Natural Computing (2022) 21:219–242
https://doi.org/10.1007/s11047-020-09820-4

A new taxonomy of global optimization algorithms

Jörg Stork[1] · A. E. Eiben[2] · Thomas Bartz-Beielstein[1]

Accepted: 6 November 2020 / Published online: 27 November 2020

Abstract
Surrogate-based optimization, nature-inspired metaheuristics, and hybrid combinations have become state of the art in algorithm design for solving real-world optimization problems. Still, it is difficult for practitioners to get an overview that explains their advantages in comparison to a large number of available methods in the scope of optimization. Available taxonomies lack the embedding of current approaches in the larger context of this broad field. This article presents a taxonomy of the field, which explores and matches algorithm strategies by extracting similarities and differences in their search strategies. A particular focus lies on algorithms using surrogates, nature-inspired designs, and those created by automatic algorithm generation. The extracted features of algorithms, their main concepts, and search operators, allow us to create a set of classification indicators to distinguish between a small number of classes. The features allow a deeper understanding of components of the search strategies and further indicate the close connections between the different algorithm designs. We present intuitive analogies to explain the basic principles of the search algorithms, particularly useful for novices in this research field. Furthermore, this taxonomy allows recommendations for the applicability of the corresponding algorithms.

Keywords Metaheuristics · Surrogate · Hybrid optimization · Evolutionary computation · Taxonomy

1 Introduction

Modern applications in industry, business, and information systems require a tremendous amount of optimization. Global optimization (GO) tackles various severe problems emerging from the context of complex physical systems, business processes, and particular from applications of artificial intelligence. Challenging problems arise from industry on the application level, e.g., machines regarding manufacturing speed, part quality or energy efficiency, or on the business level, such as optimization of production plans, purchase, sales, and after-sales. Further, they emerge from areas of artificial intelligence and information engineering, such as machine learning, e.g., optimization of standard data models such as neural networks for different applications. Their complex nature connects all these problems: they tend to be expensive to solve, and with unknown objective function properties, as the underlying mechanisms are often not well described or unknown.

Solving optimization problems of this kind relies necessarily on performing costly computations, such as simulations, or even real-world experiments, which are frequently considered being black-box. A fundamental challenge in such systems is the different costs of function evaluations. Whether we are probing a real physical system, querying the simulator, or creating a new complex data model, a significant amount of resources is needed to fulfill these tasks. GO methods for such problems thus need to fulfill a particular set of requirements. They need to work with black-box style probes only, so without any further information on the structure of the problem. Further, they must find the best possible improvement within a limited number of function evaluations.

✉ Jörg Stork
joerg.stork@th-koeln.de

A. E. Eiben
a.e.eiben@vu.nl

Thomas Bartz-Beielstein
thomas.bartz-beielstein@th-koeln.de

[1] Technische Hochschule Köln, Steinmüllerallee 1, 51643 Gummersbach, Germany

[2] Vrije Universiteit Amsterdam, 1081 HV Amsterdam, The Netherlands

The improvement of computational power in the last decades has been influencing the development of algorithms. A massive amount of computational power became available for researchers worldwide through multi-core desktop machines, parallel computing, and high-performance computing clusters. This has contributed to the following fields of research: firstly, the development of more complex, nature-inspired, and generally applicable heuristics, so-called metaheuristics. Secondly, it faciliated significant progress in the field of accurate, data-driven approximation models, so-called surrogate models, and their embodiment in an optimization process. Thirdly, the upcoming of approaches which combine several optimization algorithms and seek towards automatic combination and configuration of the optimal optimization algorithm, known as hyperheuristics. The hyperheuristic approach shows the close connections between different named algorithms, in particular in the area of bio-inspired metaheuristics. Automatic algorithm composition has shown to be able to outperform available (fixed) optimization algorithms. All these GO algorithms differ broadly from standard approaches, define new classes of algorithms, and are not well integrated into available taxonomies.

Hence, we propose a new taxonomy, which:

1. describes a comprehensive overview of GO algorithms, including surrogate-based, model-based and hybrid algorithms,
2. can generalize well and connects GO algorithm classes to show their similarities and differences,
3. focusses on simplicity, which enables an easy understanding of GO algorithms,
4. can be used to provide underlying knowledge and best practices to select a suitable algorithm for a new optimization problem.

Our new taxonomy is created based on algorithm key features and divides the algorithms into a small number of intuitive classes: *Hill-Climbing, Trajectory, Population, Surrogate, and Hybrid*. Further, *Exact* algorithms are shortly reviewed, but not an active part of our taxonomy, which focusses on heuristic algorithms. We further utilize extended class names as descriptions founded on the abstracted human behavior in pathfinding. The analogies *Mountaineer, Sightseer, Team, Surveyor* create further understanding by using the image of a single or several persons hiking a landscape in search of the desired location (optimum) utilizing the shortest path (e.g., several iterations).

This utilized abstraction allows us to present comprehensible ideas on how the individual classes differ and moreover, how the respective algorithms perform their search. Although abstraction is necessary for developing our results, we will present results that are useful for practitioners.

This article mainly addresses different kinds of readers: Beginners will find an intuitive and comprehensive overview of GO algorithms, especially concerning common metaheuristics and developments in the field of surrogate-based and hybrid and hyperheuristic optimization. For advanced readers, we also discuss the applicability of the algorithms to tackle specific problem properties and provide essential knowledge for reasonable algorithm selection. We provide an extensive list of references for experienced users. The taxonomy can be used to create realistic comparisons and benchmarks for the different classes of algorithms. It further provides insights for users, who aim to develop new search strategies, operators, and algorithms.

In general, most GO algorithms were developed for a specific search domain, e.g., discrete or continuous. However, many algorithms and their fundamental search principles can be successful for different problem spaces with reasonably small changes. For example, evolution strategies (ES), which are popular in continuous optimization (see Hansen et al. 2003), have their origin in the discrete problem domain. Beyer and Schwefel (2002) describe how the ES moved from discrete to continuous decision variables. Based on this consideration, the article is focused but not limited to illustrating the algorithm variants for the continuous domain, while they have their origins or are most successful in the discrete domain.

Moreover, we focused this taxonomy on algorithms for objective functions without particular characteristics, such as multi-objective, constrained, noisy, or dynamic. These function characteristics pose additional challenges to any algorithm, which are often faced by enhancing available search schemes with specialized operators or even completely dedicated algorithms. We included the former as part of the objective function evaluation in our general algorithm scheme and provide references to selected overviews. Dedicated algorithms, e.g., for multi-objective search, are not discussed in detail. However, their accommodation in the presented taxonomy is possible if their search schemes are related to the algorithms described in our taxonomy. If further required, we outlined the exclusive applicability of algorithms and search operators to specific domains or problem characteristics.

We organized the remainder of this article as follows: Sect. 2 presents the development of optimization algorithms and their core concepts. Section 3 motivates a new taxonomy by reviewing the history of available GO taxonomies, illustrates algorithm design aspects, and presents extracted classification features. Section 4 introduces the new intuitive classification with examples. Section 5 introduces best practices suggestions regarding the applicability of algorithms. Section 6 summarizes and concludes the article with the recent trends and challenges in GO and currently essential research fields.

2 Modern optimization algorithms

This section describes the fundamental principles of modern search algorithms, particular the elements and backgrounds of surrogate-based and hybrid optimization.

The goal of global optimization is to find the overall best solution, i.e., for the common task of minimization, to discover decision variable values that minimize the objective function value.

We denote the global search space as compact set $\mathscr{S} = \{\mathbf{x} \mid \mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u\}$ with $\mathbf{x}_l, \mathbf{x}_u \in \mathbb{R}^n$ being the explicit, finite lower and upper bounds on $\mathbf{x}$. Given a single-objective function f: $\mathbb{R}^n \rightarrow \mathbb{R}$ with real-valued input vectors $\mathbf{x}$ we attempt to find the location $\mathbf{x}^* \in \mathbb{R}^n$ which minimizes the function: $\arg\min f(\mathbf{x}), \mathbf{x} \in \mathscr{S}$.

Finding a global optimum is always the ultimate goal and as such desirable, but for many practical problems, a solution improving the current best solution in a given budget of evaluations or time will still be a success. Particularly, in continuous GO domains an optimum commonly cannot be identified exactly; thus, modern heuristics are designed to spend their resources as efficiently as possible to find the best possible improvement in the objective function value while finding a global optimum is never guaranteed.

Törn and Zilinskas (1989) mention three principles for the construction of an optimization algorithm:

1. An algorithm utilizing all available a priori information will outperform a method using less information.
2. If no a priori information is available, the information is completely based on evaluated candidate points and their objective values.
3. Given a fixed number of evaluated points, optimization algorithms will only differ from each other in the distribution of candidate points.

If a priori information about the function is accessible, it can significantly support the search and should be considered during the algorithm design. Current research on algorithm designs that include structural operators, such as function decomposition, is known as *grey-box optimization* (Whitley et al. 2016; Santana 2017). However, many modern algorithms focus on handling black-box problems where the problem includes little or no a priori information. The principles displayed above lead to the conclusion that the most crucial design aspect of any black-box algorithm is to find a strategy to distribute the initial candidates in the search space and to generate new candidates based on a variation of solutions. These procedures define the *search strategy*, which needs to follow the two competing goals of *exploration* and *exploitation*. The balance between these two competing goals is usually part of the algorithm configuration. Consequently, any algorithm needs to be adapted to the structure of the problem at hand to achieve optimal performance. This can be considered during the construction of an algorithm, before the optimization by parameter *tuning* or during the run by parameter *control* (Bartz-Beielstein et al. 2005; Eiben et al. 1999). In general, the main goal of any method is to reach their target with high efficiency, i.e., to discover optima fast and accurate with as little resources as possible. Moreover, the goal is not mandatory finding a global optimum, which is a demanding and expensive task for many problems, but to identify a valuable local optimum or to improve the currently available solution. We will explicitly discuss the design of modern optimization algorithms in Sect. 3.2.

2.1 Exact algorithms

Exact algorithms also referred to as *non-heuristic* or *complete* algorithms (Neumaier 2004), are a special class of *deterministic, systematic* or *exhaustive* optimization techniques. They can be applied in discrete or combinatorial domains, where the search space has a finite number of possible solutions or for continuous domains, if an optimum is searched within some given tolerances. Exact algorithms have a guarantee to find a global optimum with using a predictable amount of resources, such as function evaluations or computation time (Neumaier 2004; Fomin and Kaski 2013; Woeginger 2003). This guarantee often requires sufficient a priori information about the objective function, e.g., the best possible objective function value. Without available a priori information, the stopping criterion needs to be defined by a heuristic approach, which softens the guarantee of solving to optimality. Well-Known exact algorithms are based on the *branching* principle, i.e., splitting a known problem into smaller sub-problems, which each can be solved to optimality. The *Branch-and-bound* algorithm is an example for exact algorithms (Lawler and Wood 1966).

2.2 Heuristics and metaheuristics

In modern computer-aided optimization, heuristics and metaheuristics are established solution techniques. Although presenting solutions that are not guaranteed to be optimal, their general applicability and ability to present fast sufficient solutions make them very attractive for applied optimization. Their inventors built them upon the principle of systematic search, where solution candidates are evaluated and rewarded with a *fitness*. The term *fitness* has its origins in evolutionary computation, where the fitness describes the competitive ability of an individual in the reproduction process. The fitness is in its purest form the objective function value $y = f(\mathbf{x})$ concerning the optimization goal, e.g., in a minimization problem, smaller values have higher fitness than larger values. Moreover, it can be part of the search strategy, e.g., scaled or adjusted by additional functions, particular for multi-objective or constrained optimization.

Heuristics can be defined as problem-dependent algorithms, which are developed or adapted to the particularities of a specific optimization problem or problem instance (Pearl 1985). Typically, heuristics systematically perform evaluations, although utilizing stochastic elements. Heuristics use this principle to provide fast, not necessarily exact (i.e., not optimal) numerical solutions to optimization problems. Moreover, heuristics are often greedy to provide fast solutions but get trapped in local optima and fail to find a global optimum.

Metaheuristics can be defined as problem independent, general-purpose optimization algorithms. They apply to a wide range of problems and problem instances. The term *meta* describes the higher-level general methodology, which is utilized to guide the underlying heuristic strategy (Talbi 2009).

They share the following characteristics (Boussaïd et al. 2013):

- The algorithms are nature-inspired; they follow certain principles from natural phenomena or behaviors (e.g., biological evolution, physics, social behavior).
- The search process involves stochastic parts; it utilizes probability distributions and random processes.
- As they are meant to be generally applicable solvers, they include a set of control parameters to adjust the search strategy.
- They do not rely on the information of the process which is available before the start of the optimization run, so-called a priori information. Still, they can benefit from such information (e.g., to set up control parameters)

During the remainder of this article, we will focus on heuristic, respectively, metaheuristic algorithms.

2.3 Surrogate-based optimization algorithms

Surrogate-based optimization algorithms are designed to process expensive and complex problems, which arise from real-world applications and sophisticated computational models. These problems are commonly black-box, which means that they only provide very sparse domain knowledge. Consequently, problem information needs to be exploited by experiments or function evaluations. Surrogate-based optimization is intended to model available, i.e., evaluated, information about candidate solutions to utilize it to the full extent. A surrogate model is an approximation which substitutes the original expensive objective function, real-world process, physical simulation, or computational process during the optimization. In general, surrogates are either simplified *physical* or numerical models based on knowledge about the physical system, or empirical *functional* models based on knowledge acquired from evaluations and sparse sampling of the parameter space (Søndergaard et al. 2003). In this work, we focus on the latter, so-called data-driven models. The terms *surrogate model*, *meta-model*, *response surface model* and also *posterior distribution* are used synonymously in the common literature (Mockus 1974; Jones 2001; Bartz-Beielstein and Zaefferer 2017). We will briefly refer to a surrogate model as a *surrogate*. Furthermore, we assume that it is crucial to distinguish between the use of an explicit surrogate of the objective function and general *model-based* optimization (Zlochin et al. 2004), which additionally refers to methods, where a statistical model is used to generate new candidate solutions (cf. Sect. 3.2). We thus distinguish between the two different terms *surrogate-based* and *model-based* to avoid confusion. Another term present in the literature is *surrogate-assisted* optimization, which mostly refers to the application of surrogates in combination with population-based evolutionary computation (Jin 2011).

Important publications featuring overviews or surveys on surrogates and surrogate-based optimization were presented by Sacks et al. (1989), Jones (2001), Queipo et al. (2005), Forrester and Keane (2009). Surrogate-based optimization is commonly defined for but not limited to the case of complex real-world optimization applications. We define a typical surrogate-based optimization process by three layers, where the first two are considered as *problem* layers, while the latter one is the surrogate, i.e., an approximation of the problem layers. We could transfer the defined layers to different computational problems with expensive function evaluations, such as complex algorithms or machine learning tasks.

Each layer can be the target of optimization or used to retrieve information to guide the optimization process. Figure 1 illustrates the different layers of objective functions and the surrogate-based optimization process for real-world problems. In this case, the objective function layers, from the bottom up, are:

L1 The real-world application $f_1(\mathbf{x})$, given by the physical process itself or a physical model. Direct optimization is often expensive or even impossible, due to evaluations involving resource-demanding prototype building or even dangerous experiments.

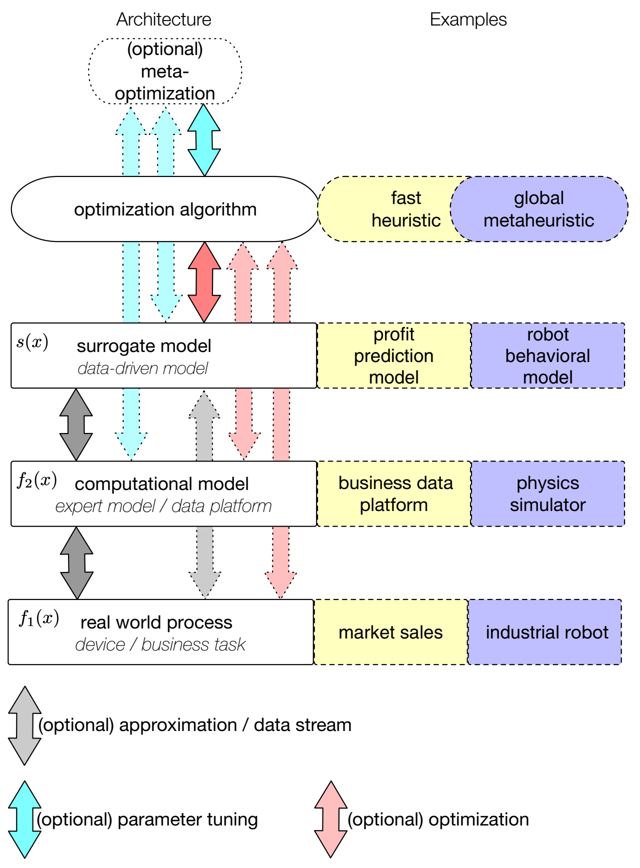

Fig. 1 A surrogate-based optimization process with the different objective function layers: real-world process, computational model, and surrogate. The arrows mark different possible data/application streams. Dotted arrows are in the background, i.e., they pass through elements; each connection always terminates with an arrow. Surrogates are typically either models for the simulation or real-world function. Direct optimization of the problem layers is also possible. Two examples of processes are given to outline the use of the architecture in a business data and robot control task

L2 The computational model $f_2(\mathbf{x})$, given by a simulation of the physical process or a complex computational model, e.g., a computational fluid dynamics model or structural dynamics model. A single computation may take minutes, hours, or even weeks to compute.

L3 The surrogate $s(\mathbf{x})$, given by a data-driven regression model. The accuracy heavily depends on the underlying surrogate type and amount of available information (i.e., function evaluations). The optimization is, compared to the other layers, typically cheap. Surrogates are constructed either for the real-world application $f_1(\mathbf{x})$ or the computational model $f_2(\mathbf{x})$.

Furthermore, the surrogate-based optimization cycle includes the optimization process itself, which is given by an adequate optimization algorithm for the selected objective function layer. No surrogate-based optimization is performed, if the optimization is directly applied to $f_1(\mathbf{x})$ or $f_2(\mathbf{x})$. The surrogate-based optimization uses $f_1(\mathbf{x})$ or $f_2(\mathbf{x})$ for verification of promising solution candidates. Moreover, the control parameters of the optimization algorithm or even the complete optimization cycle, including the surrogate modeling process, can be tuned.

Each layer imposes different evaluation costs and fidelities: the real-world problem is the most expensive to evaluate, but has the highest fidelity, while the surrogate is the cheapest to evaluate, but has a lower fidelity. The main benefit of using surrogates is thus the reduction of needed expensive function evaluations on the objective function $f_1(\mathbf{x})$ or $f_2(\mathbf{x})$ during the optimization. The studies by Loshchilov et al. (2012), Marsden et al. (2004), Ong et al. (2005) and Won and Ray (2004) feature benchmark comparisons of surrogate-based optimization. Nevertheless, the model construction and updating of the surrogates also require computational resources, as well as evaluations for verification on the more expensive function layers. An advantage of surrogate-based optimization is the availability of the surrogate model, which can be utilized to gain further global insight into the problem, which is particularly valuable for black-box problems. The surrogate can be utilized to identify important decision variables or visualize the nature of the problem, i.e., the fitness landscape.

2.4 Meta-optimization and hyperheuristics

Meta-optimization or *parameter tuning* (Mercer and Sampson 1978) describes the process of finding the optimal parameter set for an optimization algorithm. It is also an optimization process itself, which can become very costly in terms of objective function evaluations, as they are required to evaluate the parameter set of a specific algorithm. Hence, particular surrogate-model based algorithms have become very successful meta-optimizer (Bartz-Beielstein et al. 2005). Figure 1 shows where the meta-optimization is situated in an optimization process. If the algorithm adapts parameters during the active run of optimization, it is called parameter control (Eiben et al. 1999). Algorithm parameter tuning and control is further discussed in Sect. 3.2.

A *hyperheuristic* (Cowling et al. 2000, 2002) is a high-level approach that selects and combines low-level approaches (i.e., heuristics, elements from metaheuristics), to solve a specific problem or problem class. It is an optimization algorithm that automates the algorithm design process by searching an ample space of pre-defined algorithm components. A hyperheuristic can also be utilized in an online fashion, e.g., trying to find the most suitable algorithm at each state of a search process (Vermetten et al.

2019). We regard hyperheuristics as hybrid algorithms (cf. Sect. 4.5).

3 A new taxonomy

The term *taxonomy* is defined as a consistent procedure or classification scheme for separating objects into classes or categories based on specific features. The term taxonomy is mainly present in natural science for establishing hierarchical classifications. A taxonomy fulfills the task of distinction and order; it provides explanations and a greater understanding of the research area through the identification of coherence and the differences between the classes.

Several reasons drive our motivation for a new taxonomy: the first reason (I) is that considering available GO taxonomies (Sect. 3.1, cf. Fig. 2), we can conclude that during the last decades, several authors developed new taxonomies for optimization algorithms. However, new classes of algorithms have become state-of-the-art in modern algorithm design, particularly model-based, surrogate-based, and hybrid algorithms dominate the field. Existing taxonomies of GO algorithms do not reflect this situation. Although there are surveys and books which handle the broad field of optimization and give general taxonomies, they are outdated and lack the integration of the new designs. Available up-to-date taxonomies often address a particular subfield of algorithms and discuss them in detail. However, a generalized taxonomy, which includes the above-described approaches and allows to connect these optimization strategies, is missing.

This gap motivated our second reason (II) the development of a generalization scheme for algorithms. We argue that the search concepts of many algorithms are built upon each other and are related. While the algorithms have apparent differences in their strategies, they are not overall different. Many examples for similar algorithms can be found in different named nature-inspired metaheuristics that follow the same search concepts. However, certain elements are characteristic of algorithms, which allow us to define classes based on their search elements. Even different classes share a large amount of these search elements. Thus our new taxonomy is based on a generalized scheme of five crucial algorithm design elements (Sect. 3.2, cf. Fig. 3), which allows us to take a bottom to top approach to differentiate, but also connect the different algorithm classes. The recent developments in hybrid algorithms drive the urge to generalize search strategies, where we no longer use specific, individual algorithms, but combinations of search elements and operators of different classes to find and establish new strategies, which cannot merely be categorized.

Our third reason (III) is the importance of simplicity. Our new taxonomy is not only intended to divide the algorithms into classes, but also to provide an intuitive understanding of the working mechanisms of each algorithm to a broad audience. To support these ideas, we will draw *analogies* between the algorithm classes and the behavior of a human-like individual in each of the descriptive class sections.

Our last reason (IV) is that we intend our taxonomy to be helpful in practice. A common issue is the selection of an adequate optimization algorithm if faced with a new problem. Our algorithm classes are connected by individual properties, which allows us to utilize the new taxonomy to propose suitable algorithm classes based on a small set of problem features. These suggestions, in detail discussed in Sect. 5 and illustrated in Fig. 4 shall help users to find best practices for new problems.

3.1 History of taxonomies

In the literature, one can find several taxonomies trying to shed light on the vast field of optimization algorithms. The identified classes are often named by a significant feature of the algorithms in the class, with the names either being informative or descriptive. For example, Leon (1966) presented one of the first overviews on global optimization. It classified algorithms into three categories: 1. *Blind search*, 2. *Local search* 3. *Non-local search*. In this context, *blind search* refers to simple search strategies that select the candidates at random over the entire search space, but following a built-in sequential selection strategy. During the *local search*, new candidates are selected only in the immediate neighborhood of the previous candidates, which leads to a trajectory of small steps. Finally, *non-local search* allows to escape from local optima and thus enables a global search. Archetti and Schoen (1984) extends the above scheme by also adding the class of *deterministic* algorithms, i.e., those who are guaranteed to find the global optimum with a defined budget. Furthermore, the paper stands out in establishing a taxonomy, which for the first time includes the concepts to construct surrogates, as they describe *probabilistic* methods based on statistical models, which are iteratively utilized to perform the search. Törn and Zilinskas (1989) reviewed existing classification schemes and presented their classifications. They made that the most crucial distinction between two non-overlapping main classes, namely those methods with or without guaranteed accuracy. The main new feature of their taxonomy is the clear separation of the heuristic methods in those with *direct* and *indirect* objective function evaluation. Mockus (1974) also discussed the use of Bayesian optimization. Today's high availability of computational power did not exist; therefore, Törn and

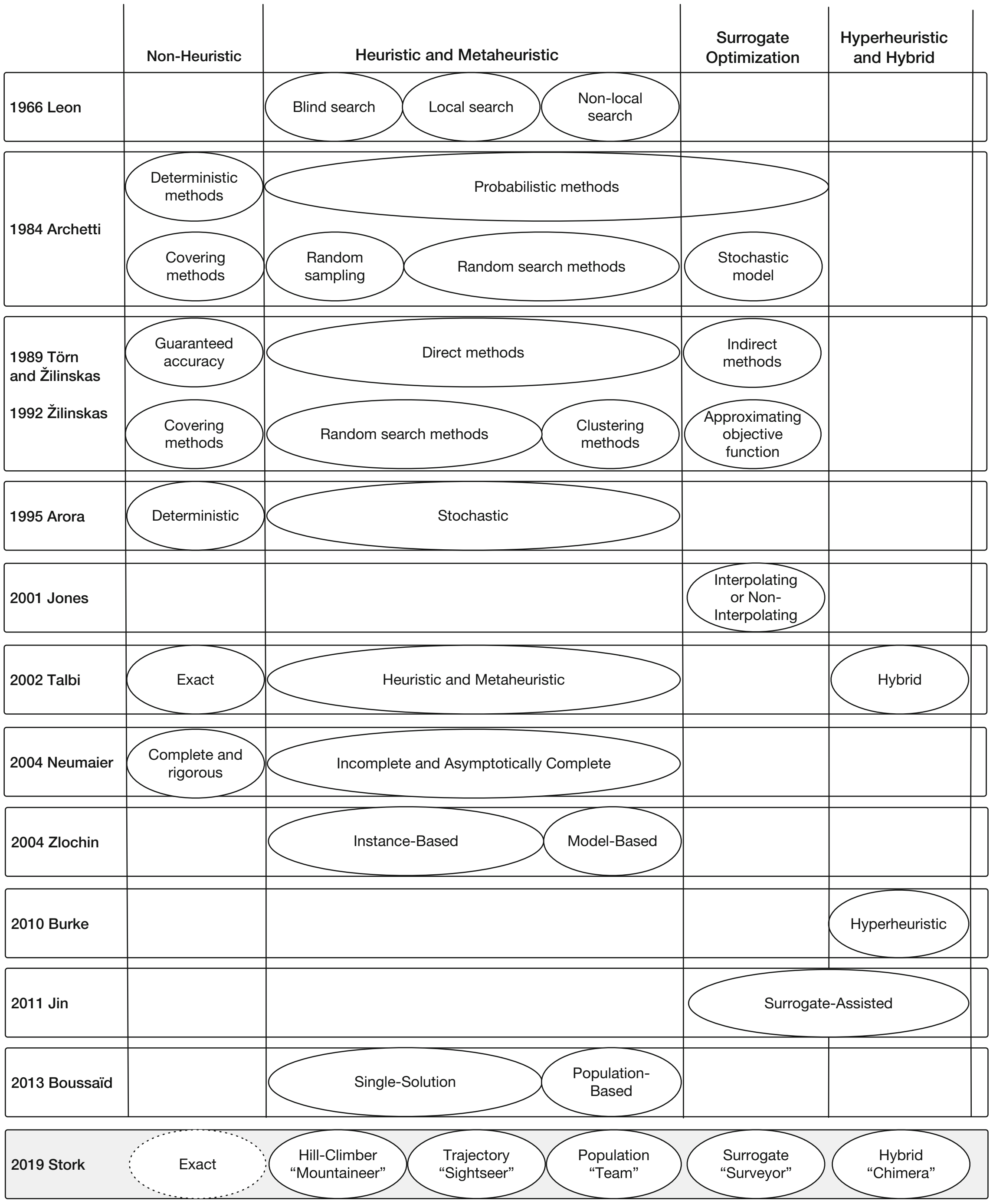

Fig. 2 Global optimization taxonomy history. Information from Leon (1966), Archetti and Schoen (1984), Törn and Zilinskas (1989),Arora et al. (1995), Jones (2001), Talbi (2002), Neumaier (2004), Zlochin et al. (2004),Burke et al. (2010), Jin (2011) and Boussaïd et al. (2013) are illustrated and compared. Different distinctions between the large set of algorithms were drawn. A comprehensive taxonomy is missing and introduced by our new taxonomy, which concludes the diagram and is further presented in Sect. 3

Zilinskas (1989) concluded the following regarding Bayesian models and their applicability for (surrogate-based) optimization:

> Even if it is very attractive, theoretically it is too complicated for algorithmic realization. Because of the fairly cumbersome computations involving operations with the inverse of the covariance matrix and complicated auxiliary optimization problems the resort has been to use simplified models.

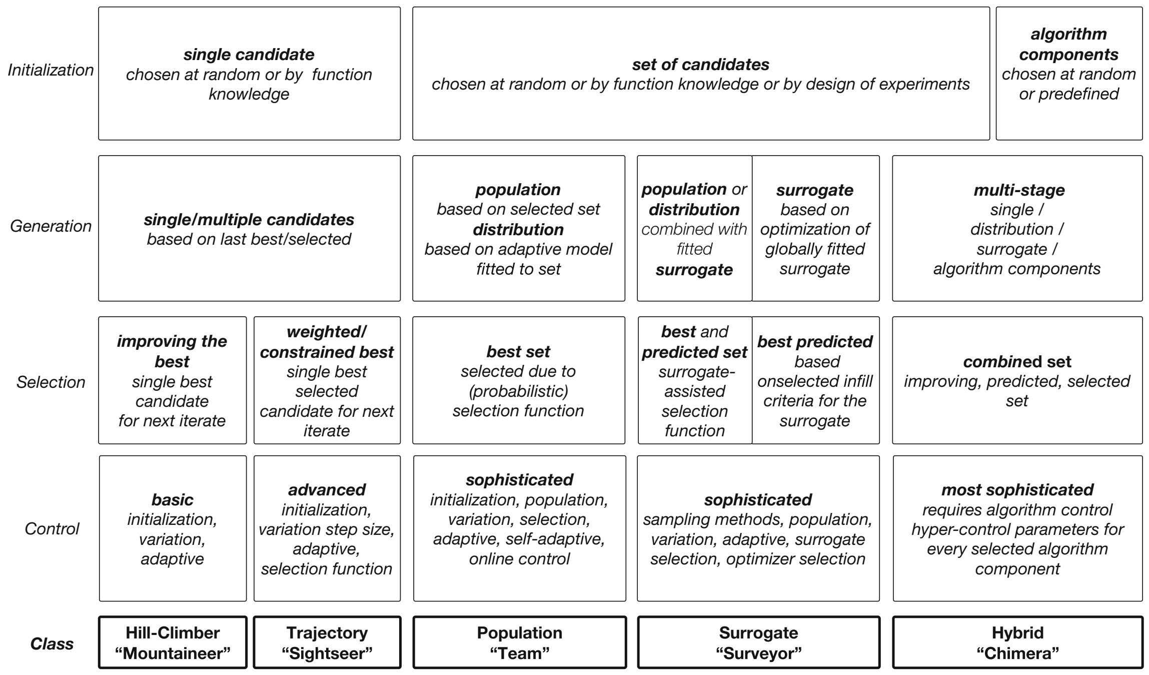
Initialization
single candidate chosen at random or by function knowledge
set of candidates chosen at random or by function knowledge or by design of experiments
algorithm components chosen at random or predefined
Generation
single/multiple candidates based on last best/selected
population based on selected set distribution based on adaptive model fitted to set
population or distribution combined with fitted surrogate
surrogate based on optimization of globally fitted surrogate
multi-stage single / distribution / surrogate / algorithm components
Selection
improving the best single best candidate for next iterate
weighted/ constrained best single best selected candidate for next iterate
best set selected due to (probabilistic) selection function
best and predicted set surrogate-assisted selection function
best predicted based onselected infill criteria for the surrogate
combined set improving, predicted, selected set
Control
basic initialization, variation, adaptive
advanced initialization, variation step size, adaptive, selection function
sophisticated initialization, population, variation, selection, adaptive, self-adaptive, online control
sophisticated sampling methods, population, variation, adaptive, surrogate selection, optimizer selection
most sophisticated requires algorithm control hyper-control parameters for every selected algorithm component
Class
Hill-Climber "Mountaineer"
Trajectory "Sightseer"
Population "Team"
Surrogate "Surveyor"
Hybrid "Chimera"

Fig. 3 Overview of defining algorithm features per search element and class. Overlapping features indicate the close connection between the individual classes

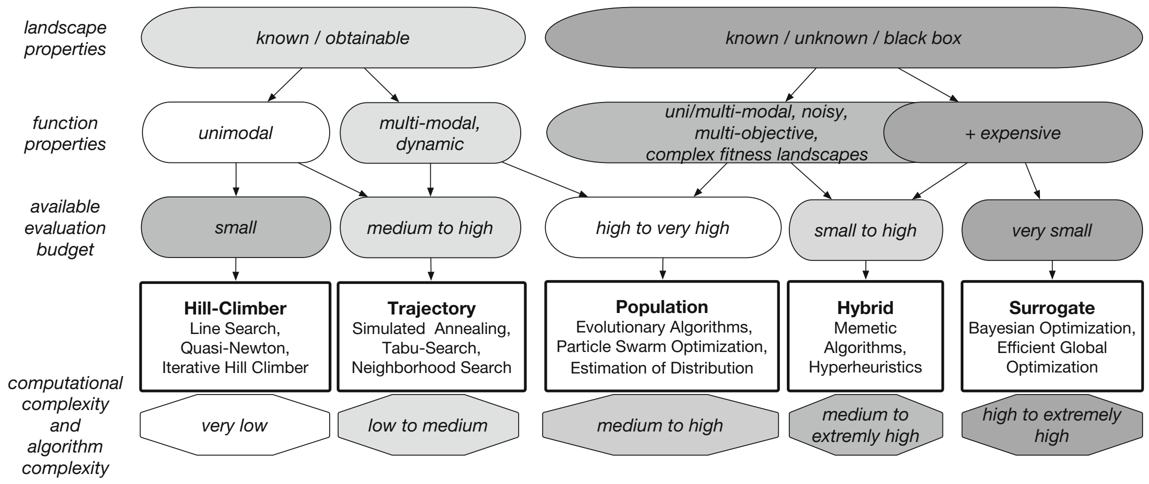
landscape properties
known / obtainable
known / unknown / black box
function properties
unimodal
multi-modal, dynamic
uni/multi-modal, noisy, multi-objective, complex fitness landscapes
+ expensive
available evaluation budget
small
medium to high
high to very high
small to high
very small
Hill-Climber Line Search, Quasi-Newton, Iterative Hill Climber
Trajectory Simulated Annealing, Tabu-Search, Neighborhood Search
Population Evolutionary Algorithms, Particle Swarm Optimization, Estimation of Distribution
Hybrid Memetic Algorithms, Hyperheuristics
Surrogate Bayesian Optimization, Efficient Global Optimization
computational complexity and algorithm complexity
very low
low to medium
medium to high
medium to extremly high
high to extremely high

Fig. 4 Algorithm selection guideline. The figure connects landscape and function properties, as well as the available budget to a suitable algorithm class and outlines their computational complexity

Still, we find the scheme of dividing algorithms into non-heuristic (or exact), random (or stochastic) and further surrogate-based frequently. Several following taxonomies added different algorithm features to their taxonomies, such as metaheuristic approaches (Arora et al. 1995), surrogate-based optimization (Jones 2001), non-heuristic methods (Neumaier 2004), hybrid methods (Talbi 2002), direct search methods (Audet 2014; Kolda et al. 2003), model-based optimization (Zlochin et al. 2004), hyper-heuristics (Burke et al. 2010), surrogate-assisted algorithms (Jin 2011), nature-inspired methods (Rozenberg et al. 2011), or population-based approaches (Boussaïd et al. 2013). We created an overview of different selected taxonomies and put them into the comparison in Fig. 2.

3.2 The four elements of algorithm design

Any modern optimization algorithm, as defined in Sect. 2, can be reduced to the four key search strategy elements *Initialization*, *Generation* and *Selection*. A fourth element controls all these key elements: the *Control* of the different functions and operators in each element. The underlying terminology is generic and based on typical concepts from the field of evolutionary algorithms. We could easily exchange it with wording from other standard algorithm classes (e.g., evaluate = test/trial, generate=produce/variate). Algorithm 3.1 displays the principal elements and the abstracted fundamental structure of optimization algorithms (Bartz-Beielstein and Zaefferer 2017). We could map this structure and elements to any modern optimization algorithm. Even if the search strategy is inherently different or elements do not follow the illustrated order or appear multiple times per iteration.

Algorithm 3.1: General Optimization Algorithm

```
set initial control parameters
begin
    $t = 0$
    initialize candidate(s)
    evaluate initial candidate(s)
    while not termination-condition do
        $t = t + 1$
        generate new candidate(s)
        evaluate new candidate(s)
        select solution(s) for next iteration
        optional: update control parameters
    end
end
```

The *initialization* of the search defines starting locations or a schema for the initial candidate solutions. Two common strategies exist:

1. If there is no available a priori knowledge about the problem and its search space, the best option is to use strategically randomized starting points. The initial distribution target is often exploration, i.e., a broad distribution of the starting points if possible. Particularly interesting for surrogate-based optimization are systematic initialization schemes by methods from the field of *design of experiments* (Crombecq et al. 2011; Bossek et al. 2020).
2. Suppose domain knowledge or other a priori information is available, such as information from the data or process from previous optimization runs. In that case, it is beneficial to utilize this information, e.g., by using a selection of these solutions, such as these with the best fitness. However, known solutions can also bias the search towards them. Thus, e.g., restart strategies intentionally discard them. In surrogate-based optimization, the initial modeling can use available data.

The initial candidates have a large impact on the balance between exploration and exploitation. Space-filling designs with large amounts of random candidates or sophisticated *design of experiments* methods will lead to an initial exploration of the search space. Starting with a single candidate will presumably lead to an exploitation of the neighborhood of the selected candidate location. Hence, algorithms using several candidates are in general more robust, while a single candidate algorithms are sensitive to the selection of the starting candidate, particular in multi-modal landscapes. Multi-start strategies can further increase the robustness and are particularly common for single-candidate algorithms, and also frequently recommended for population-based algorithms (Hansen et al. 2010b).

The *generation* during the search process defines the methods for finding new candidates, with particular regard on how they use available or obtained information about the objective function. A standard approach is the variation of existing observations, as it utilizes, and to a certain extent preserves, the information of previous iterations. Even by the simplest *hill-climber* class algorithms, which do not require any global information or stored knowledge of former iterations (Sect. 4.1), use the last obtained solution to generate new candidate(s). Sophisticated algorithms generate new candidates based on exploited and stored global knowledge about the objective function and fitness landscape. This knowledge is stored by either keeping an archive of all available or selected observations or implicitly by using distribution or data models of available observations. Another option to generate new candidates is combining information of multiple candidates by dedicated functions or operators, particular present in the trajectory class (Sect. 4.2). The exact operators for generation and variation of candidate solutions are various and an essential aspect of keeping the balance between exploration and exploitation in a search strategy.

The *selection* defines the principle of choosing the solutions for the next iteration. We use the term *selection*, which has its origins in evolutionary computation. Besides the most straightforward strategy of choosing the solution(s) with the *best* fitness, advanced selection strategies have emerged, which are mainly present in metaheuristics (Boussaïd et al. 2013). These selection strategies are particularly common in algorithms with several candidates per *generation* step; thus, evolutionary computation introduced the most sophisticated selection methods (Eiben and Smith 2015). The use of absolute differences in fitness or their relative difference is the most common strategy and called *ranked* selection, i.e., based on methods such as *truncation, tournament or proportional* selection.

The *Control parameters* determine how the search can be adapted and improved by controlling the above mentioned key elements. We distinguish between *internal* and *external* parameters: External parameters, also known as offline parameters, can be adjusted by the user and need to be set a priori to the optimization run. Typical external parameters include the number of candidates and settings influencing the above mentioned key elements. Besides standard theory-based defaults (Schwefel 1993), they are usually set by either utilizing available domain knowledge, extensive a priori benchmark experiments (Gämperle et al. 2002), or educated guessing. Sophisticated *meta-optimization* methods were developed to exploit the right parameter settings in an automated fashion. Well-known examples are sequential parameter tuning (Bartz-Beielstein et al. 2005), iterated racing for automatic algorithm tuning (López-Ibáñez et al. 2016), *bonesa* (Smit and Eiben 2011) or *SMAC* (Hutter et al. 2011). In comparison to external parameters, internal ones are not meant to be changed by the user. They are either fixed to an absolute value, which is usually based on physical constants or extensive testing by the authors of the algorithm, or are *adaptive*, or even *self-adaptive*. Adaptive parameters are changed during the search process based on fixed strategies and exploited problem information (Eiben et al. 1999) without user influence. Self-Adaptive parameters are optimized during the run, e.g., by including them into the candidate vector x as an additional decision value. Algorithms using adaptive schemes tend to have better generalization abilities than those with fixed parameters. Thus, they are especially successful for black-box problems, where no prior information about the objective function properties is available to setup parameters in advance (-Hansen et al. 2003). In general, the settings of algorithm control parameters directly affect the balance between exploration and exploitation during the search and are crucial for the search strategies and their performance.

Further, the *evaluation* step computes the fitness of the candidates. The evaluation is a crucial aspect, as it defines how and which information about any candidate solution is gathered by querying the objective function, which can significantly influence the search strategy and also the utilized search operators. However, as important aspects of the evaluation are mostly problem-dependent, such as *noise*, *constraints* and *multiple objectives*. The handling of these aspects sometimes requires unique strategies, operators, or even specialized algorithm designs. These unique algorithms will not be covered in our taxonomy. However, often strategies for handling these particular characteristics are enhanced versions of in this taxonomy presented algorithms, e.g., for handling multiple objectives. Multi-objective problems include several competing goals, i.e., an improvement in one objective leads to a deterioration in another objective. Thus, no single optimal solution is available, but a set of equivalent quality, the non-dominated solutions, or so-called Pareto-set, where reasonable solutions need to be selected from (Fonseca and Fleming 1993; Naujoks et al. 2005). A so-called decision-maker is needed to select the final solutions, which is often the user himself. Further, Multi-objective algorithms can include special search operators, such as hyper-volume-based selection or non-dominated sorting for rank-based selection (Deb et al. 2002; Beume et al. 2007). While most computer experiments are deterministic, i.e., iterations using the same value set for the associated decision variables should deliver the same results, real-world problems are often non-deterministic. They include non-observable disturbance variables and stochastic *noise*. Typical noise handling techniques include multiple evaluations of solutions to reduce the standard deviation and special sampling techniques. The interested reader can find a survey on noise handling by Arnold and Beyer (2003). Moreover, optimization problems frequently include different constraints, which we need to consider during the optimization process. Constraint handling techniques can be directly part of the optimization algorithm, but most algorithms are designed to minimize the objective function and add constraint handling on top. Thus, algorithms integrate it by adjusting the fitness, e.g., by penalty terms. Different techniques for constraint handling are discussed by Coello (2002) and Arnold and Hansen (2012).

4 The definition of intuitive algorithm classes

In his work about evolution strategies, Rechenberg (1994) illustrated a visual approach to an optimization process: a mountaineer in an alpine landscape, attempting to find and climb the highest mountain. The usage of analogies to the natural world is a valuable method to explain the behavior of search algorithms. In the area of metaheuristics, the behavior of the nature and animals inspired the search procedure of the algorithms: Evolutionary algorithms follow the evolution theory (Rechenberg 1994; Eiben and Smith 2015); particle swarm optimization (Kennedy and Eberhart 1995; Shi and Eberhart 1998) utilizes a strategy similar to the movement of bird flocks; ant colony optimization (Dorigo et al. 2006) mimics, as the name suggests, the ingenious pathfinding and food search principles of ant populations.

We take up the idea of optimization processes being human-like individuals and use it in the definition of our extended class names: the *mountaineer*, *sightseer*, *team*, *surveyor* and *chimera*. This additional naming shall accomplish the goal of presenting an evident and straightforward idea of the search strategies of the algorithms in the associated class.

4.1 Hill-climbing class: "The Mountaineer"

Intuitive Description 1 (The Mountaineer) The mountaineer is a single individual who hikes through a landscape, concentrating on achieving his ultimate goal: finding and climbing the highest mountain. He is utterly focussed on his goal to climb up that mountain. So while he checks different paths, he will always choose the ascending way and not explore the whole landscape.

Hill-Climbing algorithms focus their search strategy on greedy exploitation with minimal exploration. Hence, this class encompasses fundamental optimization algorithms with direct search strategies, which include gradient-based algorithms as well as deterministic or stochastic hill-climbing algorithms. Gradient-based algorithms, also known as first-order methods, are in first case applicable to differentiable functions, where the gradient information is available. If the gradient is not directly available, it can be approximated or estimated, for example, by *stochastic gradient descent* (SGD) algorithms (Ruder 2016).

These algorithms have, by design, fast convergence to a local optimum situated in a region of attraction and commonly no explicit strategy for exploration. Overviews of associated algorithms were presented by Lewis et al. (2000) and Kolda et al. (2003). Common algorithms include the quasi-Newton *Broyden-Fletcher–Goldfarb-Shanno* algorithm (Shanno 1970), *conjugate gradients* (CG) (Fletcher 1976), the direct search algorithm *Nelder-Mead* (Nelder and Mead 1965), and stochastic hill climbers such as the *(1+1)-Evolution Strategy* (Rechenberg 1973; Schwefel 1977).

Famous SGD algorithms are *adaptive moment estimation* (ADAM) (Kingma and Ba 2014) and the *adaptive gradient algorithm* (AdaGrad) (Duchi et al. 2011). They are frequently applied in machine learning, particularly for optimizing neural network weights with up to millions of parameters.

As this class defines fundamental search strategies, hill-climbers are often part of sophisticated algorithms as a fast-converging local optimizer. Hill-climbers do not utilize individual operators for the initialization of the single starting point. Thus, it is typically selected at random in the valid search space or based on prior knowledge.

The variation of the last observed selected candidate generates new candidates, commonly in the current solution's vicinity. For example, the stochastic hill climber utilizes random variation with a small step size compared to the range of the complete search interval. Gradient-based methods directly compute or approximate the gradients of the objective function to find the best direction and strength for the variation. Algorithms such as Nelder-Mead create new candidates by computing a search direction using simplexes.

The most common selection methods are elitist strategies, which evaluate the new candidate, compare it to the old solution, and keep the one with the best fitness as a new solution. Always selecting the best is known as a greedy search strategy, as it tries to improve as fast as possible. This greedy strategy leads to the outlined *hill-climbing* search which performs a trajectory of small, fitness-improving steps, which forms in the ideal case a direct line to the nearest optimum. In general, these algorithms search locally for an optimum and do not exploit or use global function information.

The most critical control parameter is the variation step size, which directly influences the speed of convergence. As a result of this, the state of the art is to use an adaptive variation step size that changes online during the search, often based on previous successful steps, for example as defined in the famous *1/5 success rule* (Rechenberg 1973).

4.2 Trajectory class: "The Sightseer"

Intuitive Description 2 (The Sightseer) The intuitive idea of this class is a single hiker looking for interesting places. During the search, the sightseer takes into account that multiple places of interest exist. It thus explores the search space or systematically visits areas to gather information about multiple locations and utilizes this to find the most desired ones.

Trajectory class algorithms still focus on exploitation but are supported by defined exploration methods. This class encompasses algorithms that utilize information from consecutive function evaluations.

They are the connecting link between the hill-climbing and population class. While trajectory algorithms are a step towards population algorithms and also allow the sampling of several solutions in one iteration, they use the principle of initializing and maintaining a single solution. This solution is the basis for variation in each iteration. Again, this variation forms a trajectory in the search space over consecutive iterations, similar to the hill-climbing class. Thus these methods are known as *trajectory methods*

(Boussaïd et al. 2013). While the initialization and generation of the trajectory class are similar to those of the hill-climbing class, the main differences can be found during the *selection*, as they utilize operators to guide the search process in a global landscape in specific directions. Two different strategies can be differentiated, which define two subclasses:

(i) The *exploring trajectory* class utilizes functions to calculate a probability of accepting a candidate as the (current) solution.
(ii) The *systematic trajectory* class utilizes a separation of the search space into smaller sub-spaces to guide the search into specific directions.

These different strategies are susceptible to the correct parametrization, which need to be selected adequate to the underlying objective function.

4.2.1 Exploring trajectory algorithms

The exploring trajectory subclass encompasses algorithms that implement *selection* operators to balance exploration and exploitation to enable global optimization. The introduction of selection functions that allow to expand the search space and escape the *region of attraction* of a local optimum achieves exploration. *Simulated annealing* (SANN) (Kirkpatrick et al. 1983), which is known to be a fundamental contribution to the field of metaheuristic search algorithms, exemplifies this class. The continuous version (Goffe et al. 1994; Siarry et al. 1997; Van Groenigen and Stein 1998) of the SANN algorithm extends the iterated stochastic hill-climber. It includes a new element for the *selection*, the so-called acceptance function. It determines the probability of accepting an inferior candidate as a solution by utilizing a parameter called temperature, in analogy to metal annealing procedures. This dynamic *selection* allows escaping local optima steps by accepting movement in the opposite direction of improvement, which is the fundamental difference to a hill-climber and ultimately allows the global search. At the end of each iteration, a so-called *cooling* operator adapts the temperature. This operator can be used to further balance the amount of exploration and exploitation during the search (Henderson et al. 2003). A common approach is to start with a high temperature and steadily reduce T according to the number of iterations or to utilize an exponential decay of T. This steady reduction of T leads to a phase of active movement and thus exploring in the early iterations, while with decreasing T, the probability of accepting inferior candidates reduces. With approaching a T value of zero, the behavior becomes similar to an iterative hill-climber. Modern SANN implementations integrate *self-adaptive* cooling-schemes which use alternating phases of cooling and reheating (Locatelli 2002). These allow alternating phases of exploration and exploitation but require sophisticated control.

4.2.2 Systematic trajectory algorithms

This subclass encompasses algorithms, which base their search on a space partitioning utilizing the exposed knowledge of former iterations. They create sub-spaces that are excluded from *generation* and *selection*, or *attractive* sub-spaces, where the search is focused on. These search space partitions guide the search by pushing candidate generation to new promising or previous unexplored parts of the search space. An outstanding paradigm for this class is *Tabu Search* (Glover 1989). A so-called tabu list contains the last successful candidates and defines a sub-space of all evaluated solutions. In the continuous version, Siarry and Berthiau (1997), Hu (1992) and Chelouah and Siarry (2000), small (hypersphere or hyperrectangle) regions around the candidates are utilized. The algorithm will consider these solutions or areas as forbidden for future searches, i.e., it selects no candidates situated in these regions as solutions. This process shall ensure to move away from known solutions and prevents identical cycling of candidates and getting stuck in local optima. The definition of the tabu list parameters can control exploration and exploitation by, e.g., by the number of elements or size of areas.

The areas of search can also be pre-defined, such as in *variable neighborhood search (VNS)* (Hansen and Mladenovic 2003; Hansen et al. 2010c; Mladenović et al. 2008). The search strategy of VNS is to perform sequential local searches in these sub-spaces to exploit their local optima. The idea behind this search strategy is that by using an adequate set of sub-spaces, the chance of exploiting a local optimum, which is near the global optimum, increases.

4.3 Population class: "The Team"

Intuitive Description 3 (The Team) The intuitive idea of this class is a group of individuals, which *team* up to achieve their mutual goal together. They split up to explore different locations and share their knowledge with other members of the team.

Population class algorithms utilize distributed exploration and exploitation. The idea of initializing, variation, and selection of several contemporary candidate solutions defines this class. The algorithms are commonly metaheuristics, whose search concepts follow processes found in nature. Moreover, it includes algorithms building upon the population-based concept by utilizing models of the underlying candidate distributions. Due to utilizing a population, the *generation* and *selection* strategies of these algorithms differ significantly from the hill-climber und trajectory class. We subdivide this class into the regular population and model-based population algorithms, which particularly differ in how they generate new candidates during the search:

(i) The *regular population* (Sect. 4.3.1) generate and maintain several candidates with specific population-based operators.
(ii) The *model-based population* (Sect. 4.3.2) generate and adapt models to store and process information.

4.3.1 Regular population algorithms

Well-known examples of this class are *particle swarm optimization (PSO)* (Kennedy and Eberhart 1995; Shi and Eberhart 1998) and different *evolutionary algorithms (EA)*. We regard EAs as state of the art in population-based optimization, as their search concepts are dominating for this field. Nearly all other population-based algorithms use similar concepts and are frequently associated with EAs. Fleming and Purshouse (2002) go as far to state:

> In general, any iterative, population-based approach that uses selection and random variation to generate new solutions can be regarded as an EA.

Evolutionary algorithms follow the idea of evolution, reproduction, and the natural selection concept of *survival of the fittest*. In general, the field of EAs goes back to four distinct developments, *evolution strategies* (ES) (Rechenberg 1973; Schwefel 1977), *evolutionary programming* (Fogel et al. 1966), *genetic algorithms* (Holland 1992), and *genetic programming* (Koza 1992). The naming of the methods and operators matches with their counterparts from biology: candidates are *individuals* who can be *selected* to take the role of *parents*, mate and *recombine* to give birth to *offspring*. The population of individuals is *evolved* (varied, evaluated, and selected) over several iterations, so-called generations, to improve the solutions.

Different overview articles shed light on the vast field of evolutionary algorithms (Back et al. 1997; Eiben and Smith 2003, 2015).

EAs *generate* new solutions typically by variation of a selected subset of the entire population. Typically, competition-based strategies, which also often includes probabilistic elements, select the subsets. Either random variation of this subpopulation or *recombination* by crossover, which is the outstanding concept of EAs, generates new candidates. Recombination partly swaps the variables of two or more candidates, aggregated or combined to create new candidate solutions.

The population-based *selection* strategies allow picking solutions with inferior fitness for the variation process, which allows exploration of the search space. Several selection strategies exist. For instance, in *roulette* wheel selection, the chance of being selected is proportional to the ranking while all chances sum up to one. A spin of the roulette wheel chooses each candidate, where the individual with the highest fitness also has the highest chance of being selected. Alternatively, in *tournament selection*, different small subsets of the population are randomly drawn for several tournaments. Within these tournaments, the candidates with the best fitness are selected based on comparisons to their competitors. This competition-based selection also allows inferior candidates to win their small tournament and participate in the variation.

EAs usually have several parameters, such as the selection intensity (i.e., the percent of truncation), variation step size, or recombination probability. Parameter settings, in particular adaptive and self-adaptive control for evolutionary algorithms is discussed in Angelin (1995), Eiben et al. (1999), Lobo et al. (2007), Doerr et al. (2020) and Papa and Doerr (2020).

4.3.2 Model-based population algorithms

The model-based population class encompasses algorithms that explicitly use mathematical or statistical models of the underlying candidates. These algorithms generally belong to the broad field of EAs (Sect. 4.3.1), and use similar terminology and also operators.

Estimation of distribution algorithms (EDAs) are a well-known example for this class (Larrañaga and Lozano 2001; Hauschild and Pelikan 2011). Compared to a regular population-based approach, a distribution model of selected promising candidates is learned in each iteration, which is then utilized to sample new candidates. The sampling distribution will improve and likely converge to generate only optimal or near-optimal solutions over the iterations. EDAs utilize models from univariate, over bivariate to multivariate distributions, e.g., modeled by Bayesian networks or Gaussian distributions with typical parameters, such as mean, variance, and covariance of the modeled population. The search principle of EDAs was first defined for discrete domains and later successfully extended for continuous domains (Hauschild and Pelikan 2011). Popular examples for EDAs are *population-based incremental learning* (PBIL) (Baluja 1994; Gallagher et al. 1999), the *estimation of Gaussian networks algorithm* (EGNA) (Larrañaga et al. 1999), the *extended compact genetic algorithm* (eCGA) (Harik 1999), and the *iterated density estimation evolutionary algorithm* (IDEA) (Bosman and Thierens 2000). The surrogate class distinction is that the underlying learned distribution models are directly utilized to sample new candidates, instead of substituting the objective function.

A well-known and successful model-based algorithm is the *covariance matrix adaption—evolution strategy* (CMA-ES) (Hansen et al. 2003). While it also utilizes a distribution model, its central idea extends the EDA approach by learning a multivariate Gaussian distribution model of candidate steps, i.e., their changes over iterations, instead of current locations (Hansen 2006). Moreover, instead of creating a new distribution model of selected candidates in each iteration, the model is kept and updated. This principle of updating the model is similar to applying evolutionary variation operators, such as recombination or mutation, to the candidates in a regular population-based algorithm. However, in the CMA-ES, the variation operators' target is the distribution model and not individual candidates.

Again, this class has several *control* parameters, which are often designed to be adaptive or self-adaptive. For example, the CMA-ES utilizes a sophisticated step-size control and adapts the mutation parameters during each iteration following the history of prior successful iterations, the so-called *evolution paths*. These evolution paths are exponentially smoothed sums for each distribution parameter over the consecutive prior iterative steps.

4.4 Surrogate class: "The Surveyor"

Intuitive Description 4 (The Surveyor) The intuitive idea of the surveyor is a specialist who systematically measures a landscape by taking samples of the height to create a topological map. This map resembles the real landscape with a given approximation accuracy and is typically exact at the sampled locations and models the remaining landscape by regression. It can then be examined and utilized to approximate the quality of an unknown point and further be updated if new information is acquired. Ultimately it can be used to guide an individual to the desired location.

Surrogate class algorithms utilize distributed exploration and exploitation by explicitly relying on landscape information and a landscape model. These algorithms differ from all other defined classes in their focus on acquiring, gathering, and utilizing information about the fitness landscape. They utilize evaluated, acquired information to approximate the landscape and also predict the fitness of new candidates.

As illustrated in Sect. 2.3, the surrogates depict the *maps* of the fitness landscape of an objective function in an algorithmic framework. A surrogate algorithm utilizes them for an efficient indirect search, instead of performing multiple, direct, or localized search steps. We divide this class into two subclasses:

- *Surrogate-based* algorithms utilize a global surrogate model for variation and selection.
- *Surrogate-assisted* algorithms utilize surrogates to support the search.

The distinction between the two subclasses is motivated by the different use of the surrogate model. While a surrogate-based algorithm generates new candidates solely by optimizing/prediction of the surrogate, surrogate-assisted algorithms use it to support their search by individual operators (i.e., for the selection of candidates).

For both classes, the surrogate model is a core element of the variation and selection process during optimization and essential for their performance. A perfect surrogate provides an excellent fit to observations, while ideally possessing superior interpolation and extrapolation abilities. However, a large number of available surrogate models all have significantly differing characteristics, advantages, and disadvantages. Model selection is thus a complicated and challenging task. If no domain knowledge is available, such as in real black-box optimization, it is often inevitable to test different surrogates for their applicability.

Common models are: linear, quadratic or polynomial regression, Gaussian processes (also known as Kriging) (Sacks et al. 1989; Forrester et al. 2008), regression trees (Breiman et al. 1984), artificial neural networks and radial basis function networks (Haykin 2004; Hornik et al. 1989) including deep learning networks (Collobert and Weston 2008; Hinton et al. 2006, 2012) and symbolic regression models (Augusto and Barbosa 2000; Flasch et al. 2010; McKay et al. 1995), which are usually optimized by genetic programming (Koza 1992).

Further, much effort in current studies is to research the benefits of model ensembles, which combine several distinct models (Goel et al. 2007; Müller and Shoemaker 2014; Friese et al. 2016). The goal is to create a sophisticated predictor that surpasses the performance of a single model. A well-known example is random forest regression(Freund and Schapire 1997), which uses *bagging* to fit a large number of decision trees (Breiman 2001). We regard ensemble modeling as the state of the art of current research, as they can combine the advantages of different models to generate outstanding results in both classification and regression. The drawback of these ensemble methodologies is that they are computationally expensive and pose a severe problem concerning efficient model selection, evaluation, and combination.

4.4.1 Surrogate-based algorithms

Surrogate-based algorithms explicitly utilize a global approximation surrogate in their optimization cycle by following the concept of *efficient global optimization* (EGO) (Jones et al. 1998) and Bayesian Optimization (BO) (Mockus 1974, 1994, 2012). They are either fixed algorithms designed around a specific model, such as *Kriging*, or algorithmic frameworks with a choice of possible surrogates and optimization methods *sequential parameter optimization* (Bartz-Beielstein et al. 2005; Bartz-Beielstein 2010). Further well-known examples for continuous frameworks are the *surrogate management framework* (SMF) (Booker et al. 1999) and the *surrogate modeling toolbox* (SMT) (Bouhlel et al. 2019). Versions for discrete search spaces are *mixed integer surrogate optimization* (MISO) (Müller 2016) and *efficient global optimization for combinatorial problems* (CEGO) (Zaefferer et al. 2014). The basis for our descriptions of surrogate-based algorithms is mainly EGO, and it is to note that the terminology of BO differs partly from our utilized terminology.

A general surrogate-based algorithm can be described as follows (Cf. Sect. 2.3):

1. The initialization is done by sampling the objective function at k positions with $\mathbf{y}_i = f(\mathbf{x}_i), 1 \leq i \leq k$ to generate a set of observations $\mathscr{D}_t = \{(\mathbf{x}_i, \mathbf{y}_i), 1 \leq i \leq k\}$. The sampling design plan is commonly selected according to the surrogate.
2. Selecting a suitable surrogate. The selection of the correct surrogate type can be a computational demanding step in the optimization process, as often no prior information indicating the best type is available.
3. Constructing the surrogate $s(\mathbf{x})$ using the observations.
4. Utilizing the surrogate $s(\mathbf{x})$ to predict n new promising candidates $\{\mathbf{x}^*_{1:n}\}$, e.g., by optimization of the *infill* function with a suitable algorithm. For example, it is reasonable to use algorithms that require a large number of evaluations as the surrogate itself is (comparatively) very cheap to evaluate.
5. Evaluating the new candidates with the objective function $y^*_i = f(\mathbf{x}^*_i), 1 \leq i \leq n$.
6. If the stopping criterion is not met: Updating the surrogate with the new observations $\mathscr{D}_{t+1} = \mathscr{D}_t \cup \{(\mathbf{x}^*_i, y^*_i), 1 \leq i \leq n\}$, and repeating the optimization cycle (4.-6.)

For the initialization, the model building requires a suitable sampling of the search space. The initial sampling has a significant impact on the performance and should be carefully selected. Thus, the initialization commonly uses candidates following different information criteria and suitable experimental designs. For example, it is common to built linear regression models with factorial designs and preferably couple Gaussian process models with space-filling designs, such as *Latin hypercube sampling* (Montgomery et al. 1984; Sacks et al. 1989).

The generation has two aspects: the first is the choice of surrogate itself, as it is used to find a new candidate. The accuracy of a surrogate strongly relies on the selection of the correct model type to approximate the objective function. By selecting a particular surrogate, the user makes certain assumptions regarding the characteristics of the objective function, i.e., modality, continuity, and smoothness (Forrester and Keane 2009). Most surrogates are selected to provide continuous, low-modal, and smooth landscapes, which renders the optimization process computationally inexpensive and straightforward. The second aspect is the optimizer which variates the candidates for the search on the surrogate and the approximated fitness landscape. As the surrogates are often fast to evaluate, exhaustive *exact* search strategies, such as *branch and bound* in EGO Jones et al. (1998) or multi-start hill-climbers, are often utilized, but it is also common to use sophisticated population-based algorithms.

The surrogate prediction for the expected best solution is the basis of the selection of the next candidate solution. Instead of a simple mean fitness prediction, it is common to define an infill criterion or acquisition function. Typical choices include the probability of improvement (Kushner 1964), expected improvement (Jones et al. 1998) and confidence bounds (Cox and John 1997). Expected improvement is a common infill criterion because it is a balance of exploration and exploitation by utilizing both the predicted best mean value of the model, as well as the model uncertainty. The optimization of this infill criterion then selects the candidate. Typically, in each iteration for evaluation and the model update, the algorithm selects only a single candidate. Multi-infill selection strategies are also possible.

Surrogate-based algorithms include a large number of control elements, starting with necessary components of such an algorithm, including the initialization strategy, the choice of surrogate and optimizer. In particular, the infill criteria, as part of the selection strategy, has an enormous impact on the performance. Even for a fixed algorithm, the amount of (required) control is extensive. The most important are the model parameters of the surrogate.

4.4.2 Surrogate-assisted algorithms

Surrogate-assisted algorithms utilize s search strategy similar to the population class, but employ a surrogate particular in the *selection* step to preselect candidate solutions based on their approximated fitness and *assist* the evolutionary search strategy (Ong et al. 2005; Jin 2005; Emmerich et al. 2006; Lim et al. 2010; Loshchilov et al. 2012). Commonly, only parts of the new candidates are preselected utilizing the surrogate, while another part follows a direct selection and evaluation process. The generation and selection of a new candidate are thus not based on an optimization of the surrogate landscape, which is the main difference to the surrogate-based algorithms. The surrogate can be built on the archive of all solutions, or locally on the current solution candidates. An overview of surrogate-assisted optimization is given by Jin (2011), including several examples for real-world applications, or by Haftka et al. (2016) and Rehbach et al. (2018), with focus on parallelization.

4.5 Hybrid class: "The Chimera"

Intuitive Description 5 (The Chimera) A chimera is an individual, which is a mixture, composition, or crossover of other individuals. It is an entirely new being formed out of original parts from existing species and utilizes their strengths to be versatile.

We describe the explicit combination of algorithms or their components as the hybrid class. Hybrid algorithms utilize existing components and concepts, which have their origin in an algorithm from one of the other classes, in new combinations. They are particularly present in current research regarding the automatic composition or optimization of algorithms to solve a certain problem or a problem class. Hyperheuristic algorithms also belong to this class. Overviews of hybrid algorithms were presented by Talbi (2002), Blum et al. (2011) and Burke et al. (2013). There are two kinds of hybrid algorithms:

1. *Predetermined Hybrids* (Sect. 4.5.1) have a fixed algorithm design, which is composed of certain algorithms or their components.
2. *Automated Hybrids* (Sect. 4.5.2) use optimization or machine learning to search for an optimal algorithm design or composition.

The hybrid class contains a large number of algorithms and it can be difficult to draw a distinction to a certain class. However, particularly the predetermined hybrids can be additionally described by their main components, so that their origin remains clear, e.g., an evolutionary algorithm coupled with simulated annealing could be defined as *population-trajectory hybrid*. For automated hybrids this definition is not longer possible, as they couple a large amount of different components and also the algorithms structure is part of their search, so the final algorithm structure can differ for each problem.

4.5.1 Predetermined hybrid algorithms

The search strategies of this class improve or tackle algorithm weaknesses or amplify their strengths. The algorithms are often given distinctive roles of exploration and exploitation, as they are combinations of an explorative global search method paired with a local search algorithm. For example, population-based algorithms with remarkable exploration abilities pair with local algorithms with fast convergence. This approach gives some benefits, as the combined algorithms can be adapted or tuned to fulfill their distinct tasks. Also well known are *Memetic algorithms*, as defined by Moscato et al. (1989), which are a class of search methods that combine population-based algorithms with local hill-climbers. An extensive overview of memetic algorithms is given by Molina et al. (2010). They describe how different hybrid algorithms can be constructed by looking at suitable local search algorithms with particular regard to their convergence abilities.

4.5.2 Automated hybrid algorithms

Automated hybrids are a special kind of algorithms, which do not use predetermined search strategies, but a pool of different algorithms or algorithm components, where the optimal strategy can be composed of (Lindauer et al. 2015; Bezerra et al. 2014). Hyperheuristics belong to this class, particularly those that generate new heuristics (Burke et al. 2010; Martin and Tauritz 2013). Automated algorithm selection tries to find the most suitable algorithm for a specific problem based on machine learning and problem information, such as explorative landscape analysis (Kerschke and Trautmann 2019). Instead of selecting individual algorithms, it is tried to select different types of operators for, e.g., generation or selection, to automatically compose a new, best-performing algorithm for a particular problem (Richter and Tauritz 2018). Similar to our defined elements, search operators or components of algorithms are identified, extracted, and then again combined to a new search strategy.

Generation, variation, and selection focus in this class on algorithm components, instead of candidate solutions. The search strategies on this upper level are similar to the presented algorithms; hence, for example, evolutionary or Bayesian techniques are common (Guo 2003; van Rijn et al. 2017).

4.6 Taxonomy: summary and examples

Figure 3 illustrates an overview of all classes and connected features. It outlines initialization, generation, selection, and control of the individual components for each class. The algorithm features are partly distinct and define their class, while others are shared. The figure shows the strong relationship between the algorithm classes; for example, the hill-climbing and trajectory class share similar characteristics. The algorithms of these classes are similar in their search strategy and built upon each other.

The presented scheme is intended to cover most concepts. However, available algorithms can also have characteristics of different classes and do not fit the presented scheme. For example, some components of the more complex classes can also be utilized in the less complex classes, e.g., self-adaptive control schemes also apply for hill-climbers, but are typically found in the population class. Table 1 describes examples for each of the defined algorithm classes and outlines their specific features. Again, the table is not intended to present a complete overview, instead, for each class and subclass, at least one example is given to present the working scheme. Other algorithms can be easily classified utilizing the scheme presented in Fig. 3.

5 Algorithm selection guideline

The selection of the best-suited algorithm poses a common problem for practitioners, even if experienced. In general, it is nearly impossible to predict the performance of any algorithm for a new, unknown problem. We thus recommend first to gather all available information about the problem if confronted with a new optimization problem. The features of a problem can be an excellent guideline to select at least an adequate algorithm class, where the users' choice and experience can select a concrete implementation.

Our guideline is strongly connected to the idea of *exploratory landscape analysis* (ELA) (Mersmann et al. 2011), which aims to derive problem features with the final goal of relating those features to suitable algorithm classes. For example, these features include information about convexity, multi-modality, the global structure of the problem, problem separability, and variable scaling. ELA's final goal is to provide the best-suited algorithms to previous unknown optimization problems based on the derived landscape features. This goal requires rigorous experimentation and benchmarking to match algorithms or algorithm classes to the landscape features (Kerschke and Trautmann 2019). As this information is not yet available, we extracted a small, high-level set of these features for our guideline, considering mainly the multi-modality and unique function properties, as being expensive to evaluate.

Moreover, our selection is based on the available resources, both in terms of available evaluations and computational resources. To help with the selection, we developed a small decision graph which builds upon these significant features. The provided guideline is experience-

Table 1 Summary of example algorithms for each class of the presented taxonomy

Name	Class	Specifics
1 + 1-evolution strategy (Rechenberg 1973; Schwefel 1977)	Hill-climber	Probabilistic, adaptive mutation rates
L-BFGS (Liu and Nocedal 1989), CG (Fletcher 1976)	Hill-climber	Approximating gradient
Variable neighborhood search (Hansen and Mladenovic 2003)	Trajectory (systematic)	Separation of search space in individually searched sub-spaces
Tabu search (Glover 1989; Siarry and Berthiau 1997)	Trajectory (systematic)	Tabu-list of search restricted solutions/areas
Simulated annealing (Kirkpatrick et al. 1983)	Trajectory (exploring)	Control variable: temperature to define exploration strength
Evolutionary algorithms (Back 1996; Eiben and Smith 2003, 2015)	Population (regular)	different variation and selection strategies, general framework
Particle swarm optimization (Kennedy and Eberhart 1995)	Population (regular)	Exploration and exploitation strategy based on behavior in swarms
Estimation of distribution algorithms (Larrañaga and Lozano 2001; Hauschild and Pelikan 2011)	Population (model)	probabilistic distribution model of underlying population
Covariance matrix adaption—ES (Hansen et al. 2003)	Population (model)	Modeling search steps, adaptive or pre-defined parameters
Efficient global optimization (Jones et al. 1998)	Surrogate (based)	Kriging models, expected improvement
Bayesian optimization (Mockus 1974, 1994, 2012)	Surrogate (based)	general framework,
Surrogate-assisted EAs (Ong et al. 2005; Lim et al. 2010)	Surrogate (assisted)	General framework, assisting evolutionary algorithms with surrogates
Memetic algorithms (Moscato et al. 1989)	Hybrid (predetermined)	Combining EAs with hill-climber class unimodal search algorithms
Hyperheuristics (Burke et al. 2003)	Hybrid (automated)	Automatic selection of entire heuristics or individual search operators

based and utilizes basic concepts; it is not intended to serve as an absolute policy; instead, as the first recommendation if a new problem is considered. The graph is outlined in Fig. 4.

A *hill-climbing* algorithm is in the first place suitable for unimodal functions or to exploit local optima or for cases where gradient information can be derived from the objective function. Gradient-based algorithms are incredibly successful in optimizing large scale optimization problems, such as frequently found in AI. However, they always have a high risk of getting stuck in local optima. It can be applied for global optimization to multimodal landscapes if an adequate multi-start strategy is employed. These multi-start strategies typically demand a high number of function evaluations and are most reasonable to be used if objective functions are not expensive.

Exploring trajectory algorithms are suitable for searches in unimodal and multimodal problems. As they do not rely on stored information of former iterations during their search, they are also an excellent choice to handle dynamic objective functions (Carson and Maria 1997; Corana et al. 1987; Faber et al. 2005). However, the rather simplistic utilization of exploited global information renders them not efficient for challenging and expensive optimization problems. Moreover, the control parameters have a significant effect on the performance of these algorithms. We thus advise to tune them in an offline or online fashion.

The central concept of *systematic trajectory* algorithms is to use the information of evaluated solutions and to direct the search to former unknown regions to avoid early convergence to a non-global optimum. The strategic use of sub-spaces allows precise control of exploration and exploitation and mainly ensures a high level of exploration. They include a large number of parameters, such as the number or size of sub-spaces, which makes them very vulnerable to false setups and less good universal solvers. If correctly tuned, algorithms from this class are suitable and efficient for multimodal problems. Algorithms using a pre-defined separation of the search space, such as VNS, can utilize domain knowledge for the initial definition of the sub-spaces. This pre-defined separation renders them useful for problems where the region of the global optimum is roughly known, but not particularly suitable for black-box problems.

Population-based algorithms are very flexible in their implementation and adaptable by tuning. They are robust and suitable to solve a large class of problems, including multimodal, multi-objective, dynamic and black-box problems, even with noise or discontinuities in the fitness landscape (Jin and Branke 2005; Marler and Arora 2004). Further, they have successfully been applied to a large number of different industrial problems (Fleming and Purshouse 2002) but typically require a relatively large number ($\gg 100 \times dim$) of function evaluations to be successful. This inefficiency makes them not the first choice for expensive problems, where the number of evaluations is sharply limited. Different mechanisms and strategies for controlling the balance between exploration and exploitation cause flexibility and robustness. A good overview is presented in the survey by Črepinšek et al. (2013). The detailed survey classifies the different available evolutionary approaches and presents an intensive discussion on which mechanisms influence exploration and exploitation. Theoretical aspects of evolutionary operators are discussed by Beyer (2013). *Parameter tuning and control* influence the performance of EAs, e.g., by the setting of population size, mutation strength, and selection probability. An extensive overview of the different on- and offline tuning approaches for parameter control in EAs was published by Eiben et al. (1999). Further common strategies of controlling exploration and exploitation and multimodal optimization are so-called niching strategies, which utilize sub-populations to maintain the diversity of the population, investigate several regions of the search space in parallel or conduct defined tasks of exploring and exploiting (Shir and Bäck 2005; Filipiak and Lipinski 2014).

One step further, *model-based* algorithms try to combine the benefits of statistical models and their capability of storing and processing information with population-based search operators. They are high-level metaheuristics and advanced EAs which intended to be flexible, robust, and applicable to a large class of problems, particularly those with unknown function properties. This generalizability makes them very successful in popular black-box benchmarks (Hansen et al. 2010b). For example, the design of the CMA-ES seeks to make the algorithm performance robust and not dependent on the objective function or tuning. The various *control parameters* of the algorithm were pre-defined based on theoretical aspects and practical benchmarks.

Surrogate-based algorithms were created to solve expensive problems with the help of a surrogate. Their focus on high evaluation efficiency renders them particularly suitable for problems were only a small number of function evaluations are possible, such as real-world optimization or physical simulations. The downside of surrogate modeling is that it can impose a high computational complexity in the surrogate fitting and prediction process. This cost is usually low compared to the resource cost of a real-world function or an expensive simulation, but it can get significant compared to a cheap (i.e., fast to evaluate, low cost) objective function. For problems that need very fast optimizations or allow large numbers of functions evaluations ($\gg 100 \times n$), a surrogate-based approach is not the best choice. Moreover, solving high-dimensional problems or those with a large amount of samples require unique strategies or specialized models (Regis and Shoemaker 2013). In general, the selection of an adequate model, experimental design, and optimizer requires both domain knowledge and expertise. Forrester and Keane (2009) and Bartz-Beielstein and Zaefferer (2017) give overviews of surrogate-based optimization, different surrogate models and infill criteria and match surrogates to problem classes and give hints about their applicability. Surrogate-based optimization was successfully applied to different applications, including expensive optimization problems (Lizotte 2008; Khan et al. 2002) and machine learning (Snoek et al. 2012; Swersky et al. 2013; Stork et al. 2019; Gaier et al. 2018) Surrogate-assisted optimization is more flexible, as it combines the strength of population-based algorithms with the evaluation efficiency of the surrogate.

Hybrid algorithms apply to a large class of problems, dependent on the origin of their algorithms or components. The most recent algorithms from this hybrid class search for the best algorithm design and composition automatically, even at different problem stages or search stages. The method of automatic algorithm selection has shown to be able to outperform a single algorithm on a set of benchmark functions (van Rijn et al. 2017; Vermetten et al. 2019). However, this procedure also requires either a large amount of function knowledge or a large evaluation budget and computation time. Their immense complexity includes the risk that the automatic composition does not lead to improved performance, due to the problematic balancing and required tuning of the distinct algorithms. Further, their sophisticated search strategies with a large number of control parameters can make them difficult to tune. For the automatic algorithm selection, numerous operators influence the convergence behavior, and the search strategy itself becomes a black-box that is challenging to comprehend.

The guideline can be utilized in the following way: If features of the objective function are known (e.g., it is unimodal as the sphere function), a suitable algorithm can be selected based on these features, e.g., a hill-climber. If no information about the function is known, e.g., it is a complete black-box, we recommend using algorithms with a high generalization ability, e.g., a CMA-ES. Another essential decision regards the number of possible function evaluations and the costs of the objective function. If the number of available function evaluations is low, and the objective function is costly, surrogate-based optimization algorithms are a robust choice. Recent algorithms based on automated algorithm selection aim to optimize this selection process by extracting problem features while optimizing and adapting the search strategy to them (van Rijn et al. 2017; Kerschke and Trautmann 2019).

6 Concluding remarks

We presented a new comprehensive overview of global optimization algorithms by creating a new taxonomy in this work. We set a particular focus on covering a broad range of optimization algorithms, including surrogates, metaheuristics, and algorithm combinations, because existing taxonomies do not well cover these.

Based on a generalized algorithm scheme we defined four characteristic elements of optimization algorithms, i.e., how they initialize, generate, and select solutions, how these solutions are evaluated, and finally, how these algorithms can be parametrized and controlled. With these elements, we created a generalized view on optimization algorithms by identifying their specific components. These components were then used to divide algorithms in the *hill-climber*, *trajectory*, *population*, *surrogate* and *hybrid* class and to identify similarities and differences in their search strategies.

We can conclude that most algorithms and algorithm classes have a close connection and share similar components, operators, and a large part of their search strategies. Current research for the automated design of algorithms builds upon this fact. It generalizes algorithms by breaking them down to their components and again combining these components to algorithms. This design process can be fully automatic, selecting components based on known features of the problem. Recent research aims in this direction (Lindauer et al. 2015; Kerschke and Trautmann 2019; Bezerra et al. 2014; van Rijn et al. 2017). These automatic methods can benefit from classifications of algorithms to build their design spaces of suitable algorithms classes. Our classes and identified elements can be utilized to create a design scheme for each class. For example, employ an algorithm generator for hill-climbers or population-based algorithms. Thus, our taxonomy is particularly useful for these automatic methods, as it includes a broad set of current developments in the presented classification scheme.

Our set of accompanying analogies, based on the human behavior in pathfinding, mainly helps novices comprehend the algorithm search strategies' fundamentals. We further outlined an algorithm selection guideline with best practices for more advanced practitioners, which can support them if they face a new problem and have to choose a suitable class of optimization algorithms.

Our taxonomy has several limitations, which are part of future work: the first is that our current taxonomy cannot cover all available algorithms, particularly those who handle specific problem characteristics, e.g., multi-objective, noisy, or dynamic objective functions. The second limitation concerns the missing benchmarking evidence for our algorithms selection guideline: There are excellent structured collections of (test) problems, e.g., the *black-box optimization benchmark* (BBOB) (Hansen et al. 2010a). Here we offer a system for algorithms providing a structured way of positioning algorithms. The combination of the two facilitates systematic empirical studies on problem (type) and algorithm (type) combination. In other words, systematic benchmarking: To enable fair comparisons, each algorithm from a selected algorithm class has to be compared to several algorithms not belonging to this class. This implies a large amount of work on a long term done by the research communities, going far beyond this paper's scope.

Exciting challenges for future algorithm design arise from problems in several categories. The first is *industry 4.0*. Digitalization in manufacturing and engineering, particularly the rapid development of communicating sensors and machines in engineering, requires new designs. Suitable optimization algorithms need to be directly included in the production cycle, adapting to generate robust solutions in challenging dynamic environments in an online manner, robustly improving themselves over a long-time period in the field. The second category is machine learning techniques, particularly the training of models with a massive amount of parameters, such as deep learning networks. They require algorithm designs that combine sampling and computation time efficiency, which is a challenging optimization problem. Further, the field of structural optimization or hyper-parameter optimization of neural networks is advancing. In this area, interesting research is dedicated to neuroevolution (Stanley et al. 2019) or population-based training (Jaderberg et al. 2017). Moreover, surrogate-based algorithms are considered to optimize neural networks (Gaier et al. 2018; Stork et al. 2019). Finally, algorithms arising from new computing paradigms, such as quantum computing (Pittenger 2012) or neuromorphic computing(Schuman et al. 2017), might completely change our current perspective on how optimization algorithms work.

Funding Open Access funding enabled and organized by Projekt DEAL.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Angeline PJ (1995) Adaptive and self-adaptive evolutionary computations. In: Computational intelligence: a dynamic systems perspective. Citeseer

Archetti F, Schoen F (1984) A survey on the global optimization problem: general theory and computational approaches. Ann Oper Res 1(2):87–110

Arnold DV, Beyer HG (2003) A comparison of evolution strategies with other direct search methods in the presence of noise. Comput Optim Appl 24(1):135–159

Arnold DV, Hansen N (2012) A (1 + 1)-CMA-ES for constrained optimisation. In: Proceedings of the 14th annual conference on genetic and evolutionary computation. ACM, pp 297–304

Arora J, Elwakeil O, Chahande A, Hsieh C (1995) Global optimization methods for engineering applications: a review. Struct Optim 9(3–4):137–159

Audet C (2014) A survey on direct search methods for blackbox optimization and their applications. In: Mathematics without boundaries. Springer, pp 31–56

Augusto DA, Barbosa HJ (2000) Symbolic regression via genetic programming. In: Sixth Brazilian symposium on neural networks, 2000. Proceedings. IEEE, pp 173–178

Back T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, Oxford

Back T, Fogel DB, Michalewicz Z (1997) Handbook of evolutionary computation. IOP Publishing Ltd, Bristol

Baluja S (1994) Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning

Bartz-Beielstein T (2010) SPOT: an R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. arXiv preprint arXiv:10064645

Bartz-Beielstein T, Zaefferer M (2017) Model-based methods for continuous and discrete global optimization. Appl Soft Comput 55:154–167

Bartz-Beielstein T, Lasarczyk CW, Preuß M (2005) Sequential parameter optimization. In: The 2005 IEEE Congress on evolutionary computation, 2005, vol 1. IEEE, pp 773–780

Beume N, Naujoks B, Emmerich M (2007) SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur J Oper Res 181(3):1653–1669

Beyer HG (2013) The theory of evolution strategies. Springer, Berlin

Beyer HG, Schwefel HP (2002) Evolution strategies: a comprehensive introduction. Nat Comput 1(1):3–52

Bezerra LC, López-Ibánez M, Stützle T (2014) Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In: International conference on parallel problem solving from nature. Springer, pp 508–517

Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11(6):4135–4151

Booker AJ, Dennis J Jr, Frank PD, Serafini DB, Torczon V, Trosset MW (1999) A rigorous framework for optimization of expensive functions by surrogates. Struct Optim 17(1):1–13

Bosman PAN, Thierens D (2000) Continuous iterated density estimation evolutionary algorithms within the idea framework

Bossek J, Doerr C, Kerschke P (2020) Initial design strategies and their effects on sequential model-based optimization: an exploratory case study based on BBOB. In: Proceedings of the 2020 genetic and evolutionary computation conference, GECCO '20. Association for Computing Machinery, New York, pp 778–786. https://doi.org/10.1145/3377930.3390155

Bouhlel MA, Hwang JT, Bartoli N, Lafage R, Morlier J, Martins JR (2019) A python surrogate modeling framework with derivatives. Adv Eng Softw 135:102662

Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and regression trees. CRC Press, Boca Raton

Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Handbook of metaheuristics. Springer, pp 457–474

Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Handbook of metaheuristics. Springer, pp 449–468

Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64(12):1695–1724

Carson Y, Maria A (1997) Simulation optimization: methods and applications. In: Proceedings of the 29th conference on Winter simulation. IEEE Computer Society, pp 118–126

Chelouah R, Siarry P (2000) Tabu search applied to global optimization. Eur J Oper Res 123(2):256–270

Coello CAC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191(11):1245–1287

Collobert R, Weston J (2008) A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceedings of the 25th international conference on Machine learning. ACM, pp 160–167

Corana A, Marchesi M, Martini C, Ridella S (1987) Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm. ACM Trans Math Softw (TOMS) 13(3):262–280

Cowling P, Kendall G, Soubeiga E (2000) A hyperheuristic approach to scheduling a sales summit. In: International conference on the practice and theory of automated timetabling. Springer, pp 176–190

Cowling P, Kendall G, Soubeiga E (2002) Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation. In: Workshops on applications of evolutionary computation. Springer, pp 1–10

Cox DD, John S (1997) SDO: a statistical method for global optimization. In: Multidisciplinary design optimization: state of the art, pp 315–329

Črepinšek M, Liu SH, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput Surv (CSUR) 45(3):35

Crombecq K, Gorissen D, Deschrijver D, Dhaene T (2011) A novel hybrid sequential design strategy for global surrogate modeling of computer experiments. SIAM J Sci Comput 33(4):1948–1974

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Doerr B, Doerr C, Yang J (2020) Optimal parameter choices via precise black-box analysis. Theor Comput Sci 801:1–34

Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39

Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12(7):2121–2159

Eiben AE, Smith JE (2003) Introduction to evolutionary computing, vol 53. Springer, Berlin

Eiben AE, Smith JE (2015) Introduction to evolutionary computing, 2nd edn. Springer, Berlin

Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evolut Comput 3(2):124–141

Emmerich MT, Giannakoglou KC, Naujoks B (2006) Single-and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. IEEE Trans Evolut Comput 10(4):421–439

Faber R, Jockenhövel T, Tsatsaronis G (2005) Dynamic optimization with simulated annealing. Comput Chem Eng 29(2):273–290

Filipiak P, Lipinski P (2014) Infeasibility driven evolutionary algorithm with feed-forward prediction strategy for dynamic constrained optimization problems. In: Applications of evolutionary computation. Springer, pp 817–828

Flasch O, Mersmann O, Bartz-Beielstein T (2010) RGP: an open source genetic programming system for the R environment. In: Proceedings of the 12th annual conference companion on genetic and evolutionary computation, GECCO '10. ACM, New York, pp 2071–2072. https://doi.org/10.1145/1830761.1830867

Fleming PJ, Purshouse RC (2002) Evolutionary algorithms in control systems engineering: a survey. Control Eng Pract 10(11):1223–1241

Fletcher R (1976) Conjugate gradient methods for indefinite systems. In: Numerical analysis. Springer, pp 73–89

Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, Hoboken

Fomin FV, Kaski P (2013) Exact exponential algorithms. Commun ACM 56(3):80–88. https://doi.org/10.1145/2428556.2428575

Fonseca CM, Fleming PJ et al (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. ICGA 93:416–423

Forrester AI, Keane AJ (2009) Recent advances in surrogate-based optimization. Prog Aerosp Sci 45(1):50–79

Forrester A, Sobester A, Keane A (2008) Engineering design via surrogate modelling: a practical guide. Wiley, Hoboken

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139

Friese M, Bartz-Beielstein T, Emmerich MTM (2016) Building ensembles of surrogates by optimal convex combination. In: Mernik M, Papa G (eds) Bioinspired optimization methods and their applications, pp 131–144

Gaier A, Asteroth A, Mouret JB (2018) Data-efficient neuroevolution with kernel-based surrogate models. In: Proceedings of the genetic and evolutionary computation conference, pp 85–92

Gallagher M, Frean MR, Downs T (1999) Real-valued evolutionary optimization using a flexible probability density estimator. GECCO 99:840–846

Gämperle R, Müller SD, Koumoutsakos P (2002) A parameter study for differential evolution. Adv Intell Syst Fuzzy Syst Evolut Comput 10:293–298

Glover F (1989) Tabu search—part I. ORSA J Comput 1(3):190–206

Goel T, Haftka RT, Shyy W, Queipo NV (2007) Ensemble of surrogates. Struct Multidiscip Optim 33(3):199–216

Goffe WL, Ferrier GD, Rogers J (1994) Global optimization of statistical functions with simulated annealing. J Econom 60(1):65–99

Guo H (2003) A Bayesian approach for automatic algorithm selection. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI03), workshop on AI and autonomic computing, Acapulco, Mexico, pp 1–5

Haftka RT, Villanueva D, Chaudhuri A (2016) Parallel surrogate-assisted global optimization with expensive functions—a survey. Struct Multidiscip Optim 54(1):3–13

Hansen N, Auger A, Finck S, Ros R (2010) Real-parameter black-box optimization benchmarking 2010: experimental setup. Research report no: RR-7215

Hansen P, Mladenovic N (2003) Variable neighbourhood search. Handbook of metaheuristics. Kluwer Academic Publishers, Dordrecht

Hansen N, Müller SD, Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolut Comput 11(1):1–18

Hansen N, Auger A, Finck S, Ros R (2010a) Real-parameter black-box optimization benchmarking 2010: experimental setup

Hansen N, Auger A, Ros R, Finck S, Pošík P (2010b) Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation. ACM, pp 1689–1696

Hansen P, Mladenović N, Pérez JAM (2010c) Variable neighbourhood search: methods and applications. Ann Oper Res 175(1):367–407

Harik G et al (1999) Linkage learning via probabilistic modeling in the ECGA. IlliGAL report 99010

Hauschild M, Pelikan M (2011) An introduction and survey of estimation of distribution algorithms. Swarm Evolut Comput 1(3):111–128

Haykin S (2004) A comprehensive foundation. Neural Netw 2:41

Henderson D, Jacobson SH, Johnson AW (2003) The theory and practice of simulated annealing. In: Handbook of metaheuristics. Springer, pp 287–319

Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554

Hinton G, Deng L, Yu D, Dahl GE, Ar M, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN et al (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Mag 29(6):82–97

Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366

Hu N (1992) Tabu search method with random moves for globally optimal design. Int J Numer Methods Eng 35(5):1055–1070

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. LION 5:507–523

Jaderberg M, Dalibard V, Osindero S, Czarnecki WM, Donahue J, Razavi A, Vinyals O, Green T, Dunning I, Simonyan K et al (2017) Population based training of neural networks. arXiv preprint arXiv:171109846

Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput Fusion Found Methodol Appl 9(1):3–12

Jin Y (2011) Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evolut Comput 1(2):61–70

Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments—a survey. IEEE Trans Evolut Comput 9(3):303–317

Jones DR (2001) A taxonomy of global optimization methods based on response surfaces. J Glob Optim 21(4):345–383

Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. J Glob Optim 13(4):455–492

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: IEEE international conference on neural networks, 1995. Proceedings, vol 4. IEEE, pp 1942–1948

Kerschke P, Trautmann H (2019) Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evolut Comput 27(1):99–127

Khan N, Goldberg DE, Pelikan M (2002) Multi-objective Bayesian optimization algorithm. In: Proceedings of the 4th annual conference on genetic and evolutionary computation, GECCO'02. Morgan Kaufmann Publishers Inc., San Francisco, p 684. http://dl.acm.org/citation.cfm?id=2955491.2955599

Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:14126980

Kirkpatrick S, Gelatt CD, Vecchi MP et al (1983) Optimization by simulated annealing. Science 220(4598):671–680

Kolda TG, Lewis RM, Torczon V (2003) Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev 45(3):385–482

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT Press, Cambridge

Kushner HJ (1964) A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. J Basic Eng 86(1):97–106

Larrañaga P, Lozano JA (2001) Estimation of distribution algorithms: a new tool for evolutionary computation, vol 2. Springer, Berlin

Larrañaga P, Etxeberria R, Lozano JA, Peña JM (1999) Optimization by learning and simulation of Bayesian and Gaussian networks

Lawler EL, Wood DE (1966) Branch-and-bound methods: a survey. Oper Res 14(4):699–719

Leon A (1966) A classified bibliography on optimization. Recent advances in optimization techniques. Wiley, New York, pp 599–649

Lewis RM, Torczon V, Trosset MW (2000) Direct search methods: then and now. J Comput Appl Math 124(1):191–207

Lim D, Jin Y, Ong YS, Sendhoff B (2010) Generalizing surrogate-assisted evolutionary computation. IEEE Trans Evolut Comput 14(3):329

Lindauer M, Hoos HH, Hutter F, Schaub T (2015) Autofolio: an automatically configured algorithm selector. J Artif Intell Res 53:745–778

Liu DC, Nocedal J (1989) On the limited memory BFGS method for large scale optimization. Math Program 45(1–3):503–528

Lizotte DJ (2008) Practical Bayesian optimization. University of Alberta

Lobo F, Lima CF, Michalewicz Z (2007) Parameter setting in evolutionary algorithms, vol 54. Springer, Berlin

Locatelli M (2002) Simulated annealing algorithms for continuous global optimization. In: Handbook of global optimization. Springer, pp 179–229

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58

Loshchilov I, Schoenauer M, Sebag M (2012) Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In: Proceedings of the 14th annual conference on genetic and evolutionary computation. ACM, pp 321–328

Marler RT, Arora JS (2004) Survey of multi-objective optimization methods for engineering. Struct Multidiscip Optim 26(6):369–395

Marsden AL, Wang M, Dennis JE Jr, Moin P (2004) Optimal aeroacoustic shape design using the surrogate management framework. Optim Eng 5(2):235–262

Martin MA, Tauritz DR (2013) Evolving black-box search algorithms employing genetic programming. In: Proceedings of the 15th annual conference companion on genetic and evolutionary computation, pp 1497–1504

McKay B, Willis MJ, Barton GW (1995) Using a tree structured genetic algorithm to perform symbolic regression. In: First international conference on genetic algorithms in engineering systems: innovations and applications, 1995. GALESIA. (Conf. Publ. No. 414). IET, pp 487–492

Mercer RE, Sampson J (1978) Adaptive search using a reproductive meta-plan. Kybernetes 7(3):215–228

Mersmann O, Bischl B, Trautmann H, Preuss M, Weihs C, Rudolph G (2011) Exploratory landscape analysis. In: Proceedings of the 13th annual conference on genetic and evolutionary computation, GECCO ’11. Association for Computing Machinery, New York, pp 829–836. https://doi.org/10.1145/2001576.2001690

Mladenović N, Dražić M, Kovačevic-Vujčić V, Čangalović M (2008) General variable neighborhood search for the continuous optimization. Eur J Oper Res 191(3):753–770

Mockus J (1974) On Bayesian methods for seeking the extremum. In: Proceedings of the IFIP Technical Conference. Springer, pp 400–404

Mockus J (1994) Application of Bayesian approach to numerical methods of global and stochastic optimization. J Glob Optim 4(4):347–365

Mockus J (2012) Bayesian approach to global optimization: theory and applications, vol 37. Springer, Berlin

Molina D, Lozano M, García-Martínez C, Herrera F (2010) Memetic algorithms for continuous optimisation based on local search chains. Evolut Comput 18(1):27–63

Montgomery DC, Montgomery DC, Montgomery DC (1984) Design and analysis of experiments, vol 7. Wiley, New York

Moscato P et al (1989) On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program, C3P Report 826:1989

Müller J (2016) MISO: mixed-integer surrogate optimization framework. Optim Eng 17(1):177–203

Müller J, Shoemaker CA (2014) Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for computationally expensive black-box global optimization problems. J Glob Optim 60(2):123–144

Naujoks B, Beume N, Emmerich M (2005) Multi-objective optimisation using s-metric selection: application to three-dimensional solution spaces. In: The 2005 IEEE Congress on Evolutionary Computation, vol 2. IEEE, pp 1282–1289

Nelder JA, Mead R (1965) A simplex method for function minimization. Comput J 7(4):308–313

Neumaier A (2004) Complete search in continuous global optimization and constraint satisfaction. Acta Numer 13:271–369

Ong YS, Nair P, Keane A, Wong K (2005) Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In: Knowledge Incorporation in Evolutionary Computation. Springer, pp 307–331

Papa G, Doerr C (2020) Dynamic control parameter choices in evolutionary computation: Gecco 2020 tutorial. In: Proceedings of the 2020 genetic and evolutionary computation conference companion, pp 927–956

Pearl J (1985) Heuristics. intelligent search strategies for computer problem solving. The Addison-Wesley series in artificial intelligence. Addison-Wesley, Reading, Reprinted version

Pittenger AO (2012) An introduction to quantum computing algorithms, vol 19. Springer, Berlin

Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Tucker PK (2005) Surrogate-based analysis and optimization. Prog Aerosp Sci 41(1):1–28

Rechenberg I (1973) Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog

Rechenberg I (1994) Evolutionsstrategie ’94. Frommann-Holzboog

Regis RG, Shoemaker CA (2013) Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. Eng Optim 45(5):529–555

Rehbach F, Zaefferer M, Stork J, Bartz-Beielstein T (2018) Comparison of parallel surrogate-assisted optimization approaches. In: Proceedings of the genetic and evolutionary computation conference. ACM, pp 1348–1355

Richter SN, Tauritz DR (2018) The automated design of probabilistic selection methods for evolutionary algorithms. In: Proceedings of the genetic and evolutionary computation conference companion, pp 1545–1552

Rozenberg G, Bck T, Kok JN (2011) Handbook of natural computing. Springer, Berlin

Ruder S (2016) An overview of gradient descent optimization algorithms. arXiv preprint arXiv:160904747

Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989) Design and analysis of computer experiments. Stat Sci 4:409–423

Santana R (2017) Gray-box optimization and factorized distribution algorithms: where two worlds collide. arXiv preprint arXiv:170703093

Schuman CD, Potok TE, Patton RM, Birdwell JD, Dean ME, Rose GS, Plank JS (2017) A survey of neuromorphic computing and neural networks in hardware. arXiv preprint arXiv:170506963

Schwefel HP (1977) Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie. Birkhäuser, Basel

Schwefel HPP (1993) Evolution and optimum seeking: the sixth generation. Wiley, Hoboken

Shanno DF (1970) Conditioning of quasi-Newton methods for function minimization. Math Comput 24(111):647–656

Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: The 1998 IEEE international conference on evolutionary computation proceedings, 1998. IEEE World Congress on Computational Intelligence. IEEE, pp 69–73

Shir OM, Bäck T (2005) Niching in evolution strategies. In: Proceedings of the 7th annual conference on genetic and evolutionary computation. ACM, pp 915–916

Siarry P, Berthiau G (1997) Fitting of tabu search to optimize functions of continuous variables. Int J Numer Methods Eng 40(13):2449–2457

Siarry P, Berthiau G, Durdin F, Haussy J (1997) Enhanced simulated annealing for globally minimizing functions of many-continuous variables. ACM Trans Math Softw (TOMS) 23(2):209–228

Smit S, Eiben A (2011) Multi-problem parameter tuning using bonesa. In: Artificial evolution, pp 222–233

Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp 2951–2959

Søndergaard J, Madsen K, Nielsen HB (2003) Optimization using surrogate models-by the space mapping technique. Ph.D. thesis, Technical University of Denmark

Stanley KO, Clune J, Lehman J, Miikkulainen R (2019) Designing neural networks through neuroevolution. Nat Mach Intell 1(1):24–35

Stork J, Zaefferer M, Bartz-Beielstein T, Eiben A (2019) Surrogate models for enhancing the efficiency of neuroevolution in reinforcement learning. In: Proceedings of the genetic and evolutionary computation conference, pp 934–942

Swersky K, Snoek J, Adams RP (2013) Multi-task Bayesian optimization. In: Advances in neural information processing systems, pp 2004–2012

Talbi EG (2002) A taxonomy of hybrid metaheuristics. J Heuristics 8(5):541–564

Talbi EG (2009) Metaheuristics: from design to implementation, vol 74. Wiley, Hoboken

Törn A, Zilinskas A (1989) Global optimization. Springer, Berlin

Van Groenigen J, Stein A (1998) Constrained optimization of spatial sampling using continuous simulated annealing. J Environ Qual 27(5):1078–1086

van Rijn S, Wang H, van Stein B, Bäck T (2017) Algorithm configuration data mining for CMA evolution strategies. In: Proceedings of the genetic and evolutionary computation conference, GECCO '17. ACM, New York, pp 737–744. https://doi.org/10.1145/3071178.3071205

Vermetten D, van Rijn S, Bäck T, Doerr C (2019) Online selection of CMA-ES variants. In: Proceedings of the genetic and evolutionary computation conference, GECCO '19. ACM, New York, pp 951–959. https://doi.org/10.1145/3321707.3321803

Whitley LD, Chicano F, Goldman BW (2016) Gray box optimization for Mk landscapes (Nk landscapes and MAX-KSAT). Evolut Comput 24(3):491–519

Woeginger GJ (2003) Exact algorithms for NP-hard problems: a survey. In: Combinatorial optimization—eureka, you shrink!. Springer, pp 185–207

Won KS, Ray T (2004) Performance of kriging and cokriging based surrogate models within the unified framework for surrogate assisted optimization. In: Congress on evolutionary computation, 2004. CEC2004, vol 2. IEEE, pp 1577–1585

Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, Bartz-Beielstein T (2014) Efficient global optimization for combinatorial problems. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation, pp 871–878

Zlochin M, Birattari M, Meuleau N, Dorigo M (2004) Model-based search for combinatorial optimization: a critical survey. Ann Oper Res 131(1–4):373–395