



Towards complementing the algorithm selection process with performance estimation from low-dimensional knee-point data

Levente Filep^{ID}*, Zoltán Gál^{ID}

University of Debrecen, Egyetem tér 1, Debrecen, 4032, Hungary

ARTICLE INFO

Keywords:

Swarm intelligence
Algorithm selection
Performance trend analysis
Dimensionality scaling
Wireless sensor networks
Energy optimization

ABSTRACT

The application of Swarm Intelligence (SI) to complex real-world optimization problems, including those arising in Wireless Sensor Networks (WSNs), is well established in the scientific literature. More than 400 SI algorithms have been proposed, with thousands of studies investigating their applications, performance, and characteristics, yet selecting the most suitable algorithm for a new optimization problem remains challenging despite ongoing research. This paper investigates how the knee-point performance of SI algorithms evolves as problem dimensionality increases, and proposes a technique that uses lower-dimensional problem instances to forecast the relative ranking of algorithms at the full dimension, where their performance trends can be reliably identified. For the majority of the tested algorithms and problems, knee-point performance follows a consistent trend type as dimensionality increases. We identify the conditions under which this behavior is expected to hold, and observe that the trend type depends on the algorithm–problem pair rather than on the algorithm alone. The proposed technique is evaluated on four WSN topologies, whereas the existence of the observed trend-type behavior is additionally investigated using nine classic benchmark functions. The method does not estimate exact full-dimensional performance values and is not intended as a standalone algorithm selector. Instead, it serves as one stage of a broader algorithm-selection pipeline, operating after portfolio creation to reduce the set of candidate algorithms prior to final selection, thereby reducing the overall required computational budget. The applicability of the method has limitations, including the need for a sufficient dimensional range for reliable trend identification, the non-negligible computational cost of lower-dimensional sampling, and limited generalizability of the observed trend behavior across all optimization problems. In addition, high performance variance may lead to trend misidentification, resulting in either premature or delayed elimination of candidate algorithms.

1. Introduction

Global optimization is a branch of applied mathematics that focuses on developing methods to find the global optimum of a given problem, typically represented by a mathematical function. The dimensionality of such a function is determined by the number of parameters whose values must be determined to identify the global optimum in the given problem space. With applications in real-world domains such as engineering, finance, and machine learning, this field remains an important and constantly evolving area of research.

As application problems become increasingly complex, evaluating the target function at a large enough number of points to accurately approximate the problem landscape and guarantee the discovery of its optimum becomes impractical. For such complex problems, even a single-point evaluation can be extremely time-consuming, taking hours or even days. We often apply meta-heuristic algorithms to search for

the global optimum in such scenarios. While these methods do not guarantee the finding of global optimum, depending on the heuristics used, they can often find “good enough” solutions.

Swarm Intelligence (SI) algorithms are a class of meta-heuristic algorithms with nature-inspired search heuristics. These draw inspiration from the collective behavior of swarms found in nature, such as those of various animals, birds, fish, or other natural phenomena, including physical particles or swarms of bodies. The individuals in swarms are basic agents in the system that follow simple rules, yet their collective actions give rise to collective intelligence, hence the name Swarm Intelligence.

The literature mentions over 400 SI algorithms [1]. With such a vast number available, selecting the most suitable one for a given optimization problem is a challenging task. No SI algorithms perform well in every problem scenario, and their key characteristics often

* Corresponding author.

E-mail addresses: filep.levente@inf.unideb.hu (L. Filep), gal.zoltan@inf.unideb.hu (Z. Gál).

<https://doi.org/10.1016/j.swevo.2026.102455>

Received 25 February 2026; Received in revised form 18 May 2026; Accepted 22 June 2026

Available online 29 June 2026

2210-6502/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC license (<http://creativecommons.org/licenses/by-nc/4.0/>).

relate to specific problems. The literature presents several attempts to improve algorithm selection, and this remains an ongoing research area for further improvements.

Wireless Sensor Networks (WSNs) play an important role in modern communication and sensing technologies. Consisting of a collection of low-cost, autonomous sensors with self-organizing and real-time data acquisition capabilities, they are an indispensable technology for applications that demand cost-effective, real-time data monitoring solutions. Hence, WSNs are essential in applications such as environmental monitoring, healthcare, and smart cities. Often deployed in remote or hard-to-reach locations, these sensor nodes (SNs) typically operate on limited energy resources, often relying on battery power; therefore, the challenge is to optimize sensor energy consumption to extend their operational life, thereby extending the overall lifespan of the network. As a sensor's primary energy consumption is that of the transmission circuit, energy optimization in WSNs primarily focuses on efficient data transmission, routing, and clustering techniques.

Many optimization problems in wireless sensor networks (WSNs), such as transmission power assignment, cluster-head selection, routing, node deployment, and sink-node placement, are NP-hard [2–4]. Metaheuristics, including swarm intelligence (SI) algorithms, are often used for these problems due to their ease of implementation and their ability to find good or near-optimal solutions in reasonable time.

In this paper, we investigate how the knee-point performance of SI algorithms behaves as the dimensionality of an optimization problem increases, using WSN topologies as the applied target. We observe that, for the majority of the tested algorithms on WSN energy-related optimization problems, the knee-point performance follows a consistent trend type as dimensionality grows. We further verify the same behavior on nine well-known classic benchmark functions (i.e., classic benchmark functions such as Ackley and Rastrigin), although the specific trend type depends on the algorithm–problem pair rather than on the algorithm alone.

Building on this, we propose a technique that uses performance measured at lower dimensions to forecast the relative ranking of algorithms at higher dimensions, and we state the conditions under which this behavior is expected to hold. The technique is not a standalone algorithm selector; it is intended as one stage of a larger selection pipeline, applied after an algorithm portfolio has been assembled and before the final selection is made, in order to reduce the number of candidate algorithms that must be evaluated at the target dimension.

The remainder of this paper is organized as follows. Section 2 reviews related work on swarm intelligence, performance variation, knee-point detection, benchmark problems, WSN optimization, and algorithm selection. Section 3 introduces the proposed trend-type model of knee-point performance along problem dimensions and the conditions under which it is expected to hold. Section 4 describes the testing methodology, including the WSN topologies, the classic benchmark functions, the selected SI algorithms, and the knee-point calculation. Section 5 presents the results on both the WSN and benchmark problems, examines the stability of the performance and knee-point measurements, and demonstrates a practical application in the algorithm selection process together with its limitations. Section 6 concludes the paper. As the investigation presented here originated from a practical WSN problem and was later tested on the classic benchmark functions, each relevant section presents the WSN problem first, followed by the benchmark functions.

2. Background and related work

Real-world problems are often computationally too expensive for the entirety of the solutions to be explored. Metaheuristic algorithms offer solutions to such complex problems by exploring and exploiting search spaces using nature-inspired strategies and techniques. These types of algorithms do not sample the entirety of the solution space, but instead use a higher level of heuristics to converge to a good solution

based on a relatively small number of samples. While these algorithms do not guarantee that the best solution will be found, they provide a good enough solution within a given computational budget.

Metaheuristics remain a very active research field, with classical paradigms such as genetic algorithms still being refined and increasingly hybridized with other soft computing techniques such as machine learning [5,6]. Production scheduling is a representative domain in which this evolution is clearly visible. Pan et al. [7] addressed the distributed energy-efficient parallel machines scheduling problem with a knowledge-based two-population optimization algorithm in which NSGA-II and Differential Evolution cooperate in parallel on a shared population, supported by problem-specific initialization heuristics and local search operators derived from structural properties of the problem; the resulting method jointly minimizes total energy consumption and total delays. Wang et al. [8] took a different route and added a learning layer on top of classical dispatching rules: their dual Q-learning scheme combines a top-level agent that handles machine assignment and load balancing with a bottom-level agent that selects among compound dispatching rules (e.g., lowest-level-code with longest or shortest processing time) to minimize the weighted earliness of jobs in an assembly job shop with uncertain assembly times, outperforming single-level Q-learning and each standalone dispatching rule. Pushing this hybridization further, [9] proposed QIHGRL for dynamic flexible job shop scheduling, which combines a heterogeneous graph neural encoder with reinforcement learning, a question-aware aggregation module that turns the scheduling objectives into attention signals, and an interoption prompt attention layer that balances exploration and exploitation across candidate actions, outperforming previous state-of-the-art algorithms. Together, these works illustrate a broader trend: metaheuristic frameworks are continuously revisited and enriched, both through algorithmic refinements and through integration with learning-based components, to keep up with the growing complexity of real-world optimization problems.

Also a long-standing and active branch of metaheuristics is Swarm Intelligence [10,11], which refers to a category of metaheuristic algorithms inspired by the behavior of swarms found in nature, such as biological swarms, including bird flocks and insect colonies, as well as physical systems like gravitational or particle interactions. Composed of multiple agents that follow simple rules, the swarm behaves as an overall intelligent system. This “intelligence” is an emerging feature of the interactions among numerous simple agents rather than from any single agent's capability.

With a large number of SI algorithms proposed in the literature [1], an open question is how to select algorithms for a novel type of optimization problem that will perform well enough within a given computational budget. Blind algorithm selection and execution to experimentally determine the best one is unfeasible for large-scale complex problems. Improving selection methods to better utilize or even reduce the computational budget requirement is an ongoing challenge. In this section, we present a brief overview of key aspects related to SI algorithms as well as some of the proposals found in the literature.

2.1. Swarm intelligence

SI algorithms operate by exploring the parameter space of an optimization problem P , attempting to converge to the global optimum in a series of steps (iterations). Let us denote the possible parameter values of P as S , and let $s \in S$ represent a potential (candidate) solution. In this case, S is the search space where the SI algorithm operates and searches for the best solution. The quality of a potential solution s is evaluated via a fitness function f , defined as $f : S \rightarrow \mathbb{R}$, where the value of $f(s)$ is called the fitness value of s . This induces a fitness landscape over the solution space, where the objective function defines a mapping from each candidate solution to its corresponding fitness value.

Furthermore, we can define this mapping explicitly as $F = \{(s, f(s)) | s \in S\} \subset S \times \mathbb{R}$, which forms an n -dimensional fitness landscape, also referred to as the objective space. The goal of any SI algorithm is to explore and exploit F efficiently, being guided by the landscape features (e.g., slopes) to converge towards the best possible solution, namely its global optimum. In most cases, F is associated with a minimization problem; therefore, most SI algorithms are designed to find solutions that minimize $f(s)$.

A computational evaluation of $f(s)$ is called a fitness function evaluation, denoted as (*FFEval*). For computationally demanding problems, the cost of a single $f(s)$ evaluation is significantly higher than the algorithm's other computational requirements. In such cases, an algorithm's computational budget is often specified only as the *maximum FFEvals*, which in turn determines how long the algorithm will run.

The *curse of dimensionality* expression was introduced by Richard E. Bellman in 1957 regarding dynamic programming [12] and used by Robert C. Taylor in 1993 regarding optimization problems [13]. SI algorithms are naturally affected by the curse of dimensionality, as the size of a candidate solution s (search space coordinates) increases with respect to f , requiring more computation. Furthermore, as dimensionality increases, the convergence time and execution speed of the algorithms display a super-linear slowdown [14,15]. The scale of impact varies depending on the algorithm and its implementation.

2.2. SI performance and its variation

When evaluating metaheuristic and swarm-intelligence algorithms, the *convergence curve* (C) — a plot of a scalar quality indicator of the search against the iteration counter or the number of objective function evaluations — is one of the most common tools for analyzing the performance of an algorithm [16,17]. It allows visual comparison between competing methods, showing properties such as convergence rate, stalls, and time-to-target metrics.

Many SI algorithms use a best-so-far fitness retention strategy, commonly called *elitism* after its analogue in evolutionary computation [18, 19]. Classical Particle Swarm Optimization, for example, applies this through its gbest memory [20]. A direct consequence is that the C curve, defined as the best fitness value C_k attained up to iteration k as a function of k , is monotonic by construction: the values of C either improve or stay the same.

All SI algorithms rely on some form of randomness; without it they would not work as intended. The use of an RNG within an algorithm can be summarized in three points:

- *Initialization*: the population is placed randomly in the search space.
- *Position updates*: most swarm agent movements have a stochastic component. This gives an agent a chance to escape a local optimum, which improves exploration.
- *Individual selection*: many algorithms randomly select individuals to perform certain roles or actions within the swarm.

From the points above, each SI algorithm has some inherent variation in its performance. In a previous paper [21], we examined the effect of the RNG type on several algorithms and found that each algorithm is affected to some degree, but the size of this effect varies. It depends on three factors: the algorithm itself, the RNG type, and the optimization problem instance.

2.3. Knee-point concept and detection in optimization algorithm performance

The κ^* knee-point (and subsequently the elbow point) is the point in an algorithm's performance curve beyond which marginal improvements in the objective function value diminish relative to the computational cost of further running it. In single-objective algorithms, it

can be utilized in parameter tuning and identifying cut-off points in algorithm execution time. In multi-objective scenarios, the knee-point often represents a balance between competing objectives, guiding the algorithm to focus on solutions that achieve the best compromise rather than pursuing marginal improvements in individual objectives [22–24]. There are several methods for detecting the knee-point position, including geometric, curvature-based, distance-based, and regression-based methods, as well as others [25].

To find the κ^* in runtime, an SI algorithm must overshoot this point by a sufficient number of iterations for the detection method used to be effective. To the best of our knowledge, in the case of optimization algorithms, the dominant approach is the use of stagnation heuristics, which can identify a stabilization phase of an algorithm, meaning a point past the knee-point, after which a form of knee-point detection method is applied to the measured points of the convergence curve.

2.4. Benchmarks

Benchmark functions play an important role in evaluating and assessing the performance, robustness, and various characteristics of optimization algorithms. They possess well-known properties, including modality, smoothness, and scalability. Unimodal functions have a single global optimum and no local optima, while multimodal functions feature multiple local optima, making them more challenging for optimization algorithms. Many such functions have been widely used in the literature for decades; therefore, we will collectively refer to them as classic benchmark functions.

It is well understood that no single algorithm can outperform all others across every optimization problem. Different algorithms perform best on certain types of problems or specific instances of the same problem, while lagging on others. In the context of arbitrary search landscapes, this concept is referred to as the no-free-lunch (NFL) theorem [26]. Furthermore, this phenomenon has been observed within most instances of NP-hard optimization problems [27]. When discussing specific problem instances, this is known as *performance complementarity* [27,28]. In a previous paper [15] on WSN benchmarking, we observed that the type of WSN landscape generation method has an impact on specific algorithms. Despite being within the same class of problem, they are different problem instances.

While the literature contains many experimental results regarding which algorithms work best on a particular class of problems, such classes usually contain a limited set of instances [27]. Most often, benchmark problems are used to evaluate and compare the performance of algorithms. The most well-known are the Competition on Evolutionary Computation (CEC) benchmark suites, which contain a variety of different problem classes. In [29], the authors benchmark 73 optimization problems and note differences between the best-performing algorithms on older CEC benchmarks and those on newer CEC 2020 benchmarks. [30] proposes guidelines for presenting new algorithms. One proposal is to account for the different characteristics of the benchmarks used to evaluate a new algorithm. It also highlights the importance of comparing performance with that of existing algorithms. [31] reviews a large set of mathematical and real-world engineering benchmarks of various meta-heuristic methods with the aim of better identifying the strengths and weaknesses of these. In [32], the authors point out a small number of problems in a benchmark, as well as the limitations of benchmarking with a fixed number of *FFEval* calls. They recommend using both a low and a high number of *FFEvals* in benchmarks to improve comparison results.

While benchmark functions can be classified [31] due to their smaller number, real-world optimization problems arguably cannot be. Nonetheless, benchmarking provides important insights into the strengths and weaknesses of these algorithms' performance. While benchmarks still limit our knowledge of algorithm performance to a known set of problems, well-performing algorithms across multiple classes of problems are expected to also perform well on similar types of

problems. In other words, it strengthens our confidence when applying these algorithms to even a different class of problems.

2.5. WSN optimizations using SI

A simple energy optimization strategy was introduced by G. L. da Silva Fré et al. [33], where the authors demonstrated the use of the PSO algorithm to minimize the transmission power of each node, reducing the overall power consumption while maintaining the WSN fully connected. Recently, A. K. Yadav et al. [34] demonstrated the use of the Artificial Ecosystem-based Optimization (AEO) [35] algorithm for the same optimization problem. Researchers and practitioners successfully applied SI algorithms to various single- and multi-objective optimization problems in the field of WSN [36], including energy minimization, routing, and clustering [37,38].

2.6. Algorithm selection

The *algorithm selection problem* (ASP) [27,28,39] is the task of automatically choosing, for a specific problem instance, the algorithm expected to perform best from a given set of candidate algorithms based on measurable characteristics (features) of the instance in question. The ASP framework can be formalized as a mapping between the optimization problem, algorithm, performance, and characteristics (features). The problem can be formulated as follows: for a given $\mathcal{F}$ fitness space and a set of $\mathcal{A}$ with n algorithms that can be applied to $\mathcal{F}$, the goal is to identify α in the portfolio of $\mathcal{A}$, for which the measure of $\rho(\mathcal{F}, \alpha)$ is the minimum, where ρ is a measure of using algorithm α to solve $\mathcal{F}$. The ρ measure is usually the algorithm's computational cost for solving $\mathcal{F}$ within a given budget, which involves a time-and-cost combination. In other words, find the α algorithm that we can expect to perform well within a given computational budget. If the computational budget allows for multiple algorithm runs, it is best to identify a set of algorithms $\{\alpha_1, \alpha_2, \dots, \alpha_m\} \in \mathcal{A}$, where $m \ll n$, that is likely to perform the best. Also, the smaller the m , the better the selection is considered. However, this selection process has its limitations. Similarities in problem characteristics do not guarantee accurate algorithm predictions. Nevertheless, forecasting algorithms that are likely to perform better rather than relying on blind selection still represents a significant advancement.

In this paper, we propose a method that supplements ASP by adding a low-dimensional sampling step: per-algorithm performance trends are identified from a small number of low-dimensional measurements and used to narrow the algorithm portfolio before evaluation at the target dimension. This approach is developed in Section 3.

3. SI performance trends along problem dimensionality

This research began with an investigation into the performance stability of SI algorithms as the number of sensor nodes in a WSN grows. More specifically, we aimed to investigate how the performance and, consequently, the ranking of algorithms change as the number of sensor nodes increases, and whether the best-performing algorithms would continue to perform best. Subsequently, our goal was to determine whether, as the sensor node count (i.e., problem dimensionality) increased, we could identify a trend in algorithmic performance. The hypothesis examined here is that, for each SI algorithm, the knee-point performance follows a parametric trend in problem dimensionality, and that this trend can be identified from a small set of low-dimensional measurements.

The motivation behind this hypothesis came from the following expectations: (a) by adding more and more SNs to the WSN, the value of the global minimum (the total SN energy required to maintain network connectivity) should increase monotonically, and (b) since

the agents in the swarm are driven by the same set of rules, we can expect each algorithm's performance slowdown to follow its own trend as the dimensionality grows. The shape of this slowdown trend is expected to differ between algorithms, which would explain why an algorithm can outperform the others at lower dimensions but, due to a steeper slowdown, fall behind at higher dimensions. Hence, if both the increase in the value of the global minimum and an SI algorithm's slowdown follow a trend curve, the algorithm's observed performance can be expected to follow the combination of these two curves as the dimensionality increases. However, since SI algorithms are stochastic processes, as mentioned in Section 2.2, their performance curves will show some degree of variation between runs; therefore, analyzing and comparing a convergence curve from a single run will not suffice.

To test this hypothesis, we chose to take the performance values at the knee-point of the SI algorithms' convergence curves. The knee-point represents the optimal balance between the cost and benefit of an algorithm. As shown in Fig. 8, the convergence of SI algorithms over their iterations can be represented as a well-defined curve, along which a knee-point can be determined whenever convergence occurs.

Let $\Xi_\alpha(d)$ be the average performance of algorithm α at the κ_α^* knee-point of its convergence curve in dimension d . We define the Θ_α curve as formed by Ξ_α values sampled across a set of test dimensions $D_s = \{d_1, d_2, \dots, d_s\}$ with $d_1 < d_2 < \dots < d_s$, as given in Eq. (1).

$$\Theta_\alpha = \Xi_\alpha(d), \text{ where } d \in D_s \quad (1)$$

We aim to determine whether Θ_α follows a clear trend. In our initial measurements, we observed three candidate trends: linear, a decaying exponential with a linear correction term (referred to hereafter as *combined*), and power law. To identify which one applies, we fit each of the three forms presented in Eqs. (2) separately and selected the one with the best fit.

$$\hat{y}(x) = \begin{cases} b \cdot x + a & (2.1) \\ a \cdot e^{-b \cdot x} + c \cdot x + d & (2.2) \\ a \cdot x^b + c & (2.3) \end{cases}$$

Ξ_α can be decomposed into sub-components as presented in Eq. (3), where f^* is the global optimum of the search space, P_α is the performance gap of algorithm α relative to f^* , and ϵ_α is the noise contributed by algorithm α . As previously mentioned, this noise arises from the use of a random number generator (RNG).

$$\Xi_\alpha(d) = f^*(d) + P_\alpha(d) + \epsilon_\alpha(d) \quad (3)$$

As the global optimum f^* is unknown — since this is the value we are attempting to identify — comparing algorithm performance to this value across multiple d is not possible. This means that if $f^*(d)$ varies irregularly with d , rather than following a smooth or monotonic dependence, the performance of all algorithms will inherit this irregularity. One idea to overcome this issue is calculating Θ relative to a reference algorithm. We denote this relative value $\Theta_{i,R}(d)$, as defined in Eq. (4).

$$\Theta_{i,R}(d) = \Xi_i(d) - \Xi_R(d) \quad (4)$$

Subtracting $\Xi_R(d)$ from every other algorithm's $\Xi_\alpha(d)$ would cancel out the variation introduced by $f^*(d)$. Furthermore, the sign of $\Theta_{i,R}$ indicates the relative performance of α_i with respect to α_R : a positive value indicates that α_i performs worse than α_R , a value close to zero indicates comparable performance, and a negative value indicates that α_i outperforms α_R . However, this approach comes with a major disadvantage: when the algorithms $\alpha \in \mathcal{A}$ exhibit mixed trend types, subtracting different curve types from one another produces a result that no longer follows a clean, predictable trend, and frequent zero-crossings of $\Theta_{i,R}$ are to be expected. A further disadvantage concerns the noise terms: if ϵ_{α_i} and ϵ_{α_R} are correlated, the noise in $\Theta_{i,R}$ will be amplified, making it harder to reliably identify the underlying trend. Identifying the individual Θ_α trends is therefore the most tractable of the available options.

A natural follow-up question is whether, when a clear trend exists in Θ_α , it can be used to forecast the algorithm's performance at higher dimensions. If so, this would have a practical application in algorithm selection. Furthermore, for a viable application, we need to forecast performance by sampling a smaller number of dimensions d such that the total computational cost is lower than evaluating all algorithms at the target dimension d_{target} .

We first describe the procedure for identifying the trend type from sampled measurements (Section 3.1), then establish the conditions under which this identification is reliable (Section 3.2), and conclude by outlining how the identified trends are integrated into the algorithm selection process (Section 3.3).

3.1. Identifying the trend type

For any algorithm $\alpha \in \mathcal{A}$ let $D_s = \{d_1, d_2, \dots, d_s\}$ be a set of s test dimensions with $d_1 < d_2 < \dots < d_s$. Let $\mathcal{T} = \{T_1, T_2, T_3\}$ denote the candidate trend families from Eq. (2), where each $T_k(\cdot; \theta)$ is a parametric model of Θ_α as a function of d , with parameter vector θ to be fitted per algorithm to the measured performance values $\{\Xi_\alpha(d_1), \Xi_\alpha(d_2), \dots, \Xi_\alpha(d_s)\}$ over D_s .

Step 1: Measurement. For every algorithm $\alpha \in \mathcal{A}$ and every test dimension $d \in D_s$, we measure $\Xi_\alpha(d)$, obtaining the discrete points of Θ_α over D_s .

Step 2: Per-trend parameter fitting. For each algorithm α and each trend type $T_k \in \mathcal{T}$, the parameter vector is estimated by least-squares fitting on the measured points:

$$\theta_{k,\alpha}^* = \arg \min_{\theta} \sum_{d \in D_s} (T_k(d; \theta) - \Xi_\alpha(d))^2. \quad (5)$$

Step 3: Trend selection. Among the three fitted trends, the one minimizing a model-selection criterion $\mathcal{J}$ (lower is better) is retained as the trend curve $\hat{\Theta}_\alpha$ of algorithm α :

$$k^*(\alpha) = \arg \min_{k \in \{1,2,3\}} \mathcal{J}(T_k(\cdot; \theta_{k,\alpha}^*), \{\Xi_\alpha(d)\}_{d \in D_s}), \quad (6)$$

$$\hat{\Theta}_\alpha(d) = T_{k^*(\alpha)}(d; \theta_{k^*(\alpha),\alpha}^*). \quad (7)$$

Common choices for $\mathcal{J}$ are the residual sum of squares, AIC, or BIC, with the latter two penalizing the model's number of free parameters.

The hypothesis is supported for algorithm α when $\hat{\Theta}_\alpha$ provides a sufficiently good fit to the measured points of Θ_α , judged by a goodness-of-fit indicator such as R^2 or the residual distribution. This is a separate check from the model-selection criterion $\mathcal{J}$ used in Step 3: $\mathcal{J}$ chooses between the three candidate trends, while the goodness-of-fit indicator decides whether any of them adequately describes the data. When the hypothesis holds across $\mathcal{A}$, the fitted trend curves can be used to forecast algorithm performance at unsampled dimensions, as developed in the next section. The candidate set $\mathcal{T}$ does not exclude trends of other forms for other problem classes; as we aimed for simplicity, we restricted it to these three types because they fit the observations reported in this paper, and because trend families with more free parameters would require substantially more sampled dimensions to fit reliably, increasing the cost of the method or making it infeasible.

3.2. Theoretical justification of trend-type stability across dimensions

In this section we formalize the hypothesis introduced above on how the knee-point performance curve Θ_α behaves as problem dimensionality increases. A universal proof is not realistically attainable: the no-free-lunch theorem [26] precludes any guarantee that our hypothesis will hold for every SI algorithm on every optimization problem, and the SI algorithm family is in any case too heterogeneous to admit a single proof across its members. Instead, we present the assumptions under which we expect the *trend type* of an individual algorithm to

remain stable as dimensionality grows. This is consistent with the results of Section 5, where Θ_α is well described by a single parametric trend family for the majority of the tested algorithm–problem combinations, while a minority of cases yield no valid fit; both outcomes are accounted for by the analysis that follows.

The decomposition of Eq. (3) separates $\Xi_\alpha(d)$ into a common term $f^*(d)$, an algorithm-specific gap $P_\alpha(d)$, and noise $\epsilon_\alpha(d)$. The common term is shared by all algorithms on a given problem: it may shift the absolute level and the apparent shape of every Θ_α , but does so identically for all α , and therefore does not affect which family best describes the algorithm-specific component $P_\alpha(d)$.

3.2.1. Assumptions

We base the justification on two per-algorithm regularity conditions (A1 and A2 below), applied to each algorithm's trend along the problem's dimensionality independently.

- **A1 (Dimension-stable convergence dynamics).** For a given algorithm α , the qualitative mechanism governing its knee-point slowdown does not change over the dimensional range of interest: the convergence dynamics are re-scaled in magnitude as d grows, but not re-parameterized in kind. Formally, $P_\alpha(d)$ is assumed to be a continuous function of d that can be well-approximated by a single trend family $T_k(\cdot; \theta)$, $k \in \{1, 2, 3\}$, over the dimensional range considered.
- **A2 (Bounded relative noise).** The stochastic term satisfies $|\epsilon_\alpha(d)| \leq \sigma_\alpha(d)$, with $\sigma_\alpha(d)$ small relative to the variation of $P_\alpha(d)$ across the sampling range. Equivalently, the coefficient of variation of $\Xi_\alpha(d)$ is bounded, a property examined empirically in Section 5.3.

Assumption A1 is a working assumption rather than a derived result. SI algorithms are typically defined by step-by-step procedures rather than by precise mathematical formulations, and the family is too diverse for any single proof to cover all of its members. Stated per algorithm, the assumption does not require different algorithms to share a trend family, and is therefore consistent with the mixed-trend behavior observed in Section 5.

3.2.2. Trend-type persistence

Observation 1 (Conditional Trend-Type Persistence). Let α be an algorithm satisfying A1 and A2 over a dimensional range $[d_1, d_{\text{target}}]$, and let $D_s = \{d_1, \dots, d_s\} \subset [d_1, d_{\text{target}}]$ be a set of sampled dimensions. Then the trend family $k^*(\alpha)$ selected by Eq. (6) on the samples $\{\Xi_\alpha(d)\}_{d \in D_s}$ coincides with the family that describes Θ_α over the full range, provided D_s spans enough of the curvature of Θ_α to distinguish the candidate families.

Justification. By A1, $P_\alpha(d)$ is well-approximated by a single family T_{k^*} over $[d_1, d_{\text{target}}]$, and the shared $f^*(d)$ does not affect family identification. By A2, the residual $\epsilon_\alpha(d)$ is small relative to the systematic variation of $P_\alpha(d)$ across D_s , so the least-squares fitting of Eq. (5) recovers the parameters of T_{k^*} rather than fitting noise. The qualifier on D_s is essential: if the sampled range is too narrow, two distinct families — for example, a linear curve and the early portion of a power curve — become numerically indistinguishable, and family identification is unreliable regardless of A1–A2. This requirement is addressed below.

Curve crossings across trend families. Observation 1 concerns each algorithm individually, and does not claim that the curves $\{\Theta_\alpha\}_{\alpha \in \mathcal{A}}$ keep the same relative order as dimensionality grows. When two algorithms follow different trend families — for example, one *linear* and one *combined* — their curves may cross, and the algorithm that performs best at low dimensions need not remain best at d_{target} . This matches the behavior reported in Section 5: on the T1 topology, for instance, BES leads at low dimensions but is overtaken by GTO as the number of sensor nodes grows. Such crossings are an expected consequence of

algorithms following different trend families, and are handled by the trend-extrapolation step of the selection method (Section 5.5).

3.2.3. Validity and failure regimes

Assumption A1 is plausible, and in part supported by existing analysis of dimensional scaling [14,15], under the following conditions:

- **Scalable landscape structure.** Problems in which the fitness function takes a separable or weakly coupled additive form

$$f(x) = \sum_{i=1}^d f_i(x_i) + \delta(x). \quad (8)$$

The term $\delta(x)$ denotes the non-separable remainder of the fitness function — the part of $f(x)$ that cannot be written as a sum of single-variable functions — and is assumed small for the problem classes considered here. For such problems both the global optimum and the difficulty of optimization scale predictably with d , so the per-coordinate contribution to the slowdown mechanism is qualitatively preserved as dimensions are added.

- **Smooth scaling of the optimum.** Problems whose global optimum $f^*(d)$ varies smoothly or monotonically with d , so that the shared term in Eq. (3) does not inject abrupt, dimension-specific distortions into every Θ_α .
- **Dimension-invariant search behavior.** Algorithms whose exploration–exploitation balance is governed by parameters that are not re-tuned as a function of d , so that the convergence mechanism is re-scaled but not re-parameterized.

Conversely, Assumption A1, and hence Observation 1, is expected to fail in the following regimes:

- **Strongly non-separable or irregular landscapes,** where the interaction term $\delta(x)$ in Eq. (8) dominates and the difficulty of optimization does not scale in a structurally consistent way with d .
- **Phase transitions or abrupt changes in difficulty,** where Θ_α changes family within the range of interest, for example by flattening or steepening sharply once a structural threshold in d is crossed.
- **Dimension-specific algorithmic heuristics,** where an algorithm alters its own behavior as a function of d , violating the re-scaled-but-not-re-parameterized premise of A1.
- **High relative noise (violation of A2).** When $\Xi_\alpha(d)$ is small in magnitude — which, for benchmark functions with an optimum at zero, occurs precisely for the best-performing algorithms — the RNG-induced ripple in Θ_α becomes large relative to its systematic variation, and no family can be identified reliably.

The last failure mode is directly corroborated by the results: the cases for which no valid fit is obtained in Section 5 are concentrated on the best-performing algorithms, such as BES and SpSA on several benchmark functions, whose near-zero Ξ values produce a large relative ripple in Θ . These no-fit cases are therefore not unexplained outliers, but the expected consequence of A2 being violated, consistent with the analysis above.

Sampling-range requirement. Observation 1 is not asymptotic: it concerns a finite sampling range D_s rather than the limit $d \rightarrow \infty$. Its practical content therefore depends on D_s being wide enough that the candidate families are numerically separable on the sampled points. We do not provide a closed-form lower bound on the required span, as such a bound would depend on the curvature of the underlying Θ_α , on the noise level $\sigma_\alpha(d)$, and on the model-selection criterion $\mathcal{J}$, and is not available in general. Instead, we state the requirement qualitatively — the sampled dimensions must be sufficiently far apart and must collectively cover enough of the curvature of Θ_α to discriminate between families — and treat it as an explicit boundary of the method. This is the same condition noted operationally in the description of

the selection algorithm (Section 5.5), here identified as the reason a minimum sampling span is needed.

3.2.4. Implications and scope

Because the knee-point performance $\Xi_\alpha(d)$ reflects the same convergence dynamics assumed dimension-stable in A1, Observation 1 transfers directly to knee-point-based measurements: under A1–A2 and an adequate sampling range, the trend family identified from low-dimensional knee-point data is the family that governs knee-point performance at d_{target} . Where algorithms belong to different families, their knee-point curves may cross; the selection method (Section 5.5) addresses these cases through trend extrapolation with uncertainty, which is supported empirically rather than established by the structural argument above.

The result established here is a conditional property, not a general guarantee, and follows from the assumptions stated above rather than from first principles. As noted at the start of this section, no general guarantee is available; what the formulation provides instead is consistency between the theoretical claim and the empirical method. The property it establishes — stability of trend *type* — is the property the selection method exploits, and it accommodates the mixed-trend, curve-crossing, and no-fit behavior observed in the experiments.

3.3. Integrating trend identification into algorithm selection

When the conditions of Section 3.2 hold for the algorithms in $\mathcal{A}$, the trend curves $\hat{\Theta}_\alpha$ identified from low-dimensional samples can be extrapolated to d_{target} and used to forecast algorithm performance. This forecast is then exploited for algorithm selection: at each sampling stage, the separations between forecasted performance curves are examined, and algorithms whose forecasts indicate significantly worse performance at d_{target} than the current best are eliminated from further consideration. Algorithms whose forecasts remain close to the leading ones are retained and sampled at additional dimensions, refining their trend fits and either confirming or revising their forecasts. The procedure terminates when no further confident separations can be drawn from the available samples, and additional criteria — such as algorithm runtime or memory cost — are used to select among the surviving candidates for evaluation at d_{target} . The full selection method, including the separation metrics and stopping conditions, is presented in Section 5.5.

4. Testing methodology

The algorithm test runs, data collection, and analysis were done in MATLAB R2024b. We utilized the publicly available implementations of each SI algorithm released on the MathWorks File Exchange. Test scenarios include generated WSN topologies, as well as nine of the well-known classic benchmark functions.

4.1. WSN topologies

In the WSN scenarios, we tested configurations with 10, 20, ..., 100 sensor nodes, yielding problem dimensions $d \in \{10, 20, \dots, 100\}$. For the proposed selection method, an additional 10 runs were performed at $d = 200$. Each node's power P_i can be adjusted in the range of $[-30 \dots 0]$ dBm. The optimization problem is as follows: minimize P_i so that the total WSN power usage is minimized, $\sum P_i \rightarrow \min$, but the network is still connected. The fitness function is presented in Eq. (9). This problem describes an optimization scenario within a WSN cluster or a small-scale network.

$$F = \begin{cases} \sum P_i, & \text{if network fully connected} \\ \infty, & \text{otherwise} \end{cases} \quad (9)$$

We generated the topology layouts using a random angle–radius generation method previously mentioned in [15], from which we obtained four different types of 20 m × 20 m 2D topologies with the sink

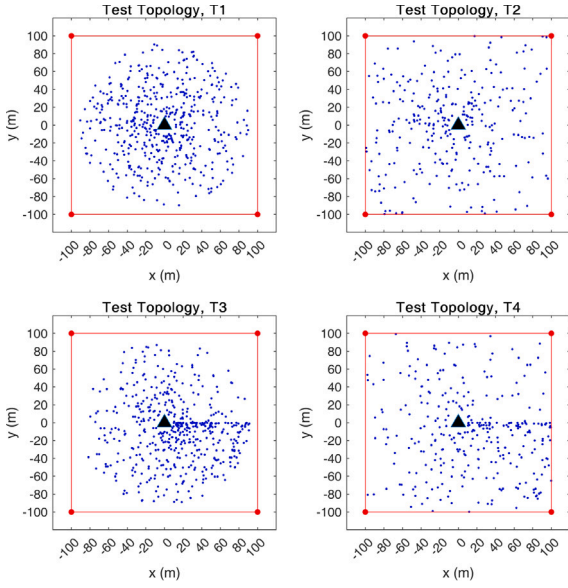


Fig. 1. WSN layout for T1, T2, T3 and T4 topologies.

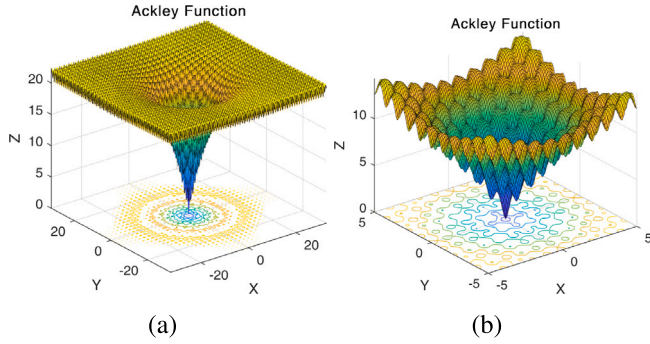


Fig. 2. Ackley function with (a) typical definition range, (b) restricted range (magnified center).

(base station) located at the center (0,0) coordinates. We named these according to their types: T1, T2, T3, and T4. Sensor nodes are placed at random angles and radii from the BS, using random and concentrated distributions. The topology characteristics are determined by the angle and radial generation, which are as follows:

- T1: uniform angle, centralized radial
- T2: uniform angle, uniform radial
- T3: nonuniform angle, centralized radial
- T4: nonuniform angle, uniform radial

The resulting topological layouts are illustrated in Fig. 1. We note that the T2 method resembles random placement. These are available for download at MATLAB file exchange [40].

4.2. Classic benchmark functions

We tested for the existence of trends in Θ on nine classic benchmark functions as well. All the selected functions used have n -dimensional definitions, some being unimodal, while others multimodal. For each of these functions, we have tested with $d \in \{10, 20, 30, \dots, 1000\}$. In the following, we will briefly present these functions. The equations for these functions are presented in Appendix A.

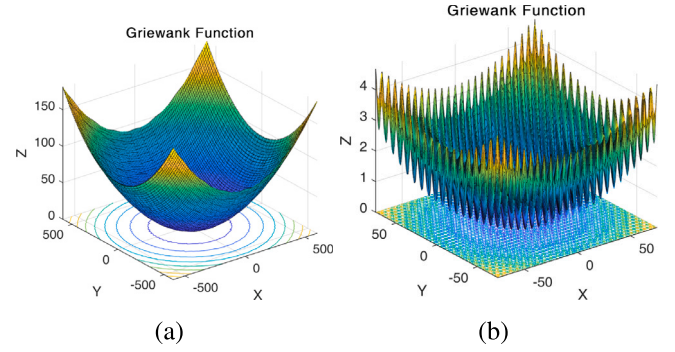


Fig. 3. Griewank function with (a) typical definition range, (b) restricted range (magnified center).

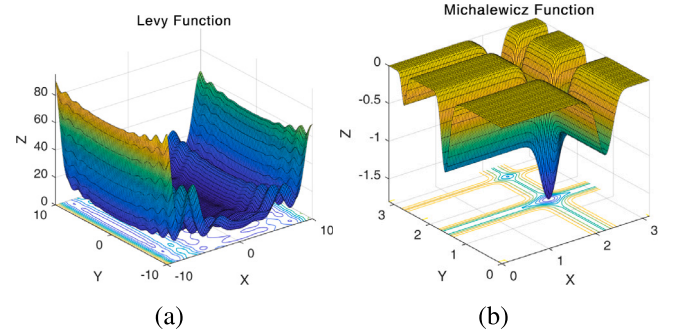


Fig. 4. Lévy (a), and Michalewicz (b) functions.

The Ackley function [41] is multimodal with many local optima. The global minimum is $f(x^*) = 0$, for $x^* = \{0, 0, \dots, 0\}$. The function is typically defined on the hypercube $x_i \in [-32.768, 32.768]$ for all $i \in \{1, 2, \dots, d\}$, although it is often restricted to a smaller definition range, such as $[-5, 5]$. Fig. 2 illustrates the Ackley function in 2D at its typical evaluation range as well as its magnified center (restricted range $[-5, 5]$).

On a small scale, the Griewank function [41] has many regularly distributed local minima. Viewed on a larger scale, its 2D image forms a bowl shape. Its typical definition range is on a hypercube $x_i \in [-32.768, 32.768]$ for all $i \in \{1, 2, \dots, d\}$, with its global minimum at $f(x^*) = 0$, for $x^* = \{0, 0, \dots, 0\}$. Fig. 3 presents this function's 2D surface at its typical definition range, and on its restricted range.

The Lévy function [41] is yet another non-convex function with many local optima, illustrated in Fig. 4(a). Its typical definition range is on a hypercube $x_i \in [-10, 10]$ for all $i \in \{1, 2, \dots, d\}$, with its global minimum at $f(x^*) = 0$, for $x^* = \{1, 1, \dots, 1\}$.

The Michalewicz function [41] is known for its high-dimensional landscape, which includes only a few minima but with very steep variations. Illustrated in Fig. 4(b), steep valleys are observable on its 2D image. The function is typically evaluated on a hypercube $x_i \in [0, \pi]$ for all $i \in \{1, 2, \dots, d\}$. The global minimum location is not fixed, and its value varies with d . For $d = 2$, $f(x^*) \approx -1.8013$ at $x^* = (2.20, 1.57)$. This is the only test function for which the global minimum value is not known throughout the test dimensions.

Similar to the Griewank function, the Rastrigin function [41] is characterized by many regularly distributed local minima. Overall, the 2D function image (Fig. 5(a)) resembles a rugged terrain with valleys separated by steep ridges, and is characterized by a low curvature towards the center, which contains its global optimum value. It is typically evaluated on a hypercube $x_i \in [-5.12, 5.12]$ for all $i \in \{1, 2, \dots, d\}$, with its global minimum at $f(x^*) = 0$, for $x^* = \{0, 0, \dots, 0\}$.

The Rosenbrock function [41] is also referred to as Rosenbrock's saddle, Valley or Banana function, due to its curved valley surrounded

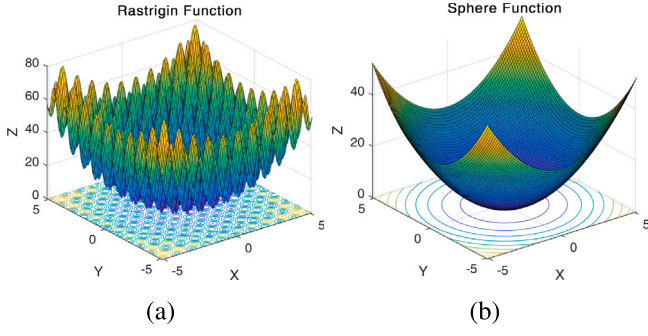


Fig. 5. Rastrigin (a), and Sphere (b) functions.

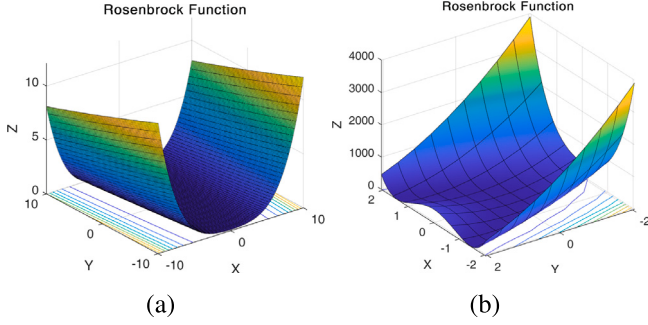


Fig. 6. Rosenbrock function with (a) typical definition range, (b) restricted range (magnified center).

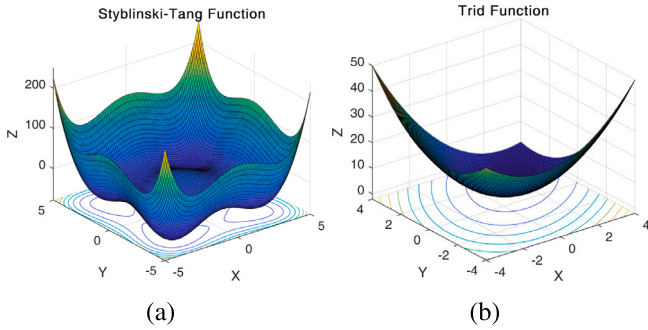


Fig. 7. Styblinski-Tang (a), Trid (b) functions.

by a plateau in its 2D image, as illustrated in Fig. 6(b). It is typically evaluated on a hypercube $x_i \in [-5, 10]$ for all $i \in \{1, 2, \dots, d\}$, or restricted to $x_i \in [-2.048, 2.048]$ for a narrower search space. Its global minimum is located at $f(x^*) = 0$, for $x^* = \{1, 1, \dots, 1\}$.

The Sphere function [41] is continuous, convex, and unimodal. Its 2D image (Fig. 5(b)) has a bowl-shaped surface. Typically evaluated on a hypercube $x_i \in [-5.12, 5.12]$ for all $i \in \{1, 2, \dots, d\}$, its single minimum is located at $f(x^*) = 0$, for $x^* = \{0, 0, \dots, 0\}$.

The Styblinski-Tang function [41] is non-convex and multimodal. Its 2D form, illustrated in Fig. 7(a), has a bowl-shaped surface with a small center hill. The definition range is a hypercube $x_i \in [-5, 5]$ for all $i \in \{1, 2, \dots, d\}$. Its global minimum value is d dependent, $f(x^*) = -39.16599 \cdot d$, at $x^* = \{-2.903534, -2.903534, \dots, -2.903534\}$.

Trid [41] is a unimodal function, with its definition range being d dependent, namely, $x_i \in [-d^2, d^2]$. Its 2D surface, illustrated in Fig. 7(b), is bowl-shaped. Its single global minimum $f(x^*) = -d(d+4)(d-1)/6$ is located at $x_i = i(d+1-i)$, for all $i = 1, 2, \dots, d$. With the increase of d , the scale of the search space also increases. We chose this function as it puts greater strain on the SI algorithms' ability to explore larger and larger spaces as d increases.

4.3. SI algorithms

For testing, we used 14 SI algorithms selected from a pool of 51 publicly available MATLAB implementations. Some of these algorithms were chosen because they are well-established methods (ABC, GWO, and PSO), while others were selected based on a preliminary evaluation of their performance on the Rastrigin function. Our aim was to include a diverse set of well-performing algorithms capable of finding feasible solutions. Therefore, we excluded algorithms that frequently stalled or could not reach solutions close to the global optimum of this function, defined as a best-cost value below 0.1. We assumed that if an algorithm cannot find a sufficiently good solution on the Rastrigin function, it is likely to perform worse on the WSN test cases.

During the preliminary selection, 19 algorithms were considered. However, in the WSN scenarios, five additional algorithms consistently stalled even at lower problem dimensions and were therefore excluded from further evaluation. Retaining these algorithms would have resulted in their early elimination at lower dimensions, and no meaningful analysis of their C curves would have been possible due to very early and frequent stalls (often after only a few iterations) or the inability to produce even a single feasible solution. In the end, 14 algorithms were retained for the final evaluation.

During the selection process, there were conflicting acronyms, for which we renamed some of these algorithms. The methodology for handling conflicting acronyms was to retain the name of the first published algorithm and rename the subsequent ones. We will indicate the algorithms affected by renaming. The algorithms used in our testing are as follows (in alphabetical order):

- ABC (2005) [42] - Artificial Bee Colony Optimization Algorithm
- AEO (2019) [35] - Artificial Ecosystem-based Optimization
- AHA (2022) [43] - Artificial Hummingbird Algorithm
- AVOA (2021) [44] - African Vultures Optimization Algorithm
- BAN (2015) [45] - Novel Bat Algorithm (renamed from NBA)
- BES (2019) [46] - Bald Eagle Search algorithm
- BSA (2015) [47] - Bird Swarm Algorithm
- CSO (2014) [48] - Chicken Swarm Optimization
- CoopSA (2021) [49] - Cooperation Search Algorithm (renamed from CSA)
- GTO (2021) [50] - Artificial Gorilla Troops Optimizer
- GWO (2014) [51] - Gray Wolf Optimizer
- MVO (2015) [52] - Multi-Verse Optimizer
- PSO (1995) [20] - Particle swarm optimization
- SpSA (2020) [53] - Sparrow Search Algorithm (renamed from SSA)

For the WSN scenarios, we ran each algorithm 100 times, with a maximum iteration count of 500. For the classic test functions, we reduced the maximum iteration count to 100, as all algorithms perform significantly better on these functions, and 100 iterations are sufficient for all of them to reach their κ^* . In these scenarios, we only ran each algorithm 50 times. Lower and upper bounds were set according to the bounds of each search space, as specified earlier. In all test cases, we used a Pop population of 30. Result curves of each run were collected and stored for later analysis. Stochastic variation affects individual run outcomes; however, performance trends are extracted from aggregated convergence curves computed over multiple independent runs under a fixed algorithm-problem-configuration setup.

For PSO we used the accelerator coefficients $c_1 = c_2 = 1.5$, inertia weight $w = 0.7$, and inertia dampening ratio $w_{damp} = 1$. The rest of the algorithms were run with their out-of-the-box configuration. Appendix B presents the parameters used for each SI algorithm.

Table 1
SI Algorithm *FFEval*s.

SI Algo.	No. of <i>FFEval</i> s
ABC	$Pop \cdot MaxIter$
AEO	$Pop + 2 \cdot Pop \cdot MaxIter$
AHA	$Pop + Pop \cdot MaxIter$
AVOA	$Pop \cdot MaxIter$
BAN	$Pop \cdot MaxIter$
BES	$Pop + 3 \cdot Pop \cdot (MaxIter - 1)$
BSA	$Pop + Pop \cdot MaxIter$
CoopSA	$Pop \cdot MaxIter$
CSO	$Pop + Pop \cdot MaxIter$
GTO	$Pop + 2 \cdot Pop \cdot MaxIter$
GWO	$Pop \cdot MaxIter$
MVO	$Pop \cdot MaxIter$
PSO	$Pop + Pop \cdot MaxIter$
SpSA	$Pop + 2 \cdot Pop \cdot MaxIter$

4.4. SI algorithm computational cost

PSO can be considered the first SI algorithm designed for continuous optimization problems. Arguably, many SI algorithms draw inspiration from the algorithmic structure of PSO. PSO is iteration-based: in each iteration, the position of each agent is updated and its new fitness value is recalculated. With few exceptions, most algorithms' structures follow this basic idea. The computational cost of such an algorithm in any given iteration is composed of two parts: (a) the algorithmic overhead, and (b) the cost of *FFEval*. Part (a) reflects the operations the algorithm performs: computing agents' positions, agent selection in some algorithms, and so on. The computational cost of this part per iteration depends on the population size. Part (b), while constant for a given topology and dimensionality, varies between algorithms in how many times *FFEval* is invoked per individual in each iteration. Table 1 presents the number of *FFEval* calls used by the algorithms investigated in this paper. Here, *Pop* denotes the number of agents and *MaxIter* denotes the number of iterations the algorithm runs. In real-life cases, the computational cost of *FFEval* far exceeds that of the algorithmic overhead.

4.5. Convergence curves and knee-point calculation

Let C be the discrete convergence curve, represented by the sequence of performance values C_i achieved by the SI algorithm at iteration i , where $i \in \{0, 1, 2, \dots, MaxIter\}$. As mentioned in Section 2.2, for most SI implementations and minimization problems, $C_i \geq C_j$ for all $i < j$, meaning the best result so far is always retained (elitism), so the curve is monotone non-increasing. The algorithms used in this paper all apply elitism, and all test problems are minimization problems, so every convergence curve considered here is of this type. Because such curves exhibit no oscillation or spurious local minima, the geometric criterion reduces to selecting the single point of maximum perpendicular distance from the chord connecting the first and last points. This point is uniquely defined even for curves with extended plateaus or several successive bends — it returns the global cost/benefit balance point rather than any local bend. Runs that do not converge are handled by the stall-point procedure and are not assigned a knee-point. We adopt this deterministic, parameter-free geometric criterion deliberately: on the restricted class of monotone elitist curves considered here it yields a unique, well-defined operating point. A broader comparison of knee-point detection methods is outside the scope of this work. Fig. 8 shows a graphical example of such curves.

Randomness influences the performance curve. As shown in Fig. 8, the initial random placement causes the performance curves to start at different values; however, after a few iterations, these stabilize. We consider these deviations as a transient component of the performance curves. For a more stable knee-point calculation, we cut these off. We will refer to this point as C_ω .

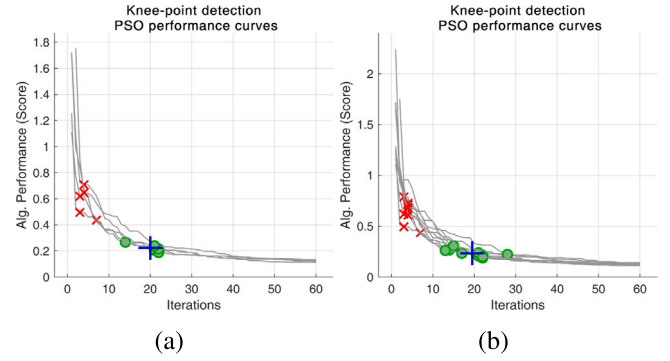


Fig. 8. Knee-point calculation example for PSO performance curves with (a) 5 curves, (b) 10 curves. Red crosses mark the transient cut-off points, green circles the individual computed knee-points, and blue plus sign the overall knee-point location.

The C_ω stall-point is the earliest iteration ω at which the convergence curve improves by less than a threshold T_{cutoff} over the remaining iterations, i.e. $C_\omega - C_{MaxIter} \leq T_{cutoff}$. Beyond C_ω the algorithm's performance does not improve significantly. In real-world applications where *FFEval* is expensive, the algorithm would be terminated as soon as the stall condition is detected; in the experiments reported here we instead ran each algorithm to *MaxIter* and identified the stall-point in post-processing, which lets us reuse the same recorded curves to simulate different stall-detection settings.

With these two cuts on C , we exclude the transient start and the converged tail region from influencing the knee-point calculation result.

We determine the knee-point of a C using a well-known geometric heuristic [25,54,55], by identifying the maximum perpendicular distance from a straight line connecting C_α and C_ω , namely, the starting point of the curve and its cut-off point.

Let $\mathbf{p}_x = (x, y_x); x \in \{1 \dots N\}$ points of the cut C curve, where $\mathbf{p}_1 = (1, y_1)$ and $\mathbf{p}_N = (N, y_N)$ denote the first and last points of the C curve. We define the line vector $\mathbf{L} = \mathbf{p}_N - \mathbf{p}_1$ and its unit vector:

$$\hat{\mathbf{L}} = \frac{\mathbf{L}}{\|\mathbf{L}\|} \quad (10)$$

For each point $\mathbf{p}_i = (i, y_i)$, we compute the vector from the first point:

$$\mathbf{v}_i = \mathbf{p}_i - \mathbf{p}_1 \quad (11)$$

The perpendicular distance from $\mathbf{p}_i$ to the line connecting $\mathbf{p}_1$ and $\mathbf{p}_N$ is obtained by decomposing $\mathbf{v}_i$ into parallel and orthogonal components with respect to $\hat{\mathbf{L}}$:

$$d_i = \left\| \mathbf{v}_i - (\mathbf{v}_i \cdot \hat{\mathbf{L}}) \hat{\mathbf{L}} \right\| \quad (12)$$

The knee-point index k^* is defined as:

$$k^* = \arg \max_{i \in \{1, \dots, N\}} d_i \quad (13)$$

4.6. Practical aspects of knee-point calculation

This section clarifies when and how the knee-point (κ^*) is computed, and quantifies its computational overhead.

4.6.1. When is the knee-point calculated?

In this work, the knee-point is determined as a post-processing step, computed on the convergence curve after the transient cut and stall-point detection have been applied. In real-world deployments, the recommended workflow is:

1. For each $\alpha \in \mathcal{A}$, run until the stall condition is met,
2. Record the convergence curve $C_\alpha = \{C_{\alpha,1}, \dots, C_{\alpha,\omega}\}$,
3. Apply transient and tail truncation to C_α (Section 4.5),
4. Compute the knee-point κ_α^* on the truncated curve.

In the experiments reported here, each $\alpha \in \mathcal{A}$ was instead run to *MaxIter* and the full curve recorded, with the stall-point and knee-point then identified in post-processing.

4.6.2. Computational cost analysis

The additional computational cost of the knee-point calculation is negligible compared to the cost of the optimization process itself.

The geometric method requires a single pass over the convergence curve of length N , with cost $\mathcal{O}(N)$. In contrast, the cost of running an SI algorithm is dominated by fitness function evaluations: from Table 1, this cost is at least $\text{Pop} \cdot \text{IterNr} \cdot \text{Cost}_{\text{FFEval}}$, where *IterNr* denotes the number of iterations actually performed per run — equal to the configured *MaxIter* in the experiments reported here, or to the stall-point iteration ω in a real-world deployment with online stall detection.

Since in real-world problems $\text{Cost}_{\text{FFEval}}$ is typically orders of magnitude larger than simple arithmetic operations, the overhead of knee-point computation is effectively negligible:

$$\frac{\mathcal{O}(N)}{\mathcal{O}(\text{Pop} \cdot \text{IterNr} \cdot \text{Cost}_{\text{FFEval}})} \approx 0 \quad (14)$$

The proposed selection strategy introduces no additional fitness evaluations beyond those required for standard algorithm execution. The only overhead is the post-processing step, which is computationally trivial.

In this work, we compute the knee-point after running the algorithm until the stall point to ensure stable and comparable measurements across algorithms. However, in real-world scenarios where computational resources are limited, the knee-point can also be estimated *online* using alternative methods, such as: (i) stagnation detection: identifying when improvement falls below a threshold over a sliding window; (ii) curvature-based methods: approximating second-order changes in the convergence curve; (iii) regression-based approaches: fitting models to partial convergence data; (iv) adaptive stopping rules: terminating when marginal improvement becomes negligible. These approaches allow early estimation of the knee-point without requiring full convergence, trading off accuracy for reduced computational cost.

4.7. Curve fitting and interpretation

The characteristic shape of Θ_α is determined using curve fitting. We used MATLAB's built-in curve fitting function to determine the best fit among three candidate types $\hat{y}$, as shown in Eq. (2): Eq. (2.1) is a *linear* fit, Eq. (2.2) is a *combined* exponential and linear fit, and Eq. (2.3) is a *power* fit. The *combined* form is intended for cases in which, after the exponential component flattens out, the curve continues with a slight linear increase.

We prefer the simplest fitting over the others if the percentage difference between them is less than 1%, namely *linear* first, *power* second, and *combined* fitting the last. For goodness-of-fit (R^2), we chose a threshold of 0.7, above which we consider that a specific type of curve can be fitted onto the data.

In cases where the rate of data change with the dimensionality increase is low, the fluctuations in the individual run results caused by the *RNG* are large enough that the fitting results in a low R^2 , despite the evident trend of the data. We will provide the fit curve plots to illustrate this phenomenon later in the paper.

Despite the different nature of the three fits, given the limited number of data points in WSN topologies, there are cases in which all three fit well, with high R^2 . Fig. 9(a) presents such a scenario.

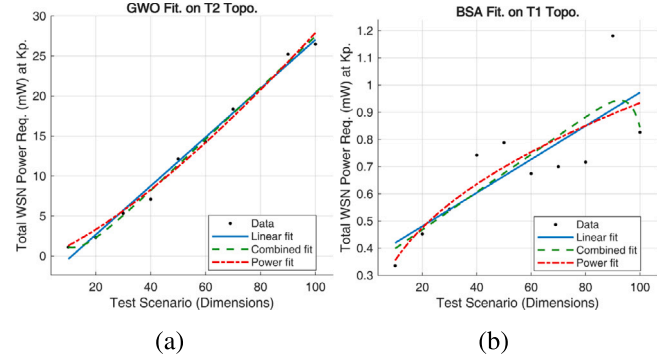


Fig. 9. Example of fits. (a) close fits, (b) invalid combined fit.

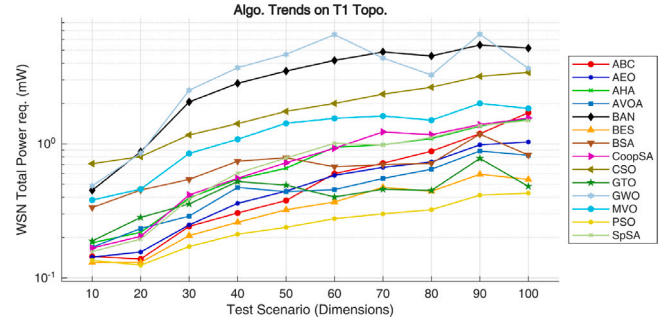


Fig. 10. WSN T1 topology algorithm performance trends from 100 runs, with 10, 20, ..., 100 sensor nodes (dimensions)

Due to the limited number of data points in WSN scenarios, the *combined* fit can produce results with extreme curvature at either end of the curve. Fig. 9(b) presents such a result. While this is technically a valid solution, it does not represent a valid performance trend; therefore, we eliminate such results. Such eliminations will be indicated by crossed-out R^2 values when presenting these results, and we select the next-best fit.

5. Results

Next, we present results for WSN topologies and nine well-known classic benchmark functions. We will use the raw knee-point performance values $\mathcal{E}$ without smoothing. For the quality of the fits, we present their R^2 values and the root mean square error *RMSE*. The trend fits and figures throughout this section are computed from 100-run averages, which provide stable curves for characterizing the trend types; the selection process, however, operates under the more realistic five-run regime, since this is the number of runs typically afforded when fitness evaluations are costly. To show the data the selection actually acted on, we additionally report, for each WSN topology, the best-performing algorithms at d_{target} over five runs.

5.1. WSN

The performance of search algorithms, as presented in previous works [15,21], is influenced by the arrangement of sensor nodes in the WSN. Analyzing the WSN simulation results, we observe similar performance across the investigated algorithms.

Fig. 10 presents the trends in Θ for the T1 topology. In the lower dimensions, BES and PSO lead, but as we climb to higher dimensions, BES is overtaken by GTO, while PSO maintains its lead. The six best-performing algorithms towards the end, over the 100-run averages, are PSO (0.42 mW), GTO* (0.48 mW), BES (0.54 mW), AVOA (0.82 mW), BSA (0.82 mW), and AEO (1.03 mW); the asterisk marks GTO, which

Table 2
Quality of fits on WSN T1 Topology.

SI Algo.	Linear	Combined	Power
	R^2	R^2	R^2
	$RMSE$	$RMSE$	$RMSE$
ABC	0.8800 1.8735e-01	0.9952 4.3043e-02	0.9656 1.0026e-01
AEO	0.9774 5.1161e-02	0.9895 4.0202e-02	0.9840 4.3024e-02
AHA	0.9853 6.0036e-02	0.9903 5.6376e-02	0.9880 5.4293e-02
AVOA	0.9299 6.6658e-02	0.9358 7.3652e-02	0.9212 7.0680e-02
BAN	0.9307 4.9718e-01	0.9840 2.6731e-01	0.9443 4.4577e-01
BES	0.9570 3.6388e-02	0.9672 3.6666e-02	0.9543 3.7482e-02
BSA	0.6547 1.4339e-01	0.7066 1.5262e-01	0.6854 1.3687e-01
CoopSA	0.9829 6.8852e-02	0.9858 7.2491e-02	0.9828 6.8971e-02
CSO	0.9896 1.0286e-01	0.9962 7.1194e-02	0.9823 1.3376e-01
GTO	0.5383 1.1346e-01	0.6321 1.1695e-01	0.5865 1.0736e-01
GWO	0.4676 1.5829e+00		0.5962 1.4738e+00
MVO	0.9059 1.8106e-01	0.9517 1.4980e-01	0.9318 1.5415e-01
PSO	0.9698 1.9676e-02	0.9824 1.7348e-02	0.9429 2.7065e-02
SpSA	0.9805 6.9270e-02	0.9815 7.7865e-02	0.9799 7.0268e-02

crashed on one of the 100 runs — terminating before reaching *MaxIter* without producing a result. This differs from a run that completes but yields a non-viable (infinite) result, as seen for other algorithms below. Under five runs — the regime in which the selection process operates — the six best-performing algorithms are PSO (0.39 mW), GTO (0.45 mW), BES (0.56 mW), BSA (0.68 mW), AEO (0.88 mW), and SpSA (0.88 mW).

Table 2 presents the quality of the fits for each type of fitted curve on the T1 topology.

Regarding the types of best-fit curves to Θ , AVOA, CoopSA, CSO, and SpSA show a *linear* trend. The *combined* fit type fits the best for ABC, AEO, AHA, BAN, MVO, and PSO, while BSA barely hits the 0.7 R^2 threshold. For GTO and GWO, none of the trends fit. In the case of GWO, although the *combined* fit crossed the 0.7 threshold, we consider it invalid due to its bell-shaped end, and neither the *linear* nor the *power* fit crossed the threshold.

Fig. 11 presents the trends in Θ for the T2 topology. Compared to T1, the overall power requirements are roughly threefold higher and the spread between best and worst is considerably wider; GTO drops from second to third, and GWO and MVO degrade sharply, with GWO rising from 3.65 to 26.49 mW. GWO and BAN remain among the worst-performing algorithms. In the lower dimensions, CoopSA and BES lead, but as we climb to higher dimensions, these are overtaken by AVOA and PSO. The six best-performing algorithms towards the end, over the 100-run averages, are PSO* (1.30 mW), AVOA (1.31 mW), GTO (1.68 mW), BES (1.70 mW), AEO (1.97 mW), and CoopSA (2.27 mW). Under five runs, the six best-performing algorithms are PSO* (1.17 mW), AVOA (1.28 mW), BES (1.30 mW), GTO (1.68 mW), AEO (1.92 mW), and CoopSA (2.25 mW); PSO* produced non-viable results where the sensor network could not be fully connected (1 out of the 5, and 8 out of the 100 runs).

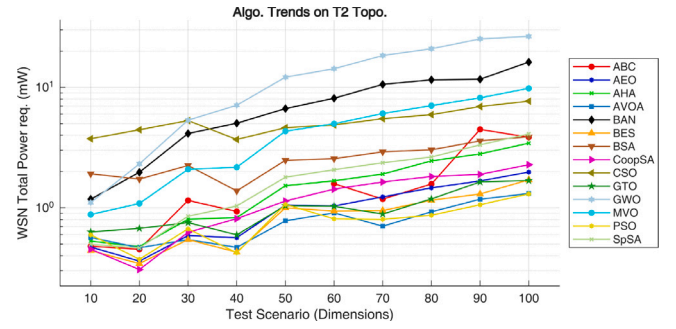


Fig. 11. WSN T2 topology algorithm performance trends from 100 runs, with 10, 20, ..., 100 sensor nodes (dimensions)

Table 3
Quality of fits on WSN T2 Topology.

SI Algo.	Linear	Combined	Power
	R^2	R^2	R^2
	$RMSE$	$RMSE$	$RMSE$
ABC	0.6884 8.5826e-01	0.8101 7.9275e-01	0.7954 7.5112e-01
AEO	0.9463 1.3467e-01		0.9421 1.3986e-01
AHA	0.9558 2.2637e-01	0.9865 1.4427e-01	0.9715 1.8169e-01
AVOA	0.8079 1.3713e-01	0.8865 1.2172e-01	0.8255 1.3972e-01
BAN	0.9737 8.2633e-01	0.9875 6.5656e-01	0.9767 7.7717e-01
BES	0.8852 1.5879e-01	0.9241 1.4908e-01	0.8708 1.6848e-01
BSA	0.8145 3.6578e-01	0.8883 3.2773e-01	0.7734 4.3226e-01
CoopSA	0.9734 1.1673e-01		0.9729 1.1780e-01
CSO	0.7615 6.7153e-01	0.8777 5.5512e-01	0.6431 8.2135e-01
GTO	0.8079 1.8277e-01	0.8916 1.5855e-01	0.7383 2.1334e-01
GWO	0.9900 9.8289e-01	0.9946 8.3435e-01	0.9903 9.6604e-01
MVO	0.9770 5.0001e-01	0.9905 3.6985e-01	0.9884 3.5415e-01
PSO	0.6607 1.8182e-01	0.7082 1.9472e-01	0.6047 1.9626e-01
SpSA	0.9642 2.4651e-01	0.9858 1.7929e-01	0.9785 1.9099e-01

Table 3 presents the goodness-of-fit on the T2 topology. Here, AEO and CoopSA show a *linear* trend for Θ , while for all the other algorithms the *combined* fits best. We note that for PSO the R^2 barely crosses the 0.7 threshold, indicating a higher noise in the data.

Fig. 12 presents the trends in Θ for the T3 topology. The behavior of the algorithms on this topology is very similar to T1: PSO, BES, and ABC lead in the beginning, and towards the end, GTO catches up to BES. The six best-performing algorithms towards the end, over the 100-run averages, are PSO (0.51 mW), GTO (0.78 mW), BES (0.81 mW), AVOA (0.89 mW), AEO (1.19 mW), and BSA (1.34 mW). Under five runs, the six best-performing algorithms are PSO (0.53 mW), AVOA (0.65 mW), GTO (0.76 mW), BES (0.93 mW), AEO (1.07 mW), and ABC (1.18 mW).

Table 4 presents the goodness-of-fit on the T3 topology. Here, except CSO, for which the *linear* trend fits the best, all have the *combined* type best fit their Θ curves.

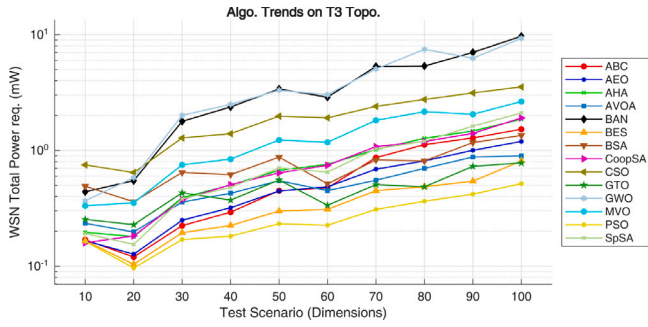


Fig. 12. WSN T3 topology algorithm performance trends from 100 runs, with 10, 20, ..., 100 sensor nodes (dimensions)

Table 4
Quality of fits on WSN T3 Topology.

SI Algo.	Linear	Combined	Power
	R^2	R^2	R^2
	$RMSE$	$RMSE$	$RMSE$
ABC	0.9198 1.5257e-01	0.9874 6.9947e-02	0.9765 8.2553e-02
AEO	0.9505 8.5842e-02	0.9939 3.4773e-02	0.9790 5.5873e-02
AHA	0.9578 1.2263e-01	0.9948 4.9791e-02	0.9842 7.5072e-02
AVOA	0.9251 7.0318e-02	0.9401 7.2638e-02	0.9091 7.7468e-02
BAN	0.9219 8.7660e-01	0.9783 5.3299e-01	0.9571 6.4939e-01
BES	0.8844 7.7496e-02	0.9718 4.4177e-02	0.9022 7.1295e-02
BSA	0.7399 1.6689e-01	0.8388 1.5168e-01	0.7972 1.5755e-01
CoopSA	0.9505 1.3394e-01	0.9916 6.3586e-02	0.9777 8.9942e-02
CSO	0.9760 1.6110e-01		0.9702 1.7958e-01
GTO	0.7484 9.8140e-02	0.8019 1.0056e-01	0.7243 1.0983e-01
GWO	0.9213 8.8004e-01	0.9419 8.7343e-01	0.9392 7.7375e-01
MVO	0.9601 1.6893e-01	0.9671 1.7709e-01	0.9640 1.6047e-01
PSO	0.8927 4.5702e-02	0.9719 2.7006e-02	0.8869 5.0166e-02
SpSA	0.9111 2.0173e-01	0.9890 8.2093e-02	0.9649 1.2679e-01

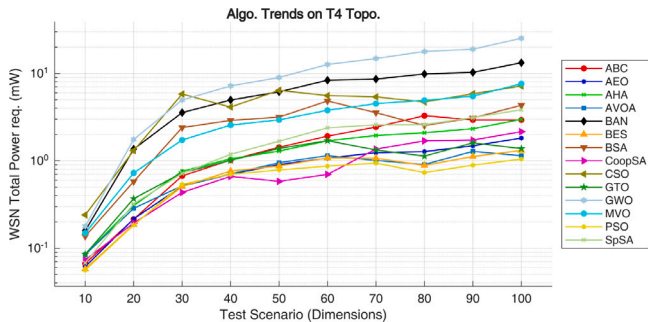


Fig. 13. WSN T4 topology algorithm performance trends from 100 runs, with 10, 20, ..., 100 sensor nodes (dimensions)

Table 5
Quality of fits on WSN T4 Topology.

SI Algo.	Linear	Combined	Power
	R^2	R^2	R^2
	$RMSE$	$RMSE$	$RMSE$
ABC	0.9426 3.0055e-01		0.9449 3.1487e-01
AEO	0.9843 7.4166e-02	0.9876 7.6106e-02	0.9814 8.0797e-02
AHA	0.9871 1.1091e-01	0.9916 1.0344e-01	0.9841 1.2327e-01
AVOA	0.8197 1.7825e-01	0.9283 1.2977e-01	0.8581 1.5812e-01
BAN	0.9790 6.4101e-01	0.9835 6.5528e-01	0.9734 7.2109e-01
BES	0.8545 1.6728e-01	0.9356 1.2847e-01	0.8785 1.5284e-01
BSA	0.5517 1.0445e+00	0.7692 8.6536e-01	0.6155 9.6723e-01
CoopSA	0.9373 1.9100e-01	0.9619 1.7193e-01	0.9607 1.5122e-01
CSO	0.5724 1.5401e+00	0.7921 1.2401e+00	0.6450 1.4033e+00
GTO	0.6658 3.2459e-01		0.7325 2.9039e-01
GWO	0.9848 1.0575e+00	0.9957 6.5018e-01	0.9874 9.6203e-01
MVO	0.9735 3.9484e-01	0.9938 2.2119e-01	0.9747 3.8607e-01
PSO	0.7788 1.6226e-01	0.9283 1.0670e-01	0.8307 1.4195e-01
SpSA	0.9807 1.8523e-01	0.9823 2.0487e-01	0.9745 2.1275e-01

Fig. 13 presents the trends in Θ for the T4 topology. The trends are broadly similar in magnitude to T2, though GTO and BES exchange ranks and, under five runs, AVOA rather than PSO is the best performer. CoopSA starts strong and at $d = 50$ and $d = 60$ overtakes the rest, but after $d = 70$ it falls behind, and by $d = 100$ ends up in 6th place. The six best-performing algorithms towards the end, over the 100-run averages, are PSO (1.04 mW), AVOA (1.14 mW), BES (1.31 mW), GTO (1.37 mW), AEO (1.81 mW), and CoopSA (2.15 mW). Under five runs, the six best-performing algorithms are AVOA (1.01 mW), PSO (1.09 mW), BES (1.13 mW), GTO (1.58 mW), AEO (2.13 mW), and CoopSA (2.29 mW). The five-run ordering differs from the 100-run averages, reflecting the higher variance under fewer runs.

Table 5 presents the goodness-of-fit on the T4 topology. Here, a linear fit is best for ABC, AEO, AHA, BAN, and SpSA. GTO has a marginal power fit, the first instance in the WSN topologies where this type of fit is the best. For the other algorithms, the best fit is the combined.

Regarding the overall performances on the WSN topologies, GWO, BAN, CSO, and MVO consistently rank last across all four topologies. However, this is not the case for the best-performing algorithms. PSO, BES, GTO, and AVOA are the best-performing ones. AVOA clearly performs best on T2 and T4, while on T1 and T3 it falls to 4th place. We see a similarity in behavior between T1 – T3 and T2 – T4, indicating that the centralized radial placement of the nodes has a much greater impact on the algorithms' performance than the generation angle.

5.2. Classic benchmark functions

In this section, we investigate whether a trend can be determined for Θ in the case of the classic benchmark functions. We ran the same set of algorithms, but for a much higher dimension count.

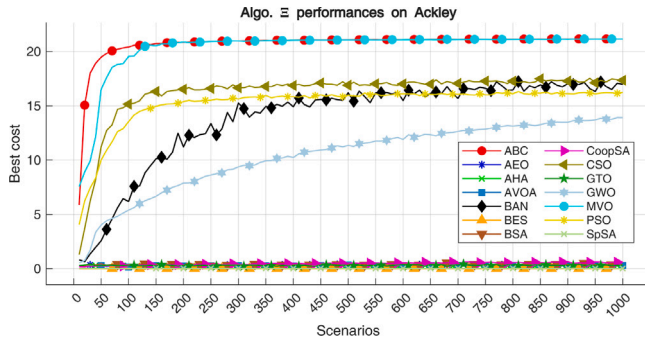


Fig. 14. Algorithm performance on Ackley's function with 10, 20, ..., 1000 dimensions.

Table 6

Trend fit quality on the Ackley function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.1422 1.5586e+00	0.9882 1.2910e-01	0.4115 1.2910e+00
AEO	0.0494 4.1685e-02	0.0815 4.1401e-02	0.0520 4.1628e-02
AHA	0.0438 4.2291e-02	0.0441 4.2722e-02	0.0282 4.2633e-02
AVOA	0.3324 3.3309e-02	0.3840 3.2327e-02	0.3834 3.2011e-02
BAN	0.6599 2.4557e+00	0.9901 4.2408e-01	0.9383 1.0512e+00
BES	0.0846 1.2369e-10	0.0852 1.2494e-10	-0.6220 1.6465e-10
BSA	0.0450 5.5670e-02	0.1251 5.3838e-02	0.0657 5.5064e-02
CoopSA	0.8311 4.6866e-02	0.9336 2.9692e-02	0.9293 3.0315e-02
CSO	0.2913 2.1804e+00	0.9927 2.2343e-01	0.6117 1.6139e+00
GTO	0.0526 4.7638e-02	0.0774 4.7497e-02	0.0331 4.8373e-02
GWO	0.8634 1.1821e+00	0.9952 2.2328e-01	0.9783 4.7107e-01
MVO	0.2333 2.0644e+00	0.9769 3.6238e-01	0.5680 1.5495e+00
PSO	0.3551 1.6972e+00	0.9969 1.1851e-01	0.7014 1.1550e+00
SpSA	0.0212 5.6821e-03	0.0261 5.7266e-03	0.0213 5.6819e-03

Fig. 14 presents the Θ lines for the tested algorithms on the Ackley benchmark function. Here, algorithms ABC, BAN, CoopSA, CSO, GWO, MVO, and PSO fit the *combined* trend. CoopSA exhibits a slight curve, and the *combined* fit shows a better R^2 when compared to the *linear* fit. The others, AEO, AHA, AVOA, BES, BSA, GTO, and SpSA have no valid fits. The best-performing algorithm here is BES, with a ε value of $3.21e-10$, followed by SpSA with a value of 0.023.

Table 6 presents the goodness-of-fit results for the Ackley function. As the trends for AEO, AHA, AVOA, BES, BSA, GTO, and SpSA are nearly flat and linear, they have a performance variation high enough to prevent a good-quality curve fit. The *combined* fit has the best R^2 for the rest of the algorithms.

Fig. 15 presents the Θ data lines for the tested algorithms on the Griewank classic function. The two best-performing algorithms here are BES with a ε value of 0, followed by SpSA with a value of 0.004.

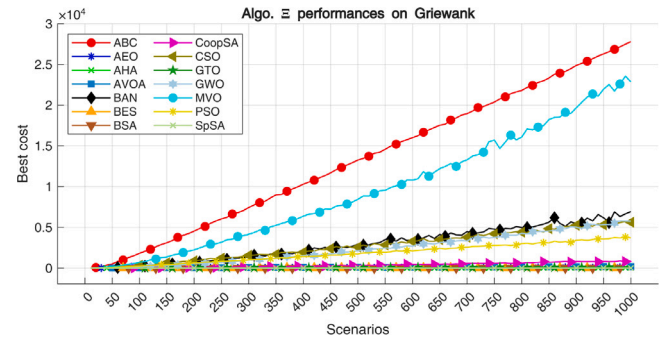


Fig. 15. Algorithm performance on Griewank with 10, 20, ..., 1000 dimensions.

Table 7

Trend fit quality on the Griewank function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9997 1.4951e+02	0.9998 1.1104e+02	0.9999 9.0376e+01
AEO	0.9146 8.1604e+00	0.9148 8.2346e+00	0.9146 8.1608e+00
AHA	0.6873 7.9209e+00	0.7036 7.7907e+00	0.6912 7.8710e+00
AVOA	0.8911 1.9232e+01	0.8957 1.9017e+01	0.8912 1.9221e+01
BAN	0.9880 2.2642e+02	0.9920 1.8697e+02	0.9914 1.9284e+02
BES	0.0355 1.1593e-06	0.0427 1.1670e-06	-0.0000 1.1805e-06
BSA	0.8881 3.2142e+00	0.8886 3.2393e+00	0.8880 3.2156e+00
CoopSA	0.9782 4.0381e+01	0.9915 2.5455e+01	0.9906 2.6528e+01
CSO	0.9966 9.9665e+01	0.9970 9.4585e+01	0.9967 9.8725e+01
GTO	0.7852 1.7362e+01	0.7914 1.7285e+01	0.7850 1.7367e+01
GWO	0.9819 2.4042e+02	0.9988 6.2571e+01	0.9988 6.3227e+01
MVO	0.9813 9.5899e+02	0.9986 2.6554e+02	0.9982 2.9544e+02
PSO	0.9983 4.6385e+01	0.9987 4.1770e+01	0.9987 4.1312e+01
SpSA	0.1633 1.1045e-03	0.1713 1.1106e-03	0.1691 1.1007e-03

In contrast, the next-best-performing one is BSA, with a significantly higher performance value of 33.50, and the distances keep growing.

From the fit qualities presented in Table 7, we observe that ABC, AEO, BSA, CSO, GTO, and PSO follow a *linear* trend, AHA, AVOA, BAN, CoopSA, GWO, and MVO best fit the *combined* fit. The *power* fit also has high R^2 values in the case of some algorithms, which is due to the slight curvature in the Θ lines. BES and SpSA have no valid fits.

Fig. 16 presents the Θ lines for the tested algorithms on the Lévy function. Here, we observe that ABC and GWO have very similar performance at their knee-points. The best-performing algorithm is SpSA, with a ε value of 0.44, followed by BES with a value of 89.09, and by GTO with a value of 90.44. AHA and BSA come close with values of 98.84 and 100.6, respectively.

Table 8 presents the fit qualities on the Θ trend in the case of the Lévy function. The majority of the algorithms' Θ lines best fit a *linear*

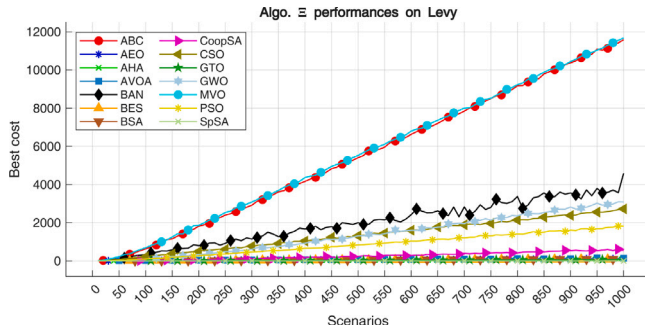


Fig. 16. Algorithm performance on Lévy's function with 10, 20, ..., 1000 dimensions.

Table 8

Trend fit quality on Lévy's function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9992 9.6674e+01	0.9996 7.1341e+01	0.9997 5.7289e+01
AEO	0.9953 2.3315e+00	0.9955 2.3041e+00	0.9953 2.3284e+00
AHA	0.9950 2.0796e+00	0.9950 2.0953e+00	0.9950 2.0807e+00
AVOA	0.9710 7.4441e+00	0.9741 7.1103e+00	0.9710 7.4489e+00
BAN	0.9811 1.5755e+02	0.9817 1.5648e+02	0.9816 1.5618e+02
BES	0.9999 2.6365e-01	1.0000 1.4482e-01	0.9997 4.6235e-01
BSA	0.9990 9.1397e-01	0.9990 9.0859e-01	0.9990 9.1499e-01
CoopSA	0.9862 2.1359e+01	0.9935 1.4819e+01	0.9933 1.4830e+01
CSO	0.9993 2.1433e+01	0.9993 2.1461e+01	0.9993 2.1539e+01
GTO	0.9077 6.9939e+00	0.9152 6.7717e+00	0.9081 6.9769e+00
GWO	0.9921 8.5903e+01	0.9976 4.7710e+01	0.9985 3.7995e+01
MVO	0.9996 7.4905e+01	0.9997 6.1304e+01	0.9998 5.2248e+01
PSO	0.9983 2.2930e+01	0.9991 1.6719e+01	0.9986 2.0409e+01
SpSA	0.6861 7.7146e-02	0.6943 7.6923e-02	0.6906 7.6600e-02

trend. In the case of BES, Θ has a perfect R^2 of 1, so we consider this the best, despite the marginal difference from the *linear* fit. The trends for CoopSA and GTO best follow the *combined* curve, whereas for SpSA, no fit R^2 exceeds the threshold.

Fig. 17 presents the Θ data lines for the tested algorithms on the Michalewicz function. This function is one of the rare ones whose $f^*(d)$ global optimum has a decreasing negative value along d . Here, MVO, ABC, and GWO are the worst-performing algorithms, while CoopSA is the best one. We observe that their performance trends are highly linear with an increasing separation as d increases.

As presented in Table 9, the best fit on the Θ values for all the algorithms is the *linear* type.

As seen in Fig. 18, the Θ lines are reasonably smooth when compared to the scale of their values, and we observe yet again many linear trends. The best-performing algorithm here is again BES, with a Ξ value of 0, followed by SpSA with a value of 21.14, and AHA with a

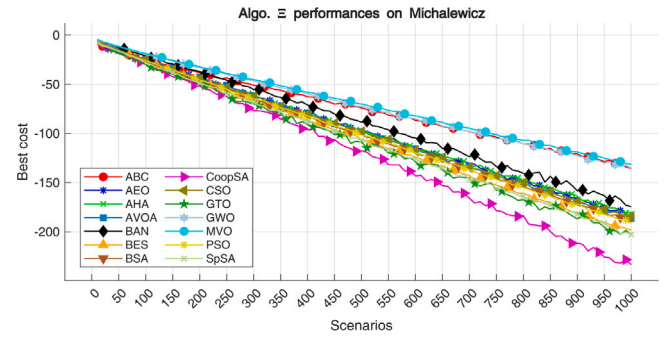


Fig. 17. Algorithm performance on Michalewicz function with 10, 20, ..., 1000 dimensions.

Table 9

Trend fit quality on the Michalewicz function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9988 1.2185e+00	0.9996 7.2017e-01	0.9993 9.0941e-01
AEO	0.9995 1.0966e+00	0.9996 9.9104e-01	0.9996 1.0126e+00
AHA	0.9993 1.3020e+00	0.9996 9.9245e-01	0.9987 1.8052e+00
AVOA	0.9994 1.2732e+00	0.9995 1.1673e+00	0.9995 1.1606e+00
BAN	0.9991 1.4417e+00	0.9991 1.4558e+00	0.9988 1.7335e+00
BES	0.9995 1.2355e+00	0.9999 5.8090e-01	0.9996 1.1546e+00
BSA	0.9992 1.5174e+00	0.9998 6.8755e-01	0.9998 6.7067e-01
CoopSA	0.9992 1.8220e+00	0.9994 1.6272e+00	0.9994 1.6596e+00
CSO	0.9995 1.1576e+00	0.9999 6.4156e-01	0.9995 1.2142e+00
GTO	0.9983 2.3657e+00	0.9988 1.9903e+00	0.9982 2.3992e+00
GWO	0.9992 1.0822e+00	0.9998 4.8789e-01	0.9994 8.9077e-01
MVO	0.9994 9.1131e-01	0.9999 4.0561e-01	0.9999 4.3878e-01
PSO	0.9998 8.2556e-01	0.9999 4.8531e-01	0.9993 1.4452e+00
SpSA	0.9993 1.4605e+00	0.9998 8.9413e-01	0.9994 1.4428e+00

value of 114.59. The worst-performing algorithms here are ABC, closely followed by MVO, then PSO, and GWO.

Table 10 presents the quality of fits in the case of the Rastrigin function. Here, we have *linear* best fit for AEO, AVOA, BAN, CSO, GWO, and PSO. In the case of BES, no trend types fit, and for the rest, the *combined* fits best. However, the qualities of the two types are very close. In the case of ABC and MVO, we have yet again a perfect R^2 of 1.

Fig. 19 presents the Θ lines for the tested algorithms on the Rosenbrock function. With a $f(x^*) = 0$ minimum value, we observe that all of the tested algorithms perform significantly worse here when compared to the other test functions. The best-performing in $d = 1000$ is SpSA with a value of 113.35, followed by BES with a value of 995.54, while the next best one is AHA, with a value of 17585.4.

As seen in Table 11, the nature of Θ is best characterized by the *linear* fit, with BES having an R^2 of 1. The *power* fit is the best in the

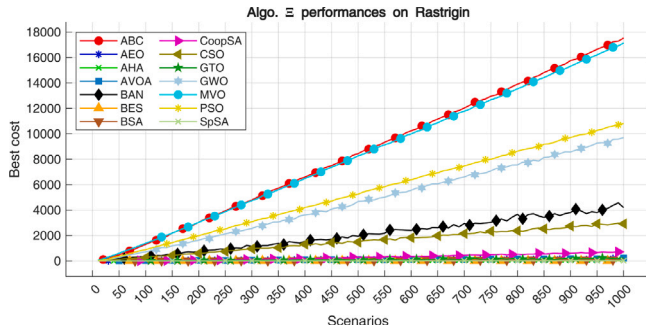


Fig. 18. Algorithm performance on Rastrigin function with 10, 20, ..., 1000 dimensions.

Table 10
Trend fit quality on the Rastrigin function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9998 7.3304e+01	1.0000 3.4799e+01	0.9999 4.2344e+01
AEO	0.9225 1.3217e+01	0.9235 1.3268e+01	0.9225 1.3215e+01
AHA	0.8580 1.4941e+01	0.8607 1.4953e+01	0.8585 1.4996e+01
AVOA	0.9305 1.5821e+01	0.9325 1.5750e+01	0.9316 1.5687e+01
BAN	0.9928 1.0734e+02	0.9942 9.7833e+01	0.9941 9.7486e+01
BES	0.0297 3.3526e-10	-0.0101 3.4561e-10	-0.0101 3.4207e-10
BSA	0.8790 1.1765e+01	0.8895 1.1360e+01	0.8788 1.1778e+01
CoopSA	0.9837 2.8135e+01	0.9962 1.3736e+01	0.9953 1.5044e+01
CSO	0.9969 4.8088e+01	0.9977 4.1793e+01	0.9977 4.1999e+01
GTO	0.8653 2.3609e+01	0.8716 2.3289e+01	0.8660 2.3547e+01
GWO	0.9992 8.1123e+01	0.9997 4.8812e+01	0.9997 4.8633e+01
MVO	0.9998 7.7269e+01	1.0000 1.9784e+01	1.0000 2.4899e+01
PSO	0.9999 2.6942e+01	1.0000 2.2800e+01	0.9999 2.3639e+01
SpSA	0.6962 3.2422e+00	0.7054 3.2259e+00	0.6955 3.2460e+00

case of CoopSA, GWO, and MVO. For GTO and SpSA, no fit crosses the R^2 threshold.

The Θ lines for the classic benchmark function Sphere are illustrated in Fig. 20. The best-performing algorithms here are BES with a Ξ value of $2.75e-20$, followed by SpSA with a value of 0.36. The worst-performing algorithms are ABC and MVO.

In the case of the Sphere function, the quality of fittings over the Θ data is presented in Table 12. Here, the best fits are the *linear* types. CoopSA and marginally GTO show the *combined* best fit, while the fits for BES and SpSA do not cross the threshold.

The Styblinski-Tang function, as illustrated in Fig. 21, is another rare benchmark function whose $f(x^*)$ values are negative and decreasing along d . Here we observe that the algorithms' Θ lines form several groups. The two best-performing algorithms on this function, which show very close performance, are SpSA and GTO, with values of -38691.8 and -38425.7 , respectively, followed closely by AVOA with a

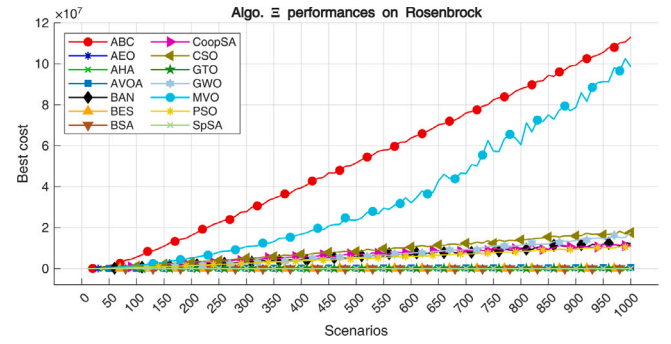


Fig. 19. Algorithm performance on Rosenbrock function with 10, 20, ..., 1000 dimensions.

Table 11
Trend fit quality on the Rosenbrock function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9990 1.0573e+06	0.9990 1.0683e+06	0.9997 6.0514e+05
AEO	0.8338 1.4003e+04	0.8344 1.4121e+04	0.8354 1.4009e+04
AHA	0.4693 1.0725e+04	0.4892 1.0631e+04	0.4729 1.0744e+04
AVOA	0.9241 4.9352e+04	0.9241 4.9863e+04	0.9256 4.9112e+04
BAN	0.9896 3.8238e+05	0.9896 3.8634e+05	0.9898 3.8203e+05
BES	1.0000 3.7781e-01	1.0000 1.6174e-01	1.0000 3.0321e-01
BSA	0.7877 5.0843e+03	0.7877 5.1370e+03	0.7886 5.0997e+03
CoopSA	0.9722 5.8472e+05	0.9766 5.4152e+05	0.9852 4.2918e+05
CSO	0.9972 2.8426e+05	0.9972 2.8720e+05	0.9972 2.8558e+05
GTO	0.5833 2.0635e+04	0.5858 2.0785e+04	0.5833 2.0739e+04
GWO	0.9768 7.5089e+05	0.9768 7.5867e+05	0.9963 2.9937e+05
MVO	0.9296 8.1105e+06	0.9296 8.1946e+06	0.9961 1.9206e+06
PSO	0.9974 1.6727e+05	0.9974 1.6900e+05	0.9989 1.1178e+05
SpSA	0.6793 1.3043e+01	0.6852 1.3056e+01	0.6788 1.3054e+01

value of -37274.5 . The worst-performing algorithms are CoopSA, MVO, and ABC.

Table 13 presents the fit qualities over the Θ values. The majority of fits are of type *linear*, with ABC being the only one with a different best-fit type, namely *power*.

The Trid function is the only one tested here, with its range defined with respect to d^2 , meaning its range increases with d , making the exploration harder and affecting the tested SI algorithms. As illustrated in Fig. 22, we observe large-scale differences in the performances. The best-performing algorithm here is BES, with a Ξ value of 893.46, followed by GTO with a value of $6.80e+10$. ABC and MVO are the two worst-performing ones here.

We present the fit data for the Trid function in Table 14. Besides BES, where the *combined* trend type fits the best, and SpSA, where no fit quality reaches the threshold, all the other algorithms' Θ follow a *power* trend.

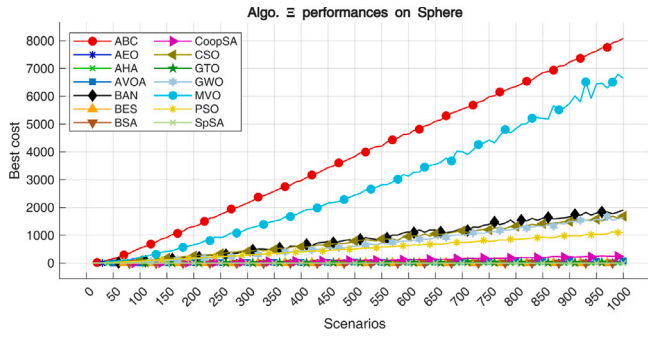


Fig. 20. Algorithm performance on the Sphere function with 10, 20, ..., 1000 dimensions.

Table 12

Trend fit quality on the Sphere function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9997 4.3417e+01	0.9999 2.2829e+01	0.9999 2.9916e+01
AEO	0.8973 2.4209e+00	0.8993 2.4216e+00	0.8978 2.4269e+00
AHA	0.7418 1.7382e+00	0.7524 1.7199e+00	0.7421 1.7371e+00
AVOA	0.9023 6.6038e+00	0.9024 6.6706e+00	0.9022 6.6083e+00
BAN	0.9924 5.1859e+01	0.9952 4.1690e+01	0.9944 4.4455e+01
BES	-0.6495 2.1624e-14	-0.0424 2.2698e-18	-0.0424 2.2465e-18
BSA	0.8950 8.3933e-01	0.8971 8.3945e-01	0.8950 8.3939e-01
CoopSA	0.9776 1.1925e+01	0.9928 6.8071e+00	0.9927 6.8120e+00
CSO	0.9968 2.8851e+01	0.9978 2.3858e+01	0.9978 2.3812e+01
GTO	0.7605 4.3960e+00	0.7848 4.2102e+00	0.7640 4.3864e+00
GWO	0.9828 6.7932e+01	0.9971 2.8315e+01	0.9980 2.3180e+01
MVO	0.9817 2.7681e+02	0.9981 8.9199e+01	0.9981 8.9321e+01
PSO	0.9984 1.3120e+01	0.9988 1.1360e+01	0.9986 1.2110e+01
SpSA	0.5924 1.1735e-01	0.5945 1.1826e-01	0.5941 1.1711e-01

5.3. Performance variation stability

The variability scale is an important factor to consider. When there is significant variation, it becomes difficult to distinguish the performance forecast of different algorithms. As dimensionality increases, SI algorithms are generally expected to perform worse, meaning that the knee-point performance value is expected to worsen with the increase in d . A useful metric for assessing the performance stability of κ^* is the coefficient of variation (CV), as it relates the variation of an algorithm to its performance.

Since some of the classic functions can have negative fitness values, we calculate CV with the absolute mean value: $CV = \sigma/|\mu|$. In the following, we present some of the CV variation measurements for the topologies discussed. We illustrate the changes in CV values graphically.

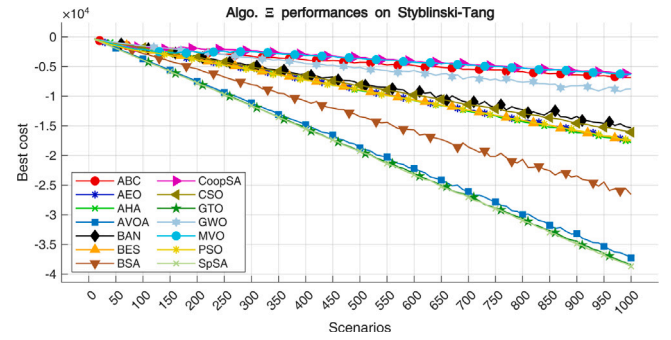


Fig. 21. Algorithm performance on the Styblinski-Tang function with 10, 20, ..., 1000 dimensions.

Table 13

Trend fit quality on the Styblinski-Tang function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.9836 2.1820e+02	0.9873 1.9429e+02	0.9977 8.2938e+01
AEO	0.9998 6.4336e+01	0.9999 6.1427e+01	0.9998 6.2591e+01
AHA	0.9999 4.9264e+01	0.9999 4.9660e+01	0.9999 4.9505e+01
AVOA	0.9999 1.1604e+02	0.9999 1.1619e+02	0.9999 1.1570e+02
BAN	0.9979 2.0095e+02	0.9980 1.9956e+02	0.9980 1.9837e+02
BES	0.9999 4.5162e+01	0.9999 4.3818e+01	0.9999 4.3932e+01
BSA	0.9991 2.2535e+02	0.9992 2.2088e+02	0.9992 2.2148e+02
CoopSA	0.9916 1.4121e+02	0.9921 1.3883e+02	0.9748 2.4445e+02
CSO	0.9999 4.7943e+01	0.9999 4.3255e+01	0.9999 4.1327e+01
GTO	1.0000 7.1198e+01	1.0000 7.1580e+01	1.0000 7.1543e+01
GWO	0.9924 2.1690e+02	0.9955 1.6904e+02	0.9967 1.4470e+02
MVO	0.9706 2.5353e+02	0.9718 2.5123e+02	0.9707 2.5451e+02
PSO	0.9997 8.3287e+01	0.9998 6.7978e+01	0.9997 8.3569e+01
SpSA	1.0000 6.0561e+01	1.0000 6.0183e+01	1.0000 6.0355e+01

On the WSN topologies (Fig. 23), we observe that the better-performing algorithms' CV curve is relatively stable. ABC is the worst-performing algorithm on T2, where, in a couple of scenarios, it fails to find viable solutions [15], and its curve shows high variability in these instances. On the other topologies, ABC performs better and has a much more stable curve. Overall, we observe that the CV of the investigated algorithms is bounded. Only the CV curve of GWO on the T1 topology appears to be increasing as it approaches an upper bound.

Next, we will present the changes to CV on some of the classic test functions.

As illustrated in Fig. 24 for Lévy and Rosenbrock functions, the CV lines are bounded; some have smaller, while others have higher variations. We observed the same for the Griewank function.

Fig. 25 illustrates the CV evolution along d in the cases of the Ackley and Michalewicz functions. On the Ackley function, a few algorithms'

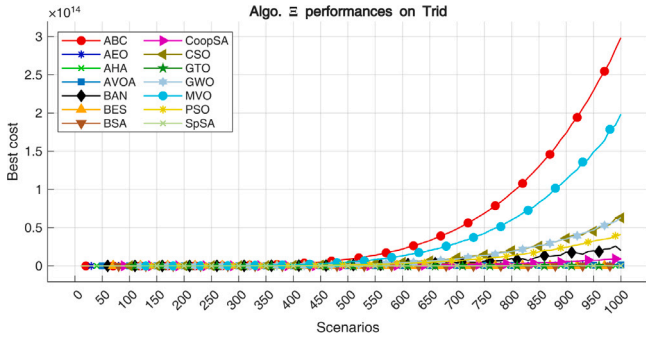


Fig. 22. Algorithm performance on Trid function with 10, 20, ..., 1000 dimensions.

Table 14
Trend fit quality on the Trid function.

SI Algo.	Linear	Combined	Power
	R^2 $RMSE$	R^2 $RMSE$	R^2 $RMSE$
ABC	0.5223	0.9969	1.0000
3.1142e+11	5.3371e+13	4.3659e+12	
AEO	0.5197	0.9594	0.9640
	1.7483e+11	5.1342e+10	4.8138e+10
AHA	0.4786	0.8656	0.8730
	7.9965e+10	4.1027e+10	3.9665e+10
AVOA	0.5626	0.9816	0.9830
	2.2246e+11	4.6151e+10	4.4107e+10
BAN	0.5307	0.9787	0.9839
	4.4516e+12	9.5900e+11	8.2961e+11
BES	0.9824	0.9935	0.9439
	4.2400e+01	2.6045e+01	7.5620e+01
BSA	0.5171	0.9566	0.9594
	5.6093e+10	1.6987e+10	1.6339e+10
CoopSA	0.5000	0.9960	0.9977
	1.7454e+12	1.5840e+11	1.1984e+11
CSO	0.5204	0.9963	0.9986
	1.1053e+13	9.8450e+11	5.9795e+11
GTO	0.4379	0.7889	0.7899
	2.6115e+10	1.6171e+10	1.6050e+10
GWO	0.5065	0.9965	0.9997
	1.1215e+13	9.5258e+11	2.9838e+11
MVO	0.5091	0.9970	0.9998
	3.5252e+13	2.8056e+12	6.9426e+11
PSO	0.5251	0.9957	0.9996
	7.4173e+12	7.1340e+11	2.2807e+11
SpSA	0.4291	0.6436	0.6551
	8.6694e+10	6.8851e+10	6.7732e+10

CVs exhibit a decreasing trend. This trend is due to the variation in these algorithms being roughly the same as their performance degrades with increasing d . In the case of Michalewicz, all the investigated algorithms' CV curves show a decreasing trend but remain bounded. The observed CV values are the smallest here, indicating better stability.

Previously, we saw that CV values are bounded. However, as CV is the ratio of the variation to the mean value, and most of these classic benchmark functions have a known optimum at the zero point, good algorithm performance, namely values close to zero, results in high CV values. The same effect is also caused by an algorithm's performance values crossing from negative to positive values. We observe this in the case of the Styblinski–Tang and Trid functions, as illustrated in Fig. 26.

We observe that some algorithms exhibit greater variation than others. There are a few large spikes in the CV curves, but overall,

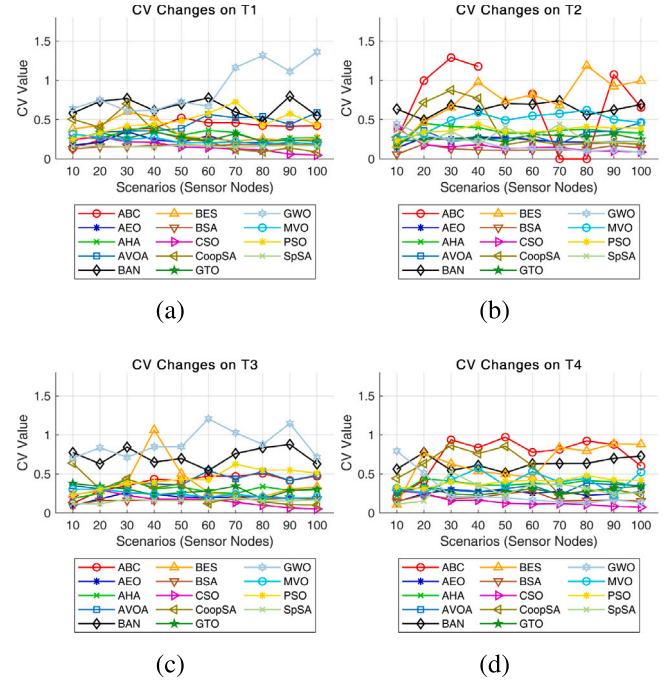


Fig. 23. Changes in CV location of algorithms on (a) T1, (b) T2, (c) T3, (d) T4 topologies.

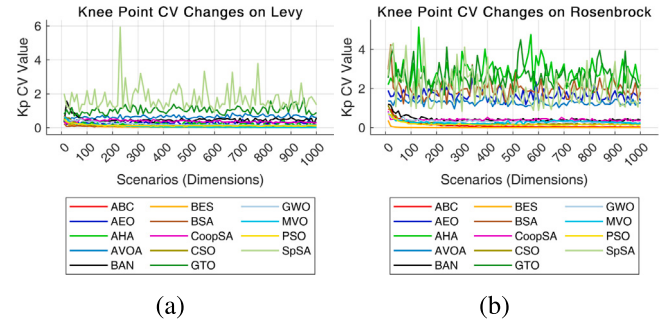


Fig. 24. Changes in CV location of algorithms on (a) Lévy, (b) Rosenbrock functions.

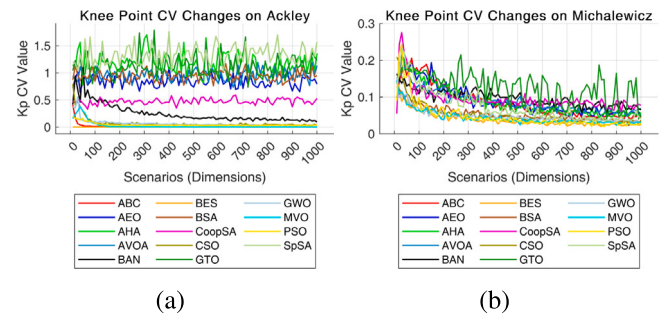


Fig. 25. Changes in CV location of algorithms on (a) Ackley, (b) Michalewicz functions.

the algorithm's CV remains stable on the investigated topologies. A bounded CV means we can estimate the scale of σ as $\sigma = CV \cdot |\mu|$, which is enough for a rough variation comparison between two algorithms.

As for the applicability of CV in performance prediction, the issues caused by small mean values can be mitigated by adding a constant c to the fitness function f to push the minima up from 0. The c constant can later be subtracted to obtain the actual performance value.

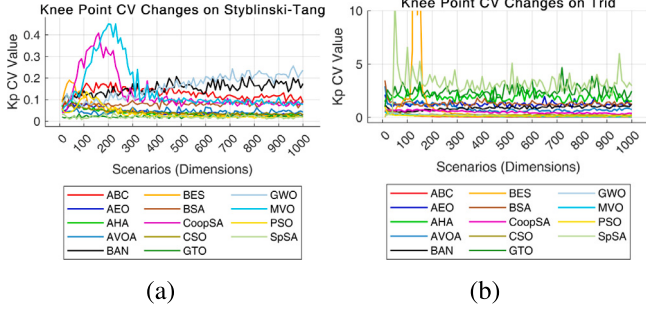


Fig. 26. Changes in CV location of algorithms on (a) Styblinski-Tang, (b) Trid functions.

5.4. Knee-point stability

The location of the knee-point (κ^*) is an important aspect of performance: it indicates the number of iterations an algorithm must run to reach κ^* , so a lower location means fewer iterations and therefore lower computational cost. If the location is unstable — meaning there is significant variation across dimensions — we cannot effectively forecast the best-performing algorithm. In such cases, a slightly worse-performing algorithm may be preferable if it reaches its knee-point in significantly fewer iterations. Next, we summarize graphically the changes in κ^* for each investigated problem.

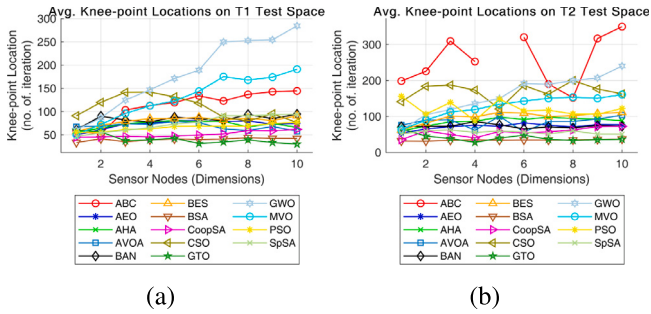


Fig. 27. Changes in the knee-point location of algorithms on the (a) T1, (b) T2 topologies.

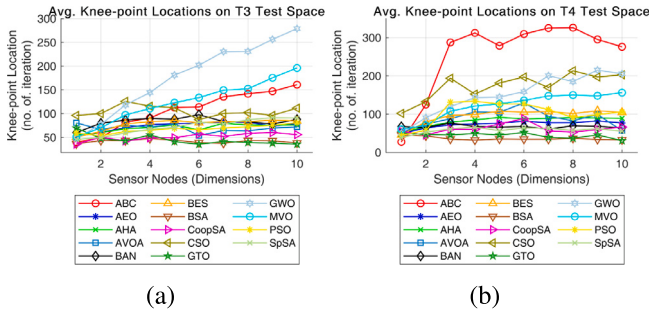


Fig. 28. Changes in the knee-point location of algorithms on the (a) T3, (b) T4 topologies.

In the WSN topologies (Figs. 27 and 28), we observe that the κ^* location of the worst-performing algorithms also increases with the number of nodes, meaning that these algorithms require more iterations to reach their κ^* . ABC has gaps in the presented data, as there are T2 scenarios in which it does not produce a viable solution; therefore, κ^* determination is not possible in those scenarios.

Fig. 29 illustrates the changes in Kp points in the case of the classic benchmark function Ackley. Some algorithms show a slight initial

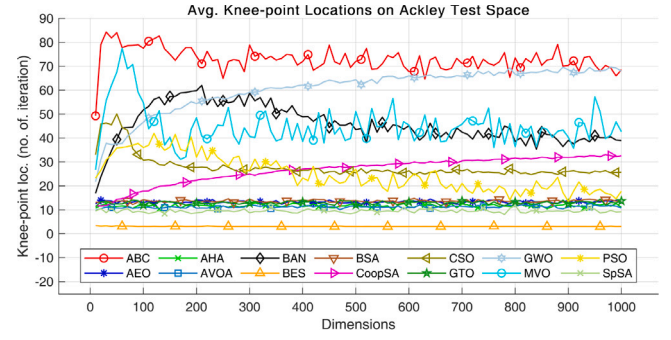


Fig. 29. Changes in the knee-point location of algorithms on Ackley's function.

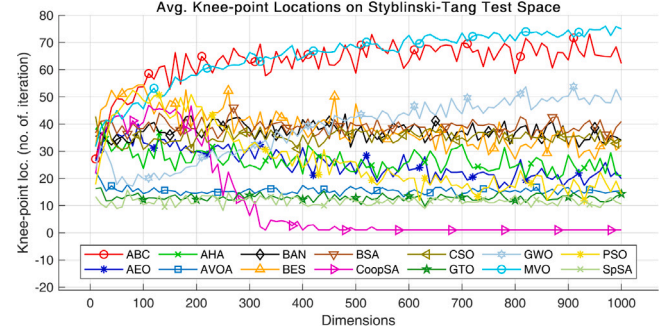


Fig. 30. Changes in the knee-point location of algorithms on the Styblinski-Tang function.

variation, after which their Kp locations stabilize; with some variations, they are bounded. We observed similar behavior in the majority of cases.

At the Styblinski-Tang function, Fig. 30, we observe a significant drift of the CoopSA algorithm's κ^* . This drift occurs because the function's performance degrades, and it stalls more quickly as d increases, thereby decreasing the κ^* location. The κ^* locations of ABC and MVO show an initial increase, which stabilizes at $d = 300$ and $d = 800$, respectively.

Overall, we observe three distinct scenarios regarding the stability of κ^* :

- κ^* values contain variations but are bounded: we consider the κ^* values for these algorithms relatively stable.
- κ^* is increasing along with d : this happens when the performance of these algorithms is getting worse and needs more and more iterations to reach their knee-points.
- κ^* is decreasing: the algorithm is gradually performing worse.

In other words, if the κ^* location is unstable, it can indicate instability in the performance of the algorithms. This criterion could also be used to eliminate an algorithm early from a selection process. Furthermore, it can also be used for late-stage selection, namely, which algorithm is likely to require fewer iterations (lower computational cost) to reach its κ^* .

5.5. Practical application in ASP

Section 3.3 outlined how the identified trends are used to narrow the algorithm portfolio: forecasts indicating clearly worse performance at d_{target} trigger elimination, while close forecasts trigger additional sampling. Here we instantiate that procedure as Algorithm 1, including the parameter values used in our tests.

In an ideal scenario, where all algorithms follow simple, comparable trends — for instance, all *linear* — a practical application would not

require identifying an exact trend: it would suffice to compare the sampled algorithms and check whether Θ_α has sufficient separation and whether the gap is widening relative to the current best-performing one. If so, algorithm α could be eliminated from the selection process. In practice, this is not so straightforward, because mixed trend types can produce misleading early behavior. A *combined* curve, for instance, can grow steeply at low dimensions and only saturate later — so its gap to a *linear* or *power* curve appears to be widening at the sampled range, when in fact the two trends will cross at higher dimensions. The trend type therefore has to be identified, not just the separation observed. A further complication is high variation in the sampled dimensions, which makes trend identification itself unreliable.

Algorithm 1 Selection algorithm

```

Initialize necessary parameters, including:
 $S = \text{all } \alpha \in \mathcal{A}$  (set of selected algorithms)
 $k = 3$  (coverage factor)
 $d_{\text{eval}} = 40$  (min.  $d$  sampled for evaluation)
 $d_{\text{cutoff}} = 70$  (sampling cutoff dimension)
 $d_{\text{target}} = 200$  (target dimension)
 $N_{\text{runs}}$  (number of samples per  $d$ )
 $i=1$  ▷ Start dimension index
stopCond = false
while  $d_i \leq d_{\text{cutoff}}$  and  $\neg \text{stopCond}$  do
  for  $\alpha \in S$  do
    sample  $\alpha$   $N_{\text{runs}}$  times in  $d_i$ 
    calculate Kp of  $\alpha$ 
  end for
  if  $d_i \geq d_{\text{eval}}$  then ▷ Alg. 2
    evaluate  $S$ 
  end if
  if terminal conditions then
    break
  end if
end while
Compare collected results
if Multiple candidate algorithms then
  Continue selection by examining:
  1. Cost of FFEvals
  2. Kp location (estimated iteration count)
  3. Expected magnitude of variations
end if

```

Algorithm 1 presents the essence of the selection algorithm. The first step is to execute each algorithm in the initial dimensions $d = \{1, 2, \dots, d_{\text{eval}}\}$ in order to obtain a minimum number of points before any evaluations can be performed.

After d_{eval} is sampled, the candidates are evaluated according to Algorithm 2, and eliminations are performed. We continue sampling d until either d_{cutoff} is reached or a stop condition is met. A simple stop condition can be when only a single candidate remains.

The final step would be to examine the remaining algorithms. In case of similar performance, the tie-breaking would be based on the cost of computation, namely, which algorithm is expected to perform best with the lowest required computational budget. Here, we have to consider the Kp location, meaning how many iterations an algorithm must run, the number of FFEvals per iteration, and the variation.

Algorithm 2 Candidate evaluation

Require: For each $\alpha \in S$, measurements of y , uncertainties σ , coverage factor k , reliability threshold τ

Ensure: Lowest curve index, candidates C

```

1: Predict final values with CI
for all  $\alpha_i \in S$  do
  Compute  $y_i$  curve rate  $r_i$ 
  if curve  $y_i$  saturating then
     $\hat{y}_i \leftarrow$  asymptote estimate
     $\sigma_{\hat{y}_i} \leftarrow$  estimate spread
  end if
end for

```

```

else
   $\hat{y}_i \leftarrow$  projected value
   $\sigma_{\hat{y}_i} \leftarrow$  propagated uncertainty
end if
end for

Find  $i^*$  where  $y_{i^*, d_{\text{current}}}$  is minimum ▷ Find best for  $d_{\text{current}}$ 

2. Evaluations for elimination
for all  $\alpha_i \in S, i \neq i^*$  do
  Compute  $n_{\sigma}^{\text{pool}}, n_{\sigma}^w$ 
  % if separated and difference growing
  if  $\max(n_{\sigma}^{\text{pool}}, n_{\sigma}^w) \geq k$  and  $\Delta r_i > -\sigma_{\Delta r}$  then
    if  $y_i$  saturating and  $\hat{y}_i < \hat{y}_{i^*} + \sigma_{\hat{y}_{i^*}}$  then
      keep  $\alpha_i$ ; continue ▷ CI overlap
    else
      eliminate  $\alpha_i$ 
    end if
  end if
  if  $|C| \leq 1$  then break
end if
end for
Remove eliminated from  $S$ 
return  $S$ 

```

Algorithm 2 presents the steps for evaluating each $\alpha \in S$ candidate algorithm. The first step here is to identify the growth rate of the performance curves for all α_i , as well as the gap relative to the best-performing one (α_{i^*}) in the currently sampled d_{current} . The main idea for the elimination is the comparison of the growth rate between α_i and (α_{i^*}), and elimination when this gap is increasing and a proper separation is observed. However, a special condition must be checked, namely, if α_i is saturating while α_{i^*} is not, then the two curves are expected to intersect. In such a case, the predicted value with uncertainty is used to differentiate between the two.

Note: the examined dimensions $d_1 < d_2 < \dots < d_{\text{cutoff}}$ must be sufficiently apart, and as a whole must cover enough of the Θ curves in order to overcome any local variation and to identify the type of the curves properly.

5.6. Separation metrics

Let $\alpha \in S$ denote a candidate algorithm with performance curve y_α , and let α^* be the best-performing (lowest-value) algorithm at the current dimension d_{current} . At each sampled point x_j , let g_j denote the gap between α and α^* , and v_j the variance of that gap. Their calculation is given in Eq. (15).

$$g_j = y_\alpha(x_j) - y_{\alpha^*}(x_j), \quad v_j = \sigma_\alpha^2(x_j) + \sigma_{\alpha^*}^2(x_j). \quad (15)$$

Curve rate calculation

Given n observations $\{x_j, y_j, \sigma_j\}_{j=1}^n$, the instantaneous rate of change is estimated as a weighted average of segment slopes, combining two complementary weighting criteria.

Segment slopes. For each consecutive pair of points, the finite-difference slope is

$$r_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad i = 1, \dots, n-1. \quad (16)$$

Slope variance. Propagating the measurement uncertainties through the finite difference gives the variance of each slope estimate,

$$\sigma^2(r_i) = \frac{\sigma_i^2 + \sigma_{i+1}^2}{(x_{i+1} - x_i)^2}. \quad (17)$$

Weighting. Each segment receives a composite weight $w_i = w_i^{\text{prec}} \cdot w_i^{\text{rec}}$ combining two factors.

Precision weight: segments with smaller slope variance contribute more:

$$w_i^{\text{prec}} = \frac{1}{\sigma^2(r_i)}. \quad (18)$$

Recency weight: segments closer to the last observation are up-weighted with an exponential kernel whose half-life equals half the total x -span $h = x_n - x_1$:

$$\lambda = \frac{\ln 2}{h/2} = \frac{2 \ln 2}{h}, \quad (19)$$

$$w_i^{\text{rec}} = \exp[\lambda(\tilde{x}_i - x_n)], \quad (20)$$

where $\tilde{x}_i = (x_i + x_{i+1})/2$ is the midpoint of segment i . By construction $w_i^{\text{rec}} \in (0, 1]$, with the latest segment receiving weight 1. If $h = 0$, all recency weights are set to 1.

Weighted rate estimate. Let $W = \sum_{i=1}^{n-1} w_i$ denote the total weight. The combined rate is

$$\hat{r} = \frac{\sum_{i=1}^{n-1} w_i r_i}{W}. \quad (21)$$

Uncertainty. Treating the segment slopes as independent and propagating the composite weights,

$$\sigma_{\hat{r}} = \frac{\sqrt{\sum_{i=1}^{n-1} w_i^2 \sigma^2(r_i)}}{W}. \quad (22)$$

Pooled gap

For n_{σ}^{pool} calculation, let $\{y_i^A, \sigma_i^A\}_{i=1}^n$ and $\{y_i^B, \sigma_i^B\}_{i=1}^n$ be two sets of n measurements with associated standard uncertainties, sampled at the same n points. All measurements are assumed to be statistically independent.

Point-wise gap. The gap at each sampling point is defined as

$$g_i = y_i^A - y_i^B, \quad i = 1, \dots, n. \quad (23)$$

Average gap. The pooled estimator of the mean gap is

$$\bar{g} = \frac{1}{n} \sum_{i=1}^n g_i. \quad (24)$$

Uncertainty of the average gap. Since the measurements are independent, the variance of g_i is $\sigma^2(g_i) = (\sigma_i^A)^2 + (\sigma_i^B)^2$. Propagating uncertainties through the mean yields

$$\sigma_{\bar{g}} = \frac{1}{n} \sqrt{\sum_{i=1}^n [(\sigma_i^A)^2 + (\sigma_i^B)^2]}. \quad (25)$$

For a curve that is uniformly offset but noisy at any single point, pooling over n points reduces the uncertainty by a factor of $\sqrt{n}$ relative to the single-point uncertainty, improving detection sensitivity accordingly.

Significance. The number of standard deviations by which the average gap differs from zero is

$$n_{\sigma}^{\text{pool}} = \frac{\bar{g}}{\sigma_{\bar{g}}}, \quad (26)$$

provided $\sigma_{\bar{g}} > 0$; otherwise n_{σ}^{pool} is taken as $\pm\infty$ according to the sign of $\bar{g}$.

Inverse-variance weighted pooled gap

For n_{σ}^w calculations, let $\{y_i^A, \sigma_i^A\}_{i=1}^n$ and $\{y_i^B, \sigma_i^B\}_{i=1}^n$ be two sets of n measurements with associated standard uncertainties, sampled at the same n points. In our case, all $\mathcal{E}$ measurements are statistically independent.

The gap at each sampling point is as previously defined in Eq. (23), with the corresponding variance:

$$\sigma^2(g_i) = (\sigma_i^A)^2 + (\sigma_i^B)^2. \quad (27)$$

Inverse-variance weights. Each point is assigned a weight inversely proportional to its variance,

$$w_i = \frac{1}{\sigma^2(g_i)} = \frac{1}{(\sigma_i^A)^2 + (\sigma_i^B)^2}, \quad (28)$$

so that high-noise points contribute less to the combined estimate. Let us denote the total weight as $W = \sum_{i=1}^n w_i$.

Weighted average gap. The inverse-variance weighted estimator of the mean gap is

$$\bar{g}_w = \frac{\sum_{i=1}^n w_i g_i}{W}. \quad (29)$$

Uncertainty of the weighted average gap. The standard uncertainty of $\bar{g}_w$ follows directly from error propagation,

$$\sigma_{\bar{g}_w} = \frac{1}{\sqrt{W}} = \left(\sum_{i=1}^n [(\sigma_i^A)^2 + (\sigma_i^B)^2]^{-1} \right)^{-1/2}. \quad (30)$$

Unlike the unweighted estimator, points with large uncertainties are down-weighted, preventing a few high-noise measurements from obscuring the signal visible in low-noise points.

Significance. The number of standard deviations by which the weighted average gap differs from zero is

$$n_{\sigma}^w = \frac{\bar{g}_w}{\sigma_{\bar{g}_w}}, \quad (31)$$

provided $\sigma_{\bar{g}_w} > 0$; otherwise n_{σ}^w is taken as $\pm\infty$ according to the sign of $\bar{g}_w$.

5.7. Value predictions

Given n observations $\{x_i, y_i, \sigma_i\}_{i=1}^n$ with $n \geq 4$, the procedure branches on whether the curve is saturating or still growing.

Saturation detection. We use two cases of the curve when estimating a future value: saturating and growing.

Case 1 — Saturating curve. An exponential-decay model

$$y(x) = C + A e^{-\lambda x} \quad (32)$$

is fitted by nonlinear least squares to estimate the asymptote C , yielding an estimate $\hat{C}$ and its uncertainty σ_C . The predicted final value and its uncertainty are

$$p = \hat{C}, \quad \sigma_p = \sigma_C. \quad (33)$$

If the fit fails (i.e. $\hat{C}$ is undefined), the last observed value is used as a fallback:

$$p = y_n, \quad \sigma_p = \sigma_n. \quad (34)$$

Case 2 — Growing curve. The growth rate is estimated from the second half of the data. Let $m = \lfloor n/2 \rfloor$ be the midpoint index. The finite-difference rate is

$$\hat{r} = \frac{y_n - y_m}{x_n - x_m}. \quad (35)$$

A coarse two-point rate is deliberate here: the projection uses only its positive part as a conservative growth estimate, so the additional precision of the weighted multi-segment rate defined earlier is not required. The projection horizon is taken as the full span of the observed data,

$$h = x_n - x_1. \quad (36)$$

The predicted value extrapolates from the last observation using only the positive part of the estimated rate,

$$p = y_n + \max(0, \hat{r}) \cdot h. \quad (37)$$

Uncertainty propagation. The uncertainty of the rate estimate follows from the uncertainties of the two endpoints used in the finite difference,

$$\sigma_{\hat{r}} = \frac{\sqrt{\sigma_n^2 + \sigma_m^2}}{x_n - x_m}. \quad (38)$$

Treating y_n and $\hat{r}$ as independent, the projected uncertainty is

$$\sigma_p = \sqrt{\sigma_n^2 + (\sigma_{\hat{r}} \cdot h)^2}. \quad (39)$$

Significance. In both cases the number of standard deviations of the predicted value from zero is

$$n_{\sigma}^{\text{pred}} = \frac{p}{\sigma_p}, \quad (40)$$

provided $\sigma_p > 0$; otherwise n_{σ}^{pred} is taken as $\pm\infty$ according to the sign of p .

5.8. Decomposition of the proposed strategy

For reference, the proposed strategy is composed of four steps, each specified earlier in the paper:

- **Low-dimensional sampling** — selecting a subset of dimensions $D_s \subset \{d_1, \dots, d_n\}$ at which the algorithms are measured (Section 3.1, Step 1).
- **Knee-point-based performance measurement** — evaluating $\Xi_a(d)$ at the knee-point κ^* of each algorithm's convergence curve (Sections 3 and 4.5).
- **Trend identification across dimensions** — fitting the three candidate families (linear, combined, power) and selecting the best fit as $\hat{\theta}_\alpha$ (Section 3.1, Steps 2–3).
- **Elimination-based selection** — comparing the fitted trends and eliminating algorithms whose forecasts indicate clearly worse performance at d_{target} (Sections 3.3 and 5.5).

Together, the first three steps produce the per-algorithm forecasts; the fourth applies them to narrow the algorithm portfolio. Detailed results of this process on the WSN topologies follow in Section 5.5.

5.9. WSN selection results

This work focuses solely on the proposed selection method. Accordingly, the results below begin from the full portfolio of 14 algorithms and end with the reduced set the method retains for further consideration. Choosing a single algorithm from a reduced set, when more than one candidate remains, is left to secondary criteria — such as per-iteration computational cost or the spread of an algorithm's results — which are outside the scope of the method evaluated here.

While the trend examination was presented for $d = 100$, the testing and evaluation of the proposed selection method were extended to $d = 200$. The corresponding Ξ values at this dimensionality are presented below for each topology.

Let us denote the computational cost required of algorithm α in dimension d as $T_\alpha(d)$ for a single run instance.

Concerning the execution times, we have recorded the run time for full iteration of each algorithm, not just until the knee-point. We will be using these full execution times here when comparing algorithm run time against portfolio time saved when presenting the results in this subsection.

We will evaluate the efficacy of the selection process by considering the execution time used to sample lower dimensional performances (T_{samples}) with the execution time of the algorithms not eliminated ($T_{\alpha \in S}$) compared to just executing all of the algorithms on the target dimension ($T_{\alpha \in A}(d_{\text{target}})$). The calculation of the total execution time for the selection process $T_{\text{selection}}$ is presented in Eq. (41), while efficiency

Table 15
ASP timing efficiency on T1.

SI algo. name	Sampled	T_{samples}	T_{target}	T_{saved}
	d	sec.	sec.	%
ABC (selected)	10, 20, 30, 40, 50, 60, 70	2792.17	2207.78	−26.47%
AEO	10, 20, 30, 40, 50, 60, 70	1033.62	3872.38	73.31%
AHA	10, 20, 30, 40	164.70	2678.00	93.85%
AVOA (selected)	10, 20, 30, 40, 50, 60, 70	2452.89	1894.58	−29.47%
BAN	10, 20, 30, 40	144.53	1500.68	90.37%
BES (selected)	10, 20, 30, 40, 50, 60, 70	8175.58	6167.11	−32.57%
BSA	10, 20, 30, 40	118.79	1466.43	91.90%
CoopSA	10, 20, 30, 40	465.52	7261.29	93.59%
CSO	10, 20, 30, 40	164.80	2972.55	94.46%
GTO	10, 20, 30, 40	264.34	4030.49	93.44%
GWO	10, 20, 30, 40	117.97	2340.50	94.96%
MVO	10, 20, 30, 40	115.24	1916.52	93.99%
PSO (selected)	10, 20, 30, 40, 50, 60, 70	2789.22	2234.53	−24.82%
SpSA	10, 20, 30, 40, 50, 60	637.09	3146.26	79.75%
	Total time saved (%):			55.51%

in percentage is given in Eq. (42).

$$T_{\text{selection}} = T_{\text{samples}} + T_{\alpha \in S}(d_{\text{target}}) \quad (41)$$

$$T_{\text{saved}} = \left(1 - \frac{T_{\text{selection}}}{T_{\alpha \in A}(d_{\text{target}})}\right) \cdot 100 \quad (42)$$

Note that we will present negative T_{saved} values for the selected algorithms: a selected algorithm is sampled across the lower dimensions and then run at d_{target} , so its individual cost exceeds that of a single d_{target} run. The net saving of the selection process comes from the eliminated algorithms, whose d_{target} runs are avoided entirely.

The measured results on T1 with $d = 200$ in best to worst order are as follows: GTO (0.819019 mW), BSA (0.943936 mW), PSO (1.112585 mW), BES (1.131919 mW), AVOA (1.149133 mW), AHA (2.565794 mW), AEO (2.676612 mW), SpSA (2.972045 mW), MVO (3.225091 mW), CoopSA (4.008558 mW), BAN (4.370596 mW), GWO (5.071377 mW), CSO (7.130528 mW), and ABC (8.869017 mW). The sampling dimensions and execution timing evaluations of the selection process are presented in Table 15. The process selects algorithms: ABC, AVOA, BES, and PSO. Due to trend misidentification, GTO is prematurely eliminated at $d = 40$, as its marginally fitted sampled trend provides no indication to continue sampling further (a failure mode discussed in Section 5.10).

The measured results on T2 with $d = 200$ in best to worst order are as follows: AVOA (1.843201 mW), PSO (1.987919 mW), GTO (2.463286 mW), BES (2.640225 mW), AEO (4.109957 mW), CoopSA (5.309690 mW), AHA (6.115554 mW), SpSA (6.832895 mW), BSA (7.829544 mW), ABC (11.020469 mW), CSO (11.192469 mW), MVO (18.530798 mW), BAN (32.865862 mW), and GWO (52.689275 mW). At the end of the selection process, AEO, AVOA, and BES remain. Algorithms ABC, CSO, CoopSA, and PSO are eliminated during selection for having found no results in one or multiple runs in the sampled dimensions, while GTO is eliminated despite being among the best performers at d_{target} . The selection process results regarding sampling dimensions and execution timing are presented in Table 16.

The measured results on T3 with $d = 200$ in best to worst order are as follows: GTO (0.527184 mW), BSA (0.616277 mW), PSO (1.046744 mW), AVOA (1.171077 mW), BES (1.523731 mW), AEO (2.259300 mW), AHA (2.579869 mW), SpSA (3.316745 mW), MVO (3.355821 mW), CoopSA (4.199234 mW), CSO (7.241116 mW), GWO (7.790576 mW), ABC (8.200233 mW), and BAN (8.656591 mW). The selection process characteristics are presented in Table 17. By the end

Table 16
ASP process time requirements on T2.

SI algo. name	Sampled	$T_{samples}$	T_{target}	T_{saved}
	d	sec.	sec.	%
	10,20	32.34	1200.11	97.30%
ABC				
AEO	10, 20, 30, 40, 50, 60, 70	4326.18	3283.12	-31.77%
(selected)				
AHA	10, 20, 30, 40, 50, 60, 70	671.84	2281.09	70.55%
AVOA	10, 20, 30, 40, 50, 60, 70	2789.89	2196.26	-27.03%
(selected)				
BAN	10, 20, 30, 40	142.35	1637.20	91.31%
BES	10, 20, 30, 40, 50, 60, 70	8427.58	6366.44	-32.38%
(selected)				
BSA	10, 20, 30, 40, 50, 60	302.91	1314.11	76.95%
CoopSA	10	24.06	6091.66	99.61%
CSO	10,20	31.98	1750.21	98.17%
GTO	10, 20, 30, 40, 50, 60, 70	1142.37	4592.02	75.12%
GWO	10, 20, 30, 40	124.66	1881.32	93.37%
MVO	10, 20, 30, 40	122.00	1858.91	93.44%
PSO	10	12.41	2098.35	99.41%
SpSA	10, 20, 30, 40, 50, 60, 70	930.89	3183.60	70.76%
Total time saved (%):				51.98%

Table 17
ASP process time requirements on T3.

SI algo. name	Sampled	$T_{samples}$	T_{target}	T_{saved}
	d	sec.	sec.	%
	10, 20, 30, 40, 50, 60, 70	2799.24	2204.04	-27.00%
ABC				
(selected)				
AEO	10, 20, 30, 40, 50, 60	713.74	4133.32	82.73%
AHA	10, 20, 30, 40	164.45	2792.95	94.11%
AVOA	10, 20, 30, 40, 50	242.39	1794.73	86.49%
BAN	10, 20, 30, 40	142.97	1538.90	90.71%
BES	10, 20, 30, 40, 50, 60, 70	8147.74	6149.93	-32.48%
(selected)				
BSA	10, 20, 30, 40	114.98	1829.33	93.71%
CoopSA	10, 20, 30, 40	455.49	7348.78	93.80%
CSO	10, 20, 30, 40	166.06	3106.00	94.65%
GTO	10, 20, 30, 40	282.20	3922.12	92.81%
GWO	10, 20, 30, 40	126.21	2337.80	94.60%
MVO	10, 20, 30, 40	122.82	1796.31	93.16%
PSO	10, 20, 30, 40, 50, 60, 70	2667.47	2078.36	-28.34%
(selected)				
SpSA	10, 20, 30, 40, 50, 60	610.35	3119.82	80.44%
Total time saved (%):				62.05%

of the selection, algorithms ABC, BES, and PSO are kept. Due to trend misidentification, GTO is once again prematurely eliminated at $d = 40$.

The measured results on T4 with $d = 200$ in best to worst order are as follows: AVOA (1.447821 mW), GTO (1.634411 mW), PSO (1.651546 mW), BES (2.658438 mW), BSA (3.365702 mW), AEO (3.632033 mW), AHA (4.677969 mW), SpSA (5.705412 mW), CoopSA (5.711277 mW), CSO (7.639055 mW), BAN (16.814832 mW), MVO (17.466521 mW), ABC (28.784132 mW), and GWO (59.569630 mW). Table 18 presents the sampled dimensions and execution timings of the selection process for T4. Here, the following algorithms were selected for further consideration: AEO, AVOA, BES, and CoopSA. Due to trend misidentification, GTO, the second-best algorithm, is prematurely eliminated at $d = 50$.

5.10. Effectiveness of the selection process

Given the scope of the proposed method evaluation, its effectiveness has two aspects: how much it reduces the candidate set, and whether that reduction is correct — that is, whether the algorithms it discards from the low-dimensional data are in fact non-competitive at d_{target} . The process targets the cost of evaluating the *portfolio* as a whole at d_{target} , not the cost of any individual algorithm.

Table 18
ASP process time requirements on T4.

SI algo. name	Sampled	$T_{samples}$	T_{target}	T_{saved}
	d	sec.	sec.	%
	10, 20, 30	73.30	1118.69	93.45%
ABC				
AEO	10, 20, 30, 40, 50, 60, 70	4231.09	3218.50	-31.46%
(selected)				
AHA	10, 20, 30, 40, 50	274.48	2180.92	87.41%
AVOA	10, 20, 30, 40, 50, 60, 70	2665.96	2093.26	-27.36%
(selected)				
BAN	10, 20, 30, 40, 50	241.59	1770.30	86.35%
BES	10, 20, 30, 40, 50, 60, 70	8552.55	6548.30	-30.61%
(selected)				
BSA	10, 20, 30, 40, 50	191.17	1447.63	86.79%
CoopSA	10, 20, 30, 40, 50, 60, 70	7598.35	5918.55	-28.38%
(selected)				
CSO	10, 20, 30	74.17	2253.13	96.71%
GTO	10, 20, 30, 40, 50	460.52	4326.33	89.36%
GWO	10, 20, 30, 40	121.56	2037.62	94.03%
MVO	10, 20, 30, 40	118.05	1660.63	92.89%
PSO	10, 20, 30	77.15	2041.14	96.22%
SpSA	10, 20, 30, 40, 50, 60	617.12	3348.60	81.57%
Total time saved (%):				36.70%

Across the four topologies, the candidate set is reduced from 14 algorithms to between 3 and 4, with a total portfolio time saving of 55.51% (T1), 51.98% (T2), 62.05% (T3), and 36.70% (T4). Since several of the retained algorithms invoke two or three fitness evaluations per agent per iteration and are intrinsically expensive to run, these savings are conservative; the number of eliminated algorithms is itself an informative measure of effectiveness, independent of the timing.

Most of the eliminations are correct. The clearly inferior algorithms are discarded, and algorithms that produce non-finite results in the sampled lower dimensions — such as PSO on several topologies — are eliminated early. Under the realistic five-run regime this is the intended behavior: an algorithm that fails at lower dimensions is not expected to improve at higher ones, and because performance typically degrades with dimension, it is likely to fail more often there — requiring more runs and further increasing the computational cost. The method is also deliberately conservative: when an algorithm has not yet separated sufficiently at the sampled dimensions, it is retained rather than discarded on insufficient evidence. ABC on T1 and T3 is such a case — retained despite being among the worst performers at d_{target} — which errs on the safe side for a pre-selection step but limits the achievable reduction when algorithms separate slowly.

However, the method fails on GTO, which is eliminated on all four topologies despite remaining among the better algorithms at d_{target} even under the five-run regime the selection operates on — ranking second on T1, fourth on T2, third on T3, and fourth on T4. The cause is not a misapplication of the elimination criterion but a limitation of the sampled range. Over the sampled dimensions ($d \leq d_{cutoff}$), GTO's knee-point performance was sufficiently separated from the running best and its gap to it was widening, so the separation-and-divergence condition of Algorithm 2 was met and GTO was eliminated — a correct decision given the data available at that point. Its performance only reverses at higher dimensions, beyond the sampled range, as shown for T4 in Section 5.12. This is a boundary case rather than a faulty decision, consistent with the conditional nature of trend-type stability established in Section 3.2: where variance is high and the sampled range too short to capture a later reversal, a correctly-applied elimination can still produce the wrong outcome.

Overall, the presented method reliably narrows the candidate portfolio at a meaningful resource saving, with its incorrect outcomes concentrated in the high-variance regime that Section 3.2 identifies as the boundary of applicability. It is intended as one stage of a larger selection pipeline — narrowing the portfolio from low-dimensional

measurements before a final choice is made among the survivors using secondary criteria such as variance or per-iteration cost — and its role within that pipeline is set out in Section 6.

5.11. Observations on the classic benchmark functions

The selection process is not applied to the classic benchmark functions. These functions were included only to test for the existence of the θ trends (Section 5), not as practical optimization targets; the applied objective of this work is the WSN setting, which motivated the original investigation. Nonetheless, the benchmark results provide a useful secondary check on the premise the selection process relies on — namely, that poorly performing algorithms separate from the rest as dimensionality grows.

On the benchmark functions with a positive optimum — Ackley, Griewank, Lévy, Rastrigin, Rosenbrock, Sphere, and Trid — this separation is pronounced and consistent. In every case a small number of algorithms diverge sharply from the others as d increases, while the remaining algorithms stay grouped near the optimum. ABC and MVO separate the most strongly and the earliest across all seven functions, and would be eliminated well before d_{target} by the selection criterion; BAN, CSO, GWO, and PSO form a secondary separating group on several of the functions, most visibly on Rastrigin (Fig. 18) and Lévy (Fig. 16). The well-performing algorithms — including BES, SpSA, AVOA, GTO, AHA, AEO, and BSA — remain tightly clustered near zero throughout these benchmark functions, and would not separate from one another within the sampled range on them. This is the behavior the selection process exploits: the clearly inferior algorithms become identifiable early, while the strong performers remain indistinguishable and are correctly retained for closer evaluation. We note that an algorithm's separation behavior is specific to the problem: GTO, for instance, is a stable strong performer here, yet on the WSN topologies its high performance variance produces the misleading trend that leads to its incorrect elimination (Section 5.12).

The remaining two functions illustrate the boundary of this behavior, consistent with the conditional nature of trend-type stability discussed in Section 3.2. On Michalewicz (Fig. 17), all algorithms remain within a narrow band, and a clear separation only emerges at larger d : ABC, GWO, and MVO separate as the worst performers at roughly $d \approx 400$, while CoopSA, the best performer, separates only after $d \approx 600$. On Styblinski-Tang (Fig. 21) the ordering differs from the other functions altogether: AVOA, GTO, and SpSA are the strongest performers, separating from the rest after $d \approx 150$, while ABC, CoopSA, and MVO are the weakest. This shows that which algorithms separate, and how quickly, is problem-dependent rather than a fixed property of the algorithms. These cases reinforce that the selection process is reliable only where the problem induces a clear and stable separation, and that its applicability must be assessed per problem rather than assumed. These benchmark observations corroborate the WSN findings; the method's practical limitations are discussed next.

5.12. Limitations in practical applications

At SI algorithm initialization, the population is randomly distributed throughout the search space, giving some members of the swarm a non-zero chance of being positioned near the global optimum of the problem. As a result, even poorly performing algorithms can occasionally outperform others. Although the likelihood of this happening decreases with increasing dimensionality, it remains non-zero. With a low number of Ξ measurements, this can introduce large enough variations in θ to draw incorrect predictions. However, in real-life cases, these algorithms are usually run only 5 times.

As all algorithms make use of RNG, when Ξ values are low the ripples in their θ lines become large, reducing the quality of any trend fits. Nonetheless, identifying a general trend direction or bounds might still be possible.

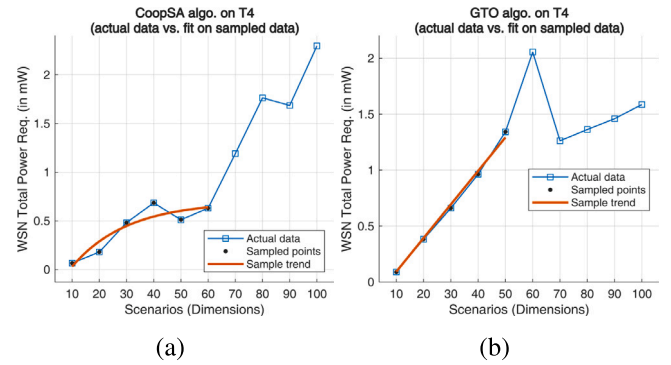


Fig. 31. Trend identification misses due to high noise in case of: (a) CoopSA, (b) GTO.

A low number of problem dimensions is a fundamental constraint. If a problem does not have a sufficient number of dimensions within a low enough computational budget, then the model either cannot be applied or will be more computationally demanding overall than running the algorithms directly in the target dimension.

High noise can be misleading. As seen in the WSN scenarios with GTO, the initial data points indicate a steady growth rate, leading to early elimination. Increasing the reliability of detection for such cases from the sampled range alone, without resorting to additional sampling and its associated computational cost, remains an open problem; while fully reliable detection may not be achievable, other types of analytical methods may still hold potential for improvement.

Fig. 31 presents the case of CoopSA and GTO on the T4 topology. For CoopSA, a favorable result at scenario 50 produces a better-than-actual trend identification. GTO is the opposite case: over the sampled dimensions (10–50) its knee-point performance separates steadily from the running best, with the gap widening, so the elimination criterion of Algorithm 2 is correctly met and there is no programmatic reason to sample GTO further. Its performance only reverses beyond the sampled range; recovering the true trend would require an additional 3 to 4 sampled dimensions, at a corresponding computational cost. GTO's trend fit over the sampled range is also marginal — barely meeting the 0.7 threshold at 0.7325 — which reflects its high performance variance on this topology. Increasing the reliability of detecting such cases from the sampled range alone, without additional sampling and its associated cost, remains an open problem; while fully reliable detection may not be achievable, other types of analytical methods may still hold potential for improvement.

6. Conclusions

In this paper, we investigated how the knee-point performance of SI algorithms behaves as the dimensionality of an optimization problem increases. We found that, for the majority of the tested algorithms and optimization problems, this performance follows a consistent trend type as dimensionality grows, and we set out the conditions under which this behavior is expected to hold, as well as the regimes in which it does not. The results further indicate that the trend type is specific to the algorithm–problem pair rather than to the algorithm alone; consequently, for a new class of optimization problem, the trends must be measured and identified anew rather than carried over from previous problems.

As a practical application, we proposed a selection technique that uses performance measured on reduced, lower-dimensional versions of a problem to forecast the relative ranking of algorithms at the full dimension, where their performance trends can be reliably identified. The technique is not a standalone algorithm selector: it is intended as one stage of a larger selection pipeline, applied after an algorithm

portfolio has been assembled and before the final choice is made, to reduce the number of candidate algorithms that must be evaluated at the target dimension. In the tested WSN scenarios it reliably narrowed the portfolio at a meaningful saving of computational time and energy, with its incorrect outcomes concentrated in the high-variance regime identified as the boundary of its applicability.

The method has limitations. Lower-dimensional sampling still carries a computational cost; the problem must have a sufficient number of dimensions for the approach to be worthwhile; the trend-type behavior does not necessarily hold for every optimization problem; and under high performance variance the trend can be misidentified, leading to the incorrect elimination of otherwise competitive algorithms. Increasing the reliability of detecting such high-variance cases from the sampled range alone, without resorting to additional sampling and its associated computational cost, remains an open problem; while fully reliable detection may not be achievable, other types of analytical methods may hold potential for improvement. Characterizing the method's applicability to other classes of optimization problem, and a systematic comparison of knee-point detection methods on the convergence curves of SI algorithms, are likewise left for future investigation.

CRediT authorship contribution statement

Levente Filep: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Zoltán Gál:** Writing – review & editing, Validation, Supervision, Methodology, Formal analysis, Data curation.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the author(s) used Claude AI (Anthropic) and Grammarly to assist with proofreading and grammar correction, text rephrasing for clarity, verification of equation definitions and their implementation. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

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.

Acknowledgments

This research was supported by the QoS-HPC-IoT Laboratory and project TKP2021-NKTA of the University of Debrecen, Hungary. Project no. TKP2021-NKTA-34 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NKTA funding scheme. This project was also supported by the Austro-Hungarian Action Foundation's OMAA, Project ID 121öu2.

Appendix A. Classic benchmark functions

Ackley function:

$$f(\mathbf{x}) = -a \exp \left(-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2} \right) - \exp \left(\frac{1}{d} \sum_{i=1}^d \cos(cx_i) \right) + a + e \quad (\text{A.1})$$

where $a = 20$, $b = 0.2$, $c = 2\pi$.

Griewank function:

$$f(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^d x_i^2 - \prod_{i=1}^d \cos \left(\frac{x_i}{\sqrt{i}} \right) + 1 \quad (\text{A.2})$$

Lévy function:

$$f(\mathbf{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1} (w_i - 1)^2 [1 + 10 \sin^2(\pi w_i + 1)] + (w_d - 1)^2 [1 + \sin^2(2\pi w_d)] \quad (\text{A.3})$$

where $w_i = 1 + \frac{x_i - 1}{4}$ for all $i = 1, \dots, d$.

Michalewicz function:

$$f(\mathbf{x}) = - \sum_{i=1}^d \sin(x_i) \sin^{2m} \left(\frac{i x_i^2}{\pi} \right) \quad (\text{A.4})$$

where typically $m = 10$.

Rastrigin function:

$$f(\mathbf{x}) = 10d + \sum_{i=1}^d [x_i^2 - 10 \cos(2\pi x_i)] \quad (\text{A.5})$$

Rosenbrock function:

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \quad (\text{A.6})$$

Sphere function:

$$f(\mathbf{x}) = \sum_{i=1}^d x_i^2 \quad (\text{A.7})$$

Styblinski–Tang function:

$$f(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^d (x_i^4 - 16x_i^2 + 5x_i) \quad (\text{A.8})$$

Trid function:

$$f(\mathbf{x}) = \sum_{i=1}^d (x_i - 1)^2 - \sum_{i=2}^d x_i x_{i-1} \quad (\text{A.9})$$

Appendix B. SI algorithm parameters

Table B.19 presents the Algorithm-specific tunable hyperparameters used in this study. Population size and maximum iterations are excluded as they are common inputs across all algorithms.

In case of AEO, and GWO, there are no algorithm-specific tunable parameters.

Table B.19

Algorithm-specific tunable hyperparameters used in this study.

Algorithm	Parameter (Purpose)	Value
ABC	<i>limit</i> (scout-bee abandonment threshold)	100
AHA	<i>M</i> (migration coefficient: period in iterations between migration-foraging events; <i>n</i> is population size)	$2n$
AVOA	P_1 (selects between two exploration strategies)	0.6
	P_2 (selects between two exploitation strategies in the first phase, $ F \in [0.5, 1]$)	0.4
	P_3 (selects between two exploitation strategies in the second phase, $ F < 0.5$)	0.6
	α (selection probability for the best vulture)	0.8
	β (selection probability for the second-best vulture)	0.2
	w (exponent shaping the satiation-rate transition curve)	2.5

(continued on next page)

Table B.19 (continued).

Algorithm	Parameter (Purpose)	Value
BAN	α (loudness decrease factor)	0.5
	γ (pulse-rate increase factor)	0.75
	$[A_{\min}, A_{\max}]$ (loudness initialization range)	[1, 5]
	$[r_{0,\min}, r_{0,\max}]$ (asymptotic maximum pulse-rate bounds)	[0, 1]
	$[f_{\min}, f_{\max}]$ (echolocation-frequency sampling range)	[0, 1.5]
	G (stagnation-reset interval for loudness and pulse-rate)	10
	$[prob_{\min}, prob_{\max}]$ (habitat-selection probability bounds)	[0.6, 0.9]
	$[\theta_{\min}, \theta_{\max}]$ (contraction-expansion coefficient bounds, linearly decreasing)	[0.5, 1]
	$[w_{\min}, w_{\max}]$ (inertia-weight bounds, linearly decreasing)	[0.5, 0.9]
	$[C_{\min}, C_{\max}]$ (Doppler-effect compensation-rate bounds)	[0.1, 0.9]
BES	α (position-update intensity in the select-space stage)	2
	a (angle between adjacent points in the spiral flight)	10
	R (number of search cycles in the spiral motion)	1.5
BSA	c_1 (cognitive accelerated coefficient in the foraging-behavior update)	1.5
	c_2 (social accelerated coefficient in the foraging-behavior update)	1.5
	a_1 (modulates indirect/mean-fitness effect in the vigilance update)	1
	a_2 (modulates direct/competitive effect in the vigilance update)	1
	FQ (flight-behavior frequency: iterations between producer/scrounger flight events)	10
CoopSA	$gbestNum$ (number of elite solutions retained in the team-communication operator)	3
	α (scales the global elite-mean term)	0.10
	β (scales the personal-best-mean term)	0.15
CSO	$rPercent$ (fraction of the swarm assigned the rooster role)	0.15
	$hPercent$ (fraction assigned the hen role; remainder are chicks)	0.7
	$mPercent$ (fraction of hens designated as mother hens)	0.5
	G (hierarchy-update interval, in iterations)	10
GTO	p (probability of migration to an unknown location)	0.03
	β (scaling coefficient of the “compete for adult females” operator)	3
	W (threshold switching between the two exploitation mechanisms)	0.8
MVO	$WEP_{\min}$ (minimum bound of the wormhole existence probability)	0.2
	$WEP_{\max}$ (maximum bound of the wormhole existence probability)	1
PSO	$c_1 = c_2$ (acceleration coefficients)	1.5
	w (inertia weight)	0.7
	w_{damp} (inertia dampening ratio)	1
SpSA	$P_{percent}$ (producer fraction; remainder are scroungers)	0.2
	R_2 (alarm-value threshold switching producer behavior)	0.8

Data availability

Data will be made available on request.

References

- [1] D. Molina, J. Poyatos, J. Del Ser, S. García, A. Hussain, F. Herrera, Comprehensive taxonomies of nature- and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations, *Cogn. Comput.* 12 (5) (2020) 897–939, <http://dx.doi.org/10.1007/s12559-020-09730-8>.
- [2] A.E.F. Clementi, P. Penna, R. Silvestri, Hardness results for the power range assignment problem in packet radio networks, in: D.S. Hochbaum, K. Jansen, J.D.P. Rolim, A. Sinclair (Eds.), *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 197–208, http://dx.doi.org/10.1007/978-3-540-48413-4_21.
- [3] E.H. Houssein, M.R. Saad, Y. Djenouri, G. Hu, A.A. Ali, H. Shaban, Metaheuristic algorithms and their applications in wireless sensor networks: review, open issues, and challenges, *Clust. Comput.* 27 (10) (2024) 13643–13673, <http://dx.doi.org/10.1007/s10586-024-04619-9>.
- [4] Q.-V. Pham, D.C. Nguyen, S. Mirjalili, D.T. Hoang, D.N. Nguyen, P.N. Pathirana, W.-J. Hwang, Swarm intelligence for next-generation networks: Recent advances and applications, *J. Netw. Comput. Appl.* 191 (2021) 103141, <http://dx.doi.org/10.1016/j.jnca.2021.103141>, URL <https://www.sciencedirect.com/science/article/pii/S1084804521001582>.
- [5] I.A. Chaudhry, A.A. Khan, A research survey: review of flexible job shop scheduling techniques, *Int. Trans. Oper. Res.* 23 (3) (2016) 551–591, <http://dx.doi.org/10.1111/itor.12199>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.12199>, URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12199>.
- [6] W. Zhang, X. Bao, X. Hao, M. Gen, Metaheuristics for multi-objective scheduling problems in industry 4.0 and 5.0: a state-of-the-arts survey, *Front. Ind. Eng. Volume 3 - 2025* (2025) <http://dx.doi.org/10.3389/fieng.2025.1540022>, URL <https://www.frontiersin.org/journals/industrial-engineering/articles/10.3389/fieng.2025.1540022>.
- [7] Z. Pan, D. Lei, L. Wang, A knowledge-based two-population optimization algorithm for distributed energy-efficient parallel machines scheduling, *IEEE Trans. Cybern.* 52 (6) (2022) 5051–5063, <http://dx.doi.org/10.1109/TCYB.2020.3026571>.
- [8] H. Wang, B.R. Sarker, J. Li, J. Li, Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning, *Int. J. Prod. Res.* 59 (19) (2021) 5867–5883, <http://dx.doi.org/10.1080/00207543.2020.1794075>, URL <https://doi.org/10.1080/00207543.2020.1794075>.
- [9] F. Zhao, Z. Fu, L. Wang, H. Sang, A heterogeneous graph reinforcement learning framework with question-aware neighborhood aggregation and interpolation prompt attention for dynamic flexible job shop scheduling problem, *IEEE Trans. Ind. Inform.* 22 (4) (2026) 2863–2874, <http://dx.doi.org/10.1109/TII.2025.3646962>.
- [10] M. Mavrovouniotis, C. Li, S. Yang, A survey of swarm intelligence for dynamic optimization: Algorithms and applications, *Swarm Evol. Comput.* 33 (2017) 1–17, <http://dx.doi.org/10.1016/j.swevo.2016.12.005>.
- [11] J. Tang, G. Liu, Q. Pan, A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends, *IEEE/CAA J. Autom. Sin.* 8 (10) (2021) 1627–1643, <http://dx.doi.org/10.1109/JAS.2021.1004129>.
- [12] R.E. Bellman, *Dynamic Programming*, Princeton University Press, 1957, A RAND Corporation research study.
- [13] C.R. Taylor, *Dynamic programming and the curses of dimensionality*, in: *Applications of Dynamic Programming To Agricultural Decision Problems*, Westview Press, 1993, pp. 1–10.
- [14] S. Chen, J. Montgomery, A. Bolufé-Röhler, Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution, *Appl. Intell.* 42 (2015) 514–526, <http://dx.doi.org/10.1007/s10489-014-0613-2>.
- [15] L. Filep, P. Polgar, Z. Gal, Energy consumption based analysis of the swarm intelligence routing mechanisms in wireless sensor networks, in: 4th IFSA Winter Conference on Automation, Robotics & Communications for Industry 4.0 / 5.0, ARCI'2024, Innsbruck, Austria, 2024, pp. 249–256, <http://dx.doi.org/10.13140/RG.2.2.20923.18722>.
- [16] E.-G. Talbi, *Metaheuristics: From Design to Implementation*, Wiley, 2009, <http://dx.doi.org/10.1002/9780470496916>.
- [17] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P.N. Suganthan, C.A. Coello Coello, F. Herrera, Bio-inspired computation: Where we stand and what's next, *Swarm Evol. Comput.* 48 (2019) 220–250,

- <http://dx.doi.org/10.1016/j.swevo.2019.04.008>, URL <https://www.sciencedirect.com/science/article/pii/S2210650218310277>.
- [18] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
 - [19] A.E. Eiben, J.E. Smith, *Introduction to evolutionary computing*, second ed., in: Natural Computing Series, Springer, 2015, <http://dx.doi.org/10.1007/978-3-662-44874-8>.
 - [20] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948, <http://dx.doi.org/10.1109/ICNN.1995.488968>.
 - [21] L. Filep, Z. Gál, Impact of random number generation methods usage on swarm intelligence algorithms for energy optimization in wireless sensor networks, in: 2024 IEEE 3rd Conference on Information Technology and Data Science, CITDS, 2024, pp. 1–8, <http://dx.doi.org/10.1109/CITDS62610.2024.10791391>.
 - [22] J. Liang, X. Zhu, C. Yue, Z. Li, B.-Y. Qu, Performance analysis on knee point selection methods for multi-objective sparse optimization problems, in: 2018 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2018, pp. 1–8.
 - [23] J. Zou, Q. Li, S. Yang, H. Bai, J. Zheng, A prediction strategy based on center points and knee points for evolutionary dynamic multi-objective optimization, *Appl. Soft Comput.* 61 (2017) 806–818, <http://dx.doi.org/10.1016/j.asoc.2017.08.004>.
 - [24] X. Zhang, Y. Tian, Y. Jin, A knee point-driven evolutionary algorithm for many-objective optimization, *IEEE Trans. Evol. Comput.* 19 (6) (2015) 761–776, <http://dx.doi.org/10.1109/TEVC.2014.2378512>.
 - [25] R. Chen, K. Li, Knee point identification based on the geometric characteristic, in: 2021 IEEE International Conference on Systems, Man, and Cybernetics, SMC, 2021, pp. 764–769, <http://dx.doi.org/10.1109/SMC52423.2021.9658848>.
 - [26] D. Wolpert, W. Macready, No free lunch theorems for optimization, *IEEE Trans. Evol. Comput.* 1 (1) (1997) 67–82, <http://dx.doi.org/10.1109/4235.585893>.
 - [27] P. Kerschke, H.H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives, *Evol. Comput.* 27 (1) (2019) 3–45, http://dx.doi.org/10.1162/evco_a_00242.
 - [28] M.A. Muñoz, Y. Sun, M. Kirley, S.K. Halgamuge, Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges, *Inform. Sci.* 317 (2015) 224–245, <http://dx.doi.org/10.1016/j.ins.2015.05.010>, URL <https://www.sciencedirect.com/science/article/pii/S0020025515003680>.
 - [29] A.P. Piotrowski, J.J. Napiorkowski, A.E. Piotrowska, Choice of benchmark optimization problems does matter, *Swarm Evol. Comput.* 83 (2023) 101378, <http://dx.doi.org/10.1016/j.swevo.2023.101378>, URL <https://www.sciencedirect.com/science/article/pii/S2210650223001517>.
 - [30] A. LaTorre, D. Molina, E. Osaba, J. Poyatos, J. Del Ser, F. Herrera, A prescription of methodological guidelines for comparing bio-inspired optimization algorithms, *Swarm Evol. Comput.* 67 (2021) 100973, <http://dx.doi.org/10.1016/j.swevo.2021.100973>, URL <https://www.sciencedirect.com/science/article/pii/S2210650221001358>.
 - [31] P. Sharma, S. Raju, Metaheuristic optimization algorithms: a comprehensive overview and classification of benchmark test functions, *Soft Comput.* 28 (2024) 3123–3186, <http://dx.doi.org/10.1007/s00500-023-09276-5>.
 - [32] A.P. Piotrowski, J.J. Napiorkowski, A.E. Piotrowska, Metaheuristics should be tested on large benchmark set with various numbers of function evaluations, *Swarm Evol. Comput.* 92 (2025) 101807, <http://dx.doi.org/10.1016/j.swevo.2024.101807>, URL <https://www.sciencedirect.com/science/article/pii/S2210650224003456>.
 - [33] G.L. da Silva Fré, J. de Carvalho Silva, F.A. Reis, L. Dias Palhão Mendes, Particle swarm optimization implementation for minimal transmission power providing a fully-connected cluster for the internet of things, in: 2015 International Workshop on Telecommunications, IWT, 2015, pp. 1–7, <http://dx.doi.org/10.1109/IWT.2015.7224573>.
 - [34] A.K. Yadav, A. Sharma, A. Yadav, A. Vijayvargiya, A. Saxena, R. Kumar, Optimization scheme for power transmission in wireless sensor network, in: 2023 International Conference on Power, Instrumentation, Energy and Control, PIECON, 2023, pp. 1–6, <http://dx.doi.org/10.1109/PIECON56912.2023.10085758>.
 - [35] W. Zhao, L. Wang, Z. Zhang, Artificial ecosystem-based optimization: a novel nature-inspired meta-heuristic algorithm, *Neural Comput. Appl.* 32 (13) (2020) 9383–9425.
 - [36] A. Srivastava, P.K. Mishra, A survey on WSN issues with its heuristics and Meta-Heuristics solutions, *Wirel. Pers. Commun.* 121 (1) (2021) 745–814, <http://dx.doi.org/10.1007/s11277-021-08659-x>.
 - [37] S. Kumar, R. Agrawal, A comprehensive survey on meta-heuristic-based energy minimization routing techniques for wireless sensor network: classification and challenges, *J. Supercomput.* 78 (5) (2022) 6612–6663, <http://dx.doi.org/10.1007/s11227-021-04128-1>.
 - [38] Anupriya, A. Malik, Clustering in IoT based wireless sensor network using swarm intelligence and machine learning approaches: Review and future directions, *Procedia Comput. Sci.* 259 (2025) 1024–1033, <http://dx.doi.org/10.1016/j.procs.2025.04.056>, Sixth International Conference on Futuristic Trends in Networks and Computing Technologies (FTNCT06), held in Uttarakhand, India.
 - [39] J.R. Rice, The algorithm selection problem, in: M. Rubinfeld, M.C. Yovits (Eds.), in: *Advances in Computers*, vol. 15, Elsevier, 1976, pp. 65–118, [http://dx.doi.org/10.1016/S0065-2458\(08\)60520-3](http://dx.doi.org/10.1016/S0065-2458(08)60520-3), URL <https://www.sciencedirect.com/science/article/pii/S0065245808605203>.
 - [40] L. Filep, P. Polgar, Z. Gal, WSN power optimization test data, 2024, <https://www.mathworks.com/matlabcentral/fileexchange/157541-wsn-power-optimization-test-data>, MATLAB Central File Exchange.
 - [41] D. Bingham, Virtual library of simulation experiments: Test functions and datasets, 2013, <http://www.sfu.ca/~ssurjano/optimization.html>. (Accessed 06 October 2025).
 - [42] D. Karaboga, B. Basturk, Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems, in: P. Melin, O. Castillo, L.T. Aguilar, J. Kacprzyk, W. Pedrycz (Eds.), *Foundations of Fuzzy Logic and Soft Computing*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 789–798, http://dx.doi.org/10.1007/978-3-540-72950-1_77.
 - [43] W. Zhao, L. Wang, S. Mirjalili, Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications, *Comput. Methods Appl. Mech. Engrg.* 388 (2022) 114194, <http://dx.doi.org/10.1016/j.cma.2021.114194>.
 - [44] B. Abdollahzadeh, F.S. Gharehchopogh, S. Mirjalili, African vultures optimization algorithm: A new nature-inspired metaheuristic algorithm for global optimization problems, *Comput. Ind. Eng.* 158 (2021) 107408, <http://dx.doi.org/10.1016/j.cie.2021.107408>.
 - [45] X.-B. Meng, X. Gao, Y. Liu, H. Zhang, A novel bat algorithm with habitat selection and Doppler effect in echoes for optimization, *Expert Syst. Appl.* 42 (17) (2015) 6350–6364, <http://dx.doi.org/10.1016/j.eswa.2015.04.026>.
 - [46] H.A. Alsattar, A.A. Zaidan, B.B. Zaidan, Novel meta-heuristic bald eagle search optimisation algorithm, *Artif. Intell. Rev.* 53 (3) (2020) 2237–2264, <http://dx.doi.org/10.1007/s10462-019-09732-5>.
 - [47] X.-B. Meng, X. Gao, L. Lu, Y. Liu, H. Zhang, A new bio-inspired optimisation algorithm: Bird swarm algorithm, *J. Exp. Theor. Artif. Intell.* 28 (4) (2016) 673–687, <http://dx.doi.org/10.1080/0952813X.2015.1042530>.
 - [48] X. Meng, Y. Liu, X. Gao, H. Zhang, A new bio-inspired algorithm: Chicken swarm optimization, in: Y. Tan, Y. Shi, C.A.C. Coello (Eds.), *Advances in Swarm Intelligence*, Springer International Publishing, Cham, 2014, pp. 86–94, http://dx.doi.org/10.1007/978-3-319-11857-4_10.
 - [49] Z. kai Feng, W. jing Niu, S. Liu, Cooperation search algorithm: A novel metaheuristic evolutionary intelligence algorithm for numerical optimization and engineering optimization problems, *Appl. Soft Comput.* 98 (2021) 106734, <http://dx.doi.org/10.1016/j.asoc.2020.106734>.
 - [50] B. Abdollahzadeh, F. Soleimani Gharehchopogh, S. Mirjalili, Artificial gorilla troops optimizer: A new nature-inspired metaheuristic algorithm for global optimization problems, *Int. J. Intell. Syst.* 36 (10) (2021) 5887–5958, <http://dx.doi.org/10.1002/int.22535>.
 - [51] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey wolf optimizer, *Adv. Eng. Softw.* 69 (2014) 46–61, <http://dx.doi.org/10.1016/j.advengsoft.2013.12.007>.
 - [52] S. Mirjalili, S.M. Mirjalili, A. Hatamlou, Multi-verse optimizer: a nature-inspired algorithm for global optimization, *Neural Comput. Appl.* 27 (2) (2016) 495–513, <http://dx.doi.org/10.1007/s00521-015-1870-7>.
 - [53] J. Xue, B. Shen, A novel swarm intelligence optimization approach: sparrow search algorithm, *Syst. Sci. Control. Eng.* 8 (1) (2020) 22–34, <http://dx.doi.org/10.1080/21642583.2019.1708830>.
 - [54] V. Satopaa, J. Albrecht, D. Irwin, B. Raghavan, Finding a “kneedle” in a haystack: Detecting knee points in system behavior, in: 2011 31st International Conference on Distributed Computing Systems Workshops, IEEE, 2011, pp. 166–171, <http://dx.doi.org/10.1109/ICDCSW.2011.20>.
 - [55] S. Salvador, P. Chan, Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms, in: 16th IEEE International Conference on Tools with Artificial Intelligence, IEEE, 2004, pp. 576–584, <http://dx.doi.org/10.1109/ICTAL.2004.50>.