Applied Soft Computing Journal 165 (2024) 112045

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

# Self-tuning multi-layer optimization algorithm (STML): An innovative parameter-less approach

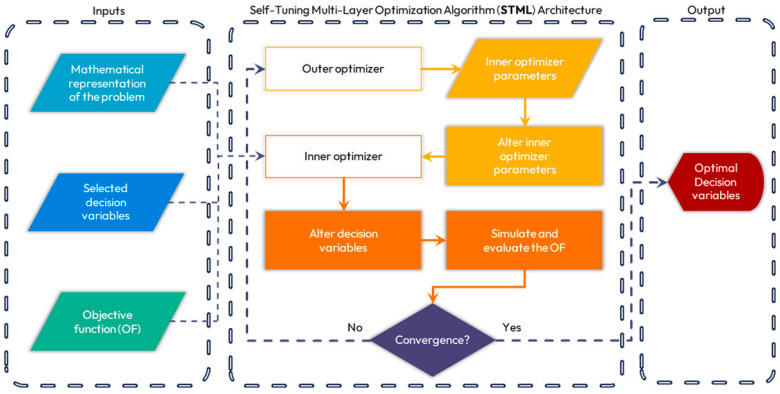
Babak Zolghadr-Asli [a,b,*], Milad Latifi [b], Ramiz Beig Zali [b], Mohammad Reza Nikoo [c], Raziyeh Farmani [b], Rouzbeh Nazari [d], Amir H. Gandomi [e,f]

[a] The Sustainable Minerals Institute, The University of Queensland, Brisbane, Australia
[b] Centre for Water Systems, University of Exeter, Exeter, UK
[c] Department of Civil and Architectural Engineering, Sultan Qaboos University, Muscat, Oman
[d] Department of Civil, Construction, and Environmental Engineering, the University of Alabama at Birmingham, USA
[e] Department of Engineering and Information Technology, University of Technology Sydney, Australia
[f] University Research and Innovation Center (EKIK), Óbuda University, Budapest 1034, Hungary

## HIGHLIGHTS

- Self-tuning multi-layer (STML) autonomously carries out the optimization.
- STML architecture eliminates manual parameter calibration.
- STML removes the need for initial guesses on parameter impact for optimal results.
- Preliminary results highlight STML's robustness and computational efficiency.

## GRAPHICAL ABSTRACT




## ARTICLE INFO

*Keywords:*
Meta-heuristic algorithms
Computational intelligence
Optimization
Self-tuning

## ABSTRACT

Computational intelligence (CI)-based methods offer a practical approach to overcoming the significant challenges posed by analytical and enumeration optimization methods when dealing with complex real-world problems. However, a notable drawback of these algorithms is the need for time-consuming and computationally demanding fine-tuning procedures to achieve optimal performance. This paper proposes a novel parameterless auto-tuning meta-heuristic architecture called the self-tuning multi-layer (STML). The fundamental concept behind this architecture involves a multi-layer structure where the inner layer optimizes the main problem. In contrast, the outer layer utilizes information obtained during the search to fine-tune the performance of the inner layer. This feature eliminates manual fine-tuning, as it can autonomously handle this task. A series of mathematical and benchmark problems were employed to demonstrate the computational prowess of the STML.

* Corresponding author at: The Sustainable Minerals Institute, The University of Queensland, Brisbane, Australia.
*E-mail addresses:* b.zolghadrasli@uq.net.au, bz267@exeter.ac.uk (B. Zolghadr-Asli), m.latifi@exeter.ac.uk (M. Latifi), rb815@exeter.ac.uk (R. Beig Zali), m.reza@squ.edu.om (M.R. Nikoo), R.Farmani@exeter.ac.uk (R. Farmani), rnazari@uab.edu (R. Nazari), gandomi@uts.edu.au (A.H. Gandomi).

https://doi.org/10.1016/j.asoc.2024.112045
Received 29 January 2024; Received in revised form 5 July 2024; Accepted 17 July 2024
Available online 10 August 2024

The results indicate its superiority over other meta-heuristic algorithms. Additionally, the STML showcases robustness, as evidenced by the numerical proximity of results obtained from different independent runs on these benchmark problems.

## 1. Introduction

Mathematical programming, commonly known as optimization, entails identifying the optimal solution from a set of feasible alternatives. This field has consistently remained a subject of great interest, particularly in disciplines such as engineering e.g., [1–3]. In recent times, the significance of optimization has grown substantially due to the gradual depletion of natural resources, necessitating the maximization of profit margins while minimizing adverse socio-economic and environmental impacts.

From a practical standpoint, optimization procedures must adhere to essential criteria, which include accuracy, simplicity, speed, and the ability to tackle complex problems. In an ideal scenario, an optimization method should possess all these qualities simultaneously. However, in practice, it is often challenging to maintain all of them, and one may need to be prioritized over others. For instance, traditional analytical methods that depend on derivatives of objective functions, providing gradient-based information for optimal solution identification, are generally highly accurate. However, they face challenges when confronted with high dimensionality, multimodality, epistasis, non-differentiability, and constraints within the search space—common characteristics of most real-world optimization problems [4,5].

On the other hand, sampling-based approaches, predominantly guided search methods, offer a more hands-on approach to practical optimization problems, as they are not strictly limited by differentiable search spaces [6]. However, the speed and flexibility of these algorithms often come at the expense of accuracy, as they tend to settle for *near-optimal solutions* rather than global optima. Ultimately, the mathematical structure of the optimization problem and the priorities set by decision-makers determine which approaches are best suited for a given problem.

Meta-heuristic optimization algorithms, one of the most notable examples of sampling-based approaches, find their theoretical foundation in the fundamental principles of computational intelligence (CI), earning them the name CI-based optimization methods. This category of optimization techniques serves as an alternative strategy that addresses the limitations of traditional analytical-based approaches and unguided sampling-based methods. These methods have successfully bridged the gap between these two vastly different approaches, giving rise to a new school of thought for tackling real-world optimization problems. However, from an implementation perspective, some pressing issues persist with these algorithms. Notably, there is no guarantee that they will converge to the optimum solutions, and they may encounter challenges such as getting stuck in local optima [7].

These algorithms incorporate parameters as a measure to allow for adjustments of their searching properties. However, the *no-free-lunch theorem*, a fundamental principle in CI, emphasizes that while fine-tuning an algorithm's parameters can enhance its performance in terms of accuracy and convergence rate for a specific problem, there can never be a universally optimal parameter set that applies to all algorithms and problem instances. This highlights the necessity of fine-tuning each algorithm based on the specific problem it aims to solve.

Fine-tuning algorithms, however, can be computationally demanding and time-consuming. Firstly, there is no guarantee that the identified settings are indeed the best parameters for a given problem. Furthermore, these algorithms' intricacy and stochastic nature often make it difficult to intuitively grasp how specific parameters influence their search properties. This lack of straightforward interpretability complicates the parameter-tuning process. To address these practical challenges, many algorithms are designed with a simpler structure and fewer parameters. Examples include the interior search algorithm [8], sine cosine algorithm [9,10], the crow search algorithm [11], and teaching-learning-based optimization [12,13]. Another approach is to leverage the modular capabilities of these algorithms by coupling them with other computational mechanisms such as greedy search, chaotic maps, quantum computing, self-adaptive strategies, or Lévy flight techniques to enhance their computational capacities. This should theoretically reduce the heavy reliance of the algorithms on parameter selection procedures [14–16]. While these options represent significant advancements, the challenge of fine-tuning meta-heuristic algorithms remains inherent in computational intelligence-based optimization methods.

This study introduces a novel approach called the Self-Tuning Multi-Layer (STML) computational architecture. The concept behind this architecture is to create a structure that autonomously fine-tunes itself through an iterative process, eliminating the need for manual fine-tuning. The proposed architecture leverages a multi-layer stack of quasi-metaheuristic algorithms, where the outer algorithm continuously adjusts the parameters of an inner optimization algorithm that handles the given optimization problem. What is significant about this architecture is that, through logical and mathematical reasoning, it was possible to pre-set the structure of the outer algorithm, making it parameterless and allowing the algorithm to operate independently without requiring user input for the optimization process. This breakthrough enables the utilization of the flexibility offered by CI-based optimization algorithms without the challenges associated with manual fine-tuning, making it a practical and efficient option. The structure of this paper is as follows: 2 will delve into the theoretical foundation and mathematical description of the proposed STML computational architecture, elucidating its inner workings and mechanisms. 3 summarizes the results obtained from implementing this architecture to solve well-known mathematical benchmark problems and a more complex real-world engineering problem, highlighting the capabilities of the proposed architecture. Finally, 4 will discuss the significance of this breakthrough, particularly its potential profound impact on creating self-tuning effects within the context of CI-based optimization algorithms.

## 2. Self-tuning multi-layer (STML) computational architecture

The underlying principle of random sampling forms the foundation of most CI-based optimization algorithms. What sets these algorithms apart from other sampling-based methods is their incorporation of guided search, which enhances their effectiveness in traversing the search space to find the optimal solution. This unique approach earned them the term "meta-heuristic," coined by [17], to distinguish them from other sampling-based optimization techniques.

Unlike analytical approaches, these guiding systems do not rely on differential equations, making them more suitable for addressing complex real-world problems. Although they follow a rigorous algorithmic procedure and are not entirely ad hoc per se, they require parameter tuning in order to adapt their structures to the specific problem at hand, as dictated by the no-free-lunch theorem. While parameter tuning grants these algorithms flexibility to handle complex problems, it has become a practical burden due to the overwhelming number of parameters in most modern CI-based optimization algorithms. Moreover, in some cases, the role of each parameter in influencing the exploration or convergence of the search process is not intuitively clear, adding to the challenges of effectively optimizing these algorithms.

The governing principle of a CI-based optimization algorithm is to

iteratively modify the positions of the searching agents based on feedback information, gradually converging towards a potential optimum solution. The fundamental idea behind this paper is to design a computational structure for parameter setting purposes so that the optimization algorithm can iteratively adjust its parameters using the information gathered from the search process. This architecture allows the meta-heuristic algorithm to fine-tune itself without needing external intervention. This concept aligns with *meta-optimization* or *hyper-heuristics* with a notable distinction. In practice, previous implementations often involved swapping the need to calibrate one set of parameters to another e.g., [18] or create a new objective function e.g., [19]. However, the proposed self-tuning multi-layer (STML) computational architecture introduces a novel approach where the algorithm can iteratively adapt its parameters through the search process, ultimately enabling self-tuning without manual parameter calibration or defining new objective functions. By incorporating this self-tuning mechanism, the STML algorithm represents a promising advancement in the field of meta-heuristics, as it streamlines the optimization process and reduces the reliance on manual parameter tuning and objective function design.

It is also important to highlight that the proposed architecture significantly differs from the adaptive strategy. While both approaches involve dynamic adjustments of parameters and searching properties during each iteration, the adaptive strategy relies on an explicitly predefined, mathematically formulated mechanism that essentially readjusts the characteristics of the exploration and exploitation phases of the algorithm. Consequently, these gradual parameter changes would occur regardless of whether they lead to practical improvements in the current searching process, as the adaptive strategy does not consider any feedback information from the searching agents or the obtained results. In contrast, the underlying concept of the proposed architecture revolves around creating an algorithmic structure that dynamically executes this idea without any pre-set notions about the optimization problem. Instead, it leverages the information acquired from the search process to make informed decisions. The objective is to design an algorithm that adapts and evolves during execution, utilizing the feedback from the ongoing search to guide its exploration and exploitation and effectively improve its performance over time. This dynamic adaptability sets it apart from the predefined and static nature of the adaptive strategy.

Implementing a self-tuning optimization method involves the algorithm dynamically adjusting its parameter values based on the information obtained from the ongoing search process. One approach to achieve this is optimizing the algorithm's parameters to better align with the properties of the specific optimization problem. This can be accomplished through a series of simultaneous optimizations, where one optimizer focuses on the main problem while others work to enhance the algorithm's overall performance. This necessitates a multi-layer structure to provide the complexity required for simultaneous optimizations.

In this paper, the STML architecture is proposed, which comprises two computational layers: An *inner optimizer* responsible for addressing the actual optimization problem and an *outer optimizer* simultaneously dedicated to optimizing the parameters of the inner layer. This dual-layered architecture enables the algorithm to fine-tune its performance dynamically throughout the execution process. The subsequent sections will provide a detailed description of the computational structure of these two layers, elucidating their roles and interactions within the self-tuning optimization framework.

### 2.1. Inner optimization layer

Drawing inspiration from the teaching-learning-based optimization algorithm (TLBO) introduced by [12], the inner optimizer within the proposed STML architecture can be described as a stochastic population-based approach grounded in the principles of swarm intelligence. Computationally, the design of the inner layer is structured around two main phases. In the first phase, the primary objective is to enforce exploitation, reinforcing the local search mechanism of the optimizer. This phase aims to exploit promising solutions within the current population to refine their quality and convergence towards better optima. Conversely, the second phase focuses on exploration, preventing the algorithm from being confined to local optima. By incorporating exploration, the optimizer seeks out new and unexplored regions of the search space, broadening its search horizon and increasing the chances of finding global optima. It should be noted that, in contrast to the original TLBO algorithm, which utilizes a greedy strategy to enhance its effectiveness by permitting only improving moves, a slightly different approach has been adopted here. This modified strategy involves stacking all tentative results in a pool once the new generation is calculated. Subsequently, the next generation is comprised solely of the best solutions from this pool. In essence, the inner optimizer can be seen as a controlled randomized local search, leveraging the approach mentioned above to consistently improve the population set's properties.

Mathematically speaking, in an optimization problem with $N$ decision variables, an $N$-dimension coordination system could be used to represent the search space. In this case, any point within the search space, say $X$, can be represented as a $1\times N$ array as follows:

$$X = (x_1, \quad x_2, \quad x_3, \quad \ldots, \quad x_j, \quad \ldots, \quad x_N) \tag{1}$$

Here, $X$ represents a search agent in the search space of an optimization problem with $N$ decision variables, and $x_j$ represents the value associated with the $j$th decision variable.

The initiation phase of the inner optimizer consists of randomly placing a series of search agents within the feasible boundaries of the search space. This bundle of arrays can be mathematically expressed as $M \times N$ matrix, where $M$ denotes the number of search agents or what is technically referred to as the population size. A population, denoted by *pop*, can be represented as follows:

$$pop = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_i \\ \vdots \\ X_M \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,j} & \ldots & x_{1,N} \\ x_{2,1} & x_{1,2} & \ldots & x_{2,j} & \ldots & x_{2,N} \\ & & & \vdots & & \\ x_{i,1} & x_{i,2} & \ldots & x_{i,j} & \ldots & x_{i,N} \\ & & & \vdots & & \\ x_{M,1} & x_{M,2} & \ldots & x_{M,j} & \ldots & x_{M,N} \end{bmatrix} \tag{2}$$

Where $X_i$ represents the $i$th agent in the population, and $x_{i,j}$ denotes the $j$th decision variable of the $i$th agent.

As stated, the algorithmic structure of the inner optimizer itself is composed of two phases, one of which serves as a local search engine, and the other is more exploratory-oriented. After the initial population set is randomly placed within the search space, the position of the search agents needs to be altered via the local search mechanism. The algorithm then needs to identify the best local optima encountered thus far in each given iteration. The mentioned search agent could be mathematically represented as follows:

$$Xbest = (x_{best,1}, \quad x_{best,\ 2}, \quad x_{best,3}, \quad \ldots, \quad x_{best,j}, \quad \ldots, \quad x_{best,N}) \tag{3}$$

where, $x_{best,j}$ denotes the position of the best-identified search agent in the population in the $j$th dimension, and *Xbest* represents the best solution encountered until the current iteration.

The main idea of this phase is to create a generic motion that pushes the entire set toward the identified local optimum position. To do so, the inner optimizer would first require computing the center of gravity for the current population set. The center of gravity, here denoted by *Xmean*, can be computed as follows:

$$Xmean = (x_{mean,1}, \quad x_{mean,\ 2}, \quad x_{mean,3}, \quad \ldots, \quad x_{mean,j}, \quad \ldots, \quad x_{mean,N}) \tag{4}$$

$$x_{mean,j} = \frac{\sum_{i=1}^{M} x_{i,j}}{M} \forall j \quad (5)$$

In which, $x_{mean,j}$ denotes the position of the center of gravity in the population in the $j$th dimension.

The inner optimizer would use the above-stated coordination to adjust the population set as follows:

$$diff_{i,j} = Rand \times \left(x_{best,j} - R_F \times x_{mean,j}\right) \forall j, T_F \in \{1, 2\} \quad (6)$$

$$x'_{i,j} = x_{i,j} + diff_{i,j} \forall j \quad (7)$$

where, $x'_{i,j}$ denotes the new position in the $j$th dimension for the tentative solution associated with the $i$th search agent; $diff_{i,j}$ denotes the value of the repositioning vector for the $i$th search agent in the $j$th dimension; *Rand* is a randomly generated value within the range 0–1; and $R_F$ denotes a random factor that assumes the value 1 or 2.

After the local search phase, the inner optimizer would continue by randomly coupling two search agents from the population set to impose a more exploratory-based motion on them. Assume the $r$th and $i$th search agents, denoted by $X_r$ and $X_i$, have been randomly coupled from the population set for this purpose. Both these agents would be evaluated. The agent with the better properties in terms of the objective function value serves as a guiding point, here denoted by *Xlocbest*, for the other agent, denoted by *Xlocworst*. Assuming that this is the $i$th search agent, the new tentative position for this agent could be computed as follows:

$$x'_{i,j} = x_{i,j} + Rand \times \left(x_{locbest,j} - x_{locworst,j}\right) \forall j \quad (8)$$

where, $x_{locbest,j}$ denotes the position of the *Xlocbest* in the $j$th dimension; and $x_{locworst,j}$ denotes the position of the *Xlocworst* in the $j$th dimension. It is essential to highlight that the same strategy employed in the previous phase is also utilized here.

The inner optimizer would iteratively execute the above procedures and update the search agents' position in each attempt. The inner optimizer terminates the search as it reaches a predefined number of iterations, here denoted by $T$. The population size ($M$) and the maximum permitted number of iterations ($T$) would serve as the parameters of the inner optimizer. The pseudo-code for the inner optimizer is depicted in Fig. 1.

### 2.2. Outer optimization layer

As mentioned earlier, the inner optimizer addresses the primary optimization problem. However, to do so, as the no-free-lunch theorem dictates, its parameter requires assuming specific values that need to be fine-tuned to get the best performance out of the inner optimizer. These would be the two parameters, namely, the population size ($M$) and the maximum permitted number of iterations ($T$). To circumvent a manual fine-tuning process for these parameters, an outer optimizer would attempt to systematically tweak these parameters using the same principles used in CI-based optimization algorithms. This, in effect, creates a hyper-two-dimensional search space composed of feasible values for the parameters. The outer optimizer would then enumerate through this new search space to identify the optimum setting for the inner optimizer.

The outer optimizer employed in this study is a stochastic single-solution local search optimization method, drawing inspiration from the pattern search (PS) algorithm introduced by [20]. It is essential to highlight that, when compared to the inner optimizer, the outer layer deals with a considerably less intricate search space. Consequently, the algorithmic structure of the outer optimizer is intentionally designed to be less complex. Furthermore, in this stage, the decision variables involved are solely the parameters of the inner optimizer. As a result, the search space comprises solely positive integer values, adding to the simplicity of the optimization process.

Like other stochastic meta-heuristic optimization methods, the initiation phase of the outer algorithm involves placing the search agent randomly within the feasible boundaries of the second search space. Given the local search-based nature of the outer optimizer, a series of trial or tentative points around the search agent's vicinity are evaluated during the optimization process. This allows the algorithm to identify a trajectory that converges toward a potential optimum solution. In each evaluation procedure, the outer optimizer repositions the search agent only if such a move improves the objective function's properties. To facilitate these local searches, the outer optimizer incorporates two mechanisms. The first mechanism involves a local search that evaluates tentative points arranged in a mesh grid network centered around the current position of the search agent. If the outer optimizer identifies an enhancing move in this search, it employs the second exploratory mechanism. This second mechanism is designed to test whether continuing this trajectory would improve the search agent's properties. If confirmed, the search agent is relocated to the new coordinates in the search space; otherwise, it remains unchanged.

To mathematically capture these procedures, a two-dimensional search space is needed where the search agent is represented with a vector similar to the one in Eq. (1), where $N$ equals 2. Next, the outer optimizer would form a gridded mesh around the current position of the search agent, where $\mu$ denotes the size of this mesh grid. In general, the tentative points created in this stage are equal to $2 \times N$, which can be mathematically expressed as follows:

```
Begin
    Generate the initial population
    Let M denote the population size
    While termination criterion is not met
        Evaluate the generated agents
        Let Xbest denote the best agent in the set
        Compute the population mean, denoted by Xmean
        Let diff denote the distance between Xbest and Xmean
        Use diff to update the position of the population
        Accept improving moves and reject the non-improving ones
        For i in range 1 to M
            Select the rth agent denoted by X_r randomly
            Use the better agent to improve the X_r position
            Accept an improving move and reject a non-improving one
        Next i
    End while
    Report the best solution
End
```

**Fig. 1.** Pseudo-code for the inner optimizer.

$$X_1^{new} = \mu.[1, \quad 0, \quad 0, \quad \ldots, \quad 0] + X$$

$$X_2^{new} = \mu.[0, \quad 1, \quad 0, \quad \ldots, \quad 0] + X$$

$$\vdots$$

$$X_N^{new} = \mu.[0, \quad 0, \quad 0, \quad \ldots, \quad 1] + X$$

$$X_{N+1}^{new} = \mu.[-1, \quad 0, \quad 0, \quad \ldots, \quad 0] + X$$

$$X_{N+2}^{new} = \mu.[0, \quad -1, \quad 0, \quad \ldots, \quad 0] + X$$

$$\vdots$$

$$X_{2N}^{new} = \mu.[, \quad 0, \quad 0, \quad \ldots, -1] + X \tag{9}$$

where, $X_i^{new}$ is the $i$th tentative solution.

Suppose the outer optimizer fails to recognize any improving move in this stage. In that case, it reduces the mesh grid size using a contraction coefficient, denoted by $\delta$, which can be expressed mathematically as follows:

$$\mu^{new} = \mu - \delta \tag{10}$$

Where, $\mu^{new}$ is the new mesh grid size. Any time during the searching process when an exploratory move leads to an improvement, the value for the mesh grid size should be restored to its initial default value.

However, if the outer optimizer recognizes any improving move in the previous stage, it executes an additional exploratory move. The idea here is to imitate and potentially amplify the improving pattern identified in the last stage. By doing so, the outer algorithm can explore the second search space with less computation effort, which, in turn, can potentially reduce the computational time of the optimization process. The described procedure can be mathematically expressed as follows:

$$X^{new} = X' + \alpha.(X - X') \tag{11}$$

where, $X'$ is the previous positions of the search agent; $X$ denotes the current positions of the search agent; $\alpha$ represents a positive acceleration factor; and $X^{new}$ is a new tentative or trial point. Similar to the previous exploratory move, the tentative point would only be accepted if it improves the performance of the search agent in terms of the objective function. As for the termination protocol for the outer optimizer, as stated, upon consecutive failure execution of the exploratory move, the size of the mesh grids could drop below a certain termination threshold, denoted by $\gamma$, in which case the optimization process would be stopped. The current position of the search agent would then be returned as the optimum solution. The pseudo-code for the outer optimizer is depicted in Fig. 2.

An essential aspect of the STML computational architecture is how the parameters of the inner and outer optimizers interact. The primary goal of the outer optimizer is to automatically fine-tune the parameters of the inner optimizers, eliminating the need for manual adjustments. However, the outer optimizer itself contains a few parameters, namely, the mesh grid size ($\mu$), contraction coefficient ($\delta$), acceleration factor ($\alpha$), and termination threshold ($\gamma$).

It is crucial to note that this unique computational structure provides considerable flexibility in handling these parameters. Unlike the inner optimizers' parameters, the outer optimizer's parameters are not directly involved in the actual optimization problem. Instead, they act as an additional buffering layer atop another meta-heuristic algorithm, which interacts with the primary optimization problem. As a result, this approach creates a more robust search mechanism and solution-finding procedure, as demonstrated in the result section. Moreover, due to this buffering mechanism, the hyper-layer search space remains limited to a two-dimensional space with positive integer values, significantly restricting the feasible values for these parameters. In practice, beyond a certain point, the performance of the STML becomes independent of these parameters. Therefore, all the stated parameters of the outer layer can be set to default values derived from a logical foundation. These default values, summarized in Table 1, must be used to construct the mathematical formulation of the outer optimizer in the STML algorithm. It is essential to emphasize that all tests and analyses of the algorithm's performance were conducted using these default parameter values. This characteristic renders the STML architecture a parameter-less optimization method. Overall, the computational structure of the STML architecture is illustrated in Fig. 3.

## 3. Numeric evaluation of the STML algorithm

In order to evaluate the performance of the STML architecture, it will be tested against a set of mathematical benchmark problems. To provide the proper context for this analysis, the TLBO and PS algorithms, which inspire the inner and outer optimizer, respectively, have been selected to solve the same mathematical benchmark problems. For more detailed

**Table 1**
Values that are used to render the outer optimizer parameterless.

| Parameter of the outer optimizer | Used default value |
|---|---|
| Mesh grid size ($\mu$) | 2 |
| Contraction coefficient ($\delta$) | 1 |
| Acceleration factor ($\alpha$) | 2 |
| Termination threshold ($\gamma$) | 1 |

```
Begin
    Generate a random point within the search space
    While mesh size is greater than the termination threshold
        Generate tentative points in the grid
        If tentative points contained a better point than the current base point
            Reset mesh size
            Implement the additional exploratory move
            If the new move is successful
                Replace the base point
                Conduct another exploratory move, and replace the base point if need be
            End if
        Otherwise
            Decrease the mesh size grid using the contraction coefficient
        End if
    End while
    Report the base point as the solution
End
```

**Fig. 2.** Pseudo-code for the outer optimizer.

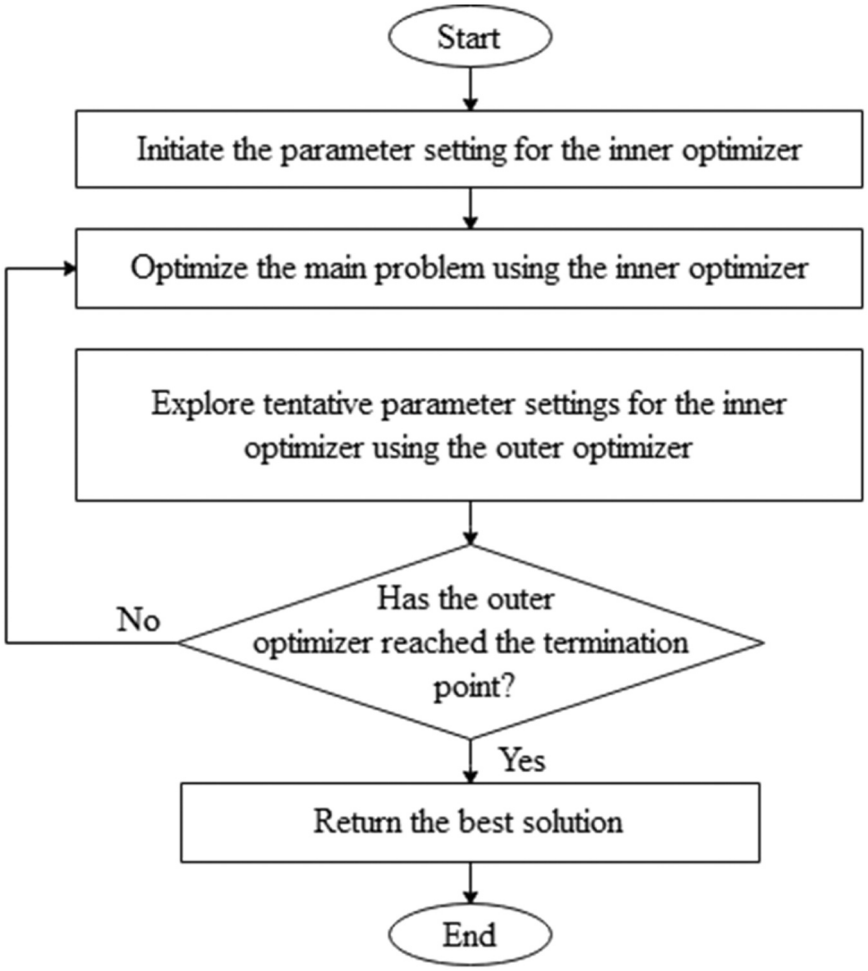



**Fig. 3.** Computational flowchart of the STML architecture.

information on the computational structure of these well-established and respected algorithms, readers can refer to [20] and [12]. As a concluding part of this numeric analysis, an engineering benchmark problem has also been chosen to demonstrate the performance of the STML architecture when confronted with more intricate optimization challenges. In both sets of benchmark problems, the outcomes were subsequently assessed through rigorous statistical analysis employing non-parametric tests — the *two-sample* Wilcoxon signed-rank test and the Friedman test. The former is applied to compare the central tendencies of two populations using paired samples, with the null hypothesis ($H_0$) being that the two samples are from the same population, while the alternative hypothesis ($H_1$) is that the samples are drawn from two statistically different populations. The latter test is designed to ordinate a set of paired samples. In assessing the performance of CI-based optimization algorithms, the Wilcoxon signed-rank test has traditionally been used to evaluate the relative efficacy of two given algorithms. Conversely, the Friedman test would often be utilized to rank the algorithms under examination statically. The significance level (i.e., $p$-value) adopted throughout this investigation is 0.05. Interested readers are encouraged to refer to [21] and [22] for a more detailed description and guidelines on these standard analytical methodologies.

### 3.1. Mathematical benchmark problems

The CEC2017 suite from the IEEE Congress on Evolutionary Computation [23] was chosen as a non-centred mathematical benchmark suite to showcase the computational capabilities of the STML architecture. This suite consists of a collection of predefined benchmarks specifically designed to challenge CI-based optimization algorithms by deliberately relocating the optimum solution away from the center of the search space. This precautionary measure ensures that central bias tendencies do not artificially enhance the performance of the evaluated algorithms. Given the stochastic nature of these algorithms, each one was independently executed fifty times for each mathematical benchmark problem, each comprising ten independent variables. This approach facilitates a more comprehensive evaluation of their performance. For more in-depth information about this benchmark suite, readers can refer to the comprehensive work by [23].

In addition to the STML architecture, the PS, TLBO, and inner optimizer were utilized to solve the mathematical benchmark suite. However, unlike the STML, which can autonomously fine-tune itself, the other tested algorithms were calibrated in advance to identify potentially suitable parameter settings for each specific benchmark problem. Subsequently, these determined settings were employed to optimize the problems through independently executed runs. The results of these evaluations are summarized in Table 2, and the distribution of emerged solutions is depicted in Fig. 4.

The results indicate that the STML algorithm has demonstrated superior performance compared to other tested algorithms in 15 cases. Among these, 6 cases were tied in ranking with the inner optimizer, which is the core of the STML architecture. Remarkably, the inner optimizer and TLBO algorithms outperformed the other options in 14

**Table 2**
Performance of different algorithms against the mathematical benchmark suite.

| | Best algorithm (s) | Best obtained result | The mean of different algorithms | Std. of different algorithms |
|---|---|---|---|---|
| $f_1$ | STML, Inner optimizer | 1.00E+02 | 1.01E+04 | 2.00E+04 |
| $f_2$ | STML, Inner optimizer | 2.00E+02 | 2.00E+02 | 2.43E−03 |
| $f_3$ | STML, Inner optimizer | 3.00E+02 | 3.00E+02 | 3.44E−01 |
| $f_4$ | STML, Inner optimizer | 4.00E+02 | 4.00E+02 | 5.20E−02 |
| $f_5$ | STML | 5.03E+02 | 5.76E+02 | 1.33E+02 |
| $f_6$ | Inner optimizer | 6.00E+02 | 6.25E+02 | 4.96E+01 |
| $f_7$ | STML | 7.07E+02 | 9.28E+02 | 4.28E+02 |
| $f_8$ | STML | 8.03E+02 | 8.50E+02 | 8.71E+01 |
| $f_9$ | STML, Inner optimizer | 9.00E+02 | 1.90E+03 | 2.01E+03 |
| $f_{10}$ | STML | 1.07E+03 | 1.68E+03 | 8.37E+02 |
| $f_{11}$ | STML, Inner optimizer | 1.10E+03 | 1.13E+03 | 4.85E+01 |
| $f_{12}$ | STML | 1.21E+03 | 9.68E+03 | 1.69E+04 |
| $f_{13}$ | STML | 1.30E+03 | 4.92E+03 | 7.23E+03 |
| $f_{14}$ | Inner optimizer | 1.40E+03 | 2.71E+03 | 2.62E+03 |
| $f_{15}$ | Inner optimizer | 1.50E+03 | 3.90E+03 | 4.81E+03 |
| $f_{16}$ | STML | 1.60E+03 | 1.78E+03 | 3.63E+02 |
| $f_{17}$ | Inner optimizer | 1.70E+03 | 6.01E+06 | 1.20E+07 |
| $f_{18}$ | Inner optimizer | 1.80E+03 | 1.39E+04 | 2.41E+04 |
| $f_{19}$ | Inner optimizer | 1.90E+03 | 6.26E+03 | 8.72E+03 |
| $f_{20}$ | Inner optimizer | 2.00E+03 | 5.12E+07 | 1.02E+08 |
| $f_{21}$ | TLBO | 2.19E+03 | 2.27E+03 | 1.35E+02 |
| $f_{22}$ | TLBO | 2.20E+03 | 2.70E+03 | 9.74E+02 |
| $f_{23}$ | TLBO | 2.59E+03 | 2.79E+03 | 3.91E+02 |
| $f_{24}$ | STML | 2.47E+03 | 2.62E+03 | 1.99E+02 |
| $f_{25}$ | TLBO | 2.62E+03 | 2.83E+03 | 1.39E+02 |
| $f_{26}$ | TLBO | 2.74E+03 | 3.27E+03 | 8.52E+02 |
| $f_{27}$ | TLBO | 3.09E+03 | 3.18E+03 | 1.78E+02 |
| $f_{28}$ | TLBO | 3.04E+03 | 3.11E+03 | 5.87E+01 |
| $f_{29}$ | Inner optimizer | 3.13E+03 | 1.28E+07 | 2.57E+07 |
| $f_{30}$ | STML | 3.39E+03 | 1.49E+04 | 2.31E+04 |

and 7 cases, respectively, making them the second and third-best alternatives. On the other hand, the PS algorithm ranked last in this comparison. Moreover, the relative outcomes for the inner optimizer and TLBO independently suggest that the former holds an edge over the latter. This observation highlights a significant improvement, considering that the inner optimizer drew inspiration from the TLBO algorithm.

The observed patterns are further validated by the results of the non-parametric Wilcoxon signed-rank test (Table 3). These findings confirm that the STML algorithm has only been statistically outperformed three times, and in all those cases, it was surpassed by the TLBO algorithm. Additionally, in 22 instances, the results were statistically indistinguishable from the top-performing algorithms. In most cases, STML performed statistically better than the other tested algorithms. These trends are consistent with the results obtained from the non-parametric Friedman test (Table 4). Based on these comprehensive results, the STML algorithm, the inner optimizer, and the TLBO algorithm are identified as the top three performing options, with the PS algorithm assuming the last position in this ranking.

Another noteworthy observation is the robustness of the tested methods, which can be inferred from the results presented in Fig. 4. The variability in an algorithm's performance across independent runs results from its stochastic nature. A *robust* algorithm, however, should exhibit consistent and statistically similar performance, leading to closely resembling results. In other words, a robust algorithm should have a smaller standard deviation of the obtained results across these independent runs than a non-robust algorithm. Based on this criterion, the STML is the most robust, showing the lowest standard deviation in 18 instances (6 times tied place). Following closely, the inner optimizer and TLBO algorithms display robustness in 14 and 3 cases, respectively, positioning them as the next most robust tested algorithms.

Finally, the last critical factor is the computational speed of these algorithms, which can be assessed by analyzing the number of function evaluations (NFEs) required for each algorithm to achieve convergence (Table 5). This metric reflects the proficiency of the algorithms in utilizing less computational power and achieving faster convergence. It is important to note that among the tested algorithms, both STML and PS, due to their unrestricted computational structures, may require different NFE values to achieve convergence, even under the same setup. Based on the results obtained, the top-performing algorithms in terms of converging with the least number of NFEs are PS (16 times), STML (9 times), and TLBO (5 times). Despite simultaneously handling two layers of optimizers autonomously, it is remarkable that STML achieved such a high ranking in terms of computational efficiency. However, NFE values alone may not reflect the algorithm's overall performance entirely. Fig. 5 illustrates the relationship between the average results obtained and the corresponding NFEs required for convergence for each example to provide a more comprehensive perspective. The size of each data point represents the number of NFEs needed for the respective algorithm to converge. Based on these results, it is evident that the unique computational structure of STML provides it with an edge over the other tested options, enabling it to achieve better results with less computational effort.

Overall, the STML architecture demonstrated reliable performance, as evidenced by the accuracy of the optimum results. Additionally, it showed robustness highlighted by greater consistency in reaching the optimum solution in each independent run. Importantly, despite having a dynamic termination point triggered automatically by the algorithm, it proved to be computationally efficient. As such, it required far fewer NFEs to reach convergence compared to other tested algorithms in most benchmarks.

### 3.2. Engineering benchmark problems

While the previous section aimed to shed light on the fundamental differences between the STML, its computational architecture, and other meta-heuristic algorithms, this section demonstrates how the algorithm would perform against a more complex real-world problem. A constrained, non-linear water resources management case study has been selected to test the performance of the computational architecture in the face of more complex real-world problems. The problem, which represents a typical water engineering problem, is set to determine the optimum water allocation scheme for a reservoir that tends to meet the downtown agricultural demands. The objective of this setup is to minimize the amount of water deficit during the operation span by opting for the optimum amount of water released from the reservoirs on a monthly time-step, which brings the number of decision variables for this designed setup to 170 decision variables. Various studies have showcased that the above-mentioned problem is a formidable option to push these algorithms to their limit due to their complex nature e.g., [24–28]. The setup, which depicts a case study described in [3], can be mathematically formulated as follows [29]:

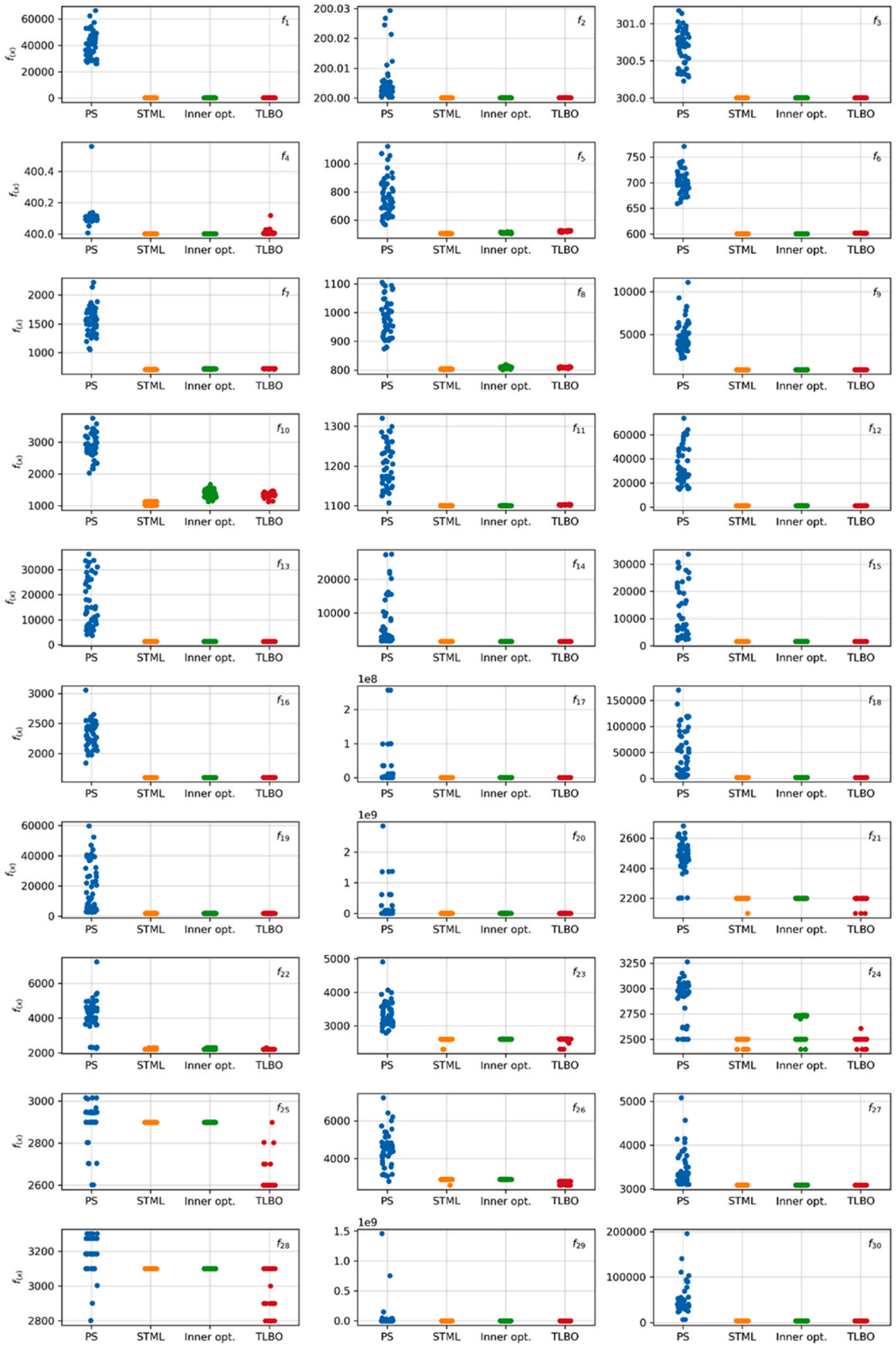



**Fig. 4.** Distribution of obtained results for each algorithm in the mathematical benchmark suite.

**Table 3**
Results for the non-parametric Wilcoxon test against the mathematical benchmark suite.

| Target | Functions | PS | Inner optimizer | TLBO |
|---|---|---|---|---|
| STML | $f_1$ | - | 0 | - |
| | $f_2$ | - | 0 | - |
| | $f_3$ | - | 0 | - |
| | $f_4$ | - | 0 | - |
| | $f_5$ | - | - | - |
| | $f_6$ | - | 0 | - |
| | $f_7$ | - | - | - |
| | $f_8$ | - | - | - |
| | $f_9$ | - | 0 | - |
| | $f_{10}$ | - | - | - |
| | $f_{11}$ | - | 0 | - |
| | $f_{12}$ | - | 0 | - |
| | $f_{13}$ | - | - | - |
| | $f_{14}$ | - | 0 | - |
| | $f_{15}$ | - | 0 | - |
| | $f_{16}$ | - | 0 | - |
| | $f_{17}$ | - | 0 | - |
| | $f_{18}$ | - | 0 | - |
| | $f_{19}$ | - | 0 | - |
| | $f_{20}$ | - | 0 | - |
| | $f_{21}$ | - | 0 | - |
| | $f_{22}$ | - | 0 | 0 |
| | $f_{23}$ | - | - | - |
| | $f_{24}$ | - | - | - |
| | $f_{25}$ | - | - | + |
| | $f_{26}$ | - | 0 | + |
| | $f_{27}$ | - | - | + |
| | $f_{28}$ | - | 0 | 0 |
| | $f_{29}$ | - | 0 | - |
| | $f_{30}$ | - | - | - |

**Note** | "-" signifies an inferior performance to the target; "+" means a superior performance to the target; and "0" identifies no statistically different performance to the target.

$$
\begin{aligned}
&Min\ \ OF = \sum_{t=1}^{T}(D_t - R_t)\\
&subjectto\\
&S_{t+1} = S_t + Q_t - R_t - loss_t - Sp_t\\
&loss_t = A_t \times Ev_t\\
&Sp_t = \begin{cases} S_t - S_{\max} & if \quad S_t > S_{\max} \\ 0 & else \end{cases}\\
&A_t = 0.03S_t + 0.8\\
&0 \le D_t \le R_t \ \ \forall_t\\
&S_{\min} \le S_t \le S_{\max} \ \ \forall_t
\end{aligned}
\tag{12}
$$

where, $OF$ denotes the objective function [10E+06 cubic meter (MCM)]; $T$ is the total number of operation periods (month); $D_t$ represents downstream water demand in the $t$th time step (MCM); $R_t$ denotes the reservoir water release in the $t$th time step (MCM); $S_{t+1}$ is the reservoir storage in the time step $t+1$ (MCM); $S_t$ denotes the reservoir storage volume in $t$th time stamp (MCM); $Q_t$ represents the reservoir inflow in the $t$th time step (MCM); $loss_t$ denotes the loss of water during period $t$ (MCM); $Sp_t$ is the spilled water in the $t$th time step (MCM); $A_t$ denotes the reservoir area in the $t$th time step ($m^2$); $Ev_t$ represents the depth of water loss during in the $t$th time step (m); $S_{min}$ is the reservoir's minimum storage capacity (MCM); and $S_{max}$ is the maximum reservoir storage capacity (MCM). Furthermore, the data input used for this case study is illustrated in Fig. 6. It is essential to mention that during the optimization process, the constraints were penalized proportionally to the extent they violated the defined boundaries. This measure, which is a standard procedure for using CI-based algorithm to handle constrained problems, was implemented to encourage the

**Table 4**
Results for the non-parametric Friedman test against the mathematical benchmark suite.

| | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ | $f_{16}$ | $f_{17}$ | $f_{18}$ | $f_{19}$ | $f_{20}$ | $f_{21}$ | $f_{22}$ | $f_{23}$ | $f_{24}$ | $f_{25}$ | $f_{26}$ | $f_{27}$ | $f_{28}$ | $f_{29}$ | $f_{30}$ | Overall Rank |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PS | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| STML | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| Inner optimizer | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 2 | 1 | 3 | 1 | 3 | 2 | 1 | 1 | 2 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 1 | 3 | 2 |
| TLBO | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 3 | 2 | 3 |

**Table 5**
Average number of NFE required for reaching coverage for each given algorithm.

| | Best algorithm (s) | Best obtained result | The mean of different algorithms | Std. of different algorithms |
|---|---|---|---|---|
| $f_1$ | Inner optimizer, TLBO | 1.50E+05 | 1.99E+05 | 6.19E+04 |
| $f_2$ | Inner optimizer, TLBO | 1.50E+05 | 2.00E+05 | 6.34E+04 |
| $f_3$ | Inner optimizer, TLBO | 1.50E+05 | 3.17E+06 | 5.97E+06 |
| $f_4$ | Inner optimizer, TLBO | 1.50E+05 | 6.10E+05 | 8.33E+05 |
| $f_5$ | TLBO | 1.50E+05 | 3.41E+05 | 1.68E+05 |
| $f_6$ | STML | 2.84E+05 | 1.60E+06 | 2.14E+06 |
| $f_7$ | PS | 2.38E+05 | 1.45E+06 | 2.23E+06 |
| $f_8$ | PS | 2.65E+05 | 1.46E+06 | 2.23E+06 |
| $f_9$ | PS | 1.90E+05 | 1.44E+06 | 2.25E+06 |
| $f_{10}$ | PS | 2.23E+05 | 1.45E+06 | 2.24E+06 |
| $f_{11}$ | STML | 2.77E+05 | 2.66E+06 | 2.48E+06 |
| $f_{12}$ | STML | 2.73E+05 | 2.67E+06 | 2.47E+06 |
| $f_{13}$ | STML | 2.84E+05 | 2.66E+06 | 2.48E+06 |
| $f_{14}$ | PS | 1.74E+05 | 2.51E+06 | 2.64E+06 |
| $f_{15}$ | STML | 2.81E+05 | 2.65E+06 | 2.49E+06 |
| $f_{16}$ | STML | 2.72E+05 | 2.61E+06 | 2.53E+06 |
| $f_{17}$ | PS | 1.71E+05 | 2.51E+06 | 2.64E+06 |
| $f_{18}$ | PS | 1.19E+05 | 2.50E+06 | 2.65E+06 |
| $f_{19}$ | PS | 1.30E+05 | 2.50E+06 | 2.66E+06 |
| $f_{20}$ | PS | 1.88E+05 | 2.52E+06 | 2.63E+06 |
| $f_{21}$ | STML | 2.73E+05 | 2.60E+06 | 2.54E+06 |
| $f_{22}$ | PS | 1.95E+05 | 2.52E+06 | 2.64E+06 |
| $f_{23}$ | PS | 1.81E+05 | 2.51E+06 | 2.64E+06 |
| $f_{24}$ | PS | 2.13E+05 | 2.53E+06 | 2.63E+06 |
| $f_{25}$ | PS | 1.28E+05 | 2.50E+06 | 2.65E+06 |
| $f_{26}$ | STML | 2.88E+05 | 3.22E+06 | 2.13E+06 |
| $f_{27}$ | PS | 2.14E+05 | 2.53E+06 | 2.63E+06 |
| $f_{28}$ | PS | 2.37E+05 | 2.53E+06 | 2.62E+06 |
| $f_{29}$ | PS | 2.37E+05 | 2.53E+06 | 2.62E+06 |
| $f_{30}$ | STML | 2.93E+05 | 3.48E+06 | 2.15E+06 |

algorithms to prioritize and return feasible solutions, effectively addressing the constraints imposed in the optimization problems [30].

The problem was solved using the TLBO and STML, with each algorithm undergoing 15 independent runs. The summarized results are presented in Table 6. The prerequisite of using this setup is for the TLBO algorithm to be fine-tuned for this specific problem while the STML inherently operates without such calibration. In the table, the results are color-coded using a green-yellow-red scheme to indicate the acceptability of the values. Green represents more desirable values, while red indicates the least acceptable solutions. Notably, the best solution, in terms of accuracy (i.e., the least objective function value) and computational efficacy (i.e., the least NFEs required for convergence), was obtained from the STML results. On average, the STML achieved an objective function value of 9.79E-02, surpassing the average result of 1.62E-01 obtained from the fine-tuned TLBO algorithm. This significant difference highlights the superior performance of the STML architecture. Additionally, while the STML algorithm required an average of 2.06E+06 NFEs to converge, the fine-tuned TLBO needed 3.75E+06 NFEs to achieve convergence. It is worth mentioning that in two independent runs, STML surpassed this NFE value to reach convergence. Nevertheless, considering that STML, in contrast to TLBO, operates without requiring any fine-tuning and still achieves better results with fewer NFEs, underscores the merits of this computational architecture. Furthermore, it is essential to note that the STML algorithm consistently returned feasible solutions in all cases, while the TLBO algorithm produced infeasible solutions in four instances.

## 4. Concluding remarks

In addressing the intricate search spaces and the overwhelming number of decision variables that often challenge conventional optimization methods, Computational Intelligence (CI)-based optimization approaches emerge as a practical solution for tackling real-world optimization problems. Nonetheless, the Achilles' heel of meta-heuristic algorithms lies in the laborious, technically demanding, and time-consuming fine-tuning process. Furthermore, pinpointing the optimal parameter settings for a given problem is not guaranteed.

The Self-Tuning Multi-Layer (STML) computational architecture has been introduced here to address this inherent challenge. This algorithmic innovation automates the fine-tuning procedure by implementing a multi-layer structure that leverages the principles of CI. The outer layer of the algorithm gradually fine-tunes the inner layer, which is responsible for the core optimization task. This architecture eliminates the need for user-driven parameter fine-tuning, ushering in a new era of autonomous optimization.

The results obtained in this study shed light on the potential of the STML architecture to address real-world optimization challenges. Our findings underscore this unique architecture's exceptional capabilities in enhancing the optimization process's computational efficiency. It consistently outperforms conventional meta-heuristic algorithms or, at the very least, demonstrates comparable performance.

It is crucial to note that, in practical applications, fine-tuning an algorithm entails a manual trial-and-error process inherently characterized by time consumption and the need for meticulous analysis to unlock potential performance enhancements through parameter adjustments. However, the outcomes of this study demonstrate that the STML architecture not only autonomously handles this fine-tuning task more efficiently in most cases but also offers a more robust alternative. These capabilities can be further elevated by integrating parallel computing, fine-tuning code implementation, and harnessing the formidable computational prowess of cutting-edge machines. In addition to testing STML against practical benchmark problems and complex real-world scenarios, another interesting future prospect is to expand the proposed architecture by increasing the number of layers to investigate any potential impact on the overall robustness of this algorithm.

## Authors contributions

Not applicable.

## Ethical approval

The authors declare all data and materials, as well as software applications or custom codes, are in line with published claims and comply with field standards.

## Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

## Consent to participate

Not applicable.

## Consent to publish

Not applicable.

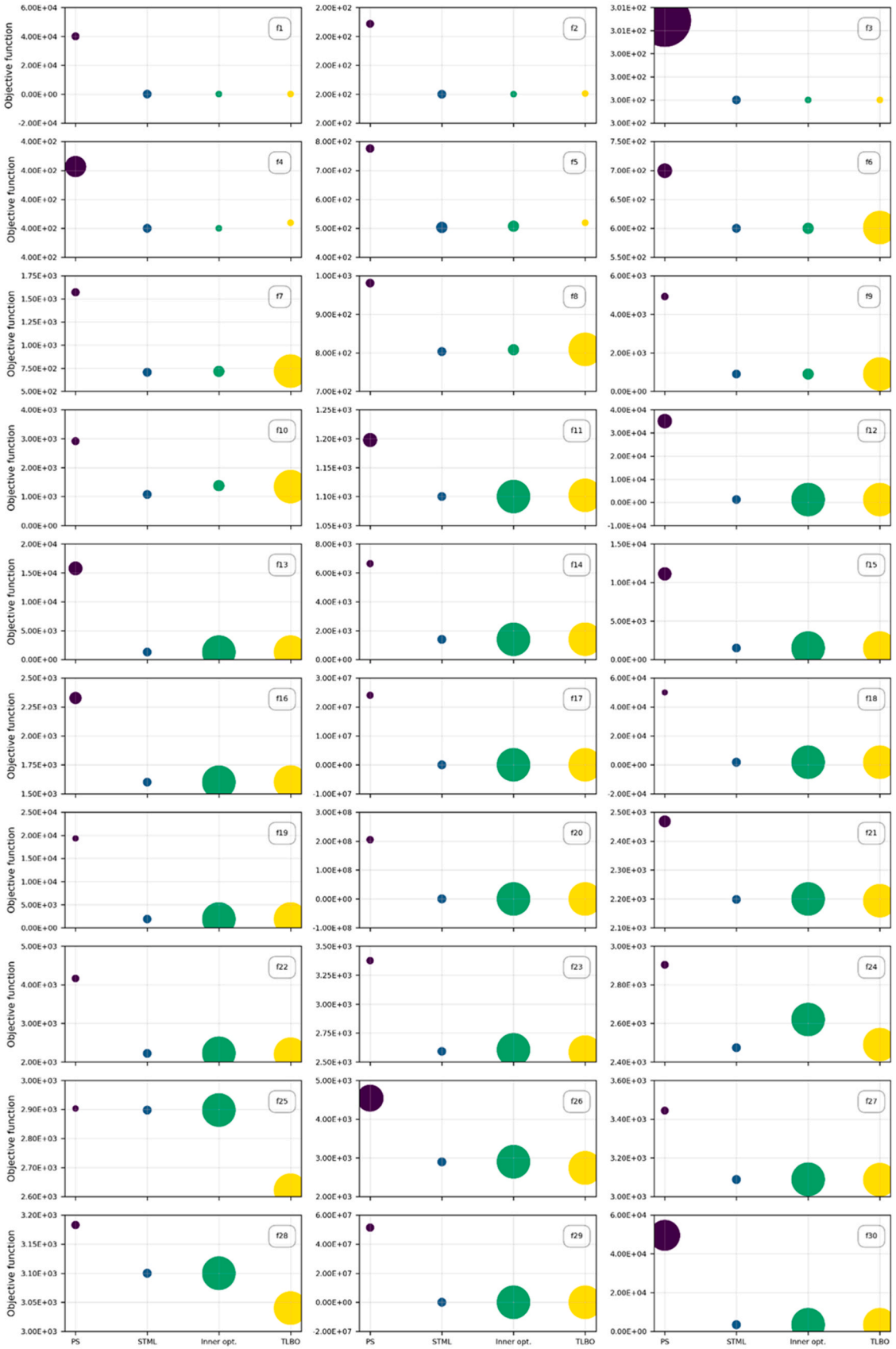



**Fig. 5.** Average obtained results for each given algorithm and the relative NFEs needed for convergence.

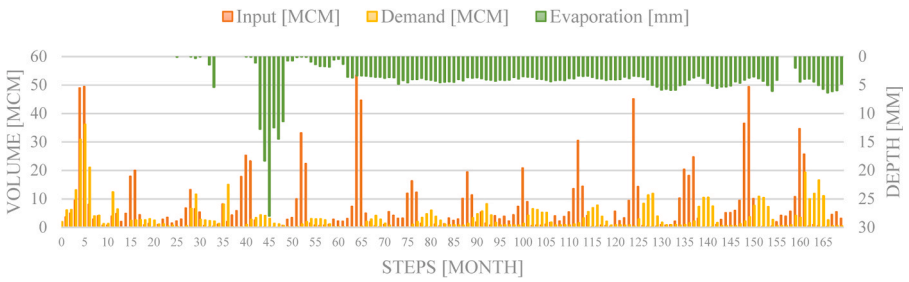



**Fig. 6.** Input values for the water allocation problem.

**Table 6**
Obtained results for the water allocation problem.

| | Run # | OF | NFE | Solution Status |
|---|---|---|---|---|
| STML | 1 | 1.05E-01 | 1.46E+06 | Feasible |
| | 2 | 9.30E-02 | 2.17E+06 | Feasible |
| | 3 | 1.22E-01 | 1.49E+06 | Feasible |
| | 4 | 1.20E-01 | 9.92E+05 | Feasible |
| | 5 | 7.53E-02 | 3.33E+06 | Feasible |
| | 6 | 7.76E-02 | 1.85E+06 | Feasible |
| | 7 | 8.42E-02 | 2.13E+06 | Feasible |
| | 8 | 9.72E-02 | 8.26E+05 | Feasible |
| | 9 | 1.15E-01 | 1.90E+06 | Feasible |
| | 10 | 1.07E-01 | 1.69E+06 | Feasible |
| | 11 | 1.08E-01 | 1.44E+06 | Feasible |
| | 12 | 6.63E-02 | 3.38E+06 | Feasible |
| | 13 | 8.31E-02 | 2.25E+06 | Feasible |
| | 14 | 1.05E-01 | 4.12E+06 | Feasible |
| | 15 | 1.08E-01 | 1.87E+06 | Feasible |
| TLBO | 1 | 1.81E-01 | 3.75E+06 | Feasible |
| | 2 | 1.70E-01 | 3.75E+06 | Feasible |
| | 3 | 1.49E-01 | 3.75E+06 | Feasible |
| | 4 | 1.53E-01 | 3.75E+06 | Unfeasible |
| | 5 | 1.62E-01 | 3.75E+06 | Feasible |
| | 6 | 1.62E-01 | 3.75E+06 | Unfeasible |
| | 7 | 1.67E-01 | 3.75E+06 | Feasible |
| | 8 | 1.69E-01 | 3.75E+06 | Feasible |
| | 9 | 1.52E-01 | 3.75E+06 | Feasible |
| | 10 | 1.53E-01 | 3.75E+06 | Unfeasible |
| | 11 | 1.84E-01 | 3.75E+06 | Feasible |
| | 12 | 1.46E-01 | 3.75E+06 | Unfeasible |
| | 13 | 1.73E-01 | 3.75E+06 | Feasible |
| | 14 | 1.62E-01 | 3.75E+06 | Feasible |
| | 15 | 1.55E-01 | 3.75E+06 | Feasible |

## CRediT authorship contribution statement

**Babak Zolghadr-Asli:** Writing – original draft, Investigation, Formal analysis, Data curation, Conceptualization. **Ramiz Beig Zali:** Investigation, Formal analysis, Data curation. **Milad Latifi:** Investigation, Formal analysis, Data curation. **Mohammad Reza Nikoo:** Writing – review & editing, Validation, Supervision, Project administration, Investigation. **Rouzbeh Nazari:** Writing – review & editing, Validation, Investigation. **Raziyeh Farmani:** Writing – review & editing, Validation, Supervision, Investigation, Conceptualization. **Amir H. Gandomi:** Writing – review & editing, Validation, Methodology, Investigation.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

All used data have been presented in the paper. The codes used to test these algorithms are available at the corresponding author's GitHub page: https://github.com/BabakZolghadrAsli/STML.

## References


[1] C.C.W. Chang, T.J. Ding, C.C.W. Ee, W. Han, J.K.S. Paw, I. Salam, G.S. Kuan, Nature-inspired heuristic frameworks trends in solving multi-objective engineering optimization problems, Arch. Comput. Methods Eng. (2024) 1–34.

[2] F. Wei, Y. Zhang, J. Li, Multi-strategy-based adaptive sine cosine algorithm for engineering optimization problems, Expert Syst. Appl. (2024) 123444.

[3] A. Yaghoubzadeh-Bavandpour, O. Bozorg-Haddad, M. Rajabi, B. Zolghadr-Asli, X. Chu, Application of swarm intelligence and evolutionary computation algorithms for optimal reservoir operation, Water Resour. Manag. (2022) 1–18.

[4] P. Sharma, S. Raju, Metaheuristic optimization algorithms: a comprehensive overview and classification of benchmark test functions, Soft Comput. 28 (4) (2024) 3123–3186.

[5] C.W. Tsai, M.C. Chiang, Handbook of metaheuristic algorithms: From fundamental theories to advanced applications, Elsevier, 2023. ISBN: 9780443191091.

[6] B. Zolghadr-Asli, Computational intelligence-based optimization algorithms: From theory to practice, CRC Press, 2023. ISBN: 9781003424765.

[7] K. Rajwar, K. Deep, S. Das, An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges, Artif. Intell. Rev. 56 (11) (2023) 13187–13257.

[8] A.H. Gandomi, Interior search algorithm (ISA): a novel approach for global optimization, ISA Trans. 53 (4) (2014) 1168–1183.

[9] S. Mirjalili, SCA: a sine cosine algorithm for solving optimization problems, Knowl. -Based Syst. 96 (2016) 120–133.

[10] R.M. Rizk-Allah, A.E. Hassanien, A comprehensive survey on the sine–cosine optimization algorithm, Artif. Intell. Rev. 56 (6) (2023) 4801–4858.

[11] A. Askarzadeh, A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm, Comput. Struct. 169 (2016) 1–12.

[12] R.V. Rao, V.J. Savsani, D.P. Vakharia, Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems, Comput. -Aided Des. 43 (3) (2011) 303–315.

[13] G. Zhou, Y. Zhou, W. Deng, S. Yin, Y. Zhang, Advances in teaching-learning-based optimization algorithm: a comprehensive survey, Neurocomputing (2023) 126898.

[14] Q. Yang, J. Liu, Z. Wu, S. He, A fusion algorithm based on whale and grey wolf optimization algorithm for solving real-world optimization problems, Appl. Soft Comput. 146 (2023) 110701.

[15] Y. Wang, H. Liu, G. Ding, L. Tu, Adaptive chimp optimization algorithm with chaotic map for global numerical optimization problems, J. Supercomput. 79 (6) (2023) 6507–6537.

[16] A. Yaghoubzadeh-Bavandpour, O. Bozorg-Haddad, B. Zolghadr-Asli, A. H. Gandomi, Improving approaches for meta-heuristic algorithms: a brief overview, Comput. Intell. Water Environ. Sci. (2022) 35–61.

[17] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13 (5) (1986) 533–549.

[18] M. Meissner, M. Schmuker, G. Schneider, Optimized particle swarm optimization (OPSO) and its application to artificial neural network training, BMC Bioinforma. 7 (1) (2006) 1–11.

[19] P. Krus, J. Ölvander, Performance index and meta-optimization of a direct search optimization method, Eng. Optim. 45 (10) (2013) 1167–1185.

[20] R. Hooke, T.A. Jeeves, "Direct Search" Solution of Numerical and Statistical Problems, J. ACM (JACM) 8 (2) (1961) 212–229.

[21] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of non-parametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evolut. Comput. 1 (1) (2011) 3–18.

[22] N.E. Zamri, M.A. Mansor, M.S.M. Kasihmuddin, S.S. Sidik, A. Alway, N.A. Romli, S. Z.M. Jamaludin, A modified reverse-based analysis logic mining model with Weighted Random 2 Satisfiability logic in Discrete Hopfield Neural Network and multi-objective training of Modified Niched Genetic Algorithm, Expert Syst. Appl. 240 (2024) 122307.

[23] Wu, G., Mallipeddi, R., & Suganthan, P.N. (2017). Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization. National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report.

[24] B. Beiranvand, P.S. Ashofteh, A systematic review of optimization of dams reservoir operation using the meta-heuristic algorithms, Water Resour. Manag. 37 (9) (2023) 3457–3526.

[25] M. Jahandideh-Tehrani, O. Bozorg-Haddad, H.A. Loáiciga, A review of applications of animal-inspired evolutionary algorithms in reservoir operation modelling, Water Environ. J. 35 (2) (2021) 628–646.

[26] A. Kangrang, H. Prasanchum, K. Sriworamas, S.M. Ashrafi, R. Hormwichian, R. Techarungruengsakul, R. Ngamsert, Application of optimization techniques for searching optimal reservoir rule curves: a review, Water 15 (9) (2023) 1669.

[27] M. Kazemi, O. Bozorg-Haddad, E. Fallah-Mehdipour, X. Chu, Optimal water resources allocation in transboundary river basins according to hydropolitical consideration, Environ., Dev. Sustain. (2022) 1–19.

[28] B. Nematollahi, M.R. Nikoo, A.H. Gandomi, N. Talebbeydokhti, G. R. Rakhshandehroo, A Multi-criteria decision-making optimization model for flood management in reservoirs, Water Resour. Manag. 36 (13) (2022) 4933–4949.

[29] B. Zolghadr-Asli, O. Bozorg-Haddad, H.A. Loáiciga, Stiffness and sensitivity criteria and their application to water resources assessment, J. Hydro-Environ. Res. 20 (2018) 93–100.

[30] O. Bozorg-Haddad, M. Solgi, H.A. Loáiciga, Meta-heuristic and evolutionary algorithms for engineering optimization, John Wiley & Sons, 2017.