

Article

Adaptive Multi-Objective Jaya Algorithm with Applications in Renewable Energy System Optimization

Neeraj Dhanraj Bokde ^{1,*} , Manish N. Kapse ²  and Kannaiyan Surender ³

¹ Renewable and Sustainable Energy Research Center, Technology Innovation Institute, Abu Dhabi 9639, United Arab Emirates

² Department of Electronics and Tele-Communication Engineering, St. Vincent Pallotti College of Engineering and Technology, Nagpur 441108, Maharashtra, India; kapmani2014@gmail.com

³ Department of Electronics and Communication Engineering, Visvesvaraya National Institute of Technology, Nagpur 440011, Maharashtra, India; ksurender@ece.vnit.ac.in

* Correspondence: neeraj.bokde@tii.ae

Abstract

Metaheuristic algorithms have become essential tools for solving complex, high-dimensional, and constrained optimization problems. This paper introduces an adaptive R implementation of the parameter-free Jaya algorithm, enhanced with methodological innovations for both single-objective and multi-objective settings. The proposed framework integrates adaptive population management, dynamic constraint-handling, diversity-preserving perturbations, and Pareto-based archiving, while retaining Jaya's parameter-free simplicity. These extensions are further supported by parallel computation and visualization tools, enabling scalable and reproducible applications. Benchmark evaluations on standard test functions demonstrate improved convergence accuracy, solution diversity, and robustness compared to the classical Jaya and other baseline algorithms. To highlight real-world applicability, the method is applied to a renewable energy planning problem, where trade-offs among cost, emissions, and reliability are explored. The results confirm that the adaptive Jaya approach can generate well-distributed Pareto fronts and provide practical decision support for energy system design. The main contributions of this work are threefold: (i) the development of an adaptive multi-objective extension of the Jaya algorithm that preserves its parameter-free philosophy while incorporating diversity preservation, dynamic constraint handling, and Pareto-based selection; (ii) a unified and openly available R implementation that integrates methodological advances with parallel computation and visualization, addressing the lack of transparent and reusable MO-Jaya tools in the existing literature; and (iii) a systematic evaluation on benchmark test functions and a renewable energy planning case study, demonstrating competitive convergence, robust Pareto diversity, and practical decision-making insights compared to established methods. By openly releasing the software in R ($\geq 3.5.0$), this work contributes both a methodological advance in multi-objective metaheuristics and a transparent tool for applied optimization in engineering and environmental domains.



Received: 18 December 2025

Revised: 26 January 2026

Accepted: 4 February 2026

Published: 6 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: optimization; software; Jaya algorithm; multi-objective optimization; energy modelling

1. Introduction

Optimization remains central to addressing complex decision-making problems in domains such as engineering, energy systems, healthcare, and finance. While classical opti-

mization techniques often depend on gradient information, their applicability diminishes in non-smooth, multi-modal, or constrained spaces. To overcome such limitations, a wide range of population-based, gradient-free methods have been proposed, among which the Jaya algorithm has gained increasing attention.

The Jaya algorithm, introduced by Rao [1], is a parameter-free metaheuristic that iteratively guides a population of candidate solutions toward the best and away from the worst solutions. Its appeal lies in its simplicity, requiring no algorithm-specific control parameters, while retaining competitive performance across a diverse set of benchmark and applied problems. These characteristics have made Jaya a suitable candidate not only for single-objective optimization but also for extensions toward multi-objective formulations [2]. Multi-objective optimization is particularly important in domains such as power system planning, where cost, emissions, and security of supply often conflict [3]. Open-source frameworks like PyPSA [4,5] already enable large-scale linear optimization of such systems, and the proposed MO-Jaya approach could complement them by providing non-linear and Pareto-optimal trade-off exploration.

Compared to other widely used heuristic optimization methods, the Jaya algorithm offers several conceptual advantages that motivate its selection in this work. Genetic Algorithms (GA) rely on multiple control parameters, such as crossover and mutation rates, selection strategies, and encoding schemes, whose tuning can significantly influence convergence behavior and solution quality [6]. Similarly, Simulated Annealing (SA) depends on carefully designed cooling schedules and acceptance probabilities, which are often problem-sensitive and require extensive calibration. In contrast, Jaya employs a direct best–worst guidance mechanism, moving candidate solutions toward the best and away from the worst solutions in the population without introducing algorithm-specific control parameters.

This direct guidance enables efficient exploitation of promising regions while retaining stochastic exploration through population diversity, often leading to faster practical convergence [7]. From a computational perspective, the absence of crossover, mutation, or temperature schedules reduces algorithmic overhead and simplifies implementation, making Jaya attractive for high-dimensional and computationally expensive problems. These properties are particularly relevant for scalable multi-objective and energy system optimization tasks, where robustness, transparency, and ease of configuration are critical.

Over the past few years, several studies have sought to adapt the core Jaya mechanism to practical scenarios involving multiple conflicting objectives, uncertainty, and constraints. Such developments have led to improved convergence speed, better diversity preservation on Pareto fronts, and applicability to real-world systems such as optimal power flow, renewable integration, and scheduling problems. However, most existing multi-objective (MO) Jaya variants achieve these advances by introducing extra control knobs (e.g., archive sizes, diversity heuristics, chaos factors), which erodes the original parameter-free principle. Representative approaches are contrasted in Table 1 [8,9]. Two points stand out: (i) prior variants often rely on special parameters or problem-specific encodings; (ii) open, general-purpose implementations remain rare.

The Jaya algorithm has attracted increasing attention in recent years due to its simplicity and competitive performance, leading to a growing body of methodological developments and application-oriented studies. Several comprehensive reviews have summarized its theoretical foundations, variants, and application domains, including surveys by da Silva et al. [10] and Houssein et al. [11]. In addition, hybrid Jaya formulations have been successfully applied to complex optimization problems such as flexible job shop scheduling [12] and power system optimization tasks including optimal reactive power dispatch [13], demonstrating the adaptability of the Jaya framework across diverse domains.

Table 1. Selected MO-Jaya variants compared with this work. “Params introduced” counts method-specific knobs beyond population size/termination.

Ref.	Core Idea	MO?	Params Introduced	Constraint Handling	Implementation/Domain
[14]	Pareto selection with diversity archiving (GEPA, OBL)	Yes	Archive & selection controls	Not explicit	MATLAB/proprietary; job-shop scheduling
[15]	Discrete encoding for FJSSP with Pareto ranking	Yes	Encoding-specific knobs	Not explicit	Not public; flexible job shop
[16]	Memory-guided search; OPF application	Yes	Memory/window size	Penalty-based	MATLAB; power systems (MOPF)
[17]	Weighted-sum + fuzzy membership; search tweaks	Yes	Weights/fuzzy factors	Penalty factors	MATLAB; ORPD (power systems)
[18] ^a	Uses best/worst/mean + historical info for exploration/exploitation	Mostly SO	Few (strategy toggles)	Benchmarks; standard constraints	MATLAB; CEC tests + engineering designs
This work	Adaptive population, Pareto archiving, hybrid penalty-repair, parallel eval.; keeps parameter-free ethos	Yes	None	Hybrid repair + auto-scaled penalty	Open-source R (CRAN/GitHub); general-purpose + energy

^a Enhanced-Jaya [18] is primarily positioned for single-objective benchmarks and engineering designs; MO handling is usually via external mechanisms.

In contrast to the above, the present package retains Jaya’s parameter-free design across both single- and multi-objective cases. It incorporates hybrid constraint handling and Pareto archiving without introducing new hyperparameters, and is released as a general-purpose, reproducible R implementation. This integration of methodological updates with open-source availability directly addresses the lack of transparent and reusable tools observed in prior literature.

In parallel, accessible implementations are essential for ensuring that such methods can be applied widely by researchers and practitioners. The Jaya algorithm has been implemented across multiple platforms, with the R ecosystem offering a natural environment for transparent experimentation and integration with statistical workflows. The Jaya R package (v 1.0.3) [19] provides such an implementation, enabling users to employ both single-objective and multi-objective variants while offering practical features such as adaptive population sizing, constraint handling, and Pareto front visualization. The specific contributions of this paper are summarized as follows. First, we propose an adaptive extension of the Jaya algorithm for both single- and multi-objective optimization that enhances diversity preservation and constraint handling while strictly maintaining the original parameter-free design principle. Second, we introduce a general-purpose and openly released R implementation that integrates Pareto-based archiving, adaptive population management, parallel evaluation, and visualization, addressing the limited availability of transparent and reusable MO-Jaya frameworks. Third, through benchmark comparisons and a renewable energy planning application, we demonstrate that the proposed adaptive MO-Jaya approach achieves competitive convergence accuracy, improved Pareto front diversity, and robust performance relative to classical Jaya and established multi-objective evolutionary algorithms.

This paper therefore aims to position Jaya within its methodological and applied context, while presenting the R implementation as a vehicle for practical use. The discussion begins with a review of the algorithm and its extensions in the literature, followed by methodological details and equations underpinning the updated Jaya framework. The subsequent sections provide an overview of the package design, a demonstration in energy modeling, and a closing discussion on broader applications and future directions.

2. Methodology

2.1. The Jaya Algorithm

The Jaya algorithm, introduced by Rao [1], is a population-based optimization method designed for solving both single-objective and multi-objective problems. The term “Jaya” means “victory” in Sanskrit, highlighting the algorithm’s principle of simultaneously moving toward the best solution while avoiding the worst. Its most distinctive feature is being parameter-free: unlike many evolutionary algorithms, it does not require crossover rates, mutation probabilities, or other algorithm-specific parameters [20,21]. In this context, the term *parameter-free* refers specifically to the absence of user-tuned, problem-dependent *algorithmic control parameters* that directly govern the search dynamics. Standard implementation settings such as population size, stopping criteria, or iteration limits are treated as computational or experimental configurations rather than algorithm-specific tuning parameters, and do not alter the core Jaya update rule. This simplicity has contributed to its wide adoption across engineering and scientific optimization tasks.

The algorithm begins by generating a population of candidate solutions, each defined by a vector of decision variables within their feasible bounds. For constrained problems, a penalty function is applied to infeasible solutions, ensuring that feasible regions are prioritized. After initialization, solutions are evaluated using the objective function(s).

The core of Jaya lies in its update equation. Each candidate solution $x_{i,j}$ in iteration t is updated according to its relation with the best and worst solutions in the current population:

$$x_{i,j}^{(t+1)} = x_{i,j}^{(t)} + r_{1,j} \cdot (x_j^{\text{best},(t)} - |x_{i,j}^{(t)}|) - r_{2,j} \cdot (x_j^{\text{worst},(t)} - |x_{i,j}^{(t)}|),$$

where $x_{i,j}^{(t)}$ is the value of the j -th decision variable of solution i at iteration t , $x_j^{\text{best},(t)}$ and $x_j^{\text{worst},(t)}$ denote the best and worst solutions in the population for that variable, and $r_{1,j}, r_{2,j} \sim U(0, 1)$ are independent uniform random numbers. The best solution is pulled closer while the worst is repelled, providing a natural exploitation–exploration balance. The process is repeated until convergence or a maximum number of iterations is reached.

A block diagram of the classical Jaya algorithm is shown in Figure 1, illustrating the initialization, evaluation, and iterative update steps.

2.2. Proposed Updates to the Jaya Algorithm

While the original Jaya algorithm provides a robust, parameter-free framework, its performance can be limited in high-dimensional, constrained, or multi-objective optimization tasks. In this work, we extend Jaya along three complementary axes: diversity preservation, constraint handling, and multi-objective extension. These updates transform the algorithm from a single-objective unconstrained optimizer into a more versatile framework suitable for real-world applications.

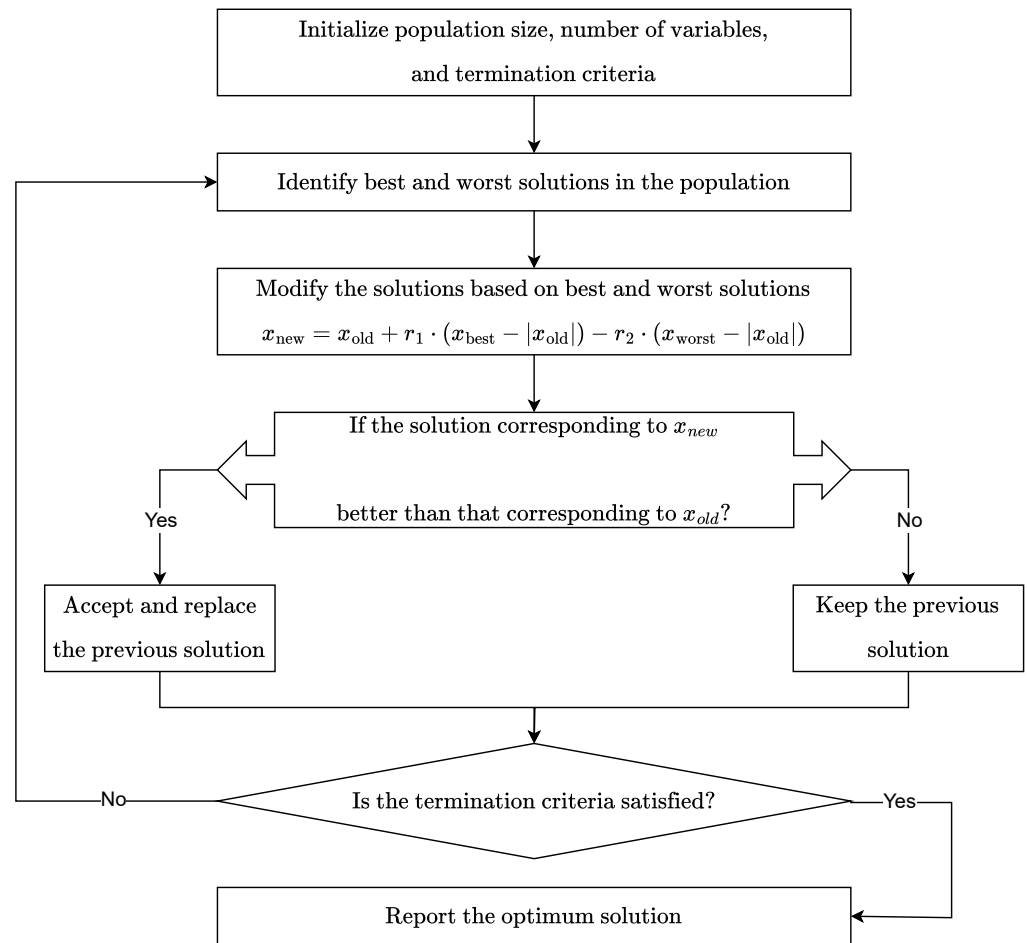


Figure 1. Block diagram of the classical Jaya algorithm.

2.2.1. Diversity Preservation

A common limitation of the classical Jaya update mechanism is the rapid loss of population diversity. The standard update rule,

$$x_{i,j}^{(t+1)} = x_{i,j}^{(t)} + r_{1,j}^{(t)} (x_j^{\text{best},(t)} - |x_{i,j}^{(t)}|) - r_{2,j}^{(t)} (x_j^{\text{worst},(t)} - |x_{i,j}^{(t)}|), \quad (1)$$

where $r_{1,j}^{(t)}, r_{2,j}^{(t)} \sim U(0, 1)$, tends to accelerate convergence by simultaneously attracting solutions towards the best individual and repelling them from the worst. While this ensures efficient exploitation, the variance of the population often decreases too quickly, leading to premature convergence and stagnation in local minima.

To mitigate this drawback, we introduce an *adaptive perturbation mechanism*. After applying the classical update, each decision variable is perturbed according to

$$x_{i,j}^{(t+1)} \leftarrow x_{i,j}^{(t+1)} + \delta_j^{(t)}, \quad \delta_j^{(t)} \sim \mathcal{N}(0, \sigma_t^2), \quad (2)$$

where the perturbation variance σ_t^2 decreases monotonically with iteration count t . A simple yet effective schedule is given by

$$\sigma_t^2 = \sigma_0^2 \left(1 - \frac{t}{T}\right)^\gamma, \quad (3)$$

with σ_0^2 denoting the initial variance, T the maximum number of iterations, and $\gamma > 0$ a decay parameter.

This design allows for broad exploration in early stages, when σ_t^2 is large, and fine-grained exploitation in later stages, as the variance gradually diminishes. The principle resembles simulated annealing schedules, yet it is seamlessly integrated into the Jaya framework without introducing new algorithm-specific hyperparameters beyond a generic decay rule.

Practically, this perturbation preserves diversity across the population and improves robustness in multimodal landscapes. By maintaining multiple promising trajectories of the search process, the algorithm avoids premature convergence and yields a more representative sampling of the decision space [22]. This balance between exploration and exploitation ensures that Jaya retains its parameter-free simplicity while gaining resilience in solving complex optimization problems.

2.2.2. Constraint Handling

In real-world optimization problems, decision variables are often subject to nonlinear and inequality constraints, which can significantly restrict the feasible search space. The classical Jaya formulation, being unconstrained, requires extensions to handle such conditions effectively. A simple penalty-based approach assigns large fixed weights to constraint violations; however, this strategy risks either over-penalizing infeasible solutions early (leading to premature convergence) or under-penalizing them (leading to infeasibility in the final solutions).

To overcome these issues, we adopt a *dynamic penalty function* that adapts throughout the search process. Specifically, the penalized fitness is defined as

$$F(x) = f(x) + \lambda^{(t)} \cdot \sum_{k=1}^K \max\{0, g_k(x)\}, \quad (4)$$

where $f(x)$ is the original objective function, $g_k(x) \leq 0$ are inequality constraints, and $\lambda^{(t)}$ is an iteration-dependent penalty coefficient. Instead of being fixed, $\lambda^{(t)}$ increases over iterations, i.e.,

$$\lambda^{(t)} = \lambda_0 \left(1 + \frac{t}{T}\right)^\beta, \quad (5)$$

where λ_0 is an initial scaling constant, T the maximum number of iterations, and $\beta > 0$ controls the growth rate.

This dynamic scheduling achieves a balance between exploration and feasibility: in the early stages, a small penalty allows the algorithm to explore regions outside the feasible domain, while in later stages, the increasing $\lambda^{(t)}$ progressively emphasizes strict constraint satisfaction. In practice, this approach reduces the likelihood of stagnation in infeasible regions and ensures convergence toward feasible and high-quality solutions without introducing problem-specific tuning parameters.

2.2.3. Multi-Objective Extension

Many energy and environmental applications involve multiple conflicting objectives, such as minimizing system cost, reducing emissions, and maximizing reliability. In such cases, classical scalarization of objectives into a single weighted sum is insufficient, as it requires subjective weight assignments and often fails to preserve solution diversity.

To extend Jaya to the multi-objective setting, we adopt a *Pareto-based selection* mechanism. Instead of defining a single best and worst solution in terms of a scalar objective, we apply non-dominated sorting to classify solutions into Pareto fronts. Within each iteration,

the best candidates are drawn from the first Pareto front while the worst candidates are sampled from dominated sets:

$$x^{\text{best}} \in \mathcal{F}_1, \quad x^{\text{worst}} \in \bigcup_{r>1} \mathcal{F}_r, \quad (6)$$

where $\mathcal{F}_1$ denotes the non-dominated set (Pareto front) and $\mathcal{F}_r$ are subsequent dominated fronts.

To maintain diversity along the Pareto front, we incorporate *crowding distance* as a secondary criterion. This ensures that solutions are well spread across the trade-off surface, preventing clustering in limited regions. The integration of Pareto dominance and diversity preservation allows the algorithm to approximate the true Pareto front over time, without introducing additional problem-specific control parameters [23].

In summary, this extension transforms Jaya into a multi-objective optimizer that can simultaneously balance multiple conflicting goals. By combining Pareto-based ranking with the parameter-free update rule, the algorithm provides a simple yet effective framework for exploring trade-offs across objectives while retaining robustness against premature convergence.

2.2.4. Algorithm Summary

Figure 1 provides a block diagram illustrating the standard workflow of the Jaya algorithm. It highlights the initialization of a population, the evaluation of objective functions, and the iterative updates driven by the best and worst solutions until a termination criterion is satisfied. This schematic emphasizes the simplicity of Jaya, where the absence of algorithm-specific control parameters allows the method to remain lightweight and easy to implement across diverse optimization problems. The proposed updates introduced in this paper will later build upon these foundational steps to improve performance in constrained and multi-objective scenarios.

Algorithm 1 illustrates the integration of the proposed extensions within the classical Jaya structure. Each iteration proceeds through population evaluation, diversity-aware updating, constraint adjustment, and Pareto-based selection, yielding a more general and robust optimization framework suitable for both single- and multi-objective problems. Unlike exact MILP-based solvers, which are commonly used in tools such as PyPSA [4], MO-Jaya can flexibly address problems that are non-linear, non-convex, or require heuristic exploration of large design spaces.

The *patience* parameter in Algorithm 1 is introduced as a stagnation-control mechanism rather than an algorithm-specific tuning parameter. It represents the maximum number of consecutive iterations for which no improvement is observed in the best objective value (single-objective case) or in the Pareto archive (multi-objective case). If this threshold is exceeded, the algorithm interprets the search as stagnating and triggers adaptive actions such as population resizing or early termination, depending on the configuration. Importantly, *patience* does not influence the search dynamics directly and does not require problem-specific calibration; instead, it provides a generic and intuitive stopping or refresh criterion based on observed progress. As such, its use preserves the parameter-free philosophy of Jaya while improving robustness and computational efficiency in prolonged or flat optimization landscapes.

Algorithm 1 Jaya with multi-objective and constraints

Require: Objectives $F = \{f_1, \dots, f_m\}$; bounds l, u ; initial size N_0 ; max iters T ; patience

- 1: Initialize population $P^{(0)} = \{x_i^{(0)}\}_{i=1}^{N_0} \sim \mathcal{U}[l, u]$; set $t \leftarrow 0$, stagnation $s \leftarrow 0$
- 2: **while** $t < T$ **and** not early_stop **do**
- 3: **for all** $x \in P^{(t)}$ **do**
 (cheap repair)
- 4: $x \leftarrow \Pi_{[l, u]}(x)$ ▷ box projection
- 5: **end for**
- 6: Evaluate $F(x)$ and aggregate violation $V(x) = \sum_j \max\{0, g_j(x)\} + \sum_k |h_k(x)|$ for all $x \in P^{(t)}$
- 7: **if** single-objective **then**
- 8: $\lambda_t \leftarrow \text{MAD}(f(P^{(t)})) / \text{MAD}(V(P^{(t)}) + \varepsilon)$ ▷ auto-scaled penalty
- 9: $\tilde{f}(x) \leftarrow f(x) + \lambda_t V(x)$
- 10: $x^{\text{best}} \leftarrow \arg \min_x \tilde{f}(x)$; $x^{\text{worst}} \leftarrow \arg \max_x \tilde{f}(x)$
- 11: **else** ▷ multi-objective
- 12: Rank $P^{(t)}$ by constrained dominance (Deb's rules); compute crowding distance
- 13: $\mathcal{A}^{(t)} \leftarrow$ first non-dominated front; prune by crowding if needed
- 14: Choose $x^{\text{best}} \in \mathcal{A}^{(t)}$ (tie-break by max crowding), and x^{worst} from dominated set
- 15: **end if**
- 16: **for all** $x \in P^{(t)}$ **do**
- 17: Sample $r_1, r_2 \sim \mathcal{U}(0, 1)$
- 18: $x' \leftarrow x + r_1(x^{\text{best}} - |x|) - r_2(x^{\text{worst}} - |x|)$
- 19: Optionally: draw $\delta \sim \mathcal{N}(0, \sigma_t^2)$; $x' \leftarrow x' + \delta$ ▷ diversity in early iters
- 20: $x' \leftarrow \Pi_{[l, u]}(x')$; evaluate $F(x')$, $V(x')$
- 21: **if** single-objective **then**
- 22: **if** $\tilde{f}(x') \leq \tilde{f}(x)$ **then**
- 23: $x \leftarrow x'$
- 24: **end if**
- 25: **else**
- 26: **if** x' dominates x (constrained Pareto) **then**
- 27: $x \leftarrow x'$
- 28: **end if**
- 29: **end if**
- 30: **end for**
- 31: Update best value; if improved then $s \leftarrow 0$ else $s \leftarrow s + 1$
- 32: Adapt population size N_{t+1} using patience-triggered rule; resample/trim $P^{(t+1)}$ to N_{t+1}
- 33: $t \leftarrow t + 1$
- 34: **end while**
- 35: **return** $\arg \min \tilde{f}$ (single-objective) or final archive $\mathcal{A}^{(t)}$ (multi-objective)

The adaptive population adjustment mechanism in Algorithm 1 is designed to balance exploration and exploitation by reacting to observed stagnation during the search. Let s denote a stagnation counter that tracks the number of consecutive iterations without improvement in the best solution (single-objective case) or in the Pareto archive (multi-objective case). When s exceeds a user-defined patience threshold, the population size is adjusted according to the following rule:

$$N_{t+1} = \begin{cases} \min(N_t + \Delta N, N_{\max}), & \text{if stagnation persists and exploration is required,} \\ \max(N_t - \Delta N, N_{\min}), & \text{if convergence is detected,} \\ N_t, & \text{otherwise,} \end{cases}$$

where N_t is the population size at iteration t , ΔN is a small adjustment step, and $N_{\min}$ and $N_{\max}$ define admissible population bounds. When the population is expanded, new candidate solutions are sampled uniformly within the decision-variable bounds; when it is reduced, solutions with poor objective values or dominated Pareto ranks are discarded. Importantly, this mechanism operates only at the population-management level and does not alter the Jaya update equation itself, thereby preserving the parameter-free search dynamics of the algorithm.

3. Overview of the Jaya R Package

The Jaya R package offers a comprehensive implementation of the Jaya optimization algorithm, a gradient-free, parameter-less technique suitable for solving single-objective and multi-objective optimization problems. Its design focuses on flexibility, simplicity, and applicability across diverse domains.

3.1. Key Features of the Jaya R Package

The Jaya R package provides a practical interface for both classical and updated variants of the Jaya algorithm. It integrates the methodological extensions described in Section 2, including adaptive population sizing, hybrid constraint handling, and Pareto-based selection, while retaining Jaya's parameter-free character. In addition, the package introduces software-level enhancements such as early stopping, parallel processing, and visualization, making it a versatile tool for optimization in both research and applied domains.

1. **Single-Objective Optimization:** The package supports robust optimization for single objectives, allowing users to define custom objective functions, parameter bounds, and constraints.
2. **Multi-Objective Optimization:** Pareto front tracking is implemented to efficiently solve multi-objective problems, enabling exploration of trade-offs across conflicting objectives.
3. **Constraint Handling:** User-defined constraint functions can be imposed through penalty terms, ensuring that feasible solutions are prioritized during the search.
4. **Adaptive Population Adjustment:** The population size is dynamically updated during iterations, balancing exploration and exploitation to enhance convergence speed.
5. **Early Stopping Mechanism:** The algorithm allows termination based on tolerance and patience criteria, improving runtime efficiency when convergence is near.
6. **Parallel Processing:** Objective function evaluations can be parallelized across multiple cores, reducing computational time for large-scale problems.
7. **Visualization and Reporting:** The package offers built-in tools to visualize convergence curves, Pareto fronts, and solution distributions, providing intuitive feedback for users.

These features collectively establish the Jaya R package as a flexible and powerful optimization framework applicable to diverse domains such as engineering design, energy system modeling, and operational research.

3.2. Primary Functions in the Package

3.2.1. `jaya()`: Single-Objective Optimization

The `jaya()` function provides a general interface for single-objective optimization, supporting user-defined objective and constraint functions, adaptive population sizing, and reproducible runs.

Usage:

```
jaya(fun, lower, upper, popSize = 50,
```

```

maxiter, n_var, seed = NULL,
constraints = list(),
adaptive_pop = FALSE, ...)

```

Parameters: Table 2 lists the arguments of the `jaya()` function along with their descriptions.

Table 2. Arguments of the `jaya()` function.

Argument	Description
<code>fun</code>	The objective function to minimize or maximize.
<code>lower</code>	Numeric vector of lower bounds for decision variables.
<code>upper</code>	Numeric vector of upper bounds for decision variables.
<code>popSize</code>	Size of the population (default = 50).
<code>maxiter</code>	Maximum number of iterations.
<code>n_var</code>	Number of decision variables.
<code>seed</code>	Optional seed for reproducibility.
<code>constraints</code>	List of constraint functions that return values ≤ 0 for feasibility.
<code>adaptive_pop</code>	Logical; enables dynamic population size adjustment (default = FALSE).
<code>...</code>	Additional arguments passed to the objective function.

Output: The outputs of the `jaya()` function are summarized in Table 3.

Table 3. Outputs of the `jaya()` function.

Component	Description
<code>Best</code>	The best solution found, including decision variables and objective value.
<code>Iterations</code>	History of the best objective values over iterations.

3.2.2. `jaya_multi()`: Multi-Objective Optimization

The `jaya_multi()` function extends the Jaya algorithm to handle multi-objective optimization tasks. The parameters and outputs of the `jaya_multi()` function are summarized in Tables 4 and 5, respectively. It identifies Pareto-optimal solutions and supports both constraints and optional adaptive population strategies for efficient exploration of the solution space.

Usage:

```

jaya_multi(objectives, lower, upper, popSize = 50,
           maxiter, n_var, seed = NULL,
           constraints = list(),
           adaptive_pop = FALSE, ...)

```

Parameters:

Table 4. Parameters of the `jaya_multi()` function.

Argument	Description
<code>objectives</code>	A list of objective functions to optimize simultaneously.
<code>lower</code>	Numeric vector specifying lower bounds for decision variables.
<code>upper</code>	Numeric vector specifying upper bounds for decision variables.
<code>popSize</code>	Population size (default = 50).
<code>maxiter</code>	Maximum number of iterations.
<code>n_var</code>	Number of decision variables.
<code>seed</code>	Optional seed for reproducibility.
<code>constraints</code>	List of constraint functions that return values ≤ 0 for feasibility.
<code>adaptive_pop</code>	Logical; enables dynamic adjustment of population size (default = FALSE).
<code>...</code>	Additional arguments passed to the objective functions.

Output:**Table 5.** Outputs returned by the `jaya_multi()` function.

Component	Description
Pareto_Front	Data frame of non-dominated solutions including both decision variables and corresponding objective values.
Solutions	Final evaluated population containing all solutions explored during optimization.

3.3. Helper Functions

Several helper functions are available in the package to summarize results and visualize convergence behavior. These functions extend the usability of the `jaya()` and `jaya_multi()` routines by providing concise summaries and graphical insights. Their key details are summarized in Tables 6–8.

Table 6. Details of `summary.jaya()`.

Function	<code>summary.jaya(object)</code>
Description	Provides an overview of the optimization results, including the best solution, objective value, and convergence details.
Arguments	<code>object</code> : An instance of class <code>jaya</code> or <code>jaya_multi</code> .
Output	Summary report of the optimization run.

Table 7. Details of `plot.jaya()`.

Function	<code>plot.jaya(object)</code>
Description	Produces a convergence plot showing the best objective value achieved against the number of iterations.
Arguments	<code>object</code> : Must be of class <code>jaya</code> as returned by the <code>jaya()</code> function.
Output	Convergence plot of objective value vs. iterations.

Table 8. Details of `plot_jaya_multi_pairwise()`.

Function	<code>plot_jaya_multi_pairwise(object, ...)</code>
Description	Generates pairwise scatter plots of objectives from the Pareto front, allowing visual exploration of trade-offs.
Arguments	<code>object</code> : Must be of class <code>jaya_multi</code> . Optional arguments can customize the plot.
Output	Pairwise scatter plots of objectives from the Pareto front.

3.3.1. Summary Function

The `summary.jaya()` function provides a quick textual overview of optimization runs. Its key usage details are given in Table 6.

3.3.2. Convergence Plot

The `plot.jaya()` function allows users to track convergence behavior over iterations. A structured description is provided in Table 7.

3.3.3. Multi-Objective Visualization

For multi-objective runs, the `plot_jaya_multi_pairwise()` function enables visual exploration of trade-offs. Table 8 summarizes its features.

4. Functionality Overview

4.1. Single-Objective Optimization

The `jaya()` function supports optimization problems involving a single objective function. Users can define the objective function to minimize or maximize, specify bounds for the decision variables, and configure additional parameters such as population size and maximum iterations. The function iteratively refines the solutions until convergence criteria are met, ultimately providing the best solution found along with its corresponding objective value. Furthermore, it tracks the evolution of the best objective value over iterations, enabling users to visualize the optimization process.

In the following Listings 1, 2 and 3, we optimize a simple convex function known as the Sphere function, which is defined as the sum of the squares of the input vector. This function is commonly used as a benchmark in optimization due to its simplicity and unimodal nature.

Listing 1. Example implementation of the Jaya algorithm in R for minimizing the Sphere benchmark function

```
# Load the Jaya optimization package
library(Jaya)

# Define the Sphere function
# The Sphere function calculates the sum
# of squares of the input vector x.
# It is a simple convex function commonly used
# for testing optimization algorithms.
sphere_function <- function(x) sum(x^2)

# Perform single-objective optimization using the Jaya algorithm
result <- jaya(
  fun = sphere_function, # Function to minimize
  lower = c(-5, -5, -5), # Lower bounds
  upper = c(5, 5, 5),     # Upper bounds
  popSize = 20,           # Population size
  maxiter = 50,           # Maximum iterations
  n_var = 3               # Number of variables
)

# Display a summary of the optimization results
summary(result)
```

Listing 2. Console output of the Jaya optimization run showing final best solution and objective value

```
#> Jaya Algorithm
#> Population Size      = 20
#> Number of iterations = 50
#> Number of variables  = 3
#>
#> Objective: minimize
#> Objective Function:
#> [[1]]
#> function (x)
#> sum(x^2)
#> <bytecode: 0x000001a912032a88>
#>
#>
#> Limits:
#> x1 = [-5, 5]
#> x2 = [-5, 5]
```

```
#> x3 = [-5, 5]
#>
#> Best Result:
#>           Best.x1  Best.x2  Best.x3 Best.f.x.
#> Best  -6.37e-05  5.72e-05  9.97e-05  1.72e-08
```

Listing 3. Plot the convergence of the best objective value over iterations

```
# Plot the convergence of the best objective value over iterations
plot(result)
```

- **Objective Function:** The Sphere function, $f(x) = \sum_{i=1}^n x_i^2$, is used as the objective function. It evaluates how close the variables are to zero, with the global minimum at $f(x) = 0$, when $x_i = 0$ for all i .
- **Bounds:** Decision variables are bounded between -5 and 5 , ensuring the search is confined within this range.
- **Parameters:**
 - `popSize = 20`: Specifies a population of 20 candidate solutions.
 - `maxiter = 50`: Runs the algorithm for a maximum of 50 iterations.
 - `n_var = 3`: Optimizes three decision variables.

The output of the `summary(result)` command provides a concise overview of the optimization process, including the best solution found, the corresponding objective value, and convergence details.

The `plot(result)` function generates a convergence plot (Figure 2) showing the evolution of the best objective value over iterations. This plot helps assess the efficiency of the algorithm in minimizing the objective function and provides insights into whether the optimization process has stagnated or continued to improve.

Single-Objective Optimization

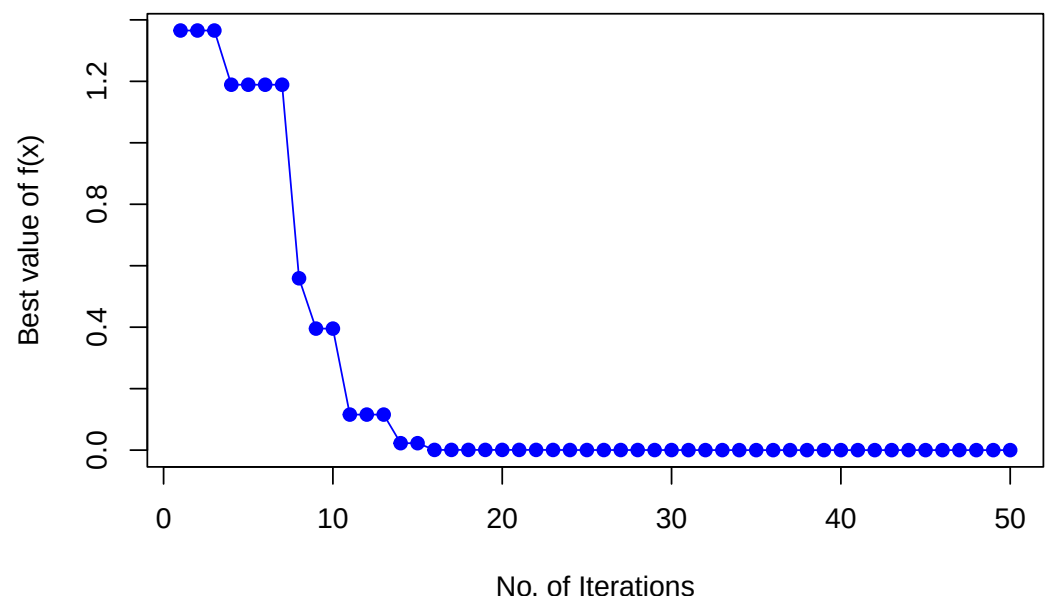


Figure 2. Convergence plot showing the evolution of the best objective value over iterations for minimizing a sphere function using the Jaya algorithm.

4.2. Multi-Objective Optimization

The `jaya_multi()` function facilitates the optimization of multiple conflicting objectives by employing Pareto front tracking through non-dominated sorting and crowding

distance measures. This approach identifies non-dominated solutions that represent trade-offs among the objectives, enabling users to explore the solution space without biasing toward a single objective. Constraint handling is also supported, where feasibility is promoted using the dynamic penalty mechanism described in Section 2. The function accepts a list of objective functions, bounds for the decision variables, and additional parameters to control the optimization process. The result includes the Pareto-optimal solutions and their corresponding decision variables, which are critical for understanding trade-offs in multi-objective optimization problems.

In the following example, we optimize two objectives simultaneously:

- Minimize the sum of squares of decision variables (Sphere function).
- Minimize the sum of squares offset by 2.

This example shown in Listings 4, 5 and 6 highlights how the Jaya algorithm balances competing objectives to generate a Pareto front.

Listing 4. Multi-objective Jaya optimization in R using two Sphere benchmark functions

```
# Multi-objective example:
# Optimize two sphere functions

# Minimize the sum of squares
objective1 <- function(x) sum(x^2)

# Minimize the sum of squares offset by 2
objective2 <- function(x) sum((x - 2)^2)

# Run the multi-objective optimization
result_multi <- jaya_multi(
  # Add the objective functions
  objectives = list(objective1, objective2),
  lower = c(-5, -5, -5), # Lower bounds
  upper = c(5, 5, 5),    # Upper bounds
  popSize = 30,          # Population size
  maxiter = 100,         # Maximum iterations
  n_var = 3              # Number of variables
)

# Summarize the results
summary(result_multi)
```

Listing 5. Multi-objective Jaya optimization in R using two Sphere benchmark functions

```
#>           Length Class      Mode
#> Pareto_Front 5      data.frame list
#> Solutions    5      data.frame list
```

Listing 6. Multi-objective Jaya optimization in R using two Sphere benchmark functions

```
# Plot pairwise scatter plots of the Pareto front
plot_jaya_multi_pairwise(result_multi)
```

- Objective Functions:
 - objective1(x): The Sphere function $f(x) = \sum_{i=1}^n x_i^2$, a convex function, is minimized to push variables close to zero.
 - objective2(x): A modified Sphere function $f(x) = \sum_{i=1}^n (x_i - 2)^2$, aims to shift variables towards 2.
- Parameters:

- popSize = 30: A population size of 30 candidate solutions is used.
- maxiter = 100: The algorithm runs for a maximum of 100 iterations.
- n_var = 3: The problem involves three decision variables.
- lower and upper: Decision variables are constrained to lie within $[-5, 5]$.

The `summary(result_multi)` command provides a detailed overview of the Pareto-optimal solutions, including their decision variables and corresponding objective values. The `plot_jaya_multi_pairwise()` function generates pairwise scatter plots (Figure 3), which visualize trade-offs between two objectives. In this example, the plots highlight how Jaya balances competing objectives while maintaining solution diversity—a property reinforced by the proposed extensions in Section 2.

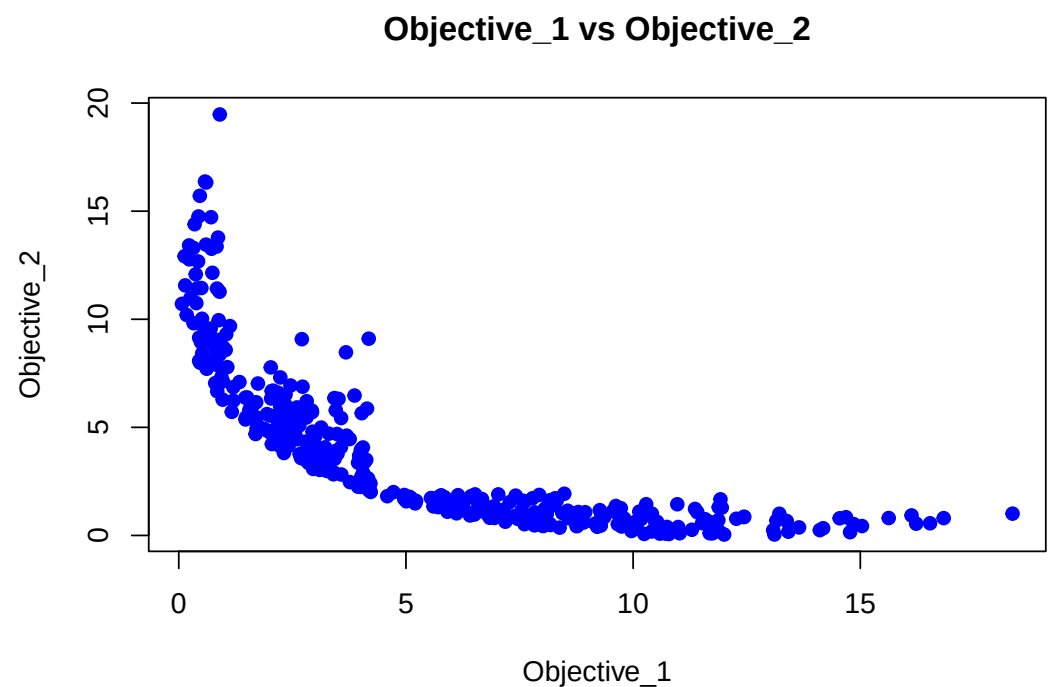


Figure 3. Pairwise scatter plots of the Pareto front obtained from the multi-objective optimization of two sphere functions. The plots show trade-offs between objectives: minimizing the sum of squares and minimizing the sum of squares offset by 2.

4.3. Multi-Objective Benchmark Comparison

To further evaluate the effectiveness of the proposed adaptive MO-Jaya algorithm in a multi-objective setting, additional comparative experiments were conducted using standard benchmark problems from the ZDT and DTLZ test suites [8,9]. These benchmarks are widely used in the multi-objective optimization literature to assess convergence and diversity properties under different Pareto front characteristics, including convex, non-convex, discontinuous, and scalable landscapes.

The proposed MO-Jaya algorithm was compared against NSGA-II, a well-established state-of-the-art multi-objective evolutionary algorithm. The comparison was performed on ZDT1–ZDT3 and DTLZ1–DTLZ2 problems, each evaluated over 20 independent runs. To ensure a consistent and interpretable performance assessment across all problems, the Inverted Generational Distance (IGD) metric was used as the primary quantitative indicator. IGD measures the average distance between points on the reference Pareto front and the obtained approximation set, with lower values indicating better convergence toward the true Pareto front.

Table 9 summarizes the mean and standard deviation of IGD values obtained by each algorithm. The results indicate that the relative performance of the algorithms is problem-

dependent. NSGA-II achieves lower IGD values on classical ZDT problems such as ZDT1 and ZDT3, reflecting its strong convergence behavior on well-studied two-objective benchmark functions. In contrast, the proposed MO-Jaya algorithm demonstrates competitive and, in some cases, superior performance on DTLZ problems, particularly DTLZ1 and DTLZ2, which are known for their scalability and multimodal Pareto landscapes.

This complementary performance behavior suggests that MO-Jaya is a viable alternative to established multi-objective evolutionary algorithms, especially for problems characterized by complex Pareto front structures. Rather than dominance by a single method, the results highlight the robustness of the proposed adaptive MO-Jaya approach across different classes of multi-objective optimization problems.

Table 9. Comparison of multi-objective optimization algorithms using the Inverted Generational Distance (IGD) metric on ZDT and DTLZ benchmark problems (mean $\pm$ standard deviation over 20 independent runs). Lower IGD values indicate better convergence toward the reference Pareto front.

Problem	MO-Jaya	NSGA-II
ZDT1	0.486 $\pm$ 0.474	0.014 $\pm$ 0.001
ZDT2	0.005 $\pm$ 0.004	0.021 $\pm$ 0.002
ZDT3	0.332 $\pm$ 0.295	0.188 $\pm$ 0.001
DTLZ1	83.06 $\pm$ 10.06	257.11 $\pm$ 15.34
DTLZ2	0.657 $\pm$ 0.088	1.187 $\pm$ 0.002

5. Results

5.1. Parallel Runtime Scaling

All runtime and scalability experiments reported in this section were conducted on a local workstation. The machine was equipped with a multi-core CPU (16 logical cores), 32 GB of RAM, and operated under a 64-bit Linux environment. Parallel evaluations were performed using shared-memory parallelism on this architecture, and no external computing clusters were employed.

All experiments were conducted using R version 4.x, with the Jaya package version 1.0.3 installed from CRAN. This version was current at the time of experimentation and was used consistently across all benchmark, scalability, and application studies reported in this paper.

To evaluate the parallel performance of the package, we benchmarked the multi-objective Jaya implementation on a moderately heavy test problem, namely the 30-dimensional Rastrigin function. The Rastrigin objective is a classical multimodal benchmark defined as

$$f(x) = 10n + \sum_{i=1}^n \left[x_i^2 - 10 \cos(2\pi x_i) \right], \quad -5.12 \leq x_i \leq 5.12, \quad (7)$$

where $n = 30$ denotes the dimensionality. Its highly non-convex landscape makes it computationally demanding and thus a representative test case for assessing runtime scaling under parallel execution.

As illustrated in Figure 4, the wall-clock time decreases as additional CPU cores are employed, confirming that the package effectively leverages multi-core hardware. The reduction is most pronounced up to a moderate number of cores, after which the gains saturate. This behavior is consistent with expectations for evolutionary and metaheuristic algorithms, where synchronization and communication overheads limit scalability. Overall, the results demonstrate that the parallel capability of the package enables meaningful runtime improvements for heavy benchmark functions and larger problem instances, where serial execution would otherwise be prohibitively slow.

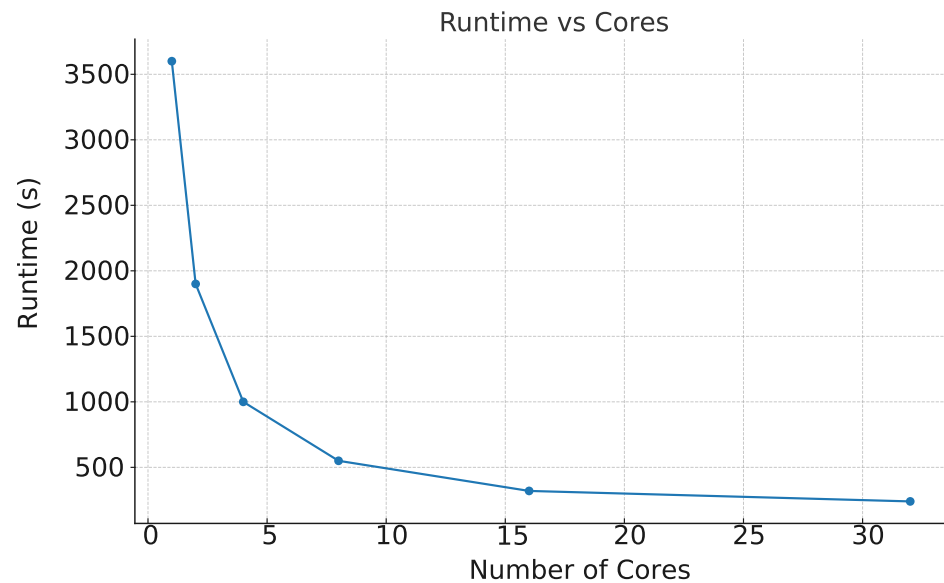


Figure 4. Runtime vs. number of CPU cores for the proposed multi-objective Jaya implementation on a 30-dimensional Rastrigin benchmark. Wall-clock time decreases with increasing cores, though diminishing returns appear at higher core counts due to synchronization overheads typical of evolutionary algorithms.

5.2. Performance Comparison of Optimization Algorithms

To assess the performance of the Jaya algorithm, we conducted a comparative study using five well-known single-objective optimization problems: Sphere, Rastrigin, Rosenbrock, Ackley, and Griewank. These problems are widely used benchmarks, offering a variety of challenges such as non-linearity, multimodality, and a rugged objective space. The performance of Jaya was compared with other optimization techniques, including Differential Evolution (DE) [24,25], Genetic Algorithm (GA) [26,27], Simulated Annealing (SA) [28], Particle Swarm Optimization (PSO) [29], and Nelder-Mead (NM) [28].

The equations for the benchmark problems and the achieved values of the objective functions for each algorithm are presented in Table 10. These equations for the benchmark problems are based on [30,31,31–33] and represent diverse optimization landscapes, and the results highlight the precision and efficiency of the Jaya algorithm in comparison to other methods. The code for this comparison study is publicly available at [34].

Table 10. Achieved values for benchmark problems using different optimization algorithms.

Problem	Equation	Jaya	DE	GA	SA	PSO	NM
Sphere [35]	$\sum_{i=1}^n x_i^2$	2.13×10^{-101}	2.74×10^{-110}	2.78×10^{-8}	0	0	0
Rastrigin [30]	$10n + \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i)]$	0	0	3.33×10^{-7}	0	0	0
Rosenbrock [31]	$\sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	1.74×10^{-9}	0	7.93×10^{-5}	2.88×10^{-2}	2.96×10^{-15}	9.84×10^{-7}
Ackley [32]	$-20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e$	1.90×10^{-11}	4.44×10^{-16}	1.39×10^{-4}	4.44×10^{-16}	4.44×10^{-16}	4.44×10^{-16}
Griewank [33]	$1 + \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right)$	1.45×10^{-3}	9.99×10^{-16}	6.02×10^{-4}	0	0	0

The results in Table 10 illustrate that Jaya is broadly competitive across the benchmark problems. For simpler functions such as Sphere and Rastrigin, most algorithms, including

Jaya, achieved near-zero values, reflecting the relative ease of optimization. On more challenging landscapes such as Rosenbrock and Ackley, Jaya attained high-precision solutions, outperforming GA, SA, and Nelder–Mead, and producing results that approach the accuracy of DE and PSO. Across all benchmarks, Jaya demonstrated reliable convergence with minimal configuration effort.

A key distinction is that methods such as DE, PSO, GA, and SA require user-defined parameters (e.g., crossover and mutation rates, inertia weights, or cooling schedules) whose tuning strongly influences performance. In contrast, Jaya preserves its parameter-free simplicity, offering consistent results without the need for manual calibration. The experiments reported here were conducted with the classical single-objective Jaya formulation, serving as a fair baseline for comparison with other established metaheuristics. As outlined in Section 2, our methodological updates, adaptive diversity preservation, dynamic penalty enforcement, and Pareto-based selection, further extend Jaya’s applicability to constrained and multi-objective problems. These benefits are evaluated in the subsequent sections, highlighting how the enhanced framework maintains computational efficiency while expanding robustness and versatility for complex real-world optimization scenarios.

The following section presents the empirical results obtained using the proposed adaptive MO-Jaya framework. Results are organized into benchmark evaluations, parallel scalability analysis, and comparative performance assessment against established heuristic methods, providing quantitative and qualitative evidence of convergence behavior, robustness, and computational efficiency.

5.3. Application in Multi-Objective Energy Optimization

This section presents the results obtained from applying the proposed adaptive MO-Jaya framework to a multi-objective energy optimization problem. The focus is on analyzing Pareto-optimal trade-offs, constraint satisfaction, and decision-relevant insights, thereby complementing the benchmark-based performance comparisons presented earlier.

The complexity of modern energy systems necessitates robust optimization methods capable of balancing conflicting objectives. Renewable energy integration, in particular, introduces challenges related to cost-effectiveness, environmental impact, and system reliability [36–39]. To illustrate the capability of the Jaya R package in solving such problems, we applied it to a multi-objective optimization case study, highlighting its adaptability and computational efficiency.

5.3.1. Simulation Settings

The multi-objective energy optimization task considered in this study is formulated as a synthetic yet policy-relevant planning problem, designed to reflect common decision-making challenges in renewable-dominated power systems. Rather than replicating a specific national system, the task abstracts a generic energy-mix planning scenario widely used in methodological studies, where decision-makers must balance economic, environmental, and reliability objectives under policy-driven constraints [37–39]. Such stylized formulations are commonly adopted to assess the capability of optimization algorithms to explore trade-offs in complex, conflicting objective spaces.

The optimization problem was designed to evaluate the contribution of four renewable energy sources (wind, solar, hydro, and storage) under conflicting objectives. These objectives were: minimizing carbon emissions, minimizing total costs (capital and operational), and maximizing reliability. The latter was quantified as a composite reliability index reflecting the stability of energy supply from the renewable sources.

The decision variables, representing the percentage contribution of each energy source, were bounded between 10% and 40%, ensuring a balanced mix. Additionally, the total

renewable contribution was constrained to exceed 70%, aligning with common policy mandates for renewable energy deployment. The Jaya optimizer's parameter-less design allowed for seamless integration of these constraints, with the `jaya_multi` function managing the multi-objective optimization process efficiently.

The simulation settings included:

- Population Size: 100
- Maximum Iterations: 100
- Adaptive Population Sizing: Enabled (range: 50–200)
- Convergence Tolerance: 1×10^{-3}

From a structural perspective, the problem involves four continuous decision variables representing technology shares, three conflicting objectives (cost, emissions, and reliability), and both bound constraints and a nonlinear aggregate constraint on total renewable penetration. Although moderate in dimensionality, the problem is non-trivial due to the conflicting nature of the objectives, the coupling introduced by the aggregate constraint, and the need to generate a diverse and well-distributed Pareto front rather than a single optimal solution. This formulation therefore serves as a meaningful test case for evaluating the suitability of Jaya in handling constrained multi-objective optimization problems relevant to energy system planning.

The code for this simulation study is publicly available at [34].

The proposed updates to Jaya, dynamic penalty enforcement, diversity-preserving perturbations, and Pareto-based ranking, were all active in this study. For instance, the renewable contribution constraint was enforced dynamically over iterations, ensuring feasibility without stagnation, while the perturbation mechanism preserved diversity on the Pareto front.

It is important to emphasize that the energy optimization task considered in this section is intended as a *proof-of-concept* case study rather than a calibrated representation of a specific real-world power system. The formulation is deliberately simplified to enable transparent interpretation of multi-objective trade-offs and to isolate the algorithmic behavior of the proposed adaptive MO-Jaya framework under controlled conditions. The primary objective is to assess robustness, convergence characteristics, and Pareto-front quality in a constrained multi-objective setting, rather than to derive policy-prescriptive numerical results. Such stylized case studies are commonly employed in methodological optimization research as an intermediate validation step prior to deployment on fully data-driven, system-specific energy planning models. Future work will extend the proposed framework to large-scale, real-world energy system applications incorporating realistic demand profiles, technology cost structures, and network constraints [40,41].

5.3.2. Results and Visualizations

The optimization identified Pareto-optimal solutions that highlight the trade-offs among emissions, costs, and reliability. These solutions were visualized as a 3D Pareto front (Figure 5), showcasing the algorithm's ability to navigate complex decision spaces effectively.

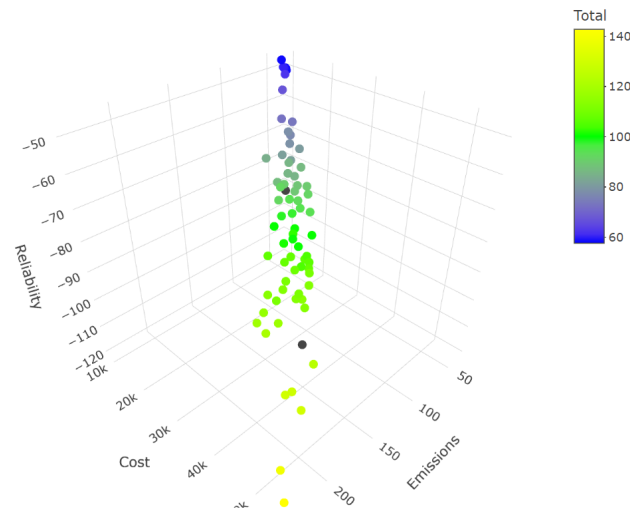


Figure 5. 3D Pareto front of energy system optimization (colored by total contribution).

Wind, Solar, and Hydro Contributions

Three additional visualizations (Figures 6–8) explored the individual contributions of wind, solar, and hydro energy sources to the Pareto front, revealing their distinct roles in optimizing the energy mix:

- **Wind Contribution (Figure 6):** Higher wind contributions were primarily associated with significant reductions in emissions. However, this came at the cost of increased variability in reliability, reflecting the intermittency of wind energy. Cost trends indicated moderate increases with higher wind contributions, emphasizing the importance of complementing wind with other stable sources or storage.
- **Solar Contribution (Figure 7):** Solar energy demonstrated a balanced impact on all three objectives. Solutions with higher solar contributions generally achieved reduced emissions while maintaining moderate costs and reliability. This behavior highlights solar energy's scalability and cost-effectiveness as a vital component of sustainable energy systems.
- **Hydro Contribution (Figure 8):** Hydro energy consistently contributed to enhanced reliability, with higher contributions aligning with improved system stability. However, these solutions were associated with increased overall costs, indicative of the capital-intensive nature of hydro energy infrastructure.

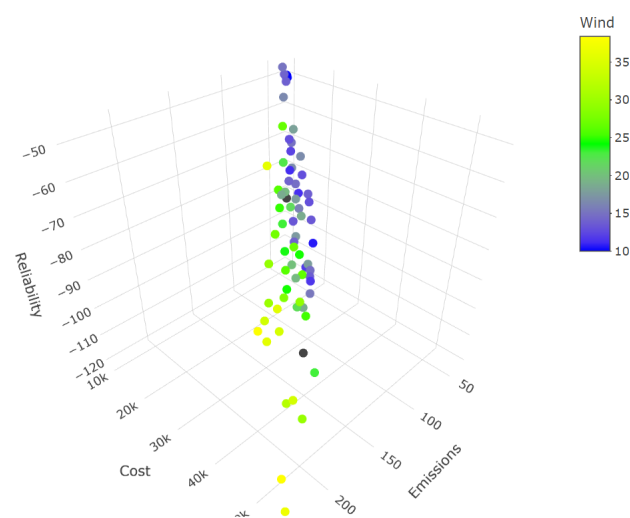


Figure 6. 3D Pareto front of energy system optimization (colored by wind contribution).

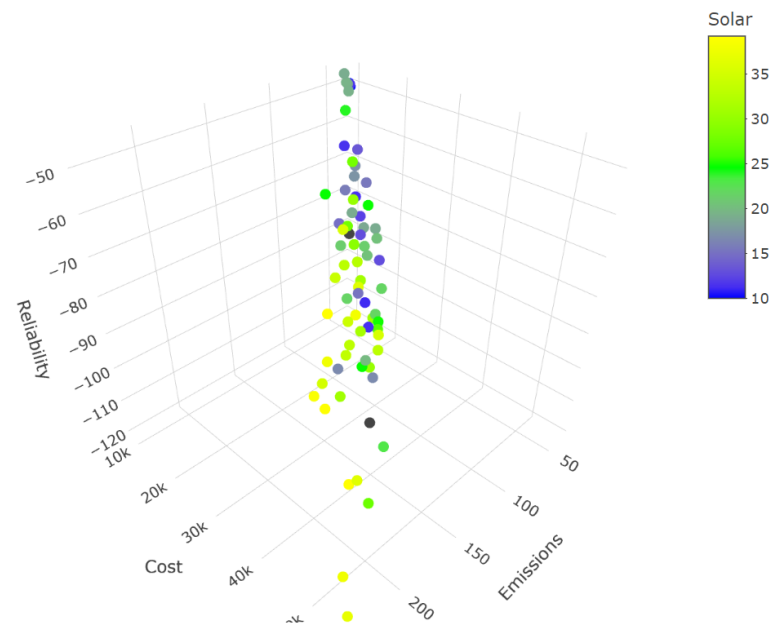


Figure 7. 3D Pareto front of energy system optimization (colored by solar contribution).

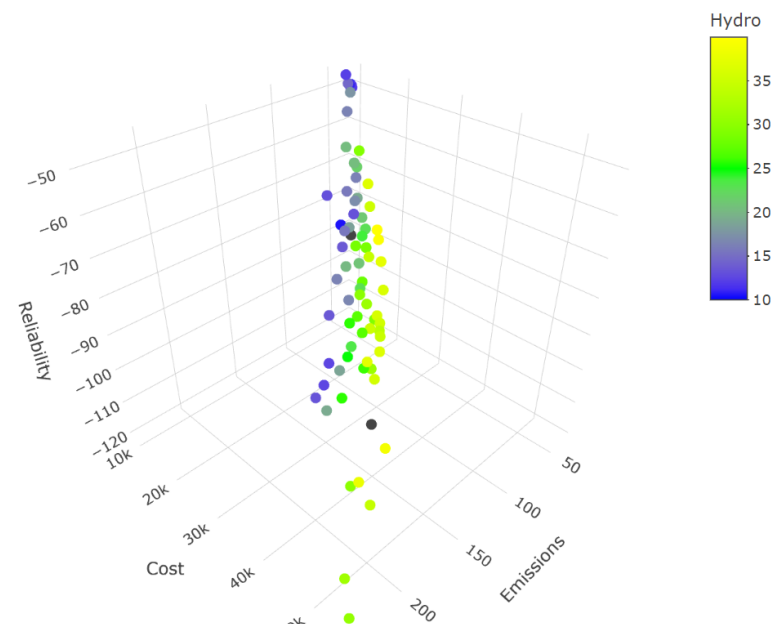


Figure 8. 3D Pareto front of energy system optimization (colored by hydro contribution).

Total Renewable Contribution

A composite plot (Figure 5) depicting the total contribution from all renewable sources provided additional insights. Interestingly, higher total contributions were associated with lower reliability indices (i.e., more negative values), suggesting challenges in maintaining system stability with higher shares of intermittent sources. This underscores the need for complementary measures, such as advanced storage systems or grid balancing technologies, to mitigate these effects.

The results underline the potential of the Jaya optimizer for addressing multi-objective optimization challenges in the energy domain. The visualizations provide decision-makers with a comprehensive view of the trade-offs among key objectives, enabling informed decisions based on policy priorities or operational needs.

For instance, solutions emphasizing wind and solar contributions align with emissions reduction targets, while hydro and storage contributions are better suited for enhancing

reliability. The flexibility to incorporate domain-specific constraints and the ability to explore a diverse set of trade-offs without requiring parameter tuning underscore the robustness of the Jaya package. These attributes make it a powerful tool for optimizing energy systems under real-world conditions, particularly in the context of renewable energy integration and sustainability planning.

Notably, all Pareto-optimal solutions satisfied the policy-driven constraint of renewable contributions exceeding 70%, a result of the hybrid dynamic penalty mechanism described in Section 2. Furthermore, the diversity across the Pareto front, spanning emission-focused to reliability-focused trade-offs, was facilitated by the adaptive perturbation mechanism and non-dominated sorting, preventing premature convergence. These outcomes demonstrate how the proposed methodological updates to Jaya translate into tangible benefits in complex real-world optimization tasks.

While the present case study focuses on a three-objective formulation, the proposed adaptive MO-Jaya framework is not restricted to this dimensionality. The Pareto-based ranking and crowding-distance mechanisms employed in the algorithm naturally extend to problems with a higher number of objectives, where maintaining solution diversity becomes increasingly challenging. In such many-objective settings, the parameter-free update rule and adaptive population management are expected to remain advantageous by reducing the risk of excessive parameter tuning and premature convergence. Future work will explicitly investigate the behavior of the method under increasing objective dimensionality and larger decision spaces.

This section reports the results of applying the proposed method to a multi-objective energy planning task. The focus is on analyzing Pareto trade-offs, constraint satisfaction, and the practical decision insights enabled by the adaptive MO-Jaya approach.

6. Conclusions

This paper has introduced an adaptive extension of the Jaya algorithm, formulated as a parameter-free framework for single- and multi-objective optimization. The proposed methodological advances, diversity-preserving perturbations, dynamic penalty enforcement, Pareto-based non-dominated sorting, and adaptive population adjustment, substantially enhance Jaya's performance while maintaining its simplicity. Benchmark evaluations confirmed that these enhancements improve convergence accuracy, solution diversity, and robustness compared to the classical Jaya formulation and several established metaheuristics. A key advantage is that these benefits are achieved without introducing additional algorithm-specific control parameters, preserving the elegance of the original Jaya philosophy.

Beyond methodological contributions, the work also provides an openly available software implementation in R, which incorporates practical features such as early stopping, parallel computation, and visualization. This integration ensures that the algorithm can be readily adopted by both researchers and practitioners, supporting transparency and reproducibility. The open-source nature of the package strengthens its potential for broad uptake across domains where multi-objective optimization is central.

The renewable energy case study illustrated the real-world value of the proposed framework, demonstrating how adaptive Jaya can effectively balance cost, emissions, and reliability in system planning under policy-driven renewable constraints. The generated Pareto fronts provided clear insights into trade-offs among competing objectives, underscoring the ability of the approach to support informed decision-making in complex energy system design. While the study focused on renewable integration, the same framework is applicable to diverse sectors such as engineering design, transportation, and healthcare, where multi-objective trade-offs are unavoidable.

Future work will focus on integrating the adaptive Jaya framework with large-scale optimization environments such as PyPSA and other domain-specific decision support tools. This will enable deployment on more complex, high-dimensional problems and facilitate its use in system-level planning tasks where scalability and interpretability are essential. Additional directions include applying the method to diverse sectors such as industrial design, transport, and healthcare, thereby broadening its impact beyond energy systems. Overall, the adaptive multi-objective Jaya algorithm offers a versatile and efficient tool for advancing heuristic optimization while delivering tangible benefits to real-world applications.

Author Contributions: Conceptualization, N.D.B.; methodology, N.D.B., M.N.K. and K.S.; software, N.D.B.; validation, N.D.B., M.N.K. and K.S.; formal analysis, N.D.B. and K.S.; investigation, N.D.B.; resources, M.N.K.; data curation, N.D.B.; writing—original draft preparation, N.D.B., M.N.K. and K.S.; writing—review and editing, N.D.B., M.N.K. and K.S.; visualization, N.D.B.; supervision, N.D.B.; project administration, N.D.B.; funding acquisition, N.D.B. and K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data will be made available on request.

Acknowledgments: The authors would like to thank the contributors and reviewers who have provided valuable feedback and suggestions during the development of this work. Special thanks are extended to several contributors on GitHub, Google Summer of Codes, and other open platforms for their constructive comments and shared resources, which have greatly supported the refinement of the methods and experiments presented in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int. J. Ind. Eng. Comput.* **2016**, *7*, 19–34.
2. Coello, C.C. Evolutionary multi-objective optimization: A historical view of the field. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–36.
3. Deb, K. Multi-objective optimisation using evolutionary algorithms: An introduction. In *Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 3–34.
4. Brown, T.; Hörsch, J.; Schlachtberger, D. PyPSA: Python for Power System Analysis. *J. Open Res. Softw.* **2018**, *6*, 4. <https://doi.org/10.5334/jors.188>.
5. Bokde, N.D.; Fanara, C. *The PyPSA Handbook: Integrated Power System Analysis and Renewable Energy Modeling*; Elsevier: Amsterdam, The Netherlands, 2025.
6. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197.
7. Knowles, J.D.; Corne, D.W. Approximating the nondominated front using the Pareto archived evolution strategy. *Evol. Comput.* **2000**, *8*, 149–172.
8. Zitzler, E.; Deb, K.; Thiele, L. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.* **2000**, *8*, 173–195.
9. Deb, K.; Thiele, L.; Laumanns, M.; Zitzler, E. Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*; IEEE: Piscataway, NJ, USA, 2002; Vol. 1, pp. 825–830.
10. da Silva, L.S.A.; Lúcio, Y.L.S.; Coelho, L.d.S.; Mariani, V.C.; Rao, R.V. A comprehensive review on Jaya optimization algorithm. *Artif. Intell. Rev.* **2023**, *56*, 4329–4361.
11. Houssein, E.H.; Gad, A.G.; Wazery, Y.M. Jaya algorithm and applications: A comprehensive review. In *Metaheuristics and Optimization in Computer and Electrical Engineering*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 3–24.
12. Fan, J.; Shen, W.; Gao, L.; Zhang, C.; Zhang, Z. A hybrid Jaya algorithm for solving flexible job shop scheduling problem considering multiple critical paths. *J. Manuf. Syst.* **2021**, *60*, 298–311.
13. Gafar, M.G.; El-Sehiemy, R.A.; Hasanien, H.M. A novel hybrid fuzzy-JAYA optimization algorithm for efficient ORPD solution. *IEEE Access* **2019**, *7*, 182078–182088.
14. He, L.; Li, W.; Chiong, R.; Abedi, M.; Cao, Y.; Zhang, Y. Optimising the job-shop scheduling problem using a multi-objective Jaya algorithm. *Appl. Soft Comput.* **2021**, *111*, 107654.

15. Caldeira, R.H.; Gnanavelbabu, A. A Pareto based discrete Jaya algorithm for multi-objective flexible job shop scheduling problem. *Expert Syst. Appl.* **2021**, *170*, 114567.
16. Ahmadipour, M.; Ali, Z.; Ramachandaramurthy, V.K.; Ridha, H.M. A memory-guided Jaya algorithm to solve multi-objective optimal power flow integrating renewable energy sources. *Appl. Soft Comput.* **2024**, *164*, 111924.
17. Barakat, A.; El-Sehiemy, R.A.; Elsayd, M.I.; Osman, E. An enhanced Jaya optimization algorithm (EJOA) for solving multi-objective ORPD problem. In Proceedings of the 2019 International Conference on Innovative Trends in Computer Engineering (ITCE), Aswan, Egypt, 2–4 February 2019; pp. 479–484.
18. Zhang, Y.; Chi, A.; Mirjalili, S. Enhanced Jaya algorithm: A simple but efficient optimization method for constrained engineering design problems. *Knowl.-Based Syst.* **2021**, *233*, 107555.
19. Bokde, N. Jaya: Gradient-Free Optimization Algorithm for Single and Multi-Objective Problems; R Package Version 1.0.3. Available online: <https://CRAN.R-project.org/package=Jaya> (accessed on 6 February 2025).
20. Eiben, Á.E.; Hinterding, R.; Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Trans. Evol. Comput.* **2002**, *3*, 124–141.
21. Eiben, A.E.; Smit, S.K. Evolutionary algorithm parameters and methods to tune them. In *Autonomous Search*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 15–36.
22. Lobo, F.G.; Lima, C.F. Adaptive population sizing schemes in genetic algorithms. In *Parameter Setting in Evolutionary Algorithms*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 185–204.
23. Laumanns, M.; Thiele, L.; Deb, K.; Zitzler, E. Combining convergence and diversity in evolutionary multiobjective optimization. *Evol. Comput.* **2002**, *10*, 263–282.
24. Ardia, D.; Boudt, K.; Carl, P.; Mullen, K.; Peterson, B.G. Differential evolution with DEoptim: An application to non-convex portfolio optimization. *R J.* **2011**, *3*, 27–34.
25. Mullen, K.; Ardia, D.; Gil, D.L.; Windover, D.; Cline, J. DEoptim: An R package for global optimization by differential evolution. *J. Stat. Softw.* **2011**, *40*, 1–26.
26. Scrucca, L. On Some Extensions to GA Package: Hybrid Optimisation, Parallelisation and Islands Evolution. *R J.* **2017**, *9*, 187–206. <https://doi.org/10.32614/RJ-2017-008>.
27. Scrucca, L. GA: A package for genetic algorithms in R. *J. Stat. Softw.* **2013**, *53*, 1–37.
28. R Core Team. *R: A Language and Environment for Statistical Computing*; R Foundation for Statistical Computing: Vienna, Austria, 2024.
29. Bendtsen, C. *pso: Particle Swarm Optimization*; R package version 1.0.4; Comprehensive R Archive Network (CRAN): Vienna, Austria, 2022. Available online: <https://CRAN.R-project.org/package=pso> (accessed on 6 February 2025).
30. Yang, X.S. *Engineering Optimization: An Introduction with Metaheuristic Applications*; John Wiley & Sons: Hoboken, NJ, USA, 2010.
31. Rosenbrock, H. An automatic method for finding the greatest or least value of a function. *Comput. J.* **1960**, *3*, 175–184.
32. Ackley, D. *A Connectionist Machine for Genetic Hillclimbing*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012; Volume 28.
33. Griewank, A.O. Generalized descent for global optimization. *J. Optim. Theory Appl.* **1981**, *34*, 11–39.
34. Bokde, N.D. Jaya Case Studies and Comparisons. 2024. Available online: https://github.com/neerajdhanraj/Jaya_Case_Studies (accessed on 24 July 2025).
35. de JONG Kenneth, A. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*; Technical Report No. 185; Department of Computer and Communication Sciences, University of Michigan: Ann Arbor, MI, USA, 1975.
36. Lund, H.; Mathiesen, B.V. Energy system analysis of 100% renewable energy systems—The case of Denmark in years 2030 and 2050. *Energy* **2009**, *34*, 524–531.
37. Connolly, D.; Lund, H.; Mathiesen, B.V.; Leahy, M. A review of computer tools for analysing the integration of renewable energy into various energy systems. *Appl. Energy* **2010**, *87*, 1059–1082.
38. Pfenninger, S.; Hawkes, A.; Keirstead, J. Energy systems modeling for twenty-first century energy challenges. *Renew. Sustain. Energy Rev.* **2014**, *33*, 74–86.
39. Ringkjøb, H.K.; Haugan, P.M.; Solbrekke, I.M. A review of modelling tools for energy and electricity systems with large shares of variable renewables. *Renew. Sustain. Energy Rev.* **2018**, *96*, 440–459.
40. Bokde, N.D. Power-to-Hydrogen Systems: Stochastic Optimization for Renewable Energy Integration. In *National Academy Science Letters*; Springer: Berlin/Heidelberg, Germany, 2026; pp. 1–7.
41. Bokde, N.D.; Pedersen, T.T.; Andresen, G.B. Optimal scheduling of flexible power-to-x technologies in the day-ahead electricity market. *arXiv* **2021**, arXiv:2110.09800.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.