Solving Inverse Kinematics Problems Using Strait of Messina Evolutionary Algorithm

Szilárd Dávid Fecht*
Márk Domonkos
János Botzheim
fecht@inf.elte.hu
domonkos@inf.elte.hu
botzheim@inf.elte.hu
Department of Artificial Intelligence
Faculty of Informatics
ELTE Eötvös Loránd University
Pázmány P. sétány 1/A, 1117 Budapest, Hungary

Abstract

In this paper, we present our novel algorithm called Strait of Messina Evolutionary Algorithm (SOMA). This algorithm takes inspiration from different already well-researched algorithm families such as Bacterial Evolutionary Algorithms, Genetic Algorithms, and Differential Evolution. In SOMA, a three-phase evolutionary process during the iterations is realized. First, the Scylla Operator is executed, followed by the Charybdis Bacterial Mutation, and finally, a Differential Evolution-like phase completes the iteration. We tested our algorithm on four continuous benchmarks (Griewank, Levy, Rosenbrock, and Schweffel) with different dimensional conditions (10, 30, 50, 100), and on a set of inverse kinematics problems consisting of 100 different problems from the same robot model. We used Grey Wolf Optimization, Crayfish Optimization Algorithm, Whale Optimization Algorithm, and Dragonfly Algorithm for comparative analysis. Results show superior results in most of the cases applied, except for Griewank and Rosenbrock functions, where the Crayfish Optimization Algorithm turned out to be a good competitor. The presented algorithm's sourcecode for reproducibility can be accessed from https://github.com/SDFecht/SOMA.

*Corresponding author.

IntRobFAIR '25, Budapest, Hungary

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts: • **Applied computing** → **Engineering**; • **General and reference** → **Empirical studies**; • **Computing methodologies** → **Continuous space search**.

Keywords: Evolutionary Computing, Optimization, Inverse Kinematics, Robotics

ACM Reference Format:
Szilárd Dávid Fecht, Márk Domonkos, and János Botzheim. 2025. Solving Inverse Kinematics Problems Using Strait of Messina Evolutionary Algorithm. In *Proceedings of IntRobFAIR '25.* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Evolutionary computing is a nature-inspired paradigm of artificial intelligence that is widely applied in various fields. In our proposed algorithm, we present two novel operators called Scylla Operator and Charybdis Bacterial mutation, named after the two Greek mythological figures, the Scylla (a huge monster with many tentacles) and the Charybdis (another huge monster capable of dragging whole ships to the bottom of the sea). The algorithm's structure proposed in this study draws inspiration from Homer's Odyssey, where the hero must navigate the treacherous Strait of Messina, flanked by the two of these mythical threats: Scylla and Charybdis. Analogously, the Strait of Messina Evolutionary Algorithm (SOMA) attempts to steer the search process between two algorithmic extremes to navigate the optimization landscape safely and efficiently, broad, multi-directional exploration (as it is done with Scylla Operator), and a greedy, rapid convergence (provided by the Charybdis Bacterial Mutation). These two main components of the algorithm, the Scylla Operator and the Charybdis Bacterial Mutation, reflect this duality in their design and function.

This problem of controlled balancing of exploration and exploitation has already become one of the central motivators for designing novel algorithms. An extensively used application is the optimization of parameters in control and robotics systems. As such, in [10] an extended version of

the Grey Wolf Optimizer (GWO) was used to optimize a nonlinear sliding mode controller, which was applied in a two-degree-of-freedom manipulator, that achieved better tracking accuracy and higher disturbance robustness compared to other standard controllers. An improvement on the Whale Optimization Algorithm (WOA) was presented in [12], which was able to find shorter and smoother collision-free paths in indoor robot navigation tasks with faster convergence. Another example of the usage of WOA in the field of control systems is in [5], where the WOA-tuned PID controller achieved better results in tracking performance for a robot arm controller compared to the Particle Swarm Optimizer and Grey Wolf Optimizer. An Evolving Robotic Dragonfly Algorithm was implemented in [3], and applied in a multi-robot environment for target search, which was able to reduce the search iterations and increase the success rate. An optimal pixel selection for stenographic embedding was successfully done by a Crayfish Optimization-based algorithm in [13]. In [9], bacterial programming was used for the estimation of the kinematic chain of a construction vehicle.

The main contribution of this paper is a new hybrid evolutionary algorithm that combines elements from genetic algorithms, differential evolution, and bacterial mutation in a non-traditional way. It introduces a dynamically adjusted, gene-level mutation factor and a simple local search method to improve stability and convergence. The algorithm is tested on well-known continuous benchmark functions as well as on a real-world-inspired inverse kinematics task. Results show that the method performs well, especially in terms of consistent fitness values and overall robustness.

After this introduction, the paper is structured as follows: in Section 2, we introduce our algorithm; which then is followed by the presentation of our experimental methodology in Section 3; in Section 4 we present the results of the experiments conducted; and finally in Section 5 we draw the conclusions of this paper.

2 Algorithm

The algorithm draws inspiration from several existing algorithms, including Bacterial Evolutionary Algorithms (BEA), Differential Evolution (DE), and Genetic Algorithms (GA). The proposed algorithm attempts to combine the advantages of the mentioned methods, integrating and utilizing them effectively to achieve its goal.

The evolutionary flow involves strong genetic evolutionary elements, although not in the classical order as visible in Fig. 1. The presented algorithm can be divided into three main components. The first two components are the Scylla Operator and the Charybdis Bacterial Mutation mechanisms, which influence the direction of the search during the third component — the differential evolution step.

In Homer's Odyssey, Scylla is described as a creature capable of coordinated attacks across multiple directions with its tentacles. Inspired by this concept, our genetic algorithm employs randomized multiplications and mutations that operate in parallel across different regions of the search space. The Scylla Operator first applies a mutation to the population, then the gene pool is refined by a uniform crossover with a population composed only of the best individual. The Scylla Operator is executed at every iteration and plays a key role in maintaining a well-balanced search by bridging between exploration and exploitation.

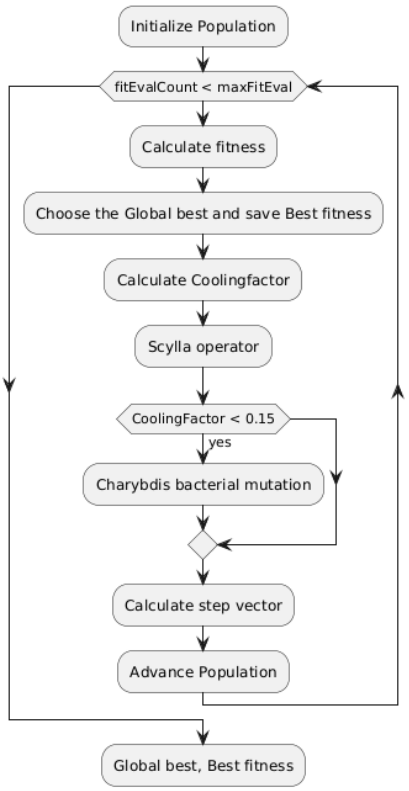

Figure 1. Flowchart of the Strait of Messina Evolutionary Algorithm (SOMA).

The Charybdis Bacterial Mutation phase is inspired by bacterial mutation strategies. In contrast to the Scylla Operator, the Charybdis Bacterial Mutation is applied only in the early stages of the search. Like its mythological namesake, which is capable of drawing ships to the depths of the sea, this algorithmic component's goal is to exhibit a rapid convergence. Owing to this behavior, the operator needs to be precisely controlled, or it can easily inhibit the optimization from achieving the global optimum.

2.1 Scylla Operator

The Scylla Operator takes inspiration from the DE algorithms, where the scaling factor is used to weight the vector difference between two randomly selected individuals, to

define the direction, and effectively guide the search toward potentially better regions.

In the Scylla Operator, this constant is being re-evaluated at each generation to influence the evolutionary process adaptively. This is individually computed for each gene within the population in each generation, based on Equation (1).

$$\mathbf{ScyM} = ((1.01+\frac{1}{1+e^{-12+r\cdot 4}}\cdot(1-CF)+(4+r\cdot 2))\cdot\|r_n\cdot\epsilon\|)^{2\cdot r-1} \tag{1}$$

where ScyM is an individuals-by-genes matrix that assigns a distinct mutation scaling factor to each gene of every individual; r is a randomly selected constant individuals × genes sized matrix that follows a uniform distribution with zero mean and 1 standard deviation; CF cooling factor is defined as the ratio of the number of fitness evaluations performed so far to the maximum allowed number of evaluations, which allows for a smooth decay of influence as the optimization progresses; r_n is a randomly selected constant individuals × genes sized matrix that follows a normal distribution with zero mean and unit variance; ϵ is an exponentially distributed random variable individuals × genes sized matrix.

The visualization of the development of the ScyM matrix values can be seen in Fig. 2. This development of the values resembles the tentacles of a monster, especially if this visualization is done after each completed run, and the resulting ScyM value developments are shown after each other.

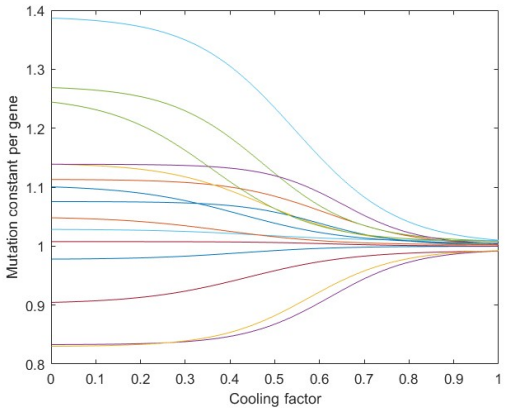

Figure 2. Example representation of ScyM values over time

We apply this mutation constant matrix to the whole population and clip any outlier values, thus obtaining our first modified output population. Then, a randomly selected individual is replaced with the arithmetic mean vector of this modified population.

We construct a set of individuals of the same size by populating it with copies of the all-time best (global best) individual. This set is also scaled with the same Scylla mutation constant matrix and is restricted to the boundaries. One individual is then replaced by a random permutation global best individual genotype to support diversity in the exchange of genes.

Finally, we combine the two populations using a uniform crossover, thus forming the output population of the Scylla operator. This whole process is visible in Fig. 3

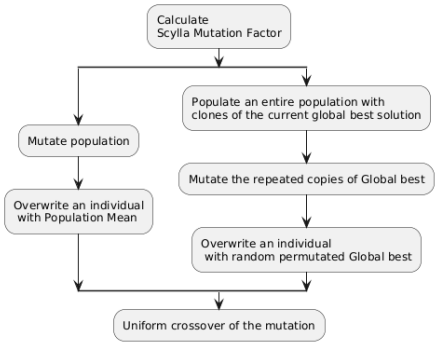

Figure 3. Flowchart of the Scylla operator

2.2 Charybdis bacterial mutation

Bacterial mutation is a simple yet effective genetic operator that improves the selected bacterium by creating multiple duplicates, mutating a chosen chromosome in each, and then selecting the one with the highest fitness to replace the original. Charybdis bacterial mutation aims to achieve a similar objective through a slightly different approach, with a calculated cooling factor to regulate the activation of this operator within the evolutionary flow to balance its greediness. This process can be seen in Fig. 4.

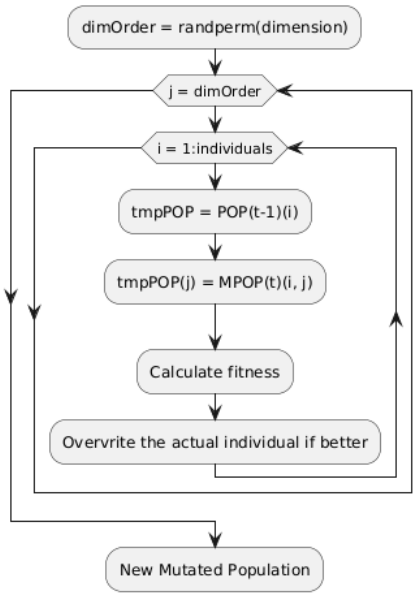

Figure 4. Flowchart of the Charybdis bacterial mutation

When the Charybdis bacterial mutation is activated, it generates a randomized gene transfer sequence. For each individual, it temporarily replaces the gene at the current position with the corresponding gene from the output population of the Scylla Operator and evaluates the cost. If the new solution is not worse than the original, the change is accepted. This position-wise local search allows gradual improvement by selectively adopting beneficial gene mutations without affecting the rest of the solution. However, this method must be used with caution, as it tends to converge rapidly, which may not necessarily lead toward the global minimum. To avoid this, the algorithm is applied exclusively in the early stages of optimization.

2.3 Computation of the Differential Evolution Step

The final mutated population is created by combining the results of the two operators above, and effectively contains all the essential information required for the algorithm to make progress toward the desired good solutions. Equation (2) defines the step that directs the algorithm toward a better global solution in this phase.

$$\mathbf{step(t)} = (0.75 + 0.25 \cdot cos(\pi \cdot r)) \cdot (\mathbf{MPOP(t)} - \mathbf{POP(t-1)}) \tag{2}$$

where r is a randomly selected constant individuals × genes sized matrix that follows a uniform distribution with zero mean and 1 standard deviation; **MPOP(t)** is the mutated population by the Scylla Operator and (in case) Charybdis Bacterial Mutation, **POP(t-1)** is the population of the previous generation.

The algorithm applies the computed step to each individual from the previous generation (see Eq. (3)), evaluates their fitness, selects and stores the best-performing individual as global best, and then initiates the next generation.

$$\mathbf{POP(t)} = \mathbf{POP(t-1)} + \mathbf{step(t)} \tag{3}$$

3 Methodology

The experiments were conducted using five different algorithms (including the presented algorithm) under standardized conditions. Executions were performed in MATLAB R2024b, employing the default random number generator settings. We relied on the original, publicly available MATLAB implementations for each algorithm, preserving the default parameter configurations. The number of fitness evaluations was limited to dimension × 10,000 for all algorithms.

To benchmark our method, we selected four additional algorithms for comparative analysis. We evaluated all methods on two sets of problems: four standard continuous optimization functions, and 100 inverse kinematics tasks involving the same robot model. The continuous benchmark functions were evaluated on 10, 30, 50, and 100 dimensions, with each configuration evaluated in 100 independent runs. In all cases, the objective was to minimize the objective function's value. In the inverse kinematics problem, 100 configurations were defined, each algorithm being executed 15 times for each configuration (and thus a distinct inverse kinematics problem).

3.1 Competing Algorithms

To validate both our solution to the inverse kinematics problem and the proposed algorithm itself, we selected four additional algorithms for a comparative analysis, aiming to include both classical and more modern approaches. These algorithms have publicly accessible MATLAB implementations, which ensure consistency and eliminate the potential biases arising from implementation.

The **Crayfish Optimization Algorithm (COA)** [2] is a nature-inspired metaheuristic method, based on the behavioral patterns of crayfish. The algorithm uses a randomly generated number between 20 and 35 to determine which of the three available stages to execute. If the generated value exceeds 30, the algorithm uses a Bernoulli distribution to choose between the following two options: the Summer Resort stage, which corresponds to exploration, or the Competition stage, where candidate solutions are pitted against each other. If the value is below 30, the Foraging stage is executed, representing the exploitation phase.

The **Dragonfly Algorithm (DA)** [6], is a swarm-based algorithm that tries to balance exploration and exploitation by using five distinct (static and dynamic) behavioral patterns of the dragonflies. These behaviors include Separation (avoidance of collision with the neighbors), Alignment (velocity matching with neighboring individuals), Cohesion (movement towards the center of the swarm), Attraction to food source (movement towards the optimal solution), and Distraction from enemies (to avoid poor solutions).

Grey Wolf Optimizer (GWO) [8] is said to be inspired by the hunting behavior of grey wolves. This algorithm is separating the population into four roles. "Alpha", "Beta", and "Delta" are the three best solutions in order, respectively, while all other individuals are assigned to "Omega" role. The algorithm has three main phases called "Encircling prey", "Hunting", and "Attacking prey".

Whale Optimization Algorithm (WOA) [7] is based on the bubble-net hunting strategy of humpback whales. This algorithm has three main concepts: "encircling prey" (assuming the optimal solution, and updating the position relative to it), "Bubble-net hunting" (where a shrinking encircling mechanism and spiral updating of the position is done), and a "search for prey".

3.2 Continuous Benchmarks

To assess the algorithm's performance, we used four continuous functions as benchmark functions called Griewank- [4], Levy's- [4], Rosenbrock- [4], and Schwefel function [4].

The Griewank function is a complex, multimodal optimization benchmark characterized by a product-term interaction and a rapidly oscillating landscape. Due to its many widespread local minima and a known global optimum, it is frequently used to evaluate the performance of global optimization algorithms. The problem can be formulated as shown in Eq. (4).

$$\begin{aligned} \text{minimize} \quad & f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \\ \text{subject to} \quad & -100 \le x_i \le 100, \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{4}$$

where $\mathbf{x}$ is the individual and n is the function's dimension.

The Levy's function is a multimodal and non-separable function designed to challenge the exploration capabilities of optimization algorithms. Its landscape contains numerous local minima, with a global minimum situated in a relatively narrow basin, making convergence difficult. The benchmark formulation is provided in Eq. (5).

$$\begin{aligned} \text{minimize} \quad f(x) = {} & \sin^2(\pi w_1) \\ & + \sum_{i=1}^{n-1} (w_i - 1)^2 \left[1 + 10\sin^2(\pi w_i + 1)\right] \\ & + (w_n - 1)^2 \left[1 + \sin^2(2\pi w_n)\right] \\ \text{subject to} \quad & -10 \le x_i \le 10, \quad \text{for } i = 1, \ldots, n \\ \text{with} \quad & w_i = 1 + \frac{x_i - 1}{4} \end{aligned} \tag{5}$$

where $\mathbf{x}$ is the individual and n is the function's dimension.

The Rosenbrock function, also known as the Banana function, is a classic unimodal benchmark that poses difficulties due to its narrow, curved valley leading to the global minimum. Although the function has a single global optimum, the ill-conditioning and strong non-linearity of its surface test the precision of optimization strategies. The problem is defined in Eq. (6).

$$\begin{aligned} \text{minimize} \quad & f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right] \\ \text{subject to} \quad & -100 \le x_i \le 100, \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{6}$$

where $\mathbf{x}$ is the individual and n is the function's dimension.

The Schwefel function is a highly multimodal and non-separable function, which is commonly used as an optimization benchmark. The problem can be stated as in Eq. (7).

$$\begin{aligned} \text{minimize} \quad & f(x) = 418.9829 \cdot n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|}) \\ \text{subject to} \quad & -500 \le x_i \le 500, \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{7}$$

where $\mathbf{x}$ is the individual and n is the function's dimension.

3.3 Inverse Kinematics Estimation

For a more real-life-like benchmarking, we used an inverse kinematics problem for the ROBOTIS OpenManipulator-X in the Denavit-Hartenberg (DH) convention [1]. This problem has a nonlinear nature, which is often framed as optimization. The OpenManipulator-X is a small-scale, open-source, 4+1 DoF robot (the fifth DoF being in the end-effector).

To estimate the joint coordinates, we first created the theoretical model of the robot. We relied on the manufacturer's manual [11] to create the robot model's structure and add the needed DH parameters for the modeling. The resulting structural model is visible in the following Fig. 5, and the theoretical parameters are listed in Table 1.

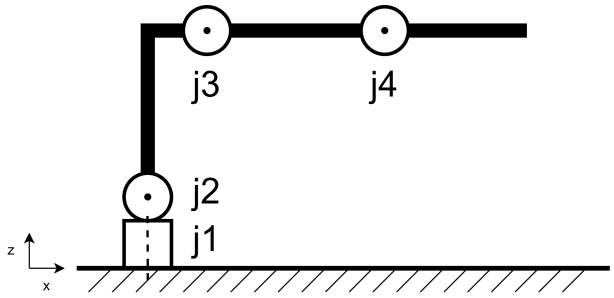

Figure 5. Mechanical structural model of the ROBOTIS Manipulator-X 4+1 DoF robotic arm. $j1$, $j2$, $j3$, and $j4$ are the robotic arm's joints. The rotational axes of the joint are represented with either a dashed line (if parallel with the origin's z axis, aligned similarly) or a dot (when parallel with the origin's y axis, rotating anti-clockwise).

Transformation	a	α	d	θ
1	0	0	0.077	0
2	0	$-\pi/2$	0	$q1$
3	0.1302	0	0	$q2 - 1.3854$
4	0.124	0	0	$q3 + 1.3854$
5	0.126	$\pi/2$	0	$q4$

Table 1. Theoretical DH parameters of the ROBOTIS OpenManipulator-X, with the fifth transformation resulting in the end effector's position and orientation. The values are rounded to 4 decimals when needed. Parameters $q1$, $q2$, $q3$, and $q4$ are the joint coordinate values of the robot. Meaning of the parameters a, α, d, and θ can be found in [1].

As it was stated earlier, we created 100 distinct and plausible joint coordinates - end effector position and orientation pairs as data for our problem. The fitness function for one case (and thus one problem) to solve can be seen in Eq. (8).

$$fitness = 100 \cdot \text{POS\_e} + \text{ANG\_e} \tag{8}$$

where *POS_e* corresponds to the calculated end effector's positional error, calculated with an Euclidean norm, while *ANG_e* corresponds to the calculated end effector's orientation error calculated with Eq. (9).

$$ANG\_e = \arccos(\cos\theta)^2 \tag{9}$$

where θ is the angular distance between two rotation matrices and,

$$\cos\theta = \frac{trace(\mathbf{ROT_{ref}}' \cdot \mathbf{ROT_{calc}}) - 1}{2} \tag{10}$$

where the $\mathbf{ROT_{ref}}$ corresponds to the rotation matrix of the end effector from the reference data, and $\mathbf{ROT_{calc}}$ corresponds to the calculated rotation matrix.

4 Results

This section presents the experimental results obtained through the comparative evaluation of the proposed algorithm and four reference methods. The results are divided into two parts: Section 4.1 reports performance on continuous benchmark functions, while Section 4.2 focuses on a domain-specific inverse kinematics task. These two test environments jointly enable a balanced assessment of the algorithms' general optimization capabilities and their behavior in a constrained, real-world-inspired problem setting.

4.1 Continuous Benchmarks

In this subsection, we evaluate the algorithms on four classical benchmark functions — Griewank, Levy, Rosenbrock, and Schwefel — using four different problem dimensionalities: 10, 30, 50, and 100. The results, shown in Table 2, represent the arithmetic mean of the best fitness values obtained over 100 independent runs for each function–dimension pair. These tests aim to assess the algorithms' general optimization capabilities, their scalability with increasing dimensionality, and their sensitivity to function landscape characteristics. Lower values in the table indicate better optimization outcomes.

In this Table 2, it is visible that in most cases SOMA outperforms the other used control algorithms. From this, only three cases are exceptions, which are the higher-dimensional problem versions of the Rosenbrock function; in these cases, COA wins, but usually with a limited difference. COA also seems a good competitor in the case of the Griewank algorithms, where in three cases it had similar results to SOMA. In the case of the Griewank function, GWO also became a similarly good competitor.

A typical evolution of the fitness is visible in Fig. 6 for the Levy's function 100D case. From this figure, we can see that SOMA has a faster convergence compared to the other competing algorithms.

4.2 Inverse Kinematics

This subsection examines the algorithmic performance in the context of inverse kinematics tasks, involving 100 distinct inverse kinematics problems. We analyzed ranking outcomes, how frequently each algorithm achieved first or second place, as it is presented in Table 3. From this Table, it is visible that in most cases, the average performance over the 15 runs is superior in the case of the SOMA algorithm, which is followed by the COA algorithm.

10 dimensions					
Function	**SOMA**	**GWO**	**COA**	**DA**	**WOA**
Griewank ↓	**0**	0, 0045	**0**	0, 20998	0, 016
Levy ↓	**0.0**	0.1067	0.2829	1.0551	0.0072
Rosenbrock ↓	**3.06**	6.2500	3.6593	12*e*4	4.8387
Schwefel ↓	**0.018**	1308.2	633.1	1386.7	369.8
30 dimensions					
Function	**SOMA**	**GWO**	**COA**	**DA**	**WOA**
Griewank ↓	**0**	7, 46*e* − 05	**0**	1, 3709	0, 0003
Levy ↓	**0.0**	1.0716	1.4393	15.2804	0.0167
Rosenbrock ↓	25.5	26.1	**21.6**	25*e*6	24.5
Schwefel ↓	**0.77**	6303.4	3396.8	6864.2	306.6
50 dimensions					
Function	**SOMA**	**GWO**	**COA**	**DA**	**WOA**
Griewank ↓	**0**	**0**	**0**	2, 3112	0, 0002
Levy ↓	**0.0**	2.4003	2.8260	40.4872	0.0128
Rosenbrock ↓	50.8	46.6	**40.3**	20*e*7	44.0
Schwefel ↓	**1.86**	11989	6628	13365	220
100 dimensions					
Function	**SOMA**	**GWO**	**COA**	**DA**	**WOA**
Griewank ↓	**0**	**0**	**0**	4, 63	8*e* − 05
Levy ↓	**0.0**	6.3	6.9	104.1	0.0176
Rosenbrock ↓	116	96.8	**89.18**	19*e*8	93.04
Schwefel ↓	**41.04**	25927.1	15997.3	31020.6	253.5

Table 2. Continuous benchmark optimization results on 10, 30, 50, and 100 dimensional problems. The results are represented as the arithmetic means of the best solution's fitness function values of the 100 individual experiments. Smaller results indicate better solutions. (Numeric values rounded differently based on relevance.)

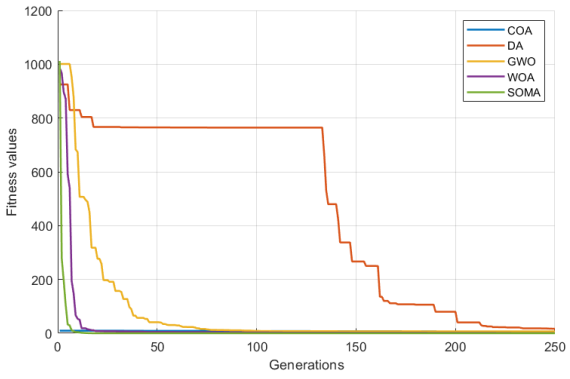

Figure 6. A typical evolution of the best fitness on the Levy's function 100D problem. Lower fitness means a better solution.

Algorithm	# 1st place	# 2nd place
SOMA	86	13
COA	12	21
DA	0	16
GWO	2	31
WOA	0	19

Table 3. Ranking frequencies of the algorithms across all 100 inverse kinematics problems.

From the representative example shown in Table 5 we can see that indeed the algorithm got close to the theoretical joint-coordinate solution.

As two examples for the presentation, we can visit Table 4, where we selected two cases of the inverse kinematics problems from the 100. In the first case (inverse kinematics problem: 53), the selection criterion was to maximize the average performance gap along the 15 runs between the best performer algorithm and the second algorithm, for the second case (inverse kinematics problem: 74), the same criterion was used but with the minimization.

A typical evolution of the fitness is visible in Fig. 7. From this figure, we can see that SOMA has a fast convergence.

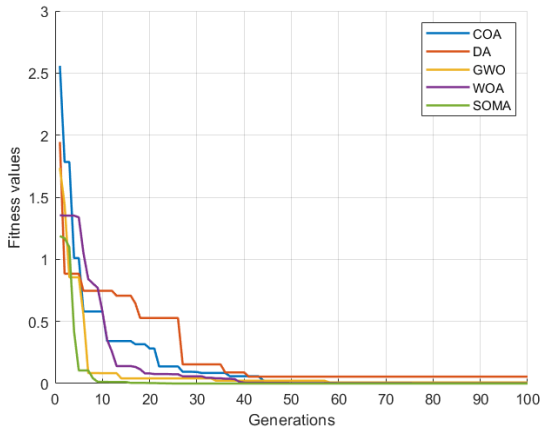

Figure 7. A typical evolution of the best fitness on the inverse kinematics problems. The shown example is the inverse kinematics problem ID53. Lower fitness means better solution.

5 Conclusions

In this paper, we presented our novel algorithm called Strait of Messina evolutionary algorithm. We assessed, and compared the performance of this algoritm on four continuous benchmark (Griewank's, Levy's, Rosenbrock's, and Schwefel's) functions, and on 100 inverse kinematics problem defined on the same robot model.

The result presented in Section 4 shows that our novel method outperforms the competing algorithms in most of cases. The only exceptions are the Rosenbrock function, where the Crayfish Optimization Algorithm wins in the 30 and 50, and 100 dimensional cases. COA turned out to be a good competitor in the case of the Griewank function as well, where a tie can be observed in multiple cases.

In the inverse kinematics task, SOMA ranked first in 86 out of 100 problems with its average performance (Table 3) and exhibited low fitness variance and stable error metrics (Tables 4 and 5). Other algorithms such as COA, DA, GWO, and WOA also demonstrated competitive behavior in specific cases, particularly in achieving second-best placements and providing accurate solutions under certain conditions.

As it is visible from Fig. 6 and Fig. 7 the SOMA algorithm facilitates a fast convergence compared to the other competing algorithms.

Overall, the results highlight SOMA's reliability within this experimental setup, while also underscoring the importance of selecting algorithms based on task-specific characteristics and performance stability.

It is visible that the algorithm underperforms in certain conditions, thus, its robustness can be enhanced. For this, further investigations are advised with other problems and other control algorithms, to reveal the exact causes.

Sourcecode availability

The presented algorithm's sourcecode for reproducibility can be accessed from https://github.com/SDFecht/SOMA.

Acknowledgments

The authors would like to thank the Faculty of Informatics of ELTE Eötvös Loránd University for supporting this work.

References

[1] Jacques Denavit and Richard S. Hartenberg. 1955. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics* 22, 2 (1955), 215–221.

[2] Heming Jia, Honghua Rao, Changsheng Wen, and Seyedali Mirjalili. 2023. Crayfish optimization algorithm. *Artificial Intelligence Review* 56, Suppl 2 (2023), 1919–1979.

[3] Dani Reagan Vivek Joseph and Shantha Selvakumari Ramapackiyam. 2025. ERDA: Evolving Robotic Dragonfly Algorithm for target search in unknown multi-robot environment. *Evolving Systems* 16, 1 (2025), 26.

[4] Manuel Laguna and Rafael Marti. 2005. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization* 33 (2005), 235–255.

[5] Fatiha Loucif, Sihem Kechida, and Abdennour Sebbagh. 2020. Whale optimizer algorithm to tune PID controller for the trajectory tracking control of robot manipulator. *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 42, 1 (2020), 1.

[6] Seyedali Mirjalili. 2015. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural computing and applications* 27 (2015), 1053–1073.

Inverse kinematics problem: 53**, where the performance gap is maximal between the best and second algorithm**												
	Avg. Fitness	Best fitness	Worst fitness	Fitness std. dev.	Best-case error				Worst-case error			
					q1	q2	q3	q4	q1	q2	q3	q4
SOMA	1.46E-05	1.00E-06	5.41E-05	1.71E-05	1.49E-04	-5.76E-04	1.23E-03	-1.08E-03	1.11E-03	-5.68E-01	1.17E+00	-6.03E-01
COA	2.08E-01	2.13E-07	1.31E+00	4.09E-01	-2.81E-04	-5.82E-01	1.19E+00	-6.12E-01	-5.76E+00	-7.76E-01	2.29E+00	4.79E+00
DA	1.46E-01	2.13E-07	9.50E-01	2.94E-01	-2.92E-04	-7.78E-04	1.65E-03	-9.29E-04	-1.26E-03	-8.26E-01	-3.93E+00	4.79E+00
GWO	1.47E-01	2.25E-06	7.02E-01	2.27E-01	-6.45E-04	2.11E-03	-3.71E-03	2.91E-03	-5.76E+00	-1.04E+00	-3.93E+00	-8.99E-01
WOA	1.98E-01	2.00E-06	9.50E-01	2.64E-01	2.89E-04	-5.81E-01	1.19E+00	-6.11E-01	-2.00E-04	-8.26E-01	2.35E+00	4.79E+00
Inverse kinematics problem: 74**, where the performance gap is minimal between the best and second algorithm**												
	Avg. Fitness	Best fitness	Worst fitness	Fitness std. dev.	Best-case error				Worst-case error			
					q1	q2	q3	q4	q1	q2	q3	q4
SOMA	1.95E-06	1.85E-07	4.85E-06	1.37E-06	2.63E-04	8.16E-04	-1.63E-03	8.41E-04	4.21E-04	-5.47E-01	1.12E+00	-5.79E-01
COA	1.13E-06	1.83E-08	1.52E-05	3.90E-06	1.27E-04	7.86E-04	-1.62E-03	8.33E-04	-1.56E-05	5.59E-03	-1.24E-02	7.03E-03
DA	1.34E-01	1.83E-08	6.63E-01	2.74E-01	1.27E-04	7.88E-04	-1.62E-03	8.33E-04	9.13E-04	-1.14E+00	2.32E+00	-1.13E+00
GWO	4.42E-02	1.47E-07	6.62E-01	1.71E-01	1.44E-04	1.21E-03	-2.55E-03	1.31E-03	5.75E-04	-1.15E+00	-3.96E+00	-1.12E+00
WOA	9.05E-02	1.08E-02	2.59E-01	8.56E-02	4.78E-03	-6.70E-01	1.25E+00	-4.96E-01	1.46E-04	4.73E-01	-8.23E-01	3.01E-01

Table 4. Two inverse kinematics problems selected from the 100. The upper part of the table showing the kinematics problem (ID: 53), where the performance gap (based on the average fitness calculation along the 15 runs) between the first and second place is maximal. The lower part shows the problem (ID: 74), where this gap is minimal. The left side shows the statistically important features of the 15 runs (average fitness, best fitness, worst fitness, and standard deviation of the fitness); in the middle we can see the best case parametric errors of the solutions, and on the right the worst case parametric errors of the solutions.

	q1	q2	q3	q4	Fitness ↓
Expected	2.7240	0.4946	−1.3296	−1.4132	0
SOMA	2.7240	0.5407	−1.4251	−1.3642	$6.4e-07$
COA	2.7241	0.5648	−1.4736	−1.3386	$1.04e-06$
DA	2.7238	0.4068	−1.1516	−1.4955	$1.1e-4$
GWO	2.7245	0.5549	−1.4540	−1.3490	$6.83e-07$
WOA	2.7265	0.5756	−1.5056	−1.3128	$1.6e-4$

Table 5. An example inverse kinematics problem with the known solution and the Corresponding Optimization Results.

[7] Seyedali Mirjalili and Andrew Lewis. 2016. The whale optimization algorithm. *Advances in engineering software* 95 (2016), 51–67.

[8] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. 2014. Grey wolf optimizer. *Advances in engineering software* 69 (2014), 46–61.

[9] Zsigmond Péntek, János Botzheim, and András Czmerk. 2021. Bacterial Programming Based Kinematic Chain Estimation of Construction Vehicle. *IEEE Access* 9 (2021), 33569–33582.

[10] Mehran Rahmani, Hossein Komijani, and Mohammad Habibur Rahman. 2020. New sliding mode control of 2-DOF robot manipulator based on extended grey wolf optimizer. *International Journal of Control, Automation and Systems* 18 (2020), 1572–1580.

[11] ROBOTIS. 2024. *OpenMANIPULATOR-X.* https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/ Accessed: 2025-04-11.

[12] Qing Si and Changyong Li. 2023. Indoor robot path planning using an improved whale optimization algorithm. *Sensors* 23, 8 (2023), 3988.

[13] Vikas K Soman and V Natarajan. 2025. Crayfish optimization based pixel selection using block scrambling based encryption for secure cloud computing environment. *Scientific Reports* 15, 1 (2025), 2406.

Received 7 May 2025; revised XX May 2025; accepted XX May 2009