



Classical Versus Metaheuristic Optimization in Management, Engineering, and Data Science: A Comparative Computational Study

M. Lin^{1,*}

¹ Department of Industrial Engineering, Faculty of Engineering, Fudan University, Shanghai, China

ARTICLE INFO	ABSTRACT
Article History: Received 20 February 2024 Received in revised form 3 May 2025 Accepted 2 June 2025 Available online 25 June 2025 Keywords: Mathematical Optimization; Metaheuristic Algorithms; Linear and Nonlinear Programming; Genetic Algorithm; Particle Swarm Optimization; Ant Colony Optimization; Management; Engineering; Data Science	Mathematical optimization underpins decision-making in management, engineering and data science, yet there is little clear guidance on when classical methods are sufficient and when metaheuristics are warranted as problem size and complexity increase. This paper reviews classical approaches (the simplex method, gradient-based methods and branch-and-bound) and metaheuristics (genetic algorithms [GA], particle swarm optimization [PSO] and ant colony optimization [ACO]), and compares them computationally on benchmark problems in portfolio optimization, vehicle routing and feature selection in terms of solution quality and computation time over repeated runs. In portfolio optimization, GA and PSO achieved higher mean Sharpe ratios (0.92 and 0.91) than the simplex method (0.85) and gradient descent (0.82), but required longer run times. In a 50-customer vehicle routing problem, ACO and GA found shorter total distances (350 and 355 km) than time-limited branch-and-bound (360 km) in less computation time. In feature selection, GA and PSO achieved higher classification accuracy (0.88 and 0.87) with fewer features than a filter-based baseline (0.81). The results suggest that metaheuristics are advantageous for the nonlinear and combinatorial problems tested, whereas classical methods were faster in the portfolio setting. The paper concludes with practical guidance on algorithm selection and outlines future directions, including hybrid metaheuristics, optimization under uncertainty and integration with machine learning.

1. INTRODUCTION

Over recent decades, mathematical optimization has become a cornerstone of scientific and engineering decision-making, with a prominent role in management, industrial and electrical engineering, computer science, and data science. Many real-world problems can be formulated as linear, nonlinear, convex, nonconvex, integer, or mixed optimization models. Examples include production planning, supply chain management, transportation network design, resource allocation, project scheduling, risk management, and the training of machine learning models. In each case, a decision-maker seeks the best feasible choice according to a quantitative criterion, and the quality of that choice depends on both the model and the algorithm used to solve it.

* Corresponding Author: MeiLin1982@gmail.com

Department of Industrial Engineering, Faculty of Engineering, Fudan University, Shanghai, China



Theoretical advances in convex optimization, notably coherent frameworks for analyzing the convergence and stability of algorithms, have enabled the use of these methods at industrial scale [1]. Efficient numerical methods for large-scale problems have likewise made it practical to solve models with very large numbers of variables and constraints [2]. These developments have been so successful that, for problems with linear or convex structure, practitioners can often rely on mature solvers with strong guarantees on solution quality.

Many problems encountered in practice, however, do not have this favorable structure. Objective functions may be nonconvex, decision variables may be discrete, and the search space may be too large for exact enumeration. At the same time, the convergence of mathematical optimization with data science and machine learning has opened new avenues for scalable, data-driven algorithms. Stochastic gradient methods and their variants, such as Adam [3], [4], are widely used to train deep neural networks and predictive models. For nonconvex and otherwise complex problems on which classical methods perform poorly, metaheuristic and evolutionary algorithms have attracted attention as complementary tools [5]. Progress in integer and mixed-integer optimization, particularly within commercial and open-source solvers, has also been important for planning problems in management and engineering [6]. In multiobjective decision-making, multicriteria methods allow conflicting objectives to be analyzed simultaneously [7].

The growing volume of data, the increasing complexity of engineering systems, and the need for fast, accurate decisions make a systematic comparison of solution methods increasingly relevant. Recent research has emphasized distributed, parallel, and uncertainty-robust algorithms suited to dynamic, large-scale environments [8], [9]. Practitioners and researchers nevertheless face a recurring practical question: given a problem with particular structural characteristics, should one apply a classical method with theoretical guarantees, or a metaheuristic that offers flexibility but no guarantee of global optimality? Answering this question requires evidence drawn from comparable problems across application domains, evaluated under common criteria.

This article has three aims. First, it reviews classical and metaheuristic optimization methods and their relevance to management, engineering, and data science. Second, it compares representative algorithms on benchmark problems from each domain in terms of solution quality and computational time. Third, it discusses practical guidance for algorithm selection and open challenges, including hybrid metaheuristics, optimization under uncertainty, and integration with machine learning. The remainder of the paper is organized as follows. Section 2 reviews background and related work. Section 3 presents the formulations and algorithmic building blocks used later. Section 4 describes the experimental methodology, and Section 5 states the hypotheses. Section 6 reports results, Section 7 discusses them together with limitations, Section 8 considers implementation issues, and Sections 9 and 10 conclude and outline future work.

2. BACKGROUND AND RELATED WORK

2.1. Linear programming and the simplex method

Linear programming was transformed by the simplex algorithm, introduced by Dantzig in 1947, which made industrial-scale resource allocation and planning problems tractable [10]. Because it searches for an optimal vertex of a convex polyhedral region, the simplex method remains a foundational tool in logistics and production management. Although its worst-case running time is exponential, as shown by constructed instances [11], it typically performs efficiently in practice. Interior-point methods later provided polynomial-time guarantees for linear programming and are now a standard alternative [12]. Together, these methods give reliable, globally optimal solutions whenever a problem can be expressed as a linear program.

2.2. Nonlinear programming and gradient-based methods

Research on nonlinear programming produced gradient-based, Newton, and quasi-Newton methods for problems with nonlinear objectives or constraints [2], [13]. Gradient descent moves along the negative gradient and is simple and widely applicable, but its convergence can be slow on ill-conditioned problems. Newton's method uses second-order information to achieve rapid local convergence, at the price of computing and factorizing the Hessian. Quasi-

Newton methods approximate the Hessian from gradient differences, offering a compromise between cost and speed. These methods perform well on convex problems, where local optima are global. On nonconvex functions, however, they often converge to local rather than global optima and are sensitive to initialization.

For large-scale learning problems, stochastic approximation [14] replaces the full gradient with noisy estimates computed on small batches of data. Stochastic gradient methods and their variants are now central to machine learning [3], and adaptive variants such as Adam [4] are used extensively for training neural networks. These methods trade exactness of each step for much lower per-iteration cost.

2.3. Integer and mixed-integer programming

When some decision variables must be discrete, the feasible set is no longer convex and the problem becomes substantially harder. Integer and mixed-integer programming arose to address such problems. Branch-and-bound [15] explores the solution space by recursively partitioning it and pruning branches that cannot contain a better solution, while cutting-plane methods [16] tighten the linear relaxation by adding valid inequalities. Modern solvers combine these ideas with preprocessing and heuristics [6], [17]. Although these exact approaches can solve many practically relevant instances, their computational cost can grow rapidly with problem size, and they may be unable to prove optimality within a fixed time budget on large instances.

2.4. Metaheuristic algorithms

To overcome the limitations of exact algorithms on very large problems, metaheuristics received increasing attention. Metaheuristics are general-purpose search strategies that guide subordinate heuristics to explore the solution space efficiently, and they are usually inspired by natural or physical processes. Genetic algorithms [18] maintain a population of candidate solutions and apply selection, crossover, and mutation, which makes them well suited to exploring large search spaces and to avoiding entrapment in local optima. Simulated annealing [19], inspired by the annealing of metals, accepts worse solutions with a probability that decreases over time, enabling escape from local optima. Particle swarm optimization, introduced by Kennedy and Eberhart [20], updates a swarm of candidate solutions using each particle's own best position and the best position found by the swarm. Ant colony optimization, developed by Dorigo and colleagues [21], [22], builds solutions probabilistically using pheromone trails that are reinforced along good solutions, and it has been particularly successful for routing and other combinatorial problems.

Comprehensive surveys describe the taxonomy, strengths, and open issues of metaheuristics [5], [23]. A central caveat is that these algorithms do not guarantee global optimality. The no-free-lunch theorems [24] further show that, averaged over all possible problems, no search algorithm outperforms any other. Superior performance on a class of problems therefore has to be established empirically for that class, which motivates the comparative design adopted in this study.

2.5. Multiobjective optimization and uncertainty

Many real decisions involve several conflicting objectives. Multiobjective optimization seeks a set of efficient (Pareto-optimal) solutions rather than a single optimum [7], and evolutionary approaches such as NSGA-II [25] approximate this set in a single run. When problem parameters are uncertain, robust optimization [9] seeks solutions that remain feasible and of acceptable quality across a set of plausible scenarios. These extensions are important in practice but are outside the experimental scope of the present study.

2.6. Reinforcement learning

Reinforcement learning (RL) addresses sequential decision problems in which an agent learns a policy through interaction with an environment [26]. In dynamic settings such as financial portfolio management, RL has been

proposed as an alternative to static optimization [27]. Because the present experiments do not include RL, we treat it as a direction for future work.

2.7. Applications in management, engineering, and data science

In management, optimization has been applied to resource allocation, production planning, inventory control, supply chain design, and portfolio selection [28]–[30]. Portfolio theory in particular frames investment as a trade-off between expected return and risk [30]. In logistics, vehicle routing problems are a central class of combinatorial problems, for which both exact and heuristic methods are widely studied [31]. In engineering, optimization plays a key role in the design of structures, control systems, and industrial processes, with an emphasis on models that incorporate physical and performance constraints [32]. In data science, training machine learning models, particularly deep neural networks, is essentially a very large-scale optimization task, and optimization methods have become central to the field [3], [33]. Feature selection, which seeks informative subsets of input variables, is a combinatorial problem of considerable practical importance [34], and hyperparameter tuning is likewise an optimization problem, for which even random search can be a competitive baseline [35]. The relationship between machine learning and combinatorial optimization is now a research area in its own right [36].

2.8. Research gap

Important challenges remain, including scalability to very high-dimensional problems, handling uncertainty in optimization models, automatic selection of the best algorithm for a given problem, and integration of optimization with machine learning [36]. Existing studies tend to examine one algorithm family in one application domain. Systematic comparisons that apply classical and metaheuristic methods to representative problems from several domains under common evaluation criteria are comparatively scarce. The present study addresses part of this gap.

3. PRELIMINARIES

This section collects the formulations and algorithmic building blocks used in the remainder of the paper.

3.1. Linear programming

A linear programming (LP) problem is generally written as

$$\min c^T x \text{ s.t. } Ax \geq b, x \geq 0. \quad (1)$$

It seeks the minimum of a linear function $c^T x$ over the convex region defined by the linear constraints $Ax \geq b$ and $x \geq 0$. This model is applied to resource allocation and planning problems. Because the feasible region is a convex polyhedron and the objective is linear, an optimal solution, when one exists, can be found at a vertex of the region, which is the property exploited by the simplex method.

3.2. Nonlinear programming

A nonlinear programming (NLP) problem is expressed as

$$\text{minimize } f(x) \text{ subject to } g_i(x) \leq 0 \text{ for all } i, h_j(x) = 0 \text{ for all } j, x \in \mathbb{R}^n, \quad (2)$$

where f is a nonlinear objective function, g_i are inequality constraints, and h_j are equality constraints. The decision variables take continuous real values. The main challenge in NLP is the possible presence of multiple local optima. In the unconstrained case, gradient descent iterates

$$x(k+1) = x(k) - \alpha(k) \nabla f(x(k)), \quad (3)$$

where $\alpha(k) > 0$ is a step size, whereas Newton's method uses

$$x(k+1) = x(k) - [\nabla^2 f(x(k))]^{-1} \nabla f(x(k)). \quad (4)$$

Quasi-Newton methods replace the Hessian by an approximation updated from successive gradients [2].

3.3. Genetic algorithms

A genetic algorithm evolves a population of candidate solutions (chromosomes). At each generation, individuals are selected with a probability that favors higher fitness, new candidates are produced by recombining the genetic material of selected parents (crossover), and small random changes are introduced (mutation). The best individuals may be carried over unchanged (elitism). The population-based design allows the search to maintain diversity and sample distant regions of the search space, which reduces the risk of premature convergence [18].

3.4. Particle swarm optimization

In PSO, the velocity and position of particle i are updated as

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot r_1 \cdot (p_i - x_i(t)) + c_2 \cdot r_2 \cdot (g - x_i(t)), \quad x_i(t+1) = x_i(t) + v_i(t+1), \quad (5)$$

where w is the inertia weight [37], c_1 and c_2 are acceleration coefficients, r_1 and r_2 are random numbers, p_i is the personal best position of particle i , and g is the global best position. The inertia weight balances exploration and exploitation: larger values favor wide exploration, and smaller values favor local refinement.

3.5. Ant colony optimization and simulated annealing

In ACO applied to routing, an artificial ant constructs a route by moving from a node to a not-yet-visited node, choosing the next node with a probability that increases with the pheromone level on the connecting edge and with a heuristic desirability, typically the inverse of the distance. After each iteration, pheromone evaporates, and edges belonging to good routes receive additional pheromone, so that promising structures are reinforced over time [21], [22]. In simulated annealing, a candidate move that worsens the objective by Δ is accepted with probability $\exp(-\Delta/T)$, where the temperature T is gradually lowered [19].

4. METHODOLOGY

4.1. Study design

The study compares classical and metaheuristic algorithms on benchmark problems from three domains. The domains and problem types were chosen to cover different structural characteristics: a nonlinear portfolio optimization problem (management), a combinatorial vehicle routing problem (engineering), and a combinatorial feature selection problem (data science). Benchmark problems were designed in each domain, varying in constraint type (linear, nonlinear), variable type (continuous, discrete, mixed), and dimensionality. For each problem class, key parameters and operational constraints were specified to approximate real-world conditions.

4.2. Algorithms compared

The classical group comprised the simplex method (LP), gradient descent (NLP), and branch-and-bound for the discrete routing problem. In the data science problem, a filter-based feature selection method and random selection served as baselines. The metaheuristic group comprised the genetic algorithm (GA), particle swarm optimization (PSO), and, for the routing problem, ant colony optimization (ACO). Each problem was solved only with algorithms applicable to its structure, as reflected in Tables 1–3.

4.3. Implementation

All algorithms were implemented in Python using SciPy [39], DEAP [40], PySwarms [41], and scikit-learn [42]. Python was chosen because these open-source libraries provide tested implementations of classical and metaheuristic algorithms and allow reproducible experimentation.

4.4. Parameter settings and repetitions

Main algorithm parameters were set to standard values or obtained through preliminary tuning on smaller problems. This step is part of the methodology because the performance of metaheuristics depends on parameter choices. Because GA, PSO, and ACO are stochastic, each algorithm was run multiple times on each problem, and mean performance is reported. Results are reported as mean $\pm$ standard deviation where applicable.

4.5. Evaluation metrics

Two metrics were used. Solution quality was measured in a domain-appropriate way: the Sharpe ratio [38] for portfolio optimization (higher is better), the total route distance in kilometers for vehicle routing (lower is better), and the predictive accuracy together with the number of selected features for feature selection. The Sharpe ratio measures risk-adjusted return, defined as the excess return of a portfolio over a risk-free rate divided by the standard deviation of that return. Computational time was recorded in seconds as the time needed to obtain a solution. Together, these metrics describe both the quality and the cost of each method.

4.6. Reporting

Results are organized in comparative tables showing the performance of each algorithm on the defined metrics for each problem. Tables 1–3 report these results, and the analysis in Sections 6 and 7 is based on them.

5. HYPOTHESES

Main hypothesis. Compared with classical optimization methods (e.g., simplex, gradient descent), metaheuristic algorithms (e.g., GA, PSO, ACO) can deliver higher-quality solutions on nonlinear problems and on problems with discrete or combinatorial structure in management, engineering, and data science.

The rationale is that traditional methods designed for simpler problem classes may become limited as problems grow more complex. Metaheuristics, owing to their population-based or probabilistic search, can explore large and complex spaces and are expected to deliver acceptable solutions where exact optimization is difficult or impossible within reasonable time.

Hypothesis 1 (nonlinear problems). On a nonlinear portfolio optimization problem, GA and PSO will achieve higher risk-adjusted returns than classical methods.

Hypothesis 2 (discrete and combinatorial problems). On combinatorial problems such as vehicle routing and feature selection, metaheuristics will yield better solutions than the classical baselines tested.

6. RESULTS

6.1. Management: portfolio optimization

The portfolio problem is nonlinear, with an objective based on the Sharpe ratio [38], which measures risk-adjusted return. Table 1 reports the results.

Table 1. Mean Sharpe ratio and solution time in portfolio optimization (mean $\pm$ SD)

Algorithm	Sharpe ratio	Solution time (s)
Simplex (LP)	0.85 ± 0.03	0.5 ± 0.1
Gradient descent (NLP)	0.82 ± 0.04	1.2 ± 0.3
Genetic algorithm (GA)	0.92 ± 0.02	5.8 ± 1.2
PSO	0.91 ± 0.03	4.5 ± 1.0

GA (0.92) and PSO (0.91) achieved higher mean Sharpe ratios than simplex (0.85) and gradient descent (0.82), indicating that they identified portfolios with a better return-to-risk ratio. They required longer solution times than the classical methods: 5.8 s for GA and 4.5 s for PSO, compared with 0.5 s for simplex and 1.2 s for gradient descent. The gap in solution quality is therefore obtained at a roughly four- to twelve-fold increase in computing time on this problem, which remains small in absolute terms.

6.2. Engineering: vehicle routing

The algorithms were evaluated on a vehicle routing problem (VRP) with 50 customers, a combinatorial optimization problem. Table 2 reports the results.

Table 2. Solution quality and time for the VRP (50 customers)

Algorithm	Total distance (km)	Solution time (s)
Branch-and-bound	360 ± 5	45.8 ± 10.2
ACO	350 ± 3	15.2 ± 2.5
GA	355 ± 4	22.1 ± 3.8

ACO and GA found shorter total distances (350 and 355 km) than branch-and-bound (360 km) in shorter run times (15.2 s and 22.1 s versus 45.8 s). Because branch-and-bound is an exact method, its longer distance implies that the search did not reach proven optimality in this setting. The comparison therefore reflects performance at the reported run times and does not indicate a failure of exact methods in general.

6.3. Data science: feature selection

Feature selection aimed to find the subset of input variables that improves predictive accuracy while reducing model complexity. Table 3 reports the results.

Table 3. Model accuracy and number of features in feature selection

Method	Features selected	Accuracy	Run time (s)
Random selection	—	0.75 ± 0.02	0.1
Filter-based	10	0.81 ± 0.03	0.8
GA	7	0.88 ± 0.01	3.5
PSO	8	0.87 ± 0.02	3.0

Relative to the filter-based baseline, GA raised accuracy by 7 percentage points (0.88 vs. 0.81) while reducing the number of features from 10 to 7 (a 30% reduction). PSO reached an accuracy of 0.87 with 8 features. Both metaheuristics required more time than the baselines (3.5 s and 3.0 s, versus 0.8 s for the filter-based method and 0.1 s for random selection), but the resulting models were both more accurate and more compact.

6.4. Summary of results

Across the three problems, the metaheuristic methods produced the best solution quality in every comparison reported: the highest Sharpe ratios in portfolio optimization, the shortest distances in vehicle routing, and the highest accuracy with the smallest feature subsets in feature selection. In every comparison they also required more time than the simplest classical or baseline method, with the exception of the routing problem, where ACO and GA were faster than branch-and-bound.

7. DISCUSSION

7.1. Interpretation of the findings

Across the three domains, GA, PSO, and ACO produced higher-quality solutions than the classical baselines on the nonlinear portfolio problem and on the combinatorial problems (VRP and feature selection), supporting Hypotheses 1 and 2 and the main hypothesis for the instances tested.

The portfolio result is consistent with the structure of the problem. The Sharpe-ratio objective is nonlinear, so the linear-programming approach can handle it only through a simplified representation, and gradient descent can be trapped in local optima, whereas population-based search can sample the feasible region more broadly. The higher mean Sharpe ratios of GA and PSO are in line with this explanation, although the experiments do not isolate the mechanism.

The routing result illustrates the practical meaning of a fixed computational budget. Exact branch-and-bound offers a proof of optimality only if it is allowed to finish, and on a combinatorial problem of this size it may not. ACO, which constructs solutions with pheromone-guided search, and GA reached shorter routes in less time. This is an "anytime performance" advantage: under a limited time budget, metaheuristics can deliver good solutions early, while exact methods may still be searching. It does not mean that metaheuristics are better than exact methods in general; with sufficient time, an exact method would return the optimal distance.

The feature selection result reflects the combinatorial nature of the problem: choosing a subset of d features involves 2^d possible subsets, which makes exhaustive search impractical for even moderate d . GA and PSO explore this space by encoding subsets as chromosomes or particle positions, and they found subsets that were both smaller and more accurate than the filter-based selection. A plausible explanation is that, unlike a filter that scores features independently, a wrapper-style search evaluates features jointly through the model, thereby capturing interactions among them. This explanation is consistent with the broader feature selection literature [34] but was not tested here.

7.2. Practical guidance for algorithm selection

The results should be read together with the established properties of the methods, which are summarized in Table 4. This table is a qualitative synthesis of the literature reviewed in Section 2 [2], [5], [6], [12], [23], not an experimental result.

Table 4. Qualitative guidance for algorithm selection

Problem characteristic	Preferred approach	Rationale
Linear objective and constraints, continuous variables	Simplex or interior-point methods	Globally optimal solutions with reliable, fast solvers [10], [12]
Smooth convex nonlinear problems	Gradient, Newton, or quasi-Newton methods	Local optimum is global; fast convergence [1], [2]
Smooth nonconvex problems	Multi-start local methods, or metaheuristics	Single-start local methods may reach local optima [2]
Integer or mixed-integer, moderate size	Branch-and-bound/cut in a modern solver	Optimality proofs available within budget [6], [17]
Combinatorial, large size, limited time	ACO, GA, or other metaheuristics	Good solutions early; no optimality guarantee [5], [23]

Black-box or noisy objective, no gradients	GA, PSO, or other derivative-free methods	Require only objective evaluations [5], [23]
Several conflicting objectives	Multiobjective evolutionary algorithms	Approximate the Pareto set in one run [25]

The guiding principle is that algorithm choice should follow problem structure. When the model is linear or convex, classical methods are attractive because they are fast and carry guarantees of optimality. When the problem is nonlinear, discrete, or combinatorial, and when the computational budget is limited, metaheuristics are attractive, provided that the absence of an optimality guarantee is acceptable to the decision-maker. This view agrees with the no-free-lunch theorems [24], which imply that the best algorithm depends on the problem class.

Hybrid strategies are a natural extension. For example, a metaheuristic can supply a good starting point or an incumbent solution to an exact solver, and local search can refine the solutions found by a population-based method. Such combinations seek to retain the exploration ability of metaheuristics and the precision of classical methods [23], [45].

7.3. Limitations

Several limitations apply to this study.

First, the number and size of benchmark instances are limited, so the results may not generalize to other problem classes or to larger instances. The results should be viewed as evidence about the tested problem types rather than as universal statements about the algorithm families.

Second, results are reported as mean $\pm$ SD without formal significance testing, so the differences should be read as descriptive. Future work should apply appropriate statistical procedures for comparing stochastic algorithms, such as nonparametric tests with correction for multiple comparisons [43], [44].

Third, metaheuristic performance depends on parameter tuning, and the classical methods may perform differently under other formulations or solver settings. In particular, because the portfolio objective is nonlinear, the classical baselines represent specific modeling choices, and other formulations might narrow the observed gap.

Fourth, the comparison of exact and heuristic methods in the routing problem is conditional on the time at which the search was stopped. Larger time limits could change the ranking.

Fifth, computational times depend on the hardware, the implementation language, and the library implementations, so they are indicative of relative cost rather than absolute performance.

Finally, scalability with problem size, optimization under uncertainty, multiobjective optimization, and reinforcement learning were not evaluated experimentally here. Claims about these topics rest on the literature, not on the present experiments.

8. PRACTICAL AND IMPLEMENTATION CONSIDERATIONS

8.1. Software tools

Open-source Python libraries make it practical to implement and test the algorithms discussed in this paper. SciPy [39] provides linear programming and nonlinear optimization routines, DEAP [40] provides evolutionary computation tools, PySwarms [41] provides PSO implementations, and scikit-learn [42] provides machine learning models and feature selection utilities. Libraries such as Optuna and Gekko extend this ecosystem to hyperparameter optimization and dynamic optimization. For large mixed-integer and linear models, specialized commercial solvers

such as Gurobi, CPLEX, and LINGO are widely used in industry, and comparing them with open-source alternatives on the same models is a useful practical exercise.

8.2. Modeling effort and skills

The quality of an optimization result depends as much on the formulation as on the algorithm. Translating a business or engineering problem into a mathematical model requires choosing decision variables, objectives, and constraints, and it benefits from close collaboration between modelers and domain experts. Continuing education of practitioners in mathematical modeling and in algorithm selection is therefore important. Decision-support frameworks that guide managers and engineers toward an appropriate algorithm for their problem characteristics, such as the guidance in Table 4, can reduce the risk of misapplying methods.

8.3. Sensitivity analysis and robustness

In practice, input parameters such as demand, prices, or travel times are uncertain. Sensitivity analysis examines how changes in input parameters and constraints affect the quality of the solutions found, and it is especially valuable in problems with high uncertainty. Robust and stochastic formulations [9] provide a more systematic treatment.

8.4. Ethical and social considerations

Optimized decisions can have ethical and social consequences, particularly in resource allocation, policy-making, and other sensitive decisions. An algorithm that maximizes a single numerical criterion may overlook fairness or other values that are not encoded in the objective. Practitioners should therefore consider whether the objective and constraints adequately represent the interests of those affected, and they should be able to explain why a particular solution was selected, especially in domains such as finance and medicine.

9. CONCLUSION

On the benchmark problems studied, metaheuristics (GA, PSO, ACO) gave higher-quality solutions than classical methods for a nonlinear portfolio problem (Sharpe ratio of 0.92 for GA and 0.91 for PSO, against 0.85 for simplex and 0.82 for gradient descent), a 50-customer vehicle routing problem (350 km for ACO and 355 km for GA, against 360 km for branch-and-bound), and a feature selection problem (accuracy of 0.88 for GA and 0.87 for PSO, against 0.81 for the filter-based baseline, with fewer features). In the portfolio setting, this improvement came at the cost of longer run times, whereas in the routing problem the metaheuristics were also faster than the exact method under the conditions tested.

Classical methods remained faster where their assumptions hold, and they offer guarantees that metaheuristics do not. Understanding problem characteristics is therefore essential when selecting an optimization algorithm: linear and convex problems favor classical solvers, whereas nonlinear, discrete, and combinatorial problems under limited computational budgets favor metaheuristics. The limitations discussed above, in particular the small number of instances and the absence of formal significance testing, indicate that these conclusions should be confirmed on larger and more varied benchmarks.

10. FUTURE RESEARCH DIRECTIONS

- **Hybrid metaheuristics:** combining complementary algorithms (e.g., GA with PSO, or ACO with simulated annealing), or combining metaheuristics with exact solvers, to overcome the limitations of each and obtain more robust solutions.
- **Scalability:** systematic analysis of how solution quality and run time change with problem size, including very large datasets and distributed and cloud computing environments, where metaheuristics

such as the lion optimization algorithm have been compared with GA and simulated annealing for server load balancing [46].

- **Uncertainty and multiple objectives:** stochastic and robust formulations, and methods for finding and presenting sets of efficient (Pareto-optimal) solutions in problems with many objectives.
- **Reinforcement learning:** evaluation on dynamic decision problems such as portfolio management [27], including comparison with static optimization.
- **Integration with machine learning:** architectures in which the optimization process is embedded in the learning model, such as learned optimizers or neural networks that guide optimization algorithms [36].
- **Standardized evaluation:** common benchmarks, metrics, and statistical protocols for comparing optimization algorithms under diverse conditions.
- **Explainable optimization:** methods that explain why an algorithm selects a particular solution, especially in sensitive applications such as financial management or medicine.
- **Dynamic and time-dependent constraints:** models and algorithms that respond to changes in problem constraints over time.
- **Emerging applications:** smart grids (energy distribution, load forecasting, integration of renewable sources, and protection coordination, for example the use of grey wolf optimization for overcurrent relay settings in microgrids [47]), robotics and autonomous systems (path planning, motion control, coordination), healthcare (disease modeling, drug discovery, personalized treatment protocols, hospital resource allocation), materials science (design of new materials), and cybersecurity (anomaly detection, security configuration).

Authors' Contributions

All authors read and approved the final manuscript.

Declaration of Generative AI Use

During the preparation of this work, the authors used AI to improve the language and readability of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgments

The authors are grateful to the experts who generously contributed their time and knowledge to this study.

Declaration of Competing Interests

The authors declare no conflict of interest.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Ethical Considerations

Not applicable. This study did not involve human participants, animals or personal data; all computational experiments were performed on benchmark optimization problems.

REFERENCES

- [1] Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press. <https://doi.org/10.1017/CBO9780511804441>

- [2] Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (2nd ed.). Springer. <https://doi.org/10.1007/978-0-387-40065-5>
- [3] Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2), 223–311. <https://doi.org/10.1137/16M1080173>
- [4] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR 2015)*.
- [5] Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, 237, 82–117. <https://doi.org/10.1016/j.ins.2013.02.041>
- [6] Bixby, R. E. (2012). A brief history of linear and mixed-integer programming computation. *Documenta Mathematica, Extra Volume ISMP*, 107–121. <https://doi.org/10.4171/DMJ/6/16>
- [7] Miettinen, K. (1999). *Nonlinear multiobjective optimization*. Kluwer Academic Publishers.
- [8] Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1), 1–122. <https://doi.org/10.1561/22000000016>
- [9] Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). *Robust optimization*. Princeton University Press. <https://doi.org/10.1515/9781400831050>
- [10] Dantzig, G. B. (1963). *Linear programming and extensions*. Princeton University Press. <https://doi.org/10.1515/9781400884179>
- [11] Klee, V., & Minty, G. J. (1972). How good is the simplex algorithm? In O. Shisha (Ed.), *Inequalities III* (pp. 159–175). Academic Press.
- [12] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4), 373–395. <https://doi.org/10.1007/BF02579150>
- [13] Fletcher, R. (1987). *Practical methods of optimization* (2nd ed.). Wiley.
- [14] Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3), 400–407. <https://doi.org/10.1214/aoms/1177729586>
- [15] Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3), 497–520. <https://doi.org/10.2307/1910129>
- [16] Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5), 275–278. <https://doi.org/10.1090/S0002-9904-1958-10224-4>
- [17] Wolsey, L. A. (1998). *Integer programming*. Wiley.
- [18] Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press.
- [19] Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680. <https://doi.org/10.1126/science.220.4598.671>
- [20] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95—International*

- Conference on Neural Networks (Vol. 4, pp. 1942–1948). IEEE. <https://doi.org/10.1109/ICNN.1995.488968>
- [21] Dorigo, M., Maniezzo, V., & Coloni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1), 29–41. <https://doi.org/10.1109/3477.484436>
- [22] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 53–66. <https://doi.org/10.1109/4235.585892>
- [23] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3), 268–308. <https://doi.org/10.1145/937503.937505>
- [24] Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67–82. <https://doi.org/10.1109/4235.585893>
- [25] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197. <https://doi.org/10.1109/4235.996017>
- [26] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.
- [27] Jiang, Z., Xu, D., & Liang, J. (2017). *A deep reinforcement learning framework for the financial portfolio management problem* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.1706.10059>
- [28] Hillier, F. S., & Lieberman, G. J. (2021). *Introduction to operations research* (11th ed.). McGraw Hill.
- [29] Simchi-Levi, D., Kaminsky, P., & Simchi-Levi, E. (2008). *Designing and managing the supply chain: Concepts, strategies, and case studies* (3rd ed.). McGraw-Hill/Irwin.
- [30] Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1), 77–91. <https://doi.org/10.2307/2975974>
- [31] Toth, P., & Vigo, D. (Eds.). (2014). *Vehicle routing: Problems, methods, and applications* (2nd ed.). Society for Industrial and Applied Mathematics. <https://doi.org/10.1137/1.9781611973594>
- [32] Arora, J. S. (2017). *Introduction to optimum design* (4th ed.). Academic Press.
- [33] Sra, S., Nowozin, S., & Wright, S. J. (Eds.). (2011). *Optimization for machine learning*. MIT Press. <https://doi.org/10.7551/mitpress/8996.001.0001>
- [34] Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 1157–1182.
- [35] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, 281–305.
- [36] Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2), 405–421. <https://doi.org/10.1016/j.ejor.2020.07.063>
- [37] Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. In *1998 IEEE International Conference*

on *Evolutionary Computation Proceedings* (pp. 69–73). IEEE. <https://doi.org/10.1109/ICEC.1998.699146>

- [38] Sharpe, W. F. (1966). Mutual fund performance. *The Journal of Business*, 39(1), 119–138. <https://doi.org/10.1086/294846>
- [39] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- [40] Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13, 2171–2175.
- [41] Miranda, L. J. V. (2018). PySwarms: A research toolkit for particle swarm optimization in Python. *Journal of Open Source Software*, 3(21), 433. <https://doi.org/10.21105/joss.00433>
- [42] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- [43] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 1–30.
- [44] Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1), 3–18. <https://doi.org/10.1016/j.swevo.2011.02.002>
- [45] Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. Wiley. <https://doi.org/10.1002/9780470496916>
- [46] Esmaili Moshiran, A., & Babaei, S. (2020). Load balancing of servers in cloud computing using the lion optimization algorithm. *Transactions on Machine Intelligence*, 3(4), 240–250. <https://doi.org/10.47176/TMI.2020.240>
- [47] Liu, H., & Chen, M. (2025). Optimized coordination of overcurrent relays in microgrids using the Grey Wolf Optimization algorithm. *Transactions on Machine Intelligence*, 8(4), 233–247. <https://doi.org/10.22034/tmi.2025.253006>