

From Performance to Understanding: A Vision for Explainable Automated Algorithm Design

Niki van Stein¹ , Anna V. Kononova¹ , and Thomas Bäck¹ 

LIACS, Leiden University, The Netherlands
`{n.van.stein,a.kononova,t.h.w.baeck}@liacs.leidenuniv.nl`

Abstract. Automated algorithm design is entering a new phase: Large Language Models can now generate full optimisation (meta)heuristics, explore vast design spaces and adapt through iterative feedback. Yet this rapid progress is largely performance-driven and opaque. Current LLM-based approaches rarely reveal why a generated algorithm works, which components matter or how design choices relate to underlying problem structures. This paper argues that the next breakthrough will come not from more automation, but from coupling automation with understanding from systematic benchmarking. We outline a vision for explainable automated algorithm design, built on three pillars: (i) LLM-driven discovery of algorithmic variants, (ii) explainable benchmarking that attributes performance to components and hyperparameters and (iii) problem-class descriptors that connect algorithm behaviour to landscape structure. Together, these elements form a closed knowledge loop in which discovery, explanation and generalisation reinforce each other. We argue that this integration will shift the field from blind search to interpretable, class-specific algorithm design, accelerating progress while producing reusable scientific insight into when and why optimisation strategies succeed.

Keywords: Automated Algorithm Design · Explainable Benchmarking · Large Language Models · Landscape Analysis.

1 Introduction

Evolutionary computation (EC) has steadily progressed toward greater *automation and abstraction*: From hand-crafted heuristics, through hyper-parameter optimisation (HPO), algorithm selection and configuration, modular algorithm spaces and now to fully automated algorithm *design* enabled by large language models (LLMs). This shift expands the design space and reduces human bottlenecks, but it also exposes two pressing needs: (i) **explainable benchmarking**, to understand *why* and *when* algorithmic components and hyperparameters matter, (ii) **problem (and instance) descriptors** that capture structural properties of problem *classes* as well as individual problem *instances*. As work in combinatorial optimisation illustrates, even within a single problem class such

as the Travelling Salesperson Problem, instances can vary dramatically in difficulty and structure and meaningful algorithm design must account for these differences.

We argue that combining LLM-driven discovery with explainable benchmarking and principled landscape descriptors offers a path toward *class- and instance-specific automated algorithm discovery*. Such an integration promises not only more effective solvers, but also a deeper scientific understanding of how algorithmic components interact with problem structure, ultimately enabling scalable, interpretable and use-case-specific algorithm design.

2 Past and Current Trends in EC

2.1 A Brief History of Algorithm Design in EC

Evolutionary Computation dates back to the 1960s (see Figure 1), when *Genetic Algorithms* (GA) [32], *Evolutionary Programming* (EP) [26] and *Evolutionary Strategies* (ES) [61,66] were proposed in the US (GA, EP), as well as in Germany (ES). Ever since their invention, their main focus has been on finding near-optimal solutions to non-linear black-box optimisation problems $\min f : \mathcal{D} \rightarrow \mathbb{R}^k$, defined over some domain \mathcal{D} , initially for the single-objective case $k = 1$ (e.g., see [5,24,52]), later on also for multi-objective problems ($k > 1$) (e.g., [21]).

In the late 1980s, automated parameter tuning by a meta-level GA has been introduced into the field [27], later on followed by combined operator selection and parameter tuning [3], as a precursor to what is being called *hyper-parameter optimisation* and *algorithm configuration* today (e.g., see [7] for an overview).

An alternative method, the continuous internal adaptation of certain hyperparameters through a process was called *self-adaptation* [39,67], has been

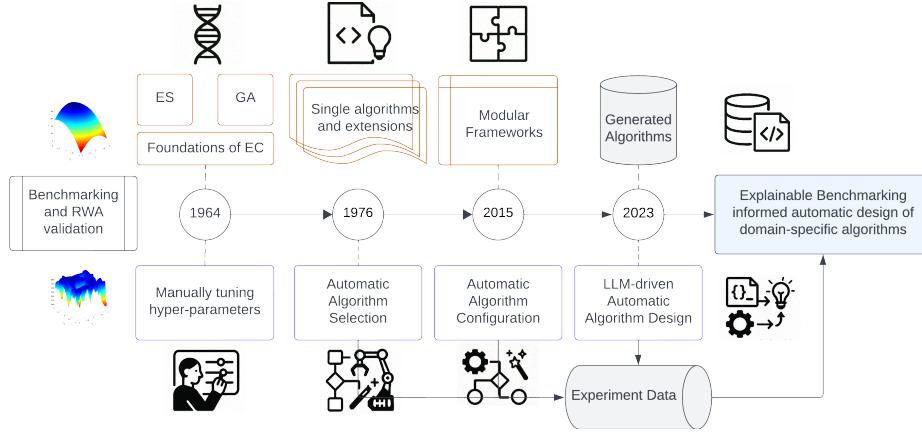


Fig. 1. General timeline of algorithm development in Evolutionary Computation – a vision.

introduced in ES from the beginning, serving as an inspiration over decades towards the *covariance matrix adaptation* (CMA-)ES [28,29] and variants thereof [2].

From the early 1990s, the development of new variants of such algorithms also accelerated significantly due to the integration of the three mainstream methods into the overarching field of *Evolutionary Computation* (EC), enabling hybridisation and cross-fertilisation among those previously separate developments (see [4] or the later version [6] for some examples). Hybridisation took highly sophisticated forms, most notably in *adaptive memetic algorithms* [53,40], which coordinate multiple local search methods tailored to different problem landscapes or optimisation stages, through complex adaptive rules [58]. These mechanisms allow algorithms to adjust their behaviour to diverse landscape characteristics, thereby extending their ability to address increasingly demanding optimisation tasks, with a relatively limited set of coordinated algorithmic components.

Ever more sophisticated algorithms and variations thereof have been proposed by experts in the field, up to a point where even in a subfield such as ES a plethora of algorithmic variants exist [2]. Beyond the field of EC, which now has a firm theoretical foundation based on decades of convergence theory and runtime [22,23], a huge number of nature-inspired metaheuristics has been proposed. However, these are often insufficiently benchmarked (ignoring state-of-the-art algorithms), insufficiently formalised (using imprecise algorithmic and no mathematical formalism) or are minor variations of existing methods - often not even able to compete with random search (see e.g. [15,16,17,70] for a discussion and [82] for a comparison against random search).

As a consequence of the manual, unstructured and incremental process of algorithm development, some researchers started around 2015 to propose modularized algorithm design frameworks in which the combinatorics of algorithmic modules (e.g., many existing variants of mutation, recombination, selection and parameter adaptation operators) is implemented in a way that allows complete enumeration or algorithmic search to be applied to these algorithm configuration design spaces. This approach resulted in the insight that one could systematically find significantly improved variants of algorithms in fields such as differential evolution [81], ES [63], particle swarm optimisation [13,18], differential evolution [81], multi-objective algorithms [11] and heuristics [49], to name the most prominent instances of such modular frameworks.

If one considers the design of optimisation algorithms to be an optimisation process by itself, the development of Large Language Models (LLM) that can generate program code enables a natural step beyond modular frameworks by allowing to generate optimisation algorithm code from scratch. Since the LLM has been trained on existing code, it re-uses existing algorithms and can combine and vary program snippets (similar to modules in a modular framework) thereof in novel ways. This new field of *LLM-driven automated algorithm design in the loop* (a - typically evolutionary - improvement loop is required) started in 2023 with a number of approaches proposed in parallel (e.g., FunSearch [64];

Large Language Model Evolutionary Algorithm, LLaMEA [71,72]; Evolution of Heuristics, EoH [44,87]).

It should be noted that a necessary ingredient for any proper algorithm development - whether manual or automated - is the availability of a sophisticated, unbiased, statistically sound and automated *benchmarking tool* \mathcal{B} , as only this allows the quality of an optimisation algorithm \mathcal{A} to be evaluated by a single scalar performance measure $\mathcal{B}(\mathcal{A}, \mathcal{F}) \in \mathbb{R} \rightarrow \max$ (which is often in practice normalised to be in the interval $[0, 1]$), aggregated on a set \mathcal{F} of test function instances. The most relevant benchmarking platforms are COCO [30], Nevergrad [10] and IOHprofiler [55,83]. Due to its relevance, benchmarking is discussed in more detail in the next section.

2.2 Evolution of Benchmarking Practices

Benchmarking plays a central role in the evaluation and development of optimisation algorithms. In its modern form, benchmarking is understood as a systematic assessment of algorithmic performance on a carefully selected set of problem instances, using clearly defined metrics to compare efficiency, accuracy and robustness under varying conditions [75]. Historically, benchmarking practices have evolved through several distinct stages.

Demonstration of applicability of the algorithm Early benchmarking focused on demonstrating that an algorithm worked on a particular class of problems, typically by plotting average fitness over time across several independent runs. These studies rarely included competing methods and often relied on qualitative, ‘verdict-like’ statements about applicability. This later evolved into comparisons of a small number of algorithms on emerging benchmark suites, although conclusions still tended to rely heavily on visual inspection rather than rigorous analysis.

Extensive convergence plots As encouraged by emerging tools such as COCO [31] and workshops such as the ACM GECCO Black Box Optimization Benchmarking series, a more extensive benchmarking strategy relies on visual analysis of standardised convergence plots for all functions in a suite. While this offers broader insight, interpreting many plots requires expertise and is sometimes complemented—although not always convincingly—by statistical tests or by summary tables that limit interpretability.

Performance per function group Benchmarking practices increasingly consider performance per function group, using high-level properties (e.g. multimodality, global structure, separability) or landscape features obtained via Exploratory Landscape Analysis [50,54] to understand when algorithms succeed or fail.

Domain-specific benchmarking Domain-specific benchmarks highlight that standard suites may not reflect real applications, risking overfitting [12]. This has motivated specialised algorithms for particular domains [37,77].

Modern benchmarking While benchmarking has become somewhat more rigorous, modern practices [8] still face several limitations: (i) ablation studies are often absent, (ii) performance aggregation typically assumes uniform problem distributions [30], while metrics such as absolute runtime distributions or performance profiles may obscure important differences in scalability. Explainability also remains limited, although initial progress has been made through analyses of algorithm complementarity and efforts to relate performance to problem features, for example via instance space analysis [69] or more structured methods for extracting semantic relations [38].

Explainable Benchmarking Recently, a new framework for experimentation and analysis of black-box iterative optimisation heuristics called IOHxplainer has been proposed based on the following [75]:

Explainable Benchmarking formalises the idea of linking an algorithm’s configuration and a benchmark problem to the performance it achieves, enriched with explainability metrics. It considers a family of algorithms, a chosen subset of configurations, a benchmarking suite and an experimental setup and seeks to describe performance as a function of both algorithm parameters and problem characteristics.

Explainable Prediction constitutes a more ambitious goal, where such mappings could be used to anticipate performance on unseen algorithms, unseen problems or untested experimental setups, although achieving this fully remains beyond current computing and methodological capabilities.

2.3 Automated Algorithm Selection, Configuration and Design

Automated Algorithm Selection (AAS) and Automated Algorithm Configuration (AAC) form the foundational layers of modern algorithmic automation. The AAS paradigm was introduced by Rice in 1976 [62] as the problem of mapping instance features to algorithm performance to select the most appropriate solver for each task. Over the past decades, this framework evolved from hand-crafted meta-models to machine-learning-based selectors that learn performance mappings directly from data [35]. Within evolutionary computation, early AAS work incorporated Exploratory Landscape Analysis features [50] to characterise problem instances and enable per-instance algorithm selection [36]. New exploratory landscape analysis approaches such as Deep-ELA [68] highlight progress towards large-scale and explainable selection models that integrate deep representations of problem structures.

In parallel, AAC seeks to automatically identify the best configuration of hyperparameters for a given algorithm and problem class. Frameworks such as ParamILS [34], SMAC [43] and irace [48] established the statistical foundations of configuration as a black-box optimisation problem. Evolutionary and surrogate-based extensions (e.g., SPOT [9], Hyperband [42]) further improved scalability and efficiency. Reviews [33] emphasise that AAC bridges automated tuning and

meta-learning by discovering configuration policies transferable across problem classes.

Together, AAS and AAC underpin the transition towards fully Automated Algorithm Design (AAD). They formalise the learning of algorithm–problem relations and provide the statistical infrastructure upon which modern generative approaches, such as LLM-driven algorithm design, can build. The evolution of AAD can be traced back to the broader development of evolutionary computation and the increasing automation of algorithmic discovery (see Figure 1). Early explorations into self-modifying systems emerged in the 1970s with *Eurisko* [41], one of the first attempts to enable computers to invent their own heuristics. This idea of algorithmic self-improvement was later grounded in evolutionary principles through the work on genetic programming by Koza in the 1990s [59], which demonstrated how evolutionary operators could evolve complete programs and algorithmic structures.

In the early 2000s, the field of *hyper-heuristics* formalised the idea of “heuristics to choose heuristics” [14], marking a conceptual shift from parameter optimisation to the automated generation and selection of heuristic strategies. Throughout the 2010s, research began to focus on the composition and assembly of heuristics, for example through self-assembling design processes [78], and surveys started consolidating the emerging methods under the umbrella of automated heuristic design [57].

In the following decade, general frameworks capable of producing metaheuristic algorithms with diverse internal structures appeared, such as AutoOpt [90], which pushed the boundary beyond configuration towards true algorithm generation. More recent surveys [76,89] positioned AAD as a distinct research field, bridging evolutionary computation, machine learning and software synthesis.

Today, the frontier of AAD integrates large language models as creative agents capable of generating novel algorithmic components or even entire algorithmic frameworks [44,56,64,72,88]. This trajectory, from handcrafted, hand-tuned heuristics to LLM-driven algorithm synthesis illustrates a clear progression toward automation and abstraction in the design of intelligent search processes.

2.4 Motivation

The evolution of algorithm design has followed a cyclical pattern of invention, refinement and automation. We have entered a *new iteration* of this cycle, but now on a higher level of abstraction: rather than designing individual algorithms, we design *methods that design algorithms*. AAD systems, now often driven by LLMs, promise rapid exploration of the open-ended algorithmic search space and continuous improvement through feedback. However, as with every previous wave of automation, new challenges emerge.

Without *explanation* and *understanding*, automation risks turning into blind exploration (e.g. random search). While recent AAD frameworks demonstrate impressive generative capabilities, they rarely provide insight into *why* a generated algorithm performs well, *which* of its components are responsible, or *how* it

relates to the underlying problem characteristics. This limits both reproducibility and scientific understanding (discovery). The field therefore needs systematic methods to attribute performance to algorithmic design choices and hyperparameters and to link these attributions to structural properties of the problems being solved.

Explainable benchmarking and problem-class characterisation can close this gap. By embedding attribution mechanisms and problem descriptors into the AAD pipeline, we can move from empirical discovery toward *interpretable discovery*. Such integration would not only accelerate progress through data-driven feedback loops but also generate reusable design knowledge, bridging the current divide between automated synthesis and human understanding.

Our motivation. This paper argues for combining LLM-driven discovery with explainable benchmarking and problem-class descriptors to enable class-specific, interpretable algorithm design. We advocate a transition from opaque performance improvement to transparent, explainable progress, where each algorithmic innovation contributes to a broader understanding of how structure, performance and design interrelate.

3 The Case for Class-Specific Discovery

As prescribed by the so-called “No Free Lunch” theorems (NFLTs) [85,65], *no* universally superior optimisation algorithm exists. The idea that one might identify the best overall algorithm across all possible problems is therefore devoid of meaning. When *no* assumptions are made about the structure of the problem class the algorithm is to be applied to, no optimisation algorithm can outperform any other in a universal sense. In such a setting, no algorithm can obtain a performance advantage on its own. Any advantage becomes possible only when the problem class is restricted, as this restriction implicitly introduces structure that some algorithms can exploit. By restricting attention to classes of functions with specified regularities or constraints, we move beyond the overly general setting assumed by the NFLTs. These more narrowly defined problem classes may theoretically permit free lunches. A clear example is continuous optimisation, where NFL does not apply because the assumptions required by the theorem cannot be satisfied by continuous function classes [1] and within such settings, some algorithms can indeed be shown to perform better on average.

Introducing additional structure or specialisation creates further opportunities for free lunches. For example, free-lunch phenomena emerge when one considers averages over structured multi-objective problem classes [19,20]. Similar effects arise in co-evolutionary contexts, where interacting populations and coupled objective landscapes induce statistical regularities that can be exploited [86]. These examples reinforce the view that meaningful performance differences become visible only when the problem class departs from the fully unrestricted case.

While continuity is sufficient to break the assumptions of the NFLTs, this insight alone has *limited* practical value. The class of continuous functions remains

so broad that it offers little guidance for understanding or predicting algorithmic performance. It is therefore necessary to consider narrower and more structured problem classes that better reflect the characteristics of real optimisation tasks. Only within such refined settings can we explain observed performance differences and formulate principled recommendations for algorithm design.

From a class-specific perspective, the goal is to use problem descriptors to identify which algorithmic elements and settings are most suitable for a given problem class with structure. However, imposing structure in practice is not straightforward. Exploratory Landscape Analysis [51,50] provides possible principled basis for defining such descriptors. With these in place, discovery can target problem classes [46,80,84], not only individual problem instances and, thus moving in the direction of reusable design knowledge and selectors [47,79].

4 Integrating Discovery and Explanation: A Vision

Automated algorithm design is moving from blind exploration to structured discovery. We envision a *closed knowledge loop*:

1. **Discover:** Use LLM-driven search (e.g., LLaMEA[72]) to propose and refine algorithms over real-world inspired benchmark suites.
2. **Explain:** Run explainable benchmarking techniques to attribute performance to components and settings across different problem classes and instances.
3. **Describe:** Learn problem descriptors that align with observed attributions and cluster functions into classes.
4. **Generalise:** Induce class-specific design rules and selectors; feed these rules back into prompts, mutation policies and priors for the next discovery cycle.

In this loop, large language models act as *creative engines*. They generate, mutate and refine optimisation algorithms from natural language prompts or feedback. Frameworks such as LLaMEA [72] and EoH [44] operationalise this idea by combining LLM-driven code generation with evolutionary selection. Candidate algorithms are rigorously evaluated on benchmark suites, their performance aggregated through measures such as area over the convergence curve or gap to the known best and improved iteratively based on the best-performing designs. To focus LLM capacity on structural innovation rather than numeric tuning, hybrid setups such as LLaMEA-HPO [74] delegate hyper-parameter optimisation to specialised tools like SMAC [43], improving both efficiency and scalability.

A future accelerator of this loop could be a tighter integration of *explainable benchmarking* [6,75]. Traditional benchmarking quantifies performance; explainable benchmarking interprets it. It attributes performance differences to algorithmic components, hyperparameters and their interactions across problem instances and classes. Frameworks such as IOHExplainer [75] build surrogate models over large configuration–performance datasets and apply explainable AI techniques or sensitivity analyses to reveal which parts of an algorithm contribute

most to success. This process transforms experimental data into *actionable* insight, identifying for example, when self-adaptation or recombination operators matter most and why. These insights can be used to steer the LLM-driven search by for example modifying mutation and selection procedures.

Good *problem descriptors* can then close the loop by providing structure on the problem side. ELA [50] and its recent deep extensions (e.g., Deep-ELA [68]) can capture landscape features that cluster functions into interpretable problem classes. Linking these descriptors with algorithmic attributions allows us to discover not just *what* works, but *where* and *why*.

Together, these elements form an iterative cycle: LLMs discover new algorithms, explainable benchmarking analyses them and problem descriptors generalise the findings into reusable design knowledge. The insights gained can then be embedded back into LLM prompts, mutation operators or priors, guiding the next generation of discoveries. This vision shifts automated algorithm design from purely performance based search toward an interpretable, data-driven *science of algorithmic behaviour*. Each iteration not only produces better solvers but also deepens our understanding of the principles behind them.

5 Implications and Research Agenda

The vision outlined above has concrete methodological and scientific consequences for the field. If automated algorithm design is to become *class-specific, explainable and reusable*, several research directions must be prioritised.

Landscape features and problem descriptors A central requirement is the development of richer, more discriminative problem descriptors. Exploratory Landscape Analysis has shown that structural regularities can be extracted from black-box problems, but current descriptors remain limited in scale, sensitivity and real-world applicability. Progress here should be driven by realistic or real-world-inspired problem sets, enabling descriptors that generalise beyond synthetic landscapes. Improved descriptors will directly strengthen AAS models and enable transfer learning within AAD, allowing design knowledge to move across related problem classes.

Advances in AAS, AAD and retrieval-augmented discovery Better problem representations will push developments in automated algorithm selection and configuration. Machine-learning models for AAS can be improved by integrating learned descriptors, while AAD pipelines can benefit from retrieval-augmented generation mechanisms that condition LLMs on prior discoveries, structural rules or examples drawn from known problem classes. This requires systematic work on how to embed algorithm–problem relations into the discovery loop.

Attribution and explainable benchmarking To move beyond blind exploration, the community needs more robust methods for attributing performance to algorithmic components, hyperparameters and their interactions. Explainable benchmarking should become a standard layer in AAD pipelines,

providing principled sensitivity analyses, component-level importance scores and interaction effects across problem classes. Such procedures not only guide discovery but also generate reusable scientific insight.

Encoding design knowledge into prompts Another research direction concerns how to inject algorithmic knowledge into LLM prompts or mutation operators. Encoding rules, motifs, constraints or high-level design principles into prompts can shape the search space and reduce wasted exploration. Equally important is distinguishing between structural innovation (algorithmic modules, operator choices, control mechanisms) and numeric tuning; delegating the latter to specialised HPO tools remains essential for scalability.

Standardisation, benchmarking and tooling The community must invest in shared protocols and tooling. Standard evaluation budgets, anytime performance metrics, aggregation rules and reporting templates will make results comparable and reproducible. Tooling such as IOH [83], BLADE [73] and LLM4AD [45] already provide strong foundations, but widespread adoption requires standard mechanisms for storing, publishing and sharing large benchmarking datasets and experiment metadata and results. Moreover, benchmarking suites need to be developed specifically for AAS and AAD.

From empirical insight to theory Finally, attribution patterns must be mapped back to known algorithmic mechanisms. This creates an opportunity to test falsifiable hypotheses [60,25] about when and why specific components matter. If done systematically, explainable benchmarking can feed the growth of new theoretical results grounded in empirical regularities, closing the loop between data-driven discovery and proof-driven understanding.

Overall, this agenda shifts the field from purely empirical improvement toward a *principled science* of algorithmic behaviour, where automated discovery, structured explanation and theoretical insight reinforce one another.

6 Conclusion

The field is poised to move from automated tuning to *explainable, problem-class-specific* algorithm discovery. LLM-driven design provides exploratory power, explainable benchmarking provides attribution of performance to algorithm components and hyperparameters and finally problem descriptors provide the semantic glue between problems and components. Together, they promise a data-to-design pipeline that learns *which* pieces matter, *why* and *for which* classes, accelerating progress while keeping it interpretable. In doing so, this agenda moves the field beyond purely empirical improvement toward a more *principled science* of algorithmic behaviour, where automated discovery is guided by structural understanding and testable explanations rather than trial and error. The future of this field is a hybridisation between evolutionary methods, artificial intelligence and real-world inspired benchmarking practices.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Auger, A., Teytaud, O.: Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica* **57**, 121–146 (2010)
2. Bäck, T., Foussette, C., Krause, P.: *Contemporary Evolution Strategies*. Springer (2013)
3. Bäck, T.: Parallel optimization of evolutionary algorithms. In: Davidor, Y., Schwefel, H., Männer, R. (eds.) *Parallel Problem Solving from Nature - PPSN III*, International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, October 9-14, 1994, Proceedings. Lecture Notes in Computer Science, vol. 866, pp. 418–427. Springer (1994). https://doi.org/10.1007/3-540-58484-6_285, https://doi.org/10.1007/3-540-58484-6_285
4. Bäck, T.: *Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press (1996)
5. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation* **1**(1), 1–23 (03 1993)
6. Bäck, T.H., Kononova, A.V., van Stein, B., Wang, H., Antonov, K.A., Kalkreuth, R.T., de Nobel, J., Vermetten, D., de Winter, R., Ye, F.: Evolutionary algorithms for parameter optimization—thirty years later. *Evolutionary Computation* **31**(2), 81–122 (2023)
7. Baratchi, M., Wang, C., Limmer, S., van Rijn, J.N., Hoos, H.H., Bäck, T., Olhofer, M.: Automated machine learning: past, present and future. *Artif. Intell. Rev.* **57**(5), 122 (2024). <https://doi.org/10.1007/S10462-024-10726-1>, <https://doi.org/10.1007/S10462-024-10726-1>
8. Bartz-Beielstein, T., Doerr, C., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., López-Ibáñez, M., Malan, K.M., Moore, J.H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., Weise, T.: Benchmarking in optimization: Best practice and open issues. *CoRR abs/2007.03488* (2020), <https://arxiv.org/abs/2007.03488>
9. Bartz-Beielstein, T., Preuss, M.: Experimental research in evolutionary computation. In: *Proceedings of the 9th annual conference companion on genetic and evolutionary computation*. pp. 3001–3020 (2007)
10. Bennet, P., Doerr, C., Moreau, A., Rapin, J., Teytaud, F., Teytaud, O.: Nevergrad: black-box optimization platform. *ACM SIGEVolution* **14**(1), 8–15 (2021)
11. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: Automatically designing state-of-the-art multi- and many-objective evolutionary algorithms. *Evol. Comput.* **28**(2), 195–226 (2020). https://doi.org/10.1162/EVCO_A_00263, https://doi.org/10.1162/evco_a_00263
12. van der Blom, K., Deist, T.M., Volz, V., Marchi, M., Nojima, Y., Naujoks, B., Oyama, A., Tušar, T.: Identifying properties of real-world optimisation problems through a questionnaire. *arXiv preprint arXiv:2011.05547* (2020)
13. Boks, R., Wang, H., Bäck, T.: A modular hybridization of particle swarm optimization and differential evolution. In: Coello, C.A.C. (ed.) *Genetic and Evolutionary Computation Conference, GECCO '20, Companion Volume*, Cancún, Mexico, July 8-12, 2020. pp. 1418–1425. ACM (2020). <https://doi.org/10.1145/3377929.3398123>, <https://doi.org/10.1145/3377929.3398123>

14. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: *Handbook of metaheuristics*, pp. 457–474. Springer (2003)
15. Camacho-Villalón, C.L., Dorigo, M., Stützle, T.: Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by *bestial* metaphors. *Int. Trans. Oper. Res.* **30**(6), 2945–2971 (2023). <https://doi.org/10.1111/ITOR.13176>, <https://doi.org/10.1111/itor.13176>
16. Camacho-Villalón, C.L., Stützle, T., Dorigo, M.: Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty. In: *International conference on swarm intelligence*. pp. 121–133. Springer (2020)
17. Camacho-Villalón, C.L., Dorigo, M., Stützle, T.: An analysis of why cuckoo search does not bring any novel ideas to optimization. *Computers & Operations Research* **142**, 105747 (2022). <https://doi.org/https://doi.org/10.1016/j.cor.2022.105747>, <https://www.sciencedirect.com/science/article/pii/S0305054822000442>
18. Camacho-Villalón, C.L., Dorigo, M., Stützle, T.: PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms. *IEEE Transactions on Evolutionary Computation* **26**(3), 402–416 (2022). <https://doi.org/10.1109/TEVC.2021.3102863>
19. Corne, D.W., Knowles, J.D.: No free lunch and free leftovers theorems for multiobjective optimisation problems. In: *Lecture Notes in Computer Science*, vol. 2632, pp. 327–341. Springer Nature (2003). https://doi.org/10.1007/3-540-36970-8_23
20. Corne, D.W., Knowles, J.D.: Some multiobjective optimizers are better than others. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*. pp. 2506–2512. IEEE Press (2003). <https://doi.org/10.1109/CEC.2003.1299403>
21. Deb, K.: *Multi-objective Optimization using Evolutionary Algorithms*. Wiley, NY (2009)
22. Doerr, B., Neumann, F. (eds.): *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Natural Computing Series, Springer (2020). <https://doi.org/10.1007/978-3-030-29414-4>, <https://doi.org/10.1007/978-3-030-29414-4>
23. Doerr, B., Neumann, F.: A survey on recent progress in the theory of evolutionary algorithms for discrete optimization. *ACM Transactions on Evolutionary Learning and Optimization* **1**(4), 16:1–16:43 (2021). <https://doi.org/10.1145/3472304>, <https://doi.org/10.1145/3472304>
24. Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computing*. Natural Computing Series, Springer (2003). <https://doi.org/10.1007/978-3-662-05094-1>, <https://doi.org/10.1007/978-3-662-05094-1>
25. Eiben, A.E., Jelasity, M.: A critical note on experimental research methodology in ec. In: *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*. pp. 582–587. IEEE (2002). <https://doi.org/10.1109/CEC.2002.1007002>
26. Fogel, L.J., Owens, A.J., Walsh, M.J.: Intelligent decision-making through a simulation of evolution. *IEEE Transactions on Human Factors in Electronics* **HFE-6**(1), 13–23 (1965). <https://doi.org/10.1109/THFE.1965.6591252>
27. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics* **16**(1), 122–128 (1986). <https://doi.org/10.1109/TSMC.1986.289288>

28. Hansen, N.: The CMA evolution strategy: A tutorial. CoRR **abs/1604.00772** (2016), <http://arxiv.org/abs/1604.00772>
29. Hansen, N., Arnold, D.V., Auger, A.: Evolution strategies. In: Springer handbook of computational intelligence, pp. 871–898. Springer (2015)
30. Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: A platform for comparing continuous optimizers in a black-box setting. Optimization Methods and Software **36**(1), 114–144 (2021)
31. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions (2009)
32. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge, MA, USA (1992, 1st edition: 1975)
33. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Autonomous search, pp. 37–71. Springer (2012)
34. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: an automatic algorithm configuration framework. Journal of artificial intelligence research **36**, 267–306 (2009)
35. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evolutionary Computation **27**(1), 3–45 (2019)
36. Kerschke, P., Trautmann, H.: Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evolutionary computation **27**(1), 99–127 (2019)
37. Kohira, T., Kemmotsu, H., Akira, O., Tatsukawa, T.: Proposal of benchmark problem based on real-world car structure design optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. p. 183–184. GECCO ’18, Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3205651.3205702>
38. Kostovska, A., Vermetten, D., Doerr, C., Džeroski, S., Panov, P., Eftimov, T.: Option: Optimization algorithm benchmarking ontology. IEEE Transactions on Evolutionary Computation pp. 1–1 (2022). <https://doi.org/10.1109/TEVC.2022.3232844>
39. Kramer, O.: Evolutionary self-adaptation: a survey of operators and strategy parameters. Evol. Intell. **3**(2), 51–65 (2010). <https://doi.org/10.1007/S12065-010-0035-Y>, <https://doi.org/10.1007/s12065-010-0035-y>
40. Krasnogor, N.: Studies on the theory and design space of memetic algorithms. Ph.D. thesis, University of the West of England, Bristol (2002)
41. Lenat, D.B.: Eurisko: a program that learns new heuristics and domain concepts: the nature of heuristics iii: program design and results. Artificial intelligence **21**(1–2), 61–98 (1983)
42. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18**(185), 1–52 (2018)
43. Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: SMAC3: a versatile Bayesian optimization package for hyperparameter optimization. J. Mach. Learn. Res. **23**, 54–1 (2022)
44. Liu, F., Tong, X., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., Zhang, Q.: Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21–27, 2024. OpenReview.net (2024), <https://openreview.net/forum?id=BwAkaxqiLB>

45. Liu, F., Zhang, R., Xie, Z., Sun, R., Li, K., Lin, X., Wang, Z., Lu, Z., Zhang, Q.: Llm4ad: A platform for algorithm design with large language model. arXiv preprint arXiv:2412.17287 (2024)
46. Long, F.X., van Stein, B., Frenzel, M., Krause, P., Gitterle, M., Bäck, T.: Learning the characteristics of engineering optimization problems with applications in automotive crash. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 1227–1236. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3512290.3528712>, <https://doi.org/10.1145/3512290.3528712>
47. Long, F.X., van Stein, N., Frenzel, M., Krause, P., Gitterle, M., Bäck, T.: Surrogate-based automated hyperparameter optimization for expensive automotive crashworthiness optimization. Structural & Multidisciplinary Optimization **68**(4) (2025). <https://doi.org/10.1007/s00158-025-03989-x>, <https://doi.org/10.1007/s00158-025-03989-x>
48. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
49. Martín-Santamaría, R., López-Ibáñez, M., Stützle, T., Colmenar, J.M.: On the automatic generation of metaheuristic algorithms for combinatorial optimization problems. Eur. J. Oper. Res. **318**(3), 740–751 (2024). <https://doi.org/10.1016/J.EJOR.2024.06.001>, <https://doi.org/10.1016/j.ejor.2024.06.001>
50. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of Genetic and Evolutionary Computation Conference. pp. 829–836. GECCO '11 (2011)
51. Mersmann, O., Preuss, M., Trautmann, H.: Benchmarking evolutionary algorithms: Towards exploratory landscape analysis. In: International Conference on Parallel Problem Solving from Nature. pp. 73–82. Springer (2010)
52. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, Third Revised and Extended Edition. Springer (1996). <https://doi.org/10.1007/978-3-662-03315-9>, <https://doi.org/10.1007/978-3-662-03315-9>
53. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Caltech Con-Current Computation Program 158-79 Technical Report C3P 826, California Institute of Technology, Pasadena (1989)
54. Muñoz, M.A., Sun, Y., Kirley, M., Halgamuge, S.K.: Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. Information Sciences **317**, 224–245 (2015). <https://doi.org/https://doi.org/10.1016/j.ins.2015.05.010>
55. de Nobel, J., Ye, F., Vermetten, D., Wang, H., Doerr, C., Bäck, T.: IOHexperimenter: Benchmarking platform for iterative optimization heuristics. arXiv preprint arXiv:2111.04077 (2021)
56. Novikov, A., Vu, N., Eisenberger, M., Dupont, E., Huang, P., Wagner, A.Z., Shirobokov, S., Kozlovskii, B., Ruiz, F.J.R., Mehrabian, A., Kumar, M.P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., Balog, M.: Alphaevolve: A coding agent for scientific and algorithmic discovery. CoRR **abs/2506.13131** (2025). <https://doi.org/10.48550/ARXIV.2506.13131>, <https://doi.org/10.48550/arXiv.2506.13131>
57. Ochoa, G., Hyde, M.V., Burke, E.K.: Automated heuristic design. In: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. pp. 1321–1342 (2011)

58. Ong, Y.S., Lim, M.H., Zhu, N., Wong, K.W.: Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **36**(1), 141–152 (2006). <https://doi.org/10.1109/TSMCB.2005.856143>
59. O'Reilly, U.M.: Genetic programming ii: automatic discovery of reusable programs. *Artificial Life* **1**(4), 439–441 (1994)
60. Popper, K.: *The Logic of Scientific Discovery*. Hutchinson, London (1959)
61. Rechenberg, I.: *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. frommann-holzboog, Stuttgart (1973)
62. Rice, J.R.: The algorithm selection problem. In: *Advances in computers*, vol. 15, pp. 65–118. Elsevier (1976)
63. van Rijn, S., Wang, H., van Leeuwen, M., Bäck, T.: Evolving the structure of evolution strategies. In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. pp. 1–8 (2016). <https://doi.org/10.1109/SSCI.2016.7850138>
64. Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J.R., Ellenberg, J.S., Wang, P., Fawzi, O., Kohli, P., Fawzi, A.: Mathematical discoveries from program search with large language models. *Nat.* **625**(7995), 468–475 (2024). <https://doi.org/10.1038/s41586-023-06924-6>
65. Rowe, J.E., Vose, M.D., Wright, A.H.: Reinterpreting no free lunch. *Evolutionary Computation* **17**(1), 117–129 (03 2009). <https://doi.org/10.1162/evco.2009.17.1.117>
66. Schwefel, H.P.: Numerische optimierung von computer-modellen mittels der evolutionsstrategie.(Teil 1, Kap. 1-5). Birkhäuser (1977)
67. Schwefel, H.P.: *Numerical optimization of computer models*. John Wiley & Sons, Inc. (1981)
68. Seiler, M.V., Kerschke, P., Trautmann, H.: Deep-ela: Deep exploratory landscape analysis with self-supervised pretrained transformers for single-and multi-objective continuous optimization problems. *Evolutionary Computation* pp. 1–27 (2025)
69. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. *Computers & Operations Research* **45**, 12–24 (2014). <https://doi.org/10.1016/j.cor.2013.11.015>, <https://www.sciencedirect.com/science/article/pii/S0305054813003389>
70. Sørensen, K.: Metaheuristics—the metaphor exposed. *International Transactions in Operational Research* **22**(1), 3–18 (2015)
71. van Stein, N., Bäck, T.: Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *CoRR abs/2405.20132* (2024). <https://doi.org/10.48550/ARXIV.2405.20132>, <https://doi.org/10.48550/arXiv.2405.20132>
72. van Stein, N., Bäck, T.: Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Trans. Evol. Comput.* **29**(2), 331–345 (2025). <https://doi.org/10.1109/TEVC.2024.3497793>, <https://doi.org/10.1109/TEVC.2024.3497793>
73. van Stein, N., V. Kononova, A., Yin, H., Bäck, T.: BLADE: Benchmark suite for llm-driven automated design and evolution of iterative optimisation heuristics. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. p. 2336–2344. GECCO '25 Companion, Association for Computing Machinery, New York, NY, USA (2025). <https://doi.org/10.1145/3712255.3734347>, <https://doi.org/10.1145/3712255.3734347>

74. van Stein, N., Vermetten, D., Bäck, T.: In-the-loop hyper-parameter optimization for ILM-based automated design of heuristics. *ACM Transactions on Evolutionary Learning* (2024)
75. van Stein, N., Vermetten, D., V. Kononova, A., Bäck, T.: Explainable benchmarking for iterative optimization heuristics. *ACM Trans. Evol. Learn. Optim.* **5**(2) (May 2025). <https://doi.org/10.1145/3716638>, <https://doi.org/10.1145/3716638>
76. Stützle, T., López-Ibáñez, M.: Automated design of metaheuristic algorithms. In: *Handbook of metaheuristics*, pp. 541–579. Springer (2018)
77. Tanabe, R., Ishibuchi, H.: An easy-to-use real-world multi-objective optimization problem suite. *Applied Soft Computing* **89**, 106078 (2020). <https://doi.org/10.1016/j.asoc.2020.106078>
78. Terrazas, G., Landa-Silva, D., Krasnogor, N.: Towards the design of heuristics by means of self-assembly. *arXiv preprint arXiv:1006.1681* (2010)
79. Thomaser, A., Vogt, M.E., Kononova, A.V., Bäck, T.: Transfer of multi-objectively tuned CMA-ES parameters to a vehicle dynamics problem. In: Emmerich, M., et al (eds.) *Evolutionary Multi-Criterion Optimization (EMO 2023)*, Lecture Notes in Computer Science, vol. 13970, pp. 546–560. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-27250-9_39
80. Thomaser, A., Kononova, A.V., Vogt, M.E., Bäck, T.: One-shot optimization for vehicle dynamics control systems: towards benchmarking and exploratory landscape analysis. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. p. 2036–2045. GECCO ’22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3520304.3533979>, <https://doi.org/10.1145/3520304.3533979>
81. Vermetten, D., Caraffini, F., Kononova, A.V., Bäck, T.: Modular differential evolution. In: Silva, S., Paquete, L. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023*, Lisbon, Portugal, July 15-19, 2023. pp. 864–872. ACM (2023). <https://doi.org/10.1145/3583131.3590417>, <https://doi.org/10.1145/3583131.3590417>
82. Vermetten, D., Doerr, C., Wang, H., Kononova, A.V., Bäck, T.: Large-scale benchmarking of metaphor-based optimization heuristics. In: Li, X., Handl, J. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2024*, Melbourne, VIC, Australia, July 14-18, 2024. ACM (2024). <https://doi.org/10.1145/3638529.3654122>, <https://doi.org/10.1145/3638529.3654122>
83. Wang, H., Vermetten, D., Ye, F., Doerr, C., Bäck, T.: IOHanalyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Transactions on Evolutionary Learning and Optimization* **2**(1), 1–29 (2022)
84. de Winter, R., Long, F.X., Thomaser, A., Bäck, T.H.W., van Stein, N., Kononova, A.V.: Landscape analysis based vs. domain-specific optimization for engineering design applications: A clear case. In: *Proceedings of the 2024 IEEE Conference on Artificial Intelligence (CAI)*. pp. 776–781. IEEE (2024). <https://doi.org/10.1109/CAI59869.2024.00148>
85. Wolpert, D., Macready, W.: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1**, 67–82 (1997). <https://doi.org/10.1109/4235.585893>
86. Wolpert, D.H., Macready, W.G.: Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation* **9**(6), 721–735 (2005). <https://doi.org/10.1109/TEVC.2005.856205>

87. Yao, S., Liu, F., Lin, X., Lu, Z., Wang, Z., Zhang, Q.: Multi-objective evolution of heuristic using large language model. In: Walsh, T., Shah, J., Kolter, Z. (eds.) AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA. pp. 27144–27152. AAAI Press (2025). <https://doi.org/10.1609/AAAI.V39I25.34922>, <https://doi.org/10.1609/aaai.v39i25.34922>
88. Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., Song, G.: Reevo: Large language models as hyper-heuristics with reflective evolution. Advances in neural information processing systems **37**, 43571–43608 (2024)
89. Zhao, Q., Duan, Q., Yan, B., Cheng, S., Shi, Y.: Automated design of metaheuristic algorithms: A survey. arXiv preprint arXiv:2303.06532 (2023)
90. Zhao, Q., Yan, B., Hu, T., Chen, X., Yang, J., Cheng, S., Shi, Y.: Autoopt: A general framework for automatically designing metaheuristic optimization algorithms with diverse structures. IEEE Transactions on Emerging Topics in Computational Intelligence **9**(5), 3690–3703 (2025). <https://doi.org/10.1109/TETCI.2025.3561629>