dimension. The presented values visualise the impact of the number of threads and dimension as well as clearly show how increase in hypersphere dimension leads to significant increase in computational complexity and how to combat this with multithreading technology. Figure 9 shows the plot of time dependency on other parameters. Here, the

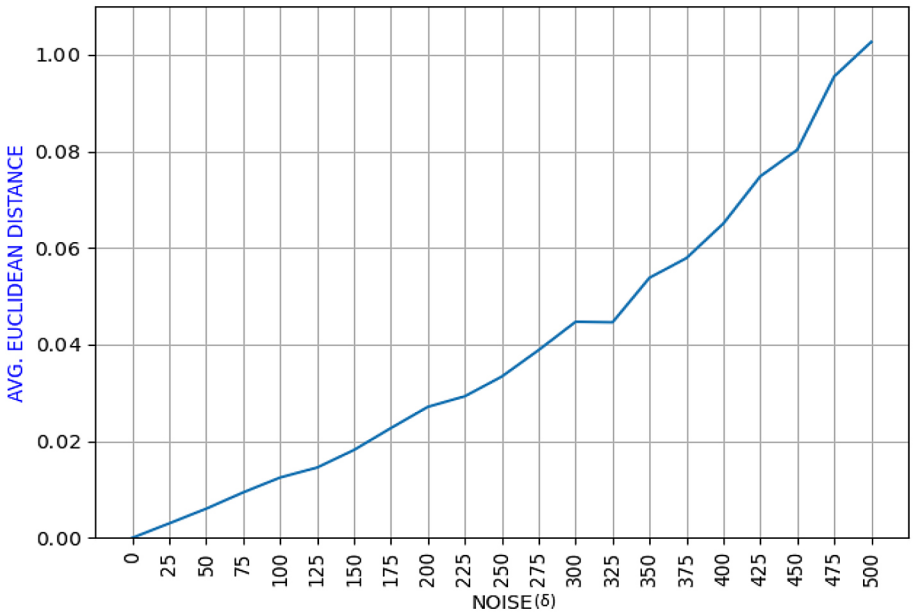

Figure 7. The dependence of the scatter of results on the noise level

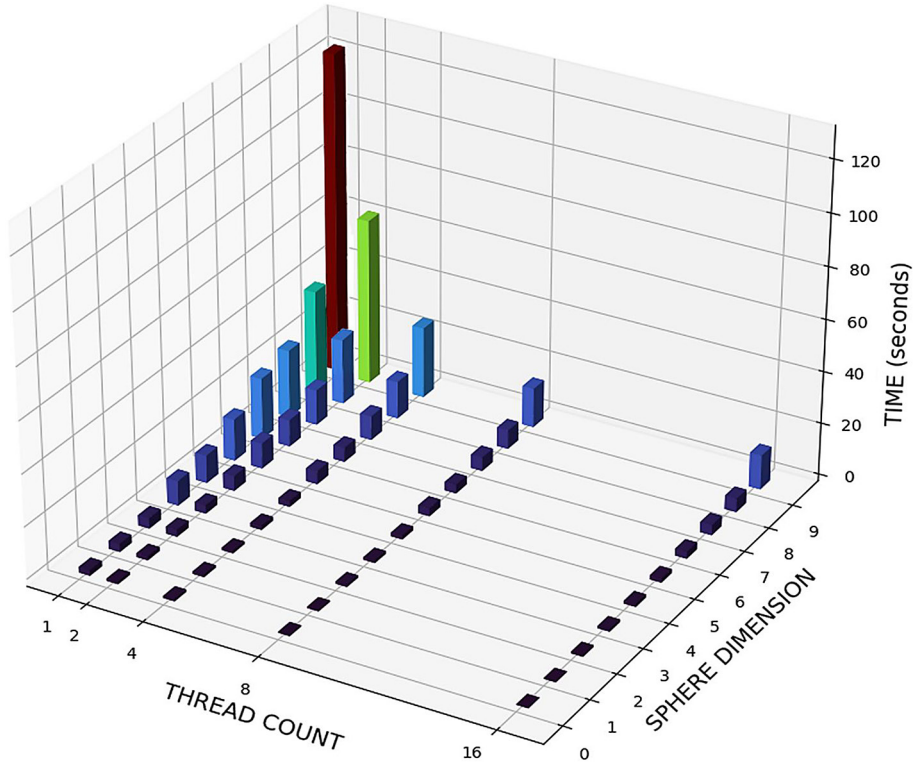

Figure 8. The dependence of time on thread count and hypersphere dimension

number of threads is replaced with population size to verify how much the new parameter affects the result (i.e. the execution time). The result for this test is fairly straightforward – the more individuals are in the population, the longer it takes for the algorithm to find an optimal solution. Additionally, while it could not be shown on this graph, with an increase in population size, the number of generations needed decreases. This test hints that there may be a certain population size for which the optimal time needed to reach a solution is shorter. For this reason another test is conducted: on a single hypersphere (namely, a 3-dimensional one) an attempt to find a solution

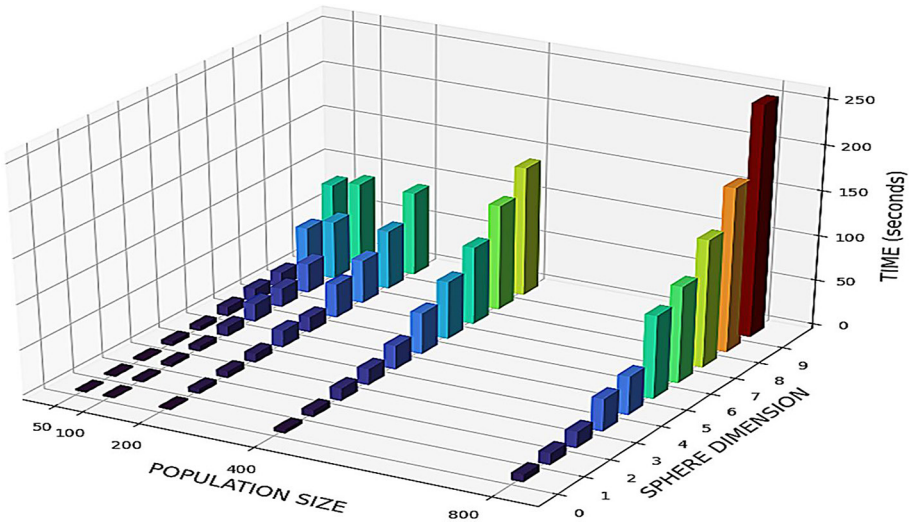

Figure 9. The dependence of time on population size and hypersphere dimension

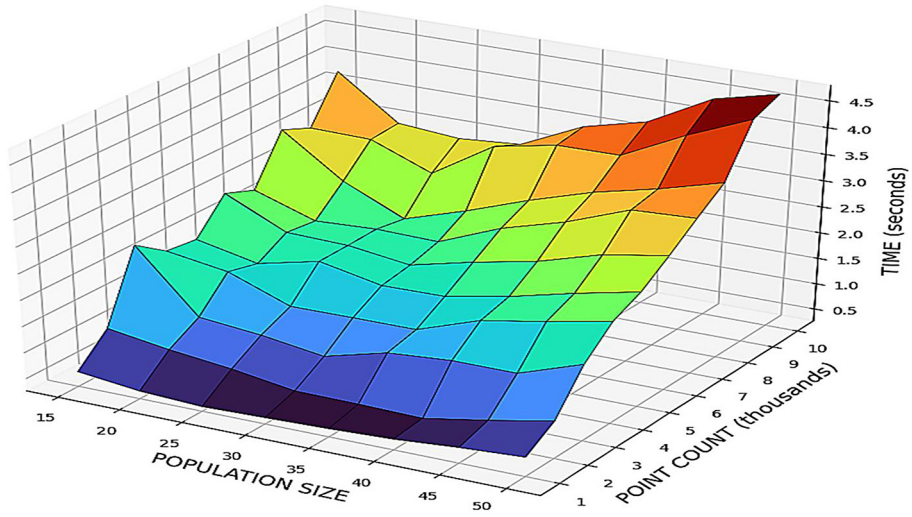

Figure 10. The dependence of time on population size and size of point cloud

for it is made, using as few individuals as possible in the population

Additionally, since hypersphere dimension is constant, this parameter is replaced with another one (size of point cloud) to extract more information. This new test (and data gained from it shown in Figure 4) is the only one for which default parameters are not used. – the tournament size for each population size is changes to half of the population size (rounded down). The results for this are shown in Figure 10 below.

Figure 10 shows the result of the test aiming to find an optimal size for the population. Upon inspecting the plot, it is visible that the optimal solution ranges between 25 and 30 individuals per population. If too small a number is used, the time to find a solution greatly increases, and sometimes, the optimal solution cannot be found. Additionally, this plot hints how the number of points in the cloud impacts on execution time, and the results are twofold. These values clearly show that an increasing size of the point cloud increases the time of finding solution. Visibly, the performance of the algorithm depends on the size of the point cloud, but this influence is small.

CONCLUSIONS

In this paper, we demonstrated how an evolutionary algorithm can be used to approximate an n-dimensional hypersphere. Multiple experiments were conducted (see the previous section) involving the estimation of centres and radii of

hyperspheres of various dimensions, with different numbers of points and diverse initial parameter settings. In all cases, the algorithm consistently produced optimal or near-optimal solutions. Taking into account the results of experiments, several additional observations are also made. Firstly, the algorithm successfully finds acceptable solutions regardless of the dimensionality of the hypersphere in question. Secondly, the execution time required to reach a satisfactory solution increases nonlinearly with the number of dimensions. Notably, this increase is marginal for dimensions below 7, indicating good scalability in low to moderate-dimensional spaces. Thirdly, the total execution time decreases approximately linearly with the number of threads used, with minor deviations attributable to thread scheduling and system-specific factors. The proposed method offers broad applicability across diverse fields of science and technology. The effectiveness of such methods in surface-to-point cloud matching has been demonstrated in both medical and engineering domains [39–41]. Notably, the hypersphere approximation technique facilitates segmentation of high-dimensional objects with exceptional efficiency, owing to its rapid execution, parallelisability, and strong resilience to various forms of noise and interference.

REFERENCES

1. Etemadpour, R., Kubik, T., Gracia, J., Tory, M., Forbes, A. Choosing visualization techniques for multidimensional data analysis. Communications in Computer and Information Science, 2016; 662: 15–30.
2. Cootes, T. F., Taylor, C. J., Lanitis, A. Active shape models: Evaluation of a multi-resolution method for improving image search. BMVC 1994; 1: 327–336.
3. Jahne, B. Digital Image Processing. Springer Verlag. 1995. https://doi.org/10.1007/978-3-662-03174-2
4. Malina, W., Smiatacz, M. Cyfrowe Przetwarzanie Obrazów. EXIT. 2012.
5. Bovik, A. Handbook of Image and Video Processing. Academy Press. 2000.
6. Lenkiewicz, P., Pereira, M., Freire, M. M., Fernandes, J. The whole mesh deformation model: a fast image segmentation method suitable for effective parallelization. EURASIP Journal on Advances in Signal Processing: 2013; 1–17. https://doi.org/10.1186/1687-6180-2013-168
7. Anđelić, N., Baressi Šegota, S., Glučina, M., Car, Z. Estimation of interaction locations in super cryogenic dark matter search detectors using genetic programming - symbolic regression method. Appl. Sci. 2023; 13: 2059. https://doi.org/10.3390/app13042059
8. Rodtook, A., Kirimasthong, K., Lohitvisate, W., Makhanov, S. S. Automatic initialization of active contours and level set method in ultrasound images of breast abnormalities. Pattern Recognition 2018; 79: 172–182. https://doi.org/10.1016/j.patrec.2017.05.001
9. Papadimitriou C.H. Computational Complexity. Addison-Wesley Publishing Company. 2012.
10. Sanjeev A., Boaz B. Computational Complexity: A Modern Approach. Cambridge University Press. 2009.
11. Tan, Z., Luo, L., Zhong, J. Knowledge transfer in evolutionary multi-task optimization: A survey. Appl. Soft Comput. 2023; 138: 110182. https://doi.org/10.1016/j.asoc.2023.110182
12. Hoser P., Antoniuk I., Strzęciwilk D. Algorithm for optimization of multi-spindle drilling machine based on evolution method. Advances in Soft and Hard Computing, January 2019.
13. Koza J.R. Genetic Programing. A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems. Report No. STAN-CS-90-1314, Stanford University. 1990.
14. Cicirello, V.A. Evolutionary Computation: Theories, Techniques, and Applications. Computer Science, Stockton University, 101 Vera King Farris Dr, Galloway, NJ 08205, USA. Appl. Sci. 2024; 14(6): 2542. https://doi.org/10.3390/app14062542
15. Czech. Z., Introduction to Parallel Computations (in Polish), PWN, 2013.
16. Karbowski A., Niewiadomska-Szynkiewicz E. (eds.), Parallel and Distributed Programming: Collective Work. (in Polish), OWPW, 2009.
17. Hoser P., Kubica B.J., Ochnio L., The problem of thread synchronization frequency in the computer simulation of a dynamic system, ESM'2023, 2023; 26–30.
18. Shi W., Tong P., Bi X. Moving-least-squares-enhanced 3D object detection for 4D millimeter-wave radar. Remote Sensing, Aug. 2025; 17(8): 1465, https://doi.org/10.3390/rs17081465
19. Fischler, M. A., Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 1981; 24(6): 381–395. https://doi.org/10.1145/358669.358692
20. Martínez-Otzeta J. M., Rodríguez-Moreno I., I. Mendialdua, Sierra B., "RANSAC for robotic applications: A survey," Sensors, Jan. 2023; 23(1): 327. https://doi.org/10.3390/s23010327
21. Cavalli L., Barath D., Pollefeys M., Larsson V. Consensus-adaptive RANSAC. arXiv preprint arXiv:2307.14030, Jul. 2023. https://doi.org/10.48550/arXiv.2307.14030

22. Hough P. V. C. Analysis of Bubble Chamber Pictures. Conf. Proc. C. 1959; 590914: 554–558.
23. Hart P. E. How the hough transform was invented. IEEE Signal Processing Magazine 2009; 26(6); 18–22, https://doi.org/10.1109/MSP.2009.934181
24. van Ginkel M., Luengo Hendriks C. L., Van Vliet L. J. A short introduction to the Radon and Hough transforms and how they relate to each other. Quantitative Imaging Group Technical Report Series. Number QI-2004-01. 2004.
25. Hassanein A. S., Mohammad S., Sameer M., Ragab M. E. A Survey on hough transform, theory, techniques and applications. Source-arXiv. February 2015.
26. Lenkiewicz, P., Pereira, M., Freire, M. M., Fernandes, J. The whole mesh deformation model: a fast image segmentation method suitable for effective parallelization. EURASIP Journal on Advances in Signal Processing: 2013; 1–17.
27. Sefti R., Sbibih D., Jennane R. A CNN-based spline active surface method with an after-balancing step for 3D medical image segmentation. Mathematics and Computers in Simulation. Elsevier. 2024; 225: 607–618.
28. Molnar J., Tasnadi E., Kintses B., Farkas Z., Pal C., Horvath P.. Active Surfaces for Selective Object Segmentation in 3D. 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA). https://doi.org/10.1109/DICTA.2017.8227401 IEEE, Sydney, NSW, Australia. 2017.
29. Kennedy, J., Eberhart, R. Particle swarm optimization. In Proceedings of the International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; 4: 1942–1948.
30. Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S. GSA: A gravitational search algorithm. Information Sciences, 2009; 179(13): 2232–2248.
31. Yang X.-S., Firefly algorithm, stochastic test functions and design optimization. International Journal of Bio-Inspired Computation, 2010; 2: 78-84.
32. Yang X.-S., S. Deb. Cuckoo search via Lévy flights. Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, 2009; 210–214. https://doi.org/10.1108/02644401211235834
33. Yang X.-S., Gandomi A.H. Bat algorithm: a novel approach for global engineering optimization. Eng Comput, 2012; 29(5): 464–483, https://doi.org/10.1108/02644401211235834
34. Mirjalili S., Mirjalili S.M., Lewis A. Grey wolf optimizer. Adv Eng Softw, 2014; 69: 46-61, https://doi.org/10.1016/j.advengsoft.2013.12.007
35. Mirjalili S., Mirjalili S.M., Hatamlou A. Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl, 2015; 27(2): 495–513. https://doi.org/10.1007/s00521-015-1870-7
36. Rainer S., Price K. Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997; 11(4): 341–359. https://doi.org/10.3390/robotics12040100
37. Ding Z., Sun Y., Xu S. et al. Recent advances and perspectives in deep learning techniques for 3D point cloud data processing. Robotics, 2023; 12(4): 100. https://doi.org/10.3390/robotics12040100
38. Bartz-Beielstein, T., Branke, J., Mersmann, O. Overview: Evolutionary Algorithms, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, May 2014.
39. Berger M., Tagliasacchi A., Seversky L., Alliez P., Guennebaud G. A survey of surface reconstruction from point clouds. Computer Graphics Forum, 2016; 27. hal-01348404v2.
40. Tian H., Xu K. Surface reconstruction from point clouds via grid-based intersection prediction. Computer Science, Computer Vision and Pattern Recognition, arXiv:2403.14085v1 [cs.CV]. 9 April 2024.
41. Trung-Thien T., Van-Toan C., Denis L. eSphere: extracting spheres from unorganized point clouds. Visual Computer 2016; 32: 1205–1222.
42. Galar, R. Simulation of local evolutionary dynamics of small populations. Biol. Cybern. 1991; 65: 37–45. https://doi.org/1007/BF00197288