Effective modifications of Self-organizing migrating algorithm

Radka Poláková[1,*], Daniel Valenta[1,†]

[1]Silesian University in Opava, Faculty of Philosophy and Science, Institute of Computer Science, 746 01 Opava, Czech Republic

Abstract

This paper deals with global optimization and focuses on the Self-Organizing Migrating Algorithm (SOMA). Three new versions of SOMA are proposed, where the first two introduce different mechanisms to maintain population diversity and the third combines both approaches. The algorithms were tested on the *CEC2014* benchmark set at two levels of dimension, $Dim = 10$ and $Dim = 30$. The results of the experiments suggest that two of the proposed modifications can improve the performance of the original SOMA-T3 variant in many cases.

Keywords

Optimization algorithms, SOMA, Population diversity

1. Introduction

In real life, many problems require finding the best (optimal) solution, which often means minimizing a mathematical function. This process is called optimization. Instead of searching for a maximum, we can focus on finding a minimum, since maximizing a function f is equivalent to minimizing $-f$. The goal is to find a point in the search space with dimension *Dim* (a set of all possible solutions that an optimization algorithm should explore; defined by the type and range of variables, sometimes including some constraints), where the function has its lowest value. Sometimes, it is hard or even impossible to solve the problem exactly, especially when the function is complex or has many local extremes. In such cases, stochastic (random-based) algorithms can be used to find a good enough solution. These methods are not exact, but they can provide a useful estimate at a lower cost. However, according to the No Free Lunch Theorem, no single optimization algorithm works best for all problems. [12] That is why researchers continue to develop new optimization methods.

Building on our previous work two years ago, where we combined the Grey Wolf Optimizer (GWO) and jSO algorithms and observed promising improvements over their standalone variants [1], we now extend this line of research by exploring a different population-based metaheuristic. In this study, we focus on the Self-Organizing Migrating Algorithm (SOMA), which offers a distinct search mechanism and has shown potential to solve complex optimization problems. [2]

Among the various SOMA variants (see [13] for example), its SOMA-T3A variant has shown particularly strong and consistent performance in demanding benchmark settings. In particular, it achieved a shared third place (alongside HyDE-DF) out of 36 algorithms in the IEEE *Œ*C2019 100-Digit Challenge, confirming its robustness in solving high-dimensional optimization problems. [10] Unlike T3A, the modified T3B version restructures the evaluation step to occur after all selected individuals have moved, allowing efficient parallel execution and making it better suited for computationally expensive real-world problems. [7]

In the following chapters, we focus on SOMA-T3 and its three variants proposed by us, which introduce different mechanisms to maintain population diversity. These variants will be tested on 10- and 30-dimensional problems using the *Œ*C2014 benchmark set, and the results will be analyzed. Based on these tests, we draw conclusions regarding the performance of the modified versions.

ITAT'25: Information Technologies – Applications and Theory, September 26–30, 2025, Telgárt, Slovakia

*Corresponding author.

†These authors contributed equally.

radka.polakova@fpf.slu.cz (R. Poláková); daniel.valenta@fpf.slu.cz (D. Valenta)
0000-0002-0782-5668 (R. Poláková); 0009-0005-0781-7755 (D. Valenta)

2. SOMA Algorithm

SOMA (Self-organizing Migrating Algorithm) was introduced in 2000 by Zelinka and Lampinen [3]. The algorithm can be viewed as a model inspired by the cooperative hunting behavior of animal packs.

It works with a population of points from the search space. At the beginning of the run, the algorithm generates a random initial population and evaluates the objective (optimized) function in each of them. Then it works in cycles, so-called migrations. After each migration, the algorithm checks whether the stopping condition is met. When done, it returns the best solution found.

In each migration, each point travels towards the best point of the population which is named the leader. The point can also skip the leader. The length of the path depends on the parameters of the algorithm. Each point visits several places (positions) on its way to the leader. We can say that the point jumps on the line (its path). The best position of such visited places is the new possible position of this jumping point. The member of the population is moved to this new possible position if it is better than the original position of the member (in the sense of optimized function value). A migration includes such jumping and eventual moving for each member of population except the best point of population – the leader.

The movement of a member to the leader need not be straight. It can move only in some dimensions. There is a parameter which calls *PRT*, this is a number between 0 and 1, more precisely the number of the interval $[0, 1]$. *PRT* determines the probability that the movement of a point to the leader is done in a dimension. Before computing new possible positions for a population member, a *PRTVector* is computed. It consists of *Dim* elements, each element can be from the set $\{0, 1\}$. The relevant member is moved only in dimension(s) where *PRTVector* has the value 1 in the corresponding position(s). Firstly, a vector of random numbers (from a uniform distribution) is generated. Then, where the vector has lower value than *PRT*, the *PRTVector* has number 1 at the corresponding place, at other places there are zeros.

The population member x jumps onto the line determined by its position, the leader position, and *PRTVector* from its position to the leader position in steps. The $STEP$ is the algorithm parameter, a number in the interval $(0, 1)$, and the length of each jump of the point is $STEP \times (Leader - x)$. As already written, the movement is done only in some dimensions (according to *PRTVector*). The recommended value of $STEP$ is 0.11 or 0.33.

The next parameter of the algorithm is called *PathLength*. It determines the maximal length of the entire path from a population member to the leader. The maximum length is $PathLength \times (Leader - x)$. The recommended value of *PathLength* ranges from 1.1 to 3.

Let us now look at how the new position is calculated in more detail. First, new positions are calculated for all members of the population during a migration. Then each point is moved in a single step to a better position (only in the dimensions selected by *PRTVector*), so that all points move together.

The last parameter is the size of the population NP, which is recommended to be set to 10, when the dimension of the solved problem is less than 100, otherwise it is recommended to have a population of $20 - 50$ members.

The duration of the algorithm run is determined by the number of migrations or by the number of evaluations of the optimized function. [3]

2.1. Migration Strategies

The authors of the algorithm proposed several different migration strategies. The one described in the previous paragraphs is called *AllToOne*. The other specification of the migration strategies is as follows: [3]

- *AllToOneRand*: The leader is selected at random rather than based on the optimized function value here, introducing an element of stochasticity.
- *AllToOneAdaptive*: In this migration strategy, the point is moved immediately after finding its better position (not at the end of the whole migration as in *AllToOne* migration). So for other members, the new position of the point can already be the leader if it is better than the previous leader.

- *AllToAll*: Each point migrates to all other members of the population. The point is then moved to the best position of all its migration paths if the position is better than its original position. This strategy is computationally demanding.
- *AllToAllAdaptive*: Similar strategy to *AllToAll*, but each individual immediately moves to a better position found during its migration path and then migrates to the next point of the population from its new position, instead of waiting (with moving) to find the best position of all its migration paths.

3. SOMA-T3 Algorithm

The SOMA-T3 algorithm is a very efficient modification of Self-organizing Migrating Algorithm. The algorithm is described in detail in Algorithm (1). The changes from the original SOMA algorithm follow (the authors' implementation in Python [10] was very helpful in identifying them precisely).

It uses dynamic setting of parameters PRT and $STEP$. Both are set separately for each individual of the population, the PRT according to equation (1) and $STEP$ according to equation (2). One can see that PRT is small at the beginning of an algorithm run and it is almost 0.95 at the end of the algorithm run. In contrast, $STEP$ decreases as the algorithm runs.

$$PRT = 0.05 + 0.90 \times (FEs/Max_{FEs}), \tag{1}$$

$$STEP = 0.15 - 0.08 \times (FEs/Max_{FEs}). \tag{2}$$

Such a dynamic setting of the parameters results in the modification of only a few dimensions at the beginning of the run, and the jumps along the aforementioned line are relatively large due to a high $STEP$, which supports a broad exploration. Later in the run, PRT increases and $STEP$ decreases, so more dimensions are updated, but jumps are smaller. This helps the algorithm smoothly shift from exploring the search space to fine-tuning the best found solutions.

The algorithm does not use the parameter *PathLength*. It uses the parameter $STEP$ with the original meaning together with a parameter N_{jump}. The number N_{jump} indicates how many positions on the path from the original position of point towards the leader position (in some coordinates; with step $STEP \times (Leader - x)$, where x is the position of the currently migrated point of population) the algorithm explores.

The SOMA-T3 algorithm also does not use only one population of points to solve an optimization problem. It uses a relatively large population, but it migrates only a part of this population, in each cycle the part is different. It has a population NP of np points. For each migration, it randomly chooses the population M of m members from NP. And finally, only the best n members of M make the final population N and migrate. Also, leader is not one of them. In this algorithm, k members are randomly selected from NP to form the population K of possible leaders. It is done separately for each migrant. The best point of K is the *Leader*.

4. Three Modifications of SOMA-T3

We studied the SOMA-T3 algorithm and the goal was to improve its efficiency. Our first modification lies in the following. We wanted to empower the exploration of the algorithm, but also to have a very similar algorithm to the SOMA-T3 at the end of the search. We decided to add a tool that allows us to extend the length of the whole path the point surmounts from its original position when it migrates at the beginning of the algorithm and to cut this path at the end of the run. Originally we proposed the number equal to $\pi \times (1 - FEs/Max_{FEs})$, it decreased from the beginning to the end of the algorithm run. But such a number without any randomness may, of course, make a behavior which is not suitable for all solved problems. So, we multiply this by the random number *rand*() between 0 and 1 and get the final g from the equality (3).

$$g = \pi \times rand() \times \left(1 - \frac{FEs}{Max_{FEs}}\right), \tag{3}$$

Algorithm 1 Pseudocode of the SOMA-T3 Algorithm

```
1: Initialize problem dimension $Dim$, population $NP$ of $np$ size (here $np = 100$), maximum number
   of function evaluations $Max_{FEs}$, domain (search space $x_j \in [x_{min}, x_{max}]$ for each dimension $j \in$
   $\{1, \dots, Dim\}$) and parameters $m, n, k$, where:
     1. $m$ represents the number of individuals selected randomly from population $NP$; $m$ individuals
        make population $M$; (here $m = 10$),
     2. $n$ indicates the number of individuals selected from the $M$ population according to optimized
        function value to make population $N$ for migration; (here $n = 5$),
     3. $k$ is the number of individuals randomly chosen from the population $NP$ to serve as potential
        leaders during migration; (here $k = 15$).
2: Initialize $N_{jump}$ which is the number of positions of a migrant on its path towards leader, here
   $N_{jump} = 45$.
3: Evaluate optimized function in all members of $NP$ population; $FEs = np$ ; the best member of $NP$ is
   solution of the problem
4: while $FEs <= Max_{FEs}$ do
5:     $newPop = \emptyset$
6:     $M \leftarrow$ random selection (with uniform distribution)
         of $m$ migrant candidates from $NP$
7:     $N \leftarrow n$ best migrants from $M$
         according to optimized function value
8:     for each $Migrant$ from $N$ do
9:         $K \leftarrow$ random selection (with uniform
             distribution) of $k$ leader candidates from $NP$
10:        $Leader \leftarrow$ best from $K$
             according to optimized function value
11:        if $Leader == Migrant$ then
12:            $Leader \leftarrow$ second best from $K$
                 according to optimized function value
13:        Compute $PRT$ according to (1)
14:        Compute $STEP$ according to (2)
15:        Compute $PRTVector$
16:        Compute all $N_{jump}$ positions of $Migrant$
             on the path towards $Leader$ using
             $PRTVector$
17:        Check each coordinate $x_j$ of each new position
             if it is in search space, if not, set the $x_j$
             to the new random position in search space
18:        Store all $N_{jump}$ new positions of
             the $Migrant$ into $newPop$
19:    Evaluate optimized function for all members
         of $newPop$
20:    $FEs = FEs + n \times N_{jump}$
21:    for each $Migrant$ in $N$ do
22:        Choose the best position for $Migrant$
             from its new positions in $newPop$
23:        if the best position of $Migrant$ is better than
             the $Migrant$ then
24:            The migration is accepted and
                 the new individual is included in the $NP$
                 instead of the $Migrant$.
25:            if just included point is better than
                 the best one of $NP$ then
26:                Correct the solution of the solved
                     problem
```

Then, when the new N_{jump} positions of a migrant are computed, the vector, which was originally added to the migrant in SOMA-T3 to get the new positions, see equality (4), is multiplied by g, see equality (5).

$$Mig_j = Mig + (Lea - Mig) \times j \times STEP \times PRTVec, \tag{4}$$

$$Mig_j = Mig + (Lea - Mig) \times j \times STEP \times PRTVec \times \mathbf{g}. \tag{5}$$

In both equations, (4) and (5), j is each member of set $\{1, 2, \dots, N_{jump}\}$.

The algorithm SOMA-T3 with just described tool implemented in, we call SOMA-T3-RL (RL means something like random length).

Our second modification of the SOMA-T3 algorithm deals with another possibility of maintaining population diversity. In the original algorithm, the authors guarantee a sufficient degree of diversity of the population by working with only a part of the population in each migration. Our approach lies in a smaller population and refreshment of some points when the search process stagnates.

We proposed changing randomly selected points of np points (but not several best) of the population to a random position in the search space when the solution of the problem does not evolve in several last consecutive migrations. In this case, when the diversity of population is provided in this way, it is possible to work with a smaller population. And it is also suitable because of saving optimized function evaluations with regard to their depletion for members which position is changed, when it is needed.

So, we reduced the population size to less than about two-thirds and introduced three new parameters, *nochanges*, *refresh*, *nobest*. We are storing the last two solutions (of solved problem; the solution - the best point of population in a time) in each time and detecting if the last solution is or is not equal to the penultimate one. If they are equal for more than *notchange* times, we change the position of *refesh* randomly selected points of the population, but not the *nobest* best points. This modification of the SOMA-T3 algorithm we call SOMA-T3-DIV in the following.

The third modification, called SOMA-T3-RL-DIV, merges the approaches from the first two variants into a single algorithm. It incorporates the random path length mechanism of SOMA-T3-RL, which adjusts the migration distance during the run, and the population refresh strategy of SOMA-T3-DIV, which replaces selected individuals when stagnation is detected. This combination allows the algorithm to use both strategies at the same time and potentially benefit from their joint effect.

5. Experiments

We tested four algorithms: the original SOMA-T3 algorithm and three of its modified versions described above, SOMA-T3-RL and SOMA-T3-DIV, and also the combination of both modified variants SOMA-T3-RL-DIV. In this initial stage of our experiments, we adopted the same parameter settings as used in SOMA-T3 to ensure a consistent baseline for comparison. For SOMA-T3-DIV and SOMA-T3-RL-DIV, we reduced the population size to less than about two thirds. This reduction is feasible because population diversity is maintained by randomly changing the positions of selected points while keeping the best points unchanged. As a result, with the same number of function evaluations, the search can be in some sense longer and can obtain better result.

The evaluation is carried out on the *CEC2014* benchmark set [9], which consists of 30 functions designed for competition for single-objective optimization algorithms with real parameters. These functions include a mix of unimodal ($F1 - F3$), simple multimodal ($F4 - F16$), hybrid ($F17 - F22$), and composition ($F23 - F30$) types and offer varying levels of difficulty and complexity.

Each algorithm was run 51 times for each configuration, with $Max_{FEs} = Dim \times 10^4$. In total, this resulted in $4 \times 2 \times 30 \times 51$ runs (count of algorithms × count of dimensions × count of functions × count of runs), that is, $12,240$ runs overall. The count of runs for a combination (algorithm, dimension, function) was chosen according to [9], and the dimensions 10 and 30 are also prescribed in it – in the future, we plan to test dimensions 50 and 100 as well.

All functions share the same search space, $[-100, 100]$, for every dimension. The global minimum values range from 100 to 3000: $F1$ has a minimum of 100, $F2$ of 200, $F3$ of 300, and so on, up to $F30$ with a minimum of 3000. This makes to compute the difference between an algorithm's solution and the true minimum straightforward. Therefore, in the results below, we report only this difference.

All four algorithms tested have several parameters. Some of these parameters are set the same across all algorithms: m, n, k, and N_{jump}, with values $m = 10$, $n = 5$, $k = 15$, and $N_{jump} = 45$, following the original SOMA-T3 settings. Other parameters, such as *nochanges*, *refesh*, and *norebest*, are specific only to SOMA-T3-DIV and SOMA-T3-RL-DIV. The parameter np differs for some of the algorithms tested. The specific settings of the last mentioned parameters are listed in Table 1. In this table, SOMA-T3-RL, SOMA-T3-DIV, and SOMA-T3-RL-DIV are marked as RL, DIV, and RL-DIV, respectively.

Table 1
Parameters of tested algorithms with different setting

Alg.	np	*nochanges*	*refesh*	*norebest*
SOMA-T3	100	-	-	-
RL	100	-	-	-
DIV	30	10	5	2
RL-DIV	30	10	5	2

The testing was done on a MacBook Pro 14" with an Apple M3 Pro chip (12-core CPU, 18-core GPU) and 36GB of RAM. As a basis, we used the *SOMA_T3A-python* implementation (version 2), available on GitHub [10], modified to support the *CEC2014* benchmark functions – together with *Python* version *3.13.2*, *NumPy* version *2.3.1*, and the *Opfunu* [11] library (with *CEC2014* functions) version *1.0.4*.

As a basis, we used a Python implementation of the algorithm sourced from GitHub [10], which we extended with the modifications described above.

In summary, we tested the three modified versions of the algorithm – by running each of them 51 times for the 30 *CEC2014* benchmark functions, using the implementation provided by the *opfunu* library (version=*"1.0.4"*). [11] The same procedure was applied to the original unmodified algorithm, SOMA-T3. Moreover, all of these experiments were performed for both dimensions $Dim = 10$ and $Dim = 30$, always using the parameters described above.

6. Results

This section provides a detailed presentation and interpretation of the results obtained in our experiments performed on the 30 *CEC2014* benchmark functions in dimensions 10 and 30. All results obtained were corrected as prescribed in [9]. If the result was less than 10^{-8}, it was substituted by 0.

Some statistics calculated from the results of our experiments are printed in Tables 2, 3, 4, 5. In these tables, SOMA-T3-RL, SOMA-T3-DIV, and SOMA-T3-RL-DIV are also marked as RL, DIV, and RL-DIV, respectively. Tables 2, 3 summarize the comparison of each proposed algorithm (SOMA-T3-RL, SOMA-T3-DIV, SOMA-T3-RL-DIV) with the original algorithm SOMA-T3 in dimension $Dim = 10$. Tables 4, 5 show the summary of the same comparison in dimension $Dim = 30$. For each combination of dimension, function, and algorithm, we computed the minimum and median of the 51 results obtained. All minimums are shown in Tables 2 and 4. All medians are shown in Tables 3 and 5.

In these four tables, better statistics are underlined. We always compare a statistic (minimum, median) of a SOMA-T3 modification (SOMA-T3-RL, SOMA-T3-DIV, SOMA-T3-RL-DIV) with the relevant statistic of the original SOMA-T3 algorithm. That is the reason why the underlined numbers appear only in the last three columns. The last line in each of these four tables summarizes the count of wins and count of losses for each of the algorithms (SOMA-T3-RL, SOMA-T3-DIV, SOMA-T3-RL-DIV) when comparing the statistic in the dimension.

When we look at Table 2 (the numbers in the table are minimums in dimension $Dim = 10$), we can see that the SOMA-T3-DIV and SOMA-T3-RL-DIV algorithms have more wins (20, 21) than losses (7, 6).

On the other hand, the algorithm SOMA-T3-RL has a diverse situation in comparison with SOMA-T3 in this dimension. The count of wins is 7 and the count of losses is 22. When we discuss the medians in dimension $Dim = 10$ (see Table 3), the results of the comparisons are very similar. SOMA-T3-DIV and SOMA-T3-RL-DIV have more wins than losses, while the opposite is true for SOMA-T3-RL. When comparing the minimums (see Table 4) and medians (see Table 5) of the results in dimension $Dim = 30$, the counts of wins and losses for the algorithms tested differ only slightly. They are very similar to the results in the tables for dimension $Dim = 10$.

Table 2
Minimums of results for $Dim = 10$ for all tested algorithms

F	SOMA-T3	RL	DIV	RL-DIV
1	1.771E-01	5.919E-01	4.161E-04	8.684E-03
2	0.000E+00	7.219E-06	0.000E+00	0.000E+00
3	0.000E+00	1.196E-08	0.000E+00	0.000E+00
4	2.203E-05	4.976E-03	0.000E+00	2.225E-07
5	1.498E-03	5.116E-04	0.000E+00	5.536E-07
6	1.436E-04	2.042E-03	9.846E-05	1.611E-04
7	0.000E+00	7.399E-03	7.396E-03	1.478E-02
8	0.000E+00	0.000E+00	0.000E+00	0.000E+00
9	1.990E+00	1.990E+00	1.990E+00	1.990E+00
10	2.060E-06	7.327E-03	0.000E+00	0.000E+00
11	6.830E+00	6.455E+00	3.747E-01	3.123E-01
12	6.986E-02	9.539E-02	6.773E-03	1.161E-02
13	2.453E-02	4.390E-02	1.012E-02	9.201E-03
14	3.972E-02	2.581E-02	3.304E-02	1.558E-02
15	3.379E-01	4.810E-01	1.733E-01	3.075E-01
16	7.031E-01	9.036E-01	7.466E-01	3.447E-01
17	1.037E+01	1.401E+01	1.422E+00	2.086E+00
18	1.901E-01	1.322E-01	7.834E-02	1.030E-01
19	1.141E+00	1.469E+00	1.133E+00	1.002E+00
20	9.232E-01	8.936E-01	1.806E-01	6.714E-01
21	4.888E-01	7.019E-01	5.236E-02	3.558E-01
22	4.872E-01	8.545E-01	1.187E-01	1.440E-01
23	1.276E-05	2.000E+02	2.000E+02	0.000E+00
24	1.059E+02	1.000E+02	1.000E+02	1.000E+02
25	1.331E+02	1.359E+02	1.360E+02	1.333E+02
26	1.000E+02	1.000E+02	1.000E+02	1.000E+02
27	1.838E+00	1.906E+00	1.919E+00	1.458E+00
28	3.568E+02	3.568E+02	3.568E+02	3.568E+02
29	4.500E+06	4.500E+06	4.500E+06	4.500E+06
30	1.896E+03	1.834E+03	1.786E+03	2.059E+03
#w/#l		7/22	20/7	21/6

Without additional statistics, we can conclude that the proposed modifications SOMA-T3-DIV (which use a tool for preserving population diversity that differs from the one used by the original algorithm) and SOMA-T3-RL-DIV (an algorithm in which we implemented both proposed tools) of the SOMA-T3 algorithm are sufficiently effective. SOMA-T3-RL does not appear to be as effective as expected. Here we can observe the difference in the improvement of effectiveness when randomness is restricted to a single direction, and when it is not restricted and diversity is supported by points across the whole search space.

In order to know the true difference in the effectivity of the original algorithm and the proposed algorithms, we computed 180 Wilcoxon rank-sum statistical tests. All statistical tests were performed at the significance level set at 0.05. The results of the statistical tests performed are shown in Table 6. In this table, RL indicates SOMA-T3-RL, DIV is SOMA-T3-DIV, and the label RL-DIV means SOMA-T3-RL-DIV.

Table 3
Medians of results for *Dim* = 10 for all tested algorithms

F	SOMA-T3	RL	DIV	RL-DIV
1	9.700E+01	1.968E+02	3.654E+00	4.353E+01
2	2.200E-07	9.630E-05	0.000E+00	0.000E+00
3	0.000E+00	8.722E-07	0.000E+00	0.000E+00
4	1.175E-02	2.288E-01	1.260E-07	3.380E-03
5	2.012E+01	2.013E+01	2.000E+01	2.002E+01
6	7.553E-03	1.820E-02	4.000E-02	3.766E-02
7	3.938E-02	3.202E-02	4.678E-02	5.419E-02
8	0.000E+00	0.000E+00	0.000E+00	0.000E+00
9	3.980E+00	3.981E+00	3.980E+00	3.980E+00
10	2.501E-01	2.499E-01	2.498E-01	1.874E-01
11	1.453E+02	1.418E+02	1.336E+02	4.169E+01
12	2.930E-01	2.811E-01	7.726E-02	9.263E-02
13	8.169E-02	8.451E-02	7.422E-02	5.381E-02
14	1.506E-01	1.336E-01	8.669E-02	7.171E-02
15	7.434E-01	7.658E-01	7.113E-01	7.008E-01
16	1.843E+00	1.568E+00	1.648E+00	1.617E+00
17	4.216E+01	8.920E+01	4.572E+01	6.517E+01
18	1.573E+00	1.476E+00	1.445E+00	1.349E+00
19	2.118E+00	2.173E+00	1.838E+00	1.730E+00
20	3.795E+00	5.264E+00	1.718E+00	2.650E+00
21	2.276E+00	3.954E+00	8.603E-01	9.803E-01
22	2.150E+01	2.156E+01	2.024E+01	2.016E+01
23	2.422E+02	2.422E+02	2.422E+02	2.368E+02
24	1.104E+02	1.089E+02	1.101E+02	1.093E+02
25	1.430E+02	1.428E+02	1.421E+02	1.415E+02
26	1.001E+02	1.001E+02	1.001E+02	1.001E+02
27	3.823E+00	3.301E+00	5.789E+00	3.009E+02
28	3.601E+02	3.601E+02	3.694E+02	3.717E+02
29	4.500E+06	4.500E+06	4.500E+06	4.500E+06
30	3.728E+03	4.678E+03	3.260E+03	4.480E+03
#w/#l		10/18	22/5	21/7

When + appears in the table, it means that the respective modified algorithm is statistically better than the original, − means that the original algorithm works statistically better than the modified one, and ≈ means that both algorithms compared work with the statistically same effectivity.

When we compare the original algorithm SOMA-T3 and SOMA-T3-RL in dimension *Dim* = 10, we found that only for the 2 functions tested the results of SOMA-T3-RL are statistically better than the results of SOMA-T3, and simultaneously the results of SOMA-T3-RL are statistically worse than the results of SOMA-T3 for 10 functions of the benchmark set. So, the conclusion should be that SOMA-T3 is better than SOMA-T3-RL in *Dim* = 10. In dimension *Dim* = 30, the situation is very similar. SOMA-T3 is statistically better than SOMA-T3-RL 13 times and SOMA-T3-RL is statistically better than SOMA-T3 6 times. The following conclusion should be drawn. The SOMA-T3 algorithm is more effective than the SOMA-T3-RL algorithm also in dimension *Dim* = 30.

When we look at the columns for the comparison of SOMA-T3 and SOMA-T3-DIV, we can see that the numbers at the bottom of the columns are different. In dimension *Dim* = 10, the SOMA-T3-DIV algorithm is better than the SOMA-T3 algorithm for 12 functions. On the other hand, the SOMA-T3 algorithm is more effective than the SOMA-T3-DIV algorithm for only 5 functions from the benchmark set. A similar situation appears for dimension *Dim* = 30. SOMA-T3-DIV is better than SOMA-T3 for 16 functions and SOMA-T3 wins only on the 6 functions tested. Looking at the last two columns of the table, we can conclude that the combined use of both proposed tools does not yield significantly

Table 4
Minimums of results for *Dim* = 30 for all tested algorithms

F	SOMA-T3	RL	DIV	RL-DIV
1	5.513E+04	2.890E+05	4.782E+03	5.241E+04
2	0.000E+00	7.834E-08	0.000E+00	0.000E+00
3	0.000E+00	2.329E-07	0.000E+00	0.000E+00
4	1.494E-02	3.728E-02	5.070E-04	2.717E-02
5	2.042E+01	2.048E+01	2.000E+01	2.006E+01
6	4.811E+00	6.309E+00	4.512E+00	4.728E+00
7	0.000E+00	0.000E+00	0.000E+00	0.000E+00
8	0.000E+00	0.000E+00	0.000E+00	0.000E+00
9	1.691E+01	1.293E+01	1.791E+01	2.089E+01
10	1.367E+00	1.391E+00	2.498E-01	1.666E-01
11	1.753E+03	2.439E+03	8.942E+02	1.521E+03
12	5.261E-01	5.175E-01	1.321E-01	1.068E-01
13	1.094E-01	8.745E-02	9.848E-02	7.590E-02
14	2.148E-01	1.304E-01	1.668E-01	1.192E-01
15	1.762E+00	2.751E+00	1.726E+00	2.412E+00
16	9.661E+00	9.880E+00	9.282E+00	8.131E+00
17	1.217E+04	1.533E+04	6.493E+03	9.419E+03
18	2.228E+04	2.724E+04	2.720E+04	2.683E+04
19	1.205E+01	1.978E+01	1.021E+01	1.479E+01
20	6.490E+03	1.336E+04	2.575E+03	6.404E+03
21	4.140E+03	7.075E+03	1.642E+03	2.522E+03
22	4.256E+01	5.676E+01	2.175E+01	2.757E+01
23	2.228E+02	2.228E+02	2.228E+02	2.228E+02
24	2.187E+02	2.153E+02	2.037E+02	2.131E+02
25	2.050E+02	2.054E+02	2.040E+02	2.041E+02
26	1.001E+02	1.001E+02	1.001E+02	1.001E+02
27	4.004E+02	4.010E+02	4.004E+02	4.006E+02
28	7.743E+02	8.190E+02	8.041E+02	7.972E+02
29	4.083E+03	4.278E+03	3.177E+03	3.366E+03
30	1.927E+04	1.696E+04	9.061E+03	1.181E+04
#w/#l		6/21	21/4	18/7

different results compared to the implementation of only the DIV mechanism. The SOMA-T3-RL-DIV wins 17 times in dimension *Dim* = 10 and 12 times in dimension *Dim* = 30. On the other hand, the last comparison looks as follows. SOMA-T3 wins 4 times and 6 times in dimension *Dim* = 10 and *Dim* = 30, respectively.

When we summarize the comparison of SOMA-T3 and SOMA-T3-RL, the part of functions where SOMA-T3-RL is statistically better than the original algorithm is equal to or less than 20 % in each tested dimension. However, when we think about the two other comparisons – SOMA-T3-DIV and also SOMA-T3-RL-DIV, we find that SOMA-T3-DIV and SOMA-T3-RL-DIV are more effective than the original SOMA-T3 algorithm for more than half of the functions in a given tested dimension: SOMA-T3-DIV in *Dim* = 30 and SOMA-T3-RL-DIV in *Dim* = 10. In the other dimension, they are both better than SOMA-T3 in more than 33 % of the tested functions. Overall in both dimensions together, SOMA-T3-DIV and also SOMA-T3-RL-DIV are worse than the original SOMA-T3 algorithm only in about 16 % of the benchmark optimization problems. The results of these two algorithms are of similar or better effectivity than the original SOMA-T3 in the rest of the benchmark problems, that is, in more than 80 % of the tested problems.

Table 5
Medians of results for $Dim = 30$ for all tested algorithms

F	SOMA-T3	RL	DIV	RL-DIV
1	2.746E+05	7.564E+05	9.246E+04	2.522E+05
2	0.000E+00	8.938E-06	0.000E+00	0.000E+00
3	1.974E-07	1.165E-04	0.000E+00	0.000E+00
4	6.924E+01	7.369E+01	3.302E+00	7.042E+01
5	2.063E+01	2.059E+01	2.002E+01	2.026E+01
6	1.092E+01	1.155E+01	1.037E+01	1.075E+01
7	0.000E+00	0.000E+00	0.000E+00	0.000E+00
8	1.990E+00	9.950E-01	9.950E-01	9.950E-01
9	2.985E+01	3.086E+01	4.676E+01	4.676E+01
10	8.042E+00	5.335E+00	3.592E+00	3.561E+00
11	2.516E+03	4.017E+03	2.046E+03	2.214E+03
12	8.740E-01	8.775E-01	2.245E-01	2.435E-01
13	1.705E-01	1.561E-01	2.068E-01	1.775E-01
14	3.106E-01	2.780E-01	2.431E-01	1.864E-01
15	3.333E+00	5.697E+00	3.702E+00	3.745E+00
16	1.097E+01	1.142E+01	1.047E+01	1.062E+01
17	2.380E+04	4.465E+04	1.772E+04	3.191E+04
18	3.906E+04	3.643E+04	5.806E+04	5.491E+04
19	2.168E+01	2.479E+01	3.069E+01	3.180E+01
20	1.576E+04	2.920E+04	1.170E+04	1.993E+04
21	1.296E+04	1.607E+04	6.306E+03	1.091E+04
22	2.651E+02	2.949E+02	1.966E+02	2.790E+02
23	2.228E+02	2.228E+02	2.228E+02	2.228E+02
24	2.240E+02	2.240E+02	2.245E+02	2.220E+02
25	2.086E+02	2.092E+02	2.051E+02	2.054E+02
26	1.002E+02	1.002E+02	1.002E+02	1.002E+02
27	4.014E+02	4.026E+02	4.012E+02	4.015E+02
28	8.724E+02	8.673E+02	8.792E+02	8.963E+02
29	6.741E+03	6.903E+03	6.317E+03	6.139E+03
30	4.549E+04	1.097E+05	1.622E+04	2.723E+04
#w/#l		7/21	19/8	15/12

7. Conclusion

In this article, we focus on the SOMA optimization algorithm, in particular the SOMA-T3 variant, which we modified to improve optimization performance. Three algorithms based on SOMA-T3 were proposed: SOMA-T3-RL, SOMA-T3-DIV, and a combination of both, SOMA-T3-RL-DIV.

The results show that SOMA-T3-RL (with random length of the migration vector) does not bring significant improvements and in some cases leads to a slight decrease in performance compared to the original algorithm. However, its performance remains largely comparable to that of the original SOMA-T3, suggesting that this modification does not introduce instability or consistent deterioration.

In contrast, SOMA-T3-DIV (with a mechanism for maintaining population diversity) demonstrates stronger and more robust improvements in both dimensions tested. The combined variant, SOMA-T3-RL-DIV, also achieves competitive results, reaching a similar or even better balance of improvements compared to SOMA-T3-DIV.

In general, the proposed modifications confirm that the most effective strategy is to enhance population diversity (DIV). The combined variant SOMA-T3-RL-DIV provides results similar to SOMA-T3-DIV and in some cases is even slightly better, showing that the random length mechanism can bring a small additional benefit when used in combination with population diversity maintenance.

However, a space for further improvement remains there, suggesting that additional modifications or

Table 6
Results of Wilcoxon two-sample tests for all three proposed modifications of SOMA-T3 for both tested dimensions

	RL		DIV		RL-DIV	
F / *Dim*	10	30	10	30	10	30
1	−	−	+	+	≈	≈
2	−	−	+	+	+	+
3	−	−	+	+	+	+
4	−	−	+	+	≈	≈
5	≈	+	+	+	+	+
6	−	≈	−	≈	−	≈
7	≈	≈	−	≈	−	≈
8	≈	+	≈	≈	≈	+
9	≈	≈	≈	−	≈	−
10	≈	+	−	+	+	+
11	≈	−	≈	+	+	+
12	≈	≈	+	+	+	+
13	≈	+	≈	−	+	≈
14	+	+	+	+	+	+
15	≈	−	≈	−	≈	−
16	≈	−	≈	+	+	+
17	−	−	≈	+	≈	−
18	≈	+	≈	−	+	−
19	≈	−	≈	−	+	−
20	−	−	+	+	+	≈
21	≈	−	+	+	+	≈
22	≈	≈	+	+	+	≈
23	−	≈	≈	≈	≈	≈
24	+	≈	≈	≈	+	+
25	≈	≈	≈	+	≈	+
26	≈	≈	+	−	+	≈
27	≈	−	−	≈	−	≈
28	−	≈	−	≈	−	−
29	−	≈	+	≈	+	≈
30	≈	−	≈	+	≈	+
#+	2	6	12	16	17	12
#−	10	13	5	6	4	6
#≈	18	11	13	8	9	12

hybrid strategies could be explored in future work.

Future research will also include a comparison of the proposed methods with more recent SOMA variants introduced in recent studies [14, 15, 16].

Acknowledgments

This work was supported by the Silesian University in Opava under the Student Funding Plan, project SGS/9/2024.

Declaration on Generative AI

Either:
The authors have not employed any Generative AI tools.

References

[1] R. Poláková and D. Valenta, jSO and GWO Algorithms Optimize Together, in *Conference Information Technologies - Applications and Theory*, Slovakia, 2022, pp. 139-146.

[2] I. Zelinka, SOMA—self-organizing migrating algorithm, in *New Optimization Techniques in Engineering*, Springer, Berlin, Heidelberg, 2004, pp. 167–217.

[3] I. Zelinka, J. Lampinen, and L. Nolle, SOMA–self-organizing migrating algorithm, in *Mendel 6th International Conference on Soft Computing*, Brno, Czech Republic, pp 167–217.

[4] I. Zelinka, J. Lampinen, and L. Nolle, SOMA: self-organizing migration algorithm, in *Proc. of the 13th International Conference on Process Control'01*, 2001, pp. 100–105. ISBN 8022715425.

[5] L. Škanderová, T. Fabian, and I. Zelinka, Self-adapting self-organizing migration algorithm, *SWEVO*, vol. 51, 2019. ISSN 2210-6510. doi:10.1016/j.swevo.2019.100593.

[6] L. Škanderová, Self-organizing migrating algorithm: review, improvements and comparison, *Artificial Intelligence Review*, vol. 56, pp. 101–172, 2023. doi:10.1007/s10462-022-10167-8.

[7] Q. B. Diep, Self-Organizing Migrating Algorithm Team To Team Adaptive – SOMA T3A, in *Proc. 2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand, 2019, pp. 1182–1187. doi:10.1109/CEC.2019.8790202.

[8] E. Volná, *Neuronové sítě a genetické algoritmy.* Ostrava: Ostravská univerzita v Ostravě, 1998. ISBN 80-7042-762-0

[9] J. J. Liang, B. Y. Qu, and P. N. Suganthan, Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, Zhengzhou University, China, and Nanyang Technological University, Singapore, 2013.

[10] diepquocbao, *SOMA-T3A-Python*, GitHub repository, 2022. [Online]. Available: https://github.com/diepquocbao/SOMA-T3A-Python

[11] N. V. Thieu, Opfunu: An Open-source Python Library for Optimization Benchmark Functions, *Journal of Open Research Software*, vol. 12, 2024. doi:10.5334/jors.508.

[12] D. H. Wolpert and W. G. Macready, No Free Lunch Theorems for Optimization, *IEEE Transactions on Evolutionary Computation*, vol. 1, 1997, pp. 67–82.

[13] R. Senkerik, T. Kadavy, P. Janku, M. Pluhacek, H. Guzowski, L. Pekař, R. Matušů, A. Viktorin, M. Smołka, A. Byrski, and Z. Oplatkova, Maximizing Efficiency: A Comparative Study of SOMA Algorithm Variants and Constraint Handling Methods for Time Delay System Optimization, 2023, pp. 1821–1829. doi:10.1145/3583133.3596417.

[14] T. Kadavy, A. Viktorin, M. Pluhacek, and S. Kovář, On modifications towards improvement of the exploitation phase for SOMA algorithm with clustering-aided migration and adaptive perturbation vector control, in *Proc. 2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 1–8. doi:10.1109/SSCI50451.2021.9659916.

[15] M. Matusikova, M. Pluhacek, T. Kadavy, A. Viktorin, and R. Senkerik, Exploring adaptive components of SOMA, in *Proc. 2023 Genetic and Evolutionary Computation Conference Companion (GECCO)*, 2023, pp. 1803–1811. doi:10.1145/3583133.3596421.

[16] L. Škanderová, Self-organizing migrating algorithm: review, improvements and comparison, *Artificial Intelligence Review*, vol. 56, pp. 101–172, 2023. doi:10.1007/s10462-022-10167-8.