Progress in Artificial Intelligence (2019) 8:143–165
https://doi.org/10.1007/s13748-019-00185-z

REVIEW

A review on the self and dual interactions between machine learning and optimisation

Heda Song[1] · Isaac Triguero[1] · Ender Özcan[1]

Received: 20 December 2018 / Accepted: 16 April 2019 / Published online: 25 April 2019

Abstract
Machine learning and optimisation are two growing fields of artificial intelligence with an enormous number of computer science applications. The techniques in the former area aim to learn knowledge from data or experience, while the techniques from the latter search for the best option or solution to a given problem. To employ these techniques automatically and effectively aligning with the real aim of artificial intelligence, both sets of techniques are frequently hybridised, interacting with each other and themselves. This study focuses on such interactions aiming at (1) presenting a broad overview of the studies on self and dual interactions between machine learning and optimisation; (2) providing a useful tutorial for researchers and practitioners in both fields in support of collaborative work through investigation of the recent advances and analyses of the advantages and disadvantages of different techniques to tackle the same or similar problems; (3) clarifying the overlapping terminologies having different meanings used in both fields; (4) identifying research gaps and potential research directions.

Keywords Machine learning · Optimisation · Artificial intelligence · Meta-learning · Algorithm design · Metaheuristic · Hyper-heuristic

1 Introduction

Machine Learning (ML) and optimisation are two different fundamental research fields in computer science [21,22]. Due to the rapid progress in the performance of computing and communication techniques, those two research areas have drawn widespread attention in a wide variety of applications and grown rapidly [66,145]. Although both fields belong to different communities, they are fundamentally based on artificial intelligence and the techniques from ML and optimisation interact frequently with each other as well as themselves in order to improve their learning and/or search capabilities.

ML and data mining are disciplines that focus on the question of how to build computer programs that automatically improve from experience [113]. A typical ML task aims to extract the hidden patterns within data based on an algorithm, which is usually designed by an expert and comprises a set of adjustable hyper-parameters. Thanks to the rapid advance of technology and communication, these algorithms have been successfully applied to many real-world problems ranging from robotics [169], natural language processing [187], computer vision [67] to recommendation systems [37].

Optimisation targets the modelling, design and implementation of solution techniques for finding the best (optimal) entity from a set of candidate alternatives for a given computational search problem [22]. An optimisation process typically starts from a single or a set (population) of initial solutions and moves towards the best solution step by step guided by an objective function. Many computationally hard real-world problems have been tackled by various optimisation techniques, such as travelling salesman problems [108], scheduling problems [150], vehicle routing problems [139], timetabling problems [160], etc.

ML and optimisation have been extensively studied and achieved extraordinary progress in their respective fields; however, they both still suffer from several limitations, such

✉ Heda Song
heda.song@nottingham.ac.uk

Isaac Triguero
isaac.triguero@nottingham.ac.uk

Ender Özcan
ender.ozcan@nottingham.ac.uk

[1] The Automated Scheduling Optimisation and Planning Research Group, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK

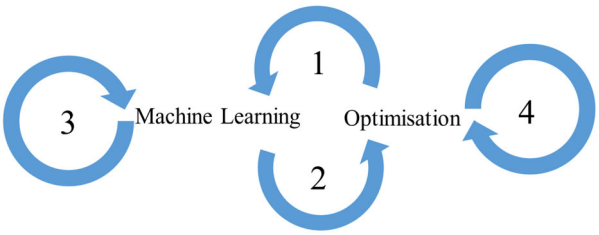

Fig. 1 The interactions between ML and optimisation

as, choosing the best set of parameter values and algorithm components based on human experience for improved performance, solving the problems requiring expensive computation (e.g., evaluation of each case/solution taking a long time), developed solution becoming tailored only to the specific cases handled and not generalising well to the unseen cases, and more. Figure 1 represents different interactions between ML and optimisation that have been proposed to address those issues. As presented in Fig. 1, ML and optimisation have been interacted with themselves to improve their respective performance (interactions 3 and 4). Thus, we define self-interaction as the use of the techniques in a domain to assist the techniques in the same domain. In addition, ML and optimisation have been enhanced by each other (interactions 1 and 2). We define dual interaction as the application of the techniques from two domains to improve each other.

As represented in interaction 1, optimisation techniques have been naturally incorporated into ML as a powerful tool for decades [79,146,189]. Many ML and data mining tasks can be formulated as optimisation problems, such as the training of an ML model by building a loss function and minimising it with respect to model parameters [146], or to optimise the preprocessing of a dataset [189], and also for clustering purposes [79]. Besides, optimisation techniques have also been introduced into ML to tackle the design issues of ML algorithms. At present, many ML algorithms are still designed by experts that manually choose suitable algorithm components and hyper-parameters in order to achieve good performance. This is not coincident with the real aim of artificial intelligence, which is to learn knowledge from experience automatically without human intervention. This has motivated the use of optimisation techniques to tune hyper-parameters of an ML algorithm at a different level [58]. From another point of view, researchers in ML community are addressing the issue by introducing another ML algorithm to extract meta-knowledge that can be further used to guide ML tasks in the base level, which is shown in interaction 3. These approaches are typically called meta-learning [142] or learning-to-learn [170]. Furthermore, interaction 3 can also be found in areas such as algorithm selection [25,141], transfer learning [130] or model training [7,102].

Similarly to ML, many optimisation algorithms also often rely on experts to fix the parameter settings that provide good performance, although there are limited theoretical studies on the parameter setting of some particular algorithms. To choose more reliable parameter values and make the optimisation process more automated, as shown in interaction 4, optimisation techniques have been introduced to improve a base optimisation algorithm, such as using a meta-optimisation algorithm at the high level to tune the parameters of a base algorithm [49] or embedding these parameters into a solution representation and adaptively altering them together with the solution [128]. From another perspective, ML techniques have also been introduced to tackle the same issues [11,195], which belongs to interaction 2. A typical optimisation process generates plenty of data including solution states, associated objective values, etc. The richness of the information within the data has motivated the use of ML techniques to extract hidden knowledge and further use it to configure the parameter settings or algorithmic components [11,195] for improving the optimisation process and the performance of the overall algorithm. In addition, the learned knowledge has also been used to assist in solving unseen optimisation problems in order to accelerate the optimisation process [80,100] and improve the quality of solutions [9,164].

In the past few years, a number of review papers have been published about the dual interactions [18,33,39], individual interactions [5,24,84,97,177] and the interactions that focused on a specific ML or optimisation algorithm [16,56,58,79,117,118,125,189,193,197] between ML and optimisation. Differently, our study provides a global overview of the interaction between ML and optimisation as well as those within themselves as shown in Fig. 1. Since the whole research field is very extensive, in this study, we mainly focus on the first three interactions where ML sits at the core. However, we also provide a brief overview of the interaction 4 to distinguish between different research areas. We clarify the same terminologies having different meanings used in both fields. We also investigate representative approaches and recent advances, presenting a taxonomy for each interaction. Besides, we analyse the advantages and disadvantages of various approaches from different interactions tackling a common task, such as the use of ML and optimisation techniques to tune hyper-parameters for ML algorithms. Furthermore, identified research gaps and potential research directions are presented for future work.

The remainder of this paper is organised as follows. Section 2 provides some background knowledge of ML and optimisation. Section 3 reviews the approaches that use optimisation to improve ML. Section 4 investigates the approaches that use ML to enhance optimisation. A survey of recent progress in meta-learning is presented in Sect. 5. Section 6 presents a brief overview of the approaches that use

optimisation to improve optimisation. Finally, some research gaps and potential research directions are discussed in Sect. 7.

2 Background

Since researchers and practitioners in ML and optimisation mainly focus on their own separate areas, this section makes a brief introduction to those fields as a background. First, Sects. 2.1 and 2.2 provide basic knowledge on ML and optimisation, respectively. Then, Sect. 2.3 distinguishes the terminologies used in both fields.

2.1 Machine learning

ML, which aims to extract knowledge from data, is one of today's hottest research topics in computer science. According to [21], ML tasks can be generally classified into three categories:

- **Supervised learning** The values of input variables and their corresponding values of output variables are known. The learning process is to automatically find some regularities between input variables and output variables. Depending on different learning tasks, supervised learning can be further categorised into classification and regression. Classical supervised learning algorithms include logistic regression, neural networks (NNs), support vector machines (SVMs), decision trees (DTs), k-nearest neighbours (k-NN), etc.
- **Unsupervised learning** The values of input variables are known while the values of output variables are unknown. The learning task is to find some hidden patterns within the data based on input variables. An example of unsupervised learning is clustering, which learns the distribution of data to gather the samples into different groups. Representative unsupervised learning algorithms are k-means, autoencoders, generative adversarial networks, etc.
- **Reinforcement learning (RL)** aims at choosing the most suitable action at a specific state in an environment to maximise the cumulative reward [167]. Classic RL algorithms include Q-Learning, Monte Carlo RL, SARSA, etc.

Although ML has been dramatically developed, it is still a young field with a number of unexplored research opportunities and challenging problems. A few ML tasks target at learning knowledge in extreme scenarios, such as imbalanced data classification [75], big data mining [186], data stream mining [60,161] and few-shot learning [98]. Imbalanced data exhibits an unequal distribution between its classes [75], on which standard ML classifiers cannot perform well because they would pay more attention to majority classes to achieve better overall performance. Big data mining concerns large-volume, complex, growing data sets with multiple, autonomous sources [186]. Data stream mining focuses on learning from streaming data containing concept drifts, which need an efficient learner with continuous learning ability [60]. Few-shot learning works towards learning knowledge quickly from very few examples, while conventional ML algorithms need a lot of data for training and they may suffer from overfitting based on such little data [155]. In addition, most existing ML algorithms are still designed by experts that select suitable hyper-parameters or algorithm components for a specific ML task. However, the real artificial intelligence aims to learn knowledge without human interventions. Therefore how to automate the design of ML algorithms is another challenging problem in ML field. These challenging problems require more sophisticated techniques like meta-optimisation or meta-learning to quickly adapt an ML algorithm without human intervention.

2.2 Optimisation

Optimisation focuses on finding the best (optimal) solution from a group of alternative candidate solutions, which has the general form [22]:

$$\begin{aligned} &\text{minimise } f_0(x) \\ &\text{subject to } f_i(x) \leq b_i,\ i = 1, \ldots, m \end{aligned} \tag{1}$$

Here, x represents the decision variables of the optimisation problem and optimisation algorithms search for the x values that optimise (minimise) f_0, which is the objective function, subject to a set of constraints denoted as f_i.

There are many orthogonal categories of optimisation, such as exact versus inexact methods, single-objective versus multi-objective optimisation, unconstrained versus constrained optimisation, deterministic versus stochastic optimisation, and convex versus non-convex optimisation. More generally, it can be classified into continuous versus discrete optimisation according to the type of decision variables. The common methods for continuous optimisation consist of simplex algorithm, gradient-based methods, such as gradient descent method and its variants, Newton's method and its variants, conjugate gradient method, interior-point methods [6], gradient-free methods such as Bayesian optimisation, heuristics and metaheuristics. These gradient-free methods are also commonly used to solve discrete optimisation problems. Besides, common methods for discrete optimisation also include approximation algorithms, mathematical programming, dynamic programming and hyper-heuristics. To make the content in the following sections more understandable, we briefly introduce some optimisation algorithms here.

Gradient-based methods search among the solution space according to the gradient of a differentiable objective func-

tion [146]. Bayesian optimisation is a sequential design strategy for global optimisation of black-box functions that does not require derivatives [114]. A heuristic is an intuitive approach that may quickly lead to a near-optimal solution, derived based on the problem domain knowledge. A metaheuristic represents "a high-level problem independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimisation algorithms" [158]. Typical methods include single point based search methods, such as tabu search [68] and iterated local search [106] and multi-point (population)-based search methods, including evolutionary algorithms (EAs) [13], ant colony optimisation algorithm [50], particle swarm optimisation (PSO) [88]. Hyper-heuristics focus on automating the design of heuristic methods to solve hard computational search problems [28]. Hyper-heuristics consist of two separate layers. At the high level, the algorithm automatically *selects* or *generates* a number of low-level heuristics performing search over the space of heuristics, while at the low-level where the domain specific components sit, the selected or generated heuristics are invoked performing search over the space of solutions. Selection hyper-heuristics contain two key components: heuristic selection and move acceptance. The improvement is achieved based on often a single point based search which makes iterative moves from a single solution to another via low level operators/heuristics. On the other hand, generation hyper-heuristics build new heuristics based on the predefined components for a given problem operating in a train and test fashion.

Although optimisation has been widely researched, it still suffers from a few weaknesses, such as high computational cost, getting trapped at local optima and choosing suitable parameters for different problems or even instances. These issues have been investigated for years and still not completely solved. In addition, nowadays, real-world optimisation problems are becoming more and more complex with increasing number of decision variables as well as constraints. Efficient and effective advanced intelligent optimisation algorithms are needed more than ever.

2.3 Terminology

Some common terminologies are both used in ML and optimisation fields, while they may represent different concepts. To avoid confusion, we clarify those common key terminologies in Table 1, providing some examples. Note that the number of distinctive terminologies in ML and optimisation is so high that we limit this section to the most common terminologies that are used in both ML and optimisation.

3 Optimisation for machine learning

A typical ML approach usually starts with collecting raw data. Then, data preparation and preprocessing are conducted to shape and clean the data and extract relevant features [53]. After that, a suitable ML algorithm is chosen and a set of hyper-parameters are set by an expert. Then, a model[1] that captures the hidden patterns within the data is built by adjusting the model parameters according to a loss function. Finally, the learned model can be used to do prediction tasks. Optimisation plays a crucial role in many stages of the ML cycle. For example, at the stage of data preprocessing, optimisation algorithms can help to search for relevant features and examples for the subsequent learning process [171,189]. Besides, in the phase of hyper-parameter tuning, instead of setting these hyper-parameters by ML experts, optimisation algorithms can be used to find suitable hyper-parameters in order to make the learning process more automatic. In addition, in the training phase, model parameters learning can be formulated as an optimisation problem by building up a loss function and minimising it. To automate a whole ML cycle, a more general research field has been formed these years known as AutoML, which covered a few above mentioned research problems, such as data preprocessing, algorithm selection and hyper-parameter tuning, etc. A recent survey presented an overview of automated ML, in which a number of approaches that used optimisation algorithms to automate ML tasks, such as feature selection or generation, ML algorithm selection and optimisation algorithm selection for model training, were investigated [192]. A recent book provided a tutorial-level overview of the methods underlying automatic ML [81]. These works also presented some advanced ML techniques for automated ML, such as meta-learning, which we discuss in Sect. 5. Besides, a few competitions on AutoML have also been organised in recent years [70,71]. Other than the stages within a common ML cycle, optimisation has also been used to improve some specific ML areas, such as clustering, ensemble learning.

This section delves into the use of optimisation for ML in five different subsections depending on where optimisation is introduced in a typical ML approach as shown in Fig. 2.

3.1 Data preprocessing

Data preprocessing techniques aim to transform raw data into a reasonable shape for the subsequent learning, which includes data cleaning, dimensionality reduction, instance reduction, discretisation, data balancing, etc. Some of those strategies can be formulated as an optimisation problem. Dimensionality reduction focuses on finding a subset of features from original feature space or extract new features,

[1] Not all ML algorithms build a model, such as k-NN.

Table 1 Clarification of the most common terminologies used in both ML and optimisation

	Machine learning	Optimisation
Instance	A sample from the data, containing one or more features	A specific expression that can be used as an input for a given problem to be solved
	E.g. A single image from a handwriting recognition dataset	E.g. A travelling salesman problem consisting of 10 cities and specific distances between each pair of them
Parameter/hyper-parameter	A parameter refers to an internal variable of an ML model and its value can be estimated from data E.g. The weights of NNs A hyper-parameter is an adjustable parameter that is related to the property of an ML algorithm E.g. The number of clusters in the *k*-means clustering algorithm	A parameter refers to the property of an optimisation algorithm which can be set before or during the optimisation process. This is similar to the term, 'hyper-parameter', used in the ML field E.g. The population size and mutation rate in EAs
Variable	The changeable factors within which we aim to extract hidden patterns	The changeable factors whose values affect the objective function in an optimisation problem
	E.g. The petal length is a variable of the iris plant, which can be used to distinguish different classes of the iris plant	E.g. For a nurse scheduling problem, the number of nurses to employ during the morning shift can be a variable
Iteration	The number of times that an algorithm's parameters are updated while training a model on a dataset	The number of times that the solutions to a problem are updated in optimisation
	E.g. The number of epochs for training NNs	E.g. The number of generations
Solution	A solution in ML can be varied depending on a specific ML task. For a classification task, a solution can be a learned model that distinguishes the samples from different classes. For clustering, a solution can be a few identified clusters. For reinforcement learning tasks, a solution can be a learned policy	A set of variables with certain values, which determines a specific value of the objective in an optimisation problem E.g. A specific route that covers all the cities and finishes at the starting city in a travelling salesman problem
	E.g. A trained SVM model	
Evaluation	The evaluation of an ML algorithm or model can be based on various performance metrics, such as accuracy, precision, F1 score and computational time	The evaluation of an optimisation algorithm is usually based on the quality of solutions and computational time
Model	A mathematical representation of patterns or inferences, which is formed by learning its parameters from the training data	An optimisation model is a mathematical representation of an optimisation problem, which consists of three main components, an objective function, a set of decision variables and constraints
	E.g. A NN model comprised by a number of units and learnable connection weights	
Objective/loss/cost/ fitness function	A loss/cost function is usually used in the ML field, which defined on data, predictions and labels, and measures the penalty E.g. Cross entropy loss function, which is widely used in classification problems	An objective/cost/fitness function is often used in the optimisation field, which represents the main goal of an optimisation problem to be maximised or minimised E.g. The function that maps a route into the total distance for the travelling salesman problem
Representation	A transformation of raw data that captures useful information	A transformation of a solution
	E.g. The vector in the fully connected layer of a convolutional NN is a deep representation of the raw image	E.g. The binary representation for EAs, which represents a solution by a bit string

which are informative, non-redundant and can facilitate the following learning process [72]. Similarly, instance reduction selects a subset of instances from original instance space or generate new instances [26]. Discretisation looks for the best set of cut points to transform numerical or continuous attributes into discrete ones, which can facilitate the subsequent learning and help researchers understand data easily [64]. Data balancing usually oversamples and/or downsamples a new set of examples when the data suffers from the class imbalanced problem [63]. These problems can be transformed into optimisation problems, in which an optimal set of features, instances or intervals are searched in a finite feature,

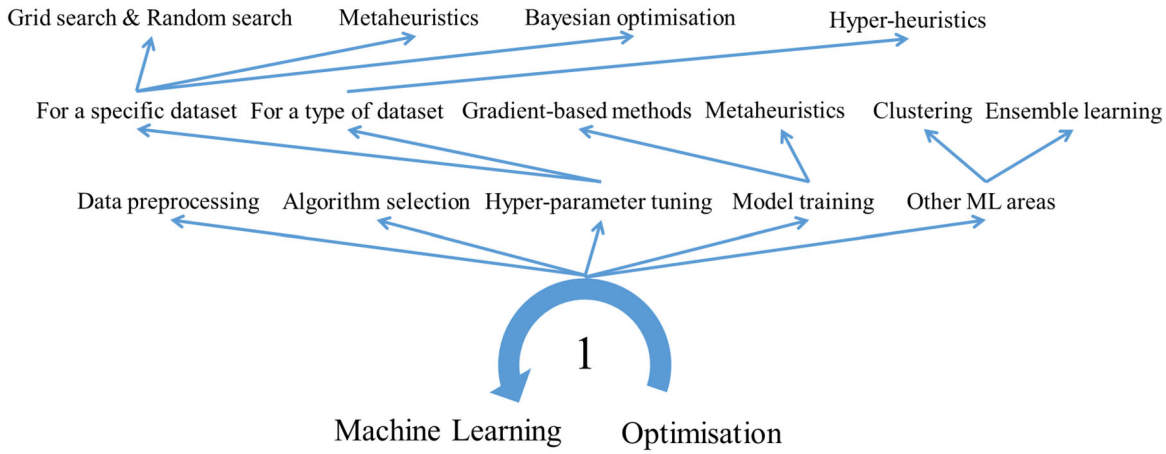

Fig. 2 Interaction 1: optimisation for machine learning

instance or interval space. These optimisation problems have been tackled by means of heuristics [133], metaheuristics [63,137,163,171,189] and hyper-heuristics [115]. Compared to heuristic-based data preprocessing, metaheuristic-based methods have better global search ability by balancing exploration and exploitation. Hyper-heuristic-based approaches provide a more general framework, which automatically chooses suitable heuristics to select features or instances during the search. From the attribute perspective, we can find heuristic optimisation methods [133], metaheuristic methods [189] and also hyper-heuristic optimisation methods [115]. A survey on state-of-the-art evolutionary computation approaches for feature selection was presented in [189], which highlighted the use of genetic algorithm (GA) [77], Genetic Programming, PSO and ant colony optimisation and different ways of representing a solution, search mechanism and performance measure. Then, from the instance perspective, most optimisation techniques are based on EAs [171], while hyper-heuristics have not been introduced yet, which might be of use similarly to feature selection [115] and improve the generalisation ability of instance selection algorithms. As for discretisation, EAs have been frequently used to search the best set of cut points for each attribute, in which binary encoding was usually utilised to determine whether the predefined cut points were adopted [137,163]. Data balancing can be seen as an instance selection problem, so that, it has been tackled by EAs similarly, in which EAs were used to search for a new set of examples from the imbalanced data by downsampling or oversampling with the purpose of balancing data and achieving good learning performance [63].

3.2 Algorithm selection in ML

Algorithm selection is an important task for solving ML problems, because different ML algorithms may have different performance on a specific problem. For example, SVMs may broadly perform better than NNs when the number of training examples is small. Since there are a number of ML algorithms, choosing a suitable one for a specific ML task has been a difficulty for ML researchers, especially for ML beginners. To address the issue, some approaches introduce optimisation techniques to search for a suitable ML algorithm from a pool of those for a specific ML problem. Several representative methods have been further developed into pieces of software for automating the design of ML algorithms, such as Auto-WEKA [168], Auto-WEKA 2.0 [93], auto-sklearn [54] and TPOT [126]. Some of these approaches used Bayesian optimisation techniques [156] to search for suitable ML algorithms and corresponding hyper-parameters as well [54,93,168]. The method in [126] transformed an ML approach into tree-based pipelines including features, algorithms and hyper-parameters, and utilised Genetic Programming [94] to optimise these pipelines.

3.3 Hyper-parameter tuning

Hyper-parameters are adjustable parameters that are related to the property of an ML algorithm, such as the number of clusters in a *k*-means clustering, the number of hidden layers in NNs and the kernel function for SVMs. These hyper-parameters have a great influence on the performance of the subsequent model training. The hyper-parameter tuning for ML algorithms mainly relies on the knowledge of ML experts. To make the learning process more automatic, optimisation algorithms can be introduced to search for promising hyper-parameters without human intervention. Hyper-parameter tuning is a kind of black-box optimisation, in which the objective function is unknown and usually approximated by the performance on the validation sets. This problem has been tackled by grid search, random search, metaheuristics, Bayesian optimisation and hyper-heuristics, and more. Depending on their generalisation ability, those

approaches can be categorised into hyper-parameter tuning for a specific dataset and a type of datasets.

3.3.1 Hyper-parameter tuning for a specific dataset

A typical ML task learns knowledge from a specific dataset, which is usually divided into training set, validation set and testing set. The training set is used for extracting knowledge, the validation set serves to tune hyper-parameters and prevent overfitting, and the testing set provides an evaluation of the learned knowledge. To achieve good generalisation performance on a dataset, hyper-parameters are often tuned by experts according to the performance of the model learned from training set on the validation set. However, to remove human intervention, a few optimisation or search techniques have been introduced to do that task. Grid search and random search are two strategies that are commonly used [19]. In addition, metaheuristics and Bayesian optimisation, which are two effective global optimisation methods for black-box functions, are also used for hyper-parameter tuning in the ML field [59,156].

- **Grid search and random search** Grid search and random search are the two most popular hyper-parameter tuning algorithms because they are easy to implement and parallelisation is trivial. Grid search performs exhaustive search in the hyper-parameter space in order to find the optimal set of hyper-parameters, while this process can be time consuming and even virtually impossible in a very high dimensional space. To address the issue, a more efficient search method that explores the hyper-parameter space randomly was proposed, which is called random search. This approach is able to achieve promising hyper-parameters with less computation time when compared to grid search [19].
- **Metaheuristic-based search** Rather than simply search exhaustively or randomly, metaheuristic-based approaches perform search according to the generalisation performance, which may be beneficial for quick convergence. Such search usually starts from a single or a set of random hyper-parameters. They are altered using some operators, such as mutation or crossover, and improved by selection mechanism according to their generalisation performance on the validation datasets. For example, the approach in [59] encoded the kernel parameter and regularisation parameter of a SVM into a candidate solution (chromosome) of an EA, and a generalisation performance measure was chosen as the fitness function. Other than SVM, several previous studies provided an overview of the approaches that used metaheuristics to tune the hyper-parameters of NNs [56,125,193].
- **Bayesian optimisation** From the point of view of probability, Bayesian optimisation based method for hyper-parameter tuning models a learning algorithm's generalisation performance as a sample from a Gaussian process and evaluates promising hyper-parameters based on the model [156]. In most cases, these approaches are better than previous methods because they use cheaper models to exploit uncertainty to balance exploration against exploitation, which can obtain better results in fewer evaluations. However, they suffer from selecting hyper-parameters of Gaussian processes. In [156], the authors identified good practices for Bayesian optimisation of ML algorithms.

3.3.2 Hyper-parameter tuning for a type of datasets

Some ML tasks learn knowledge from different but related datasets that include similar structural and statistical characteristics and belong to the same domain, such as microarray gene expression data [17]. Even though tuning hyper-parameters of an ML algorithm for each particular dataset could achieve good performance, this procedure can be time consuming. To tackle the issue, some approaches introduce optimisation algorithms to tune hyper-parameters for a type of datasets from the same domain so that we do not have to tune them for every specific task. Hyper-heuristics have been mainly utilised because of their higher generalisation ability compared to other optimisation techniques. Different from metaheuristics based methods, hyper-heuristic-based approaches usually search for a set of promising rules, strategies or methods, which can be further used to design an ML algorithm for different but related datasets.

Most of the existing methods used EA-based hyper-heuristics to select a set of generic hyper-parameters of an ML algorithm for a type of datasets, in which the selection mechanism was based on natural selection and the low-level heuristics were encoded into a gene sequence [17,46]. After the whole run of an EA, the hyper-parameters for designing an ML algorithm tailored to a type of datasets can be obtained. Although these methods can be time consuming, they have been demonstrated to achieve better generalisation performance for a type of datasets so that there is no need to spend additional time tuning hyper-parameters on unseen datasets in the same type. In [17], a hyper-heuristic EA for designing DTs was proposed. In this study, each individual in the EA was encoded as an integer vector and the genes of each individual were related to different rules and parameters of designing a DT, such as the split rules, stopping criteria. The fitness was decided by validation performance. After the whole run of EA, a specific DT induction algorithm for a type of datasets can be obtained. The approach in [46] proposed a similar approach to automatically design

Bayesian network classifier based on hyper-heuristic EA. In this approach, each individual of EA contained 11 genes, the first of which represented one of 12 search algorithms for generating the structure of a network, the other of which represented the strategies and parameters related to the corresponding search algorithm.

3.4 Model training

Model training is the core of an ML approach. Usually, this task can be transformed into an optimisation problem by building a loss function based on an ML model and minimising it with respect to the model parameters. For most ML models, the parameters are continuous variables so that a differentiable loss function can be built in order to allow gradient-based optimisation algorithms to solve the problem [134,146]. However, gradient-based algorithms may be prone to fall into local optima. To address the issue, metaheuristics are sometimes introduced to search for the optimal model parameters utilising their global search ability. In addition, if model parameters are discrete variables or the loss function is non-differentiable, metaheuristics have also been used to train an ML model.

3.4.1 Gradient-based optimisation

The optimisation problems for training an ML model can be broadly divided into convex and non-convex optimisation [82,162]. For example, training a SVM model can be formulated into a convex quadratic programming problem, which can be solved by a sequential minimal optimisation algorithm [134]. In terms of non-convex optimisation, learning the weights of deep NNs can be transformed into a non-convex optimisation problem, which can be tackled by various gradient-based algorithms. These algorithms can be classified into batch gradient descent, stochastic gradient descent (SGD) and mini-batch gradient descent based on how much data was used to compute the gradient in each iteration. These vanilla algorithms suffer from a few problems, such as fluctuation in SGD and choosing suitable learning rate. To overcome these shortcomings, a number of improvements have been proposed, such as Momentum [135], Adaptive Moment Estimation (Adam) [91], etc. An overview of different variants of gradient descent algorithms for training NNs was provided in [146].

3.4.2 Metaheuristic

To prevent model parameters from falling into local optima, metaheuristics have been introduced to train an ML model due to their global search ability. Even though these methods are usually more time consuming, they may achieve better performance. For example, in [56,125,193], several overviews of approaches that optimise the weights of NNs based on various metaheuristics were provided. For some ML algorithms, some of the model parameters are discrete, and therefore gradient-based optimisation algorithms do not work on training these models, instead metaheuristics can be used. For example, the splitting attributes of a DT are discrete model parameters. They can be encoded into a chromosome of an EA and optimised through the evolution process [16].

3.5 Other ML areas improved by optimisation

Apart from the previous stages in which optimisation have been incorporated in the conventional ML cycle, optimisation has also been used for some specific ML tasks, such as clustering [79,116], ensemble learning [34,140] or association rules mining [4,44]. Section 3.5.1 provides a brief overview of optimisation based clustering. Section 3.5.2 investigates the approaches that use optimisation techniques to improve ensemble learning.

3.5.1 Clustering based on optimisation

Different from previously discussed typical ML methods, clustering is an unsupervised learning ML task, whose goal is to determine a finite set of clusters to describe a dataset according to similarities among its objects [86]. It can be considered as an optimisation problem that maximises the homogeneity within each cluster and the heterogeneity among different clusters [8]. Optimisation techniques have been used to directly perform clustering [69,79,152] or to assist the existing clustering algorithms [151]. Most approaches used metaheuristics to directly perform clustering by transforming it into an optimisation problem [69,79,152]. The study in [79] provided a survey of using EAs to perform clustering. A taxonomy that highlighted some very important aspects in the context of evolutionary data clustering was presented in this review. A recent survey provided an investigation on multi-objective evolutionary clustering techniques [116]. Conversely, some other approaches introduced metaheuristics to assist the existing clustering methods, such as using GAs to initialise the k-means algorithm [151].

Some other approaches utilise hyper-heuristics to do clustering, in which suitable heuristics are automatically chosen during iterations. Compared to metaheuristic-based methods, these approaches are more generic for different clustering problems and have a lower probability of falling into local optima. For example, in [38], the authors proposed a hyper-heuristic-based approach for web document clustering. The approach used random selection and roulette wheel selection to choose the low-level heuristic based on their performance calculated by Bayesian information criteria. Furthermore, two acceptance strategies were used, replacing the worst and restricted competition replacement. The approach in [172]

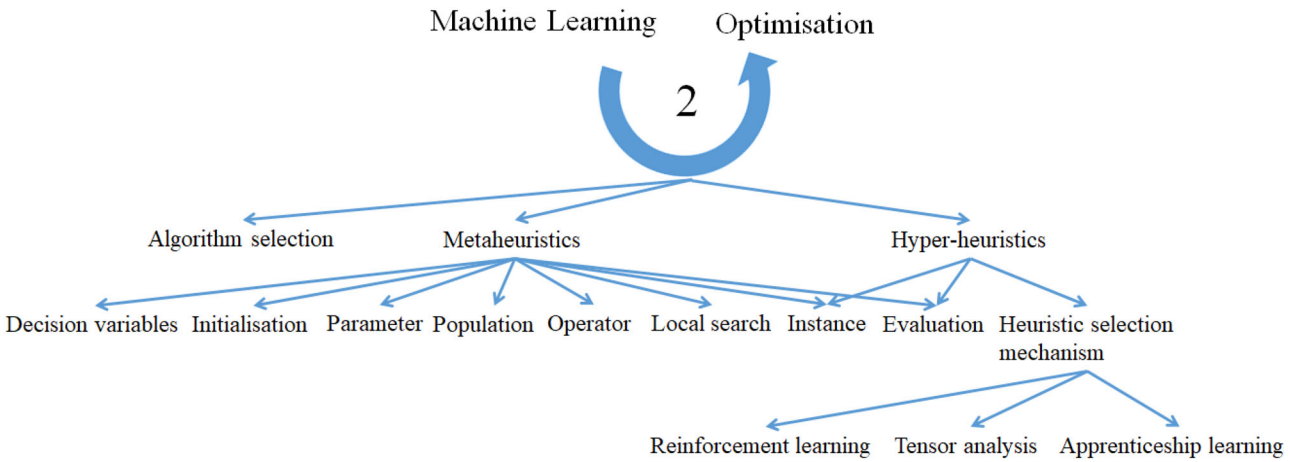

Fig. 3 Interaction 2: machine learning for optimisation . (this taxonomy is adapted from [84])

proposed a hyper-heuristic clustering algorithm, which chose three metaheuristics, simulated annealing, tabu search, GAs, and one classic clustering method, *k*-means, as low-level heuristics. One of the methods was selected iteratively based on their previous performance, which may prevent a fixed algorithm from being trapped in local optima.

3.5.2 Ensemble learning based on optimisation

Ensemble learning combines the predictions of individual ML models to obtain better performance [140]. Conventional ensemble methods include bagging, boosting and random forest, decomposition methods, negative correlation learning methods, fuzzy ensemble methods, multiple kernel learning ensemble methods, etc [140]. As [74] pointed out, the key to successful ensemble learning is to construct individual predictors which are accurate and disagree with one another. This means a good ensemble learning approach should consist of a set of accurate and diverse predictors. Therefore, it is natural to incorporate multi-objective optimisation techniques into ensemble learning to search for a set of individual predictors that are both accurate and diverse. Since multi-objective EAs can produce a set of trade-off predictors in the form of a non-dominated set [118], they have been used frequently to assist ensemble learning [34,35,61]. An example can be found in [34], this approach used multi-objective EAs to find the optimal trade-off between diversity and accuracy of an ensemble of NN classifiers. Each particular NN was treated as an individual in the population and the final ensemble was the non-dominated set of NNs in the final generation.

4 Machine learning for optimisation

There are many approaches that use ML techniques to improve optimisation algorithms, mainly metaheuristics and hyper-heuristics. The aims of introducing ML to optimisation are to speed up the search process and to improve the quality of solutions. The general approach of this field is to use ML techniques to extract knowledge from the data gathered from optimisation processes. Then, the extracted knowledge, which is usually represented by a model or rules, is used to tune or substitute for a component of an optimisation algorithm. Multiple overviews have recently covered the interactions between ML and metaheuristics [33,39,84,197], however, to the best of our knowledge, there is no survey presenting a categorisation on the way that ML is used for enhancing hyper-heuristics. In addition, ML techniques can also be used to choose the best performing algorithm for a particular optimisation problem. Section 4.1 provides a brief overview of using ML to enhance metaheuristics based on previous review papers. Section 4.2 investigates the approaches that used ML to improve hyper-heuristics. Section 4.3 presents the latest advances on algorithm selection in optimisation based on ML techniques. Overall, the ways of applying ML to improve optimisation are shown in Fig. 3.

4.1 Machine learning for metaheuristic optimisation

A metaheuristic is a high-level problem independent algorithmic framework that provides a set of strategies to develop heuristic optimisation algorithms [158]. This framework is general, and it can be applied to different optimisation problems in the same domain or even in different domains. However, this problem solving process can be time consuming and the method may be prone to fall into local optimum. To tackle these problems, ML techniques have been used to accelerate the search process and improve the quality of solutions. There are several previous review works about improving metaheuristics by ML [33,39,84,197]. They provided similar taxonomies and one of them proposed in [84] depended on three different criteria, namely localisation, aim

and kind of knowledge. Based on this taxonomy, we summarise the recent advances in the literature.

Depending on the localisation where ML techniques are embedded in an optimisation algorithm, the approaches that used ML to improve metaheuristics can be categorised into eight classes, specifically evaluation, decision variables, parameters, initialisation, population, operators, local search and problem instances.

- **Evaluation** For some cases in optimisation, the objective function is hard to be expressed by a mathematical model or computationally expensive, for example, many engineering problems require simulations to evaluate a solution. To solve the problem, ML techniques are usually used to approximate the objective function of a optimisation problem, which is also a branch of surrogate models [15], or reduce the number of solutions to be evaluated. A recent approach in [131] used generalised regression NNs to approximate the fitness function in PSO to speed up evaluation. Several overviews of using surrogate models to assist evolutionary computation were provided in [41,48,83]. To reduce the number of solutions to be evaluated, some other methods used a clustering algorithm to group the population of a metaheuristic and then only chose representative individual from each cluster to be evaluated [90]. Some other approaches focused on multi-objective optimisation problems. These methods used ML techniques, such as principal component analysis [148] or feature selection techniques [104], to reduce the number of objectives in a multi-objective optimisation problem for the sake of less computational burden and complexity.
- **Decision variables** An optimisation problem usually includes a set of decision variables, whose values need to be determined to solve the problem. Conventional metaheuristics seldom take into account the scalability of the number of decision variables. However, this factor sometimes may influence the performance of an optimisation process, especially for problems with large-scale decision variables. To fill the gap, some approaches utilise ML techniques to deal with decision variables so that they can be optimised efficiently. A recent EA-based approach introduced a clustering algorithm to separate the decision variables into convergence-related and diversity-related ones, and two different EA strategies were applied to dealing with the two classes of decision variables, respectively [198].
- **Parameters** A metaheuristic typically consists of many adjustable parameters, which have great impact on the performance of an optimisation process. To choose promising parameters automatically, ML techniques have been introduced to choose suitable ones before an optimisation process (parameter tuning) or even adjust them adaptively during the run of an optimisation algorithm (parameter control). For example, in EAs, we have population size, crossover probability and mutation probability, among others. These parameters can be adaptively set using ML algorithms. One instance was given in [196], in which a clustering algorithm was used to find different subgroups within the population of a GA, then crossover probability and mutation probability were adaptively adjusted based on the relative size of the cluster containing the best chromosome and the one containing the worst chromosome.
- **Initialisation** Normally, a metaheuristic starts searching from a single or a number of randomly generated initial solutions, which may include some bad solutions. ML algorithms can be used to help generate a number of relatively promising initial solutions, which can accelerate convergence. For example, a clustering based initial population strategy is proposed for travelling salesman problem [45], in which the cities were clustered into several groups based on k-means and then an initial solution can be obtained by using a GA to find the local optimal path for each group and a global optimal path connecting different groups.
- **Population** Some metaheuristics are population-based methods, such as EAs and PSO, in which a population comprised with a number of solutions evolves iteratively. The diversity and quality of the population highly influence the results. To improve the quality of solutions, ML techniques can be introduced to maintain population diversity or predict promising regions. Clustering has been the most frequently used ML method to maintain population diversity. The approach in [164] used a clustering algorithm to group the population of a GA into several sub-populations and operate selection within each sub-population. To predict promising region in the search space, several ML algorithms were used, such as clustering algorithms [87] and SVMs [191], etc. The approach in [191] learned the mapping between best individuals and their corresponding fitness function values based on SVMs in each generation of an EA, and then searched for the promising individuals according to the learned model.
- **Operators** A metaheuristic normally has some operators to alter solutions, such as selection, crossover and mutation operator in EAs. To accelerate the convergence of an optimisation process, ML techniques have been applied to adaptively choosing suitable operators for each problem state during searching. RL is a frequently used ML algorithm. For instance, the approach in [195] proposed an adaptive evolutionary programming algorithm based on RL, in which the optimal mutation operator was chosen for each individual based on immediate and delayed performance of mutation operators.

– **Local search** metaheuristics are often hybridised with local search to exploit local region of candidate solutions in metaheuristics. ML techniques can be applied to helping local search. For example, in [109], clustering algorithm was used to cluster the population into several groups and the best individual in each group was refined by local search. However, the local search can be computational expensive. To address the issue, the approach in [183] proposed a pruning strategy for bee colony optimisation algorithm [184,185], which employs the bi-directional extension based frequent closed pattern mining algorithm to only allow relatively better bees to undergo local optimisation.
– **Problem instances** An instance indicates a specific description that can be used as an input for a given problem in optimisation. For some optimisation problems, the number of existing real-world or even synthetic instances are limited. More importantly, there might be many problem instances but they might not differ much from each other when their characteristics (instance features) are considered. It is not trivial and reliable to evaluate an optimisation algorithm based on such insufficient instances, since they cannot possibly cover all the representative regions in the instance space. To address that issue, ML techniques have been used to generate synthetic instances with varying characteristics from different regions of the instance space. It should be noted that this method does not assume any particular optimisation method and useful purely for a better and informed performance comparison of optimisation methods, including metaheuristics and hyper-heuristics. The approach in [103] is an illustration of this research topic, which used DTs to classify the real and generated instances into their corresponding classes and the instance generator was iteratively modified according to the feedback from the classifier in order to generate more real instances.

Based on the type of knowledge that ML techniques extract from the optimisation processes, the approaches can be categorised into the ones that use a priori knowledge and the ones that use dynamic knowledge [84]. From the perspective of ML, these could also be classified as metaheuristics that use online (dynamic) and offline (a priori) ML techniques. For the methods that use dynamic knowledge or online ML techniques, knowledge is extracted in each iteration to guide the following search process. For example, as discussed previously, the approach in [195] used RL to learn optimal policy of choosing mutation operators. The policy was updated dynamically based on the performance of mutation operators every iteration. For the approaches that use a priori knowledge or offline ML techniques, knowledge is extracted by an offline ML algorithm from solving different instances in a specific optimisation problem domain and use the knowledge to help solving unseen instances in the same problem domain [190,191].

Focusing on the main goal for which ML is introduced in metaheuristics, we can classify these approaches into two categories, to speed up search process and to improve the quality of solutions. Some of previously discussed approaches aimed to speed up search process, such as using ML algorithms to approximate costly objective function [80] and to generate promising initial solutions [190]. Some of them aimed to improve the quality of solutions, such as using ML techniques to maintain population diversity [164] and to choose suitable operators for individuals [195].

4.2 Machine learning for hyper-heuristic optimisation

The main difference between metaheuristics and hyper-heuristics is that metaheuristics use a pre-designed algorithmic framework comprised by specific parameters and operators to solve a problem, while hyper-heuristics utilise selection or generation mechanisms to automatically design an optimisation algorithm by selecting or generating suitable heuristics during the search process. Hyper-heuristics typically have a stronger generalisation ability than metaheuristics and can be applied to different optimisation problems without significant modifications. In the literature, there is a certain degree of confusion between hyper-heuristics and the approaches discussed in Sect. 4.1 that used ML algorithms to adaptively select suitable parameters and operators for metaheuristics. Even though researchers in different fields use different terms to describe them, actually they share the same goal of automatically designing an algorithm for an optimisation problem. Concentrating on the difference between metaheuristics and hyper-heuristics, we divide this section into two subsections. The first presents an overview of the approaches that use ML techniques to improve selection or generation mechanism in hyper-heuristics, which is missing in metaheuristics. The second reviews the methods that aim to accelerate the evaluation of hyper-heuristics, which can also be found in a hybridised metaheuristic.

4.2.1 Learning to select heuristics

There are many approaches that apply ML techniques to learning how to select heuristics in hyper-heuristics. Depending on the ML techniques they used, these methods can be generally classified into three categories, RL-based approaches, tensor analysis based approaches and apprenticeship learning approaches.

4.2.1.1 RL-based approaches

In this branch, these methods use RL to learn a heuristic selection mechanism in an online manner, in which the selection mechanism keeps changing according to the performance of low-level heuristics. They can be further split into two classes, using only RL, hybridising RL with other algorithms.

- **Using only RL** A simple and intuitive way of learning the best low-level heuristic during the optimisation process may be based on classical RL, so that good performing heuristics are positively rewarded while bad performing heuristics are negatively punished. Two approaches in [95,121] used such simple RL scheme to reward and punish the weights of low-level heuristics based on their previous performance and selected low-level heuristic using roulette wheel method according to their weights in each iteration. Rather than using such simple RL scheme, a more complex RL method, Q-learning, was introduced to learn to select heuristics in [36], which strictly follows the criteria of RL. In [47], the author proposed a RL-based hyper-heuristic method that fulfilled the criteria of RL. In their approach, several variants of RL were investigated to test their performance on selecting heuristics.
- **Hybridising RL with other methods** Only using RL as heuristic selection mechanism may result in sticking to a specific heuristic if the heuristic is heavily rewarded previously. To address the issue, Tabu search was introduced to prevent a specific heuristic from being chosen for certain times [31]. Previous approaches that used RL to select heuristics only focused on which heuristic was most suitable in each iteration. However, choosing a suitable sequence of heuristics may benefit more for a consecutive optimisation process. Therefore, a Markov chain and RL-based hyper-heuristic method was proposed in [110], in which each state of Markov chain was represented by a low-level heuristic and the transition weights of Markov chain was updated adaptively by RL based on the probability of generating dominating solutions. This method did not only learn which low-level heuristic was effective, but also focused on which sequence of low-level heuristics was promising. Besides, RL can be utilised to improve the heuristic-based selection mechanism of a hyper-heuristic. The approach in [51] used RL to improve Choice Function heuristic selection mechanism [40] by adaptively adjusting the weights of measures in the choice function. If no improvement was made, the intensification weight would be richly rewarded; otherwise, the diversification weight would be heavily punished.

From a critical viewpoint, the author of [3] presented a theoretical analysis of the limitation of using RL as selection mechanism in hyper-heuristics. Their theoretical results showed that if the probability of improving the candidate solution in each point of the search process is less than 50% which is a mild assumption, then additive RL mechanisms perform asymptotically similar to the simple random mechanism.

4.2.1.2 Tensor analysis based approaches

Tensor analysis, also known as tensor calculus, is a collective term for the techniques that investigate high dimensional data and extract latent patterns and correlations between various modes of data [11]. It is used for elegant and compact formulation and presentation of equations and identities in mathematics, science and engineering. It is also used for algebraic manipulation of mathematical expressions and proving identities in a neat and succinct way [157]. Some hyper-heuristic approaches use such technique to extract knowledge for heuristic selection. Rather than directly choosing a promising heuristic in each iteration, these approaches select a suitable group of heuristics for a move acceptance method based on tensor analysis in order to group heuristics that work well with each move acceptance method [10,11]. The approach in [10] is an offline learning process, in which low-level heuristics were grouped based on tensor analysis once at the beginning of the search. In [11], the authors further extended the method to an online learning manner and made it more flexible. In this method, the low-level heuristics were partitioned dynamically during the search process and the selected heuristics can be overlapped between different groups.

4.2.1.3 Apprenticeship learning approaches

Promising hyper-heuristics, such as AdapHH [112] which was the winner of a cross domain heuristic search challenge among twenty competitors, consist of effective manually designed heuristic selection mechanism. Instead of directly improving these methods, some approaches use ML algorithms to learn selection mechanism from them and may achieve better performance on some problem instances. These approaches can be seen as apprenticeship learning, which is a process of learning by observing an expert [1]. These apprenticeship learning methods use different ML algorithms, such as NNs and DTs, to learn heuristic selection mechanism from experts, such as AdapHH and choice function [40]. Most of them learn in an offline manner. The approach in [173] trained a time delay NN to extract heuristic selection mechanisms from the training examples that were generated by a promising heuristic selection method, choice function. Then, the time delay NN was used as a

classifier to select suitable low-level heuristic for solving unseen problems. In [9], several DTs were trained to imitate heuristic selection of a promising hyper-heuristic method, AdapHH. Before training, several datasets were constructed by applying AdapHH to an optimisation problem and collecting search states. Then, predictors were trained based on these datasets by means of supervised learning. The previous approaches used promising hyper-heuristics as experts, while some other methods chose some evaluation functions as experts. In [127], the author used NNs to learn the hidden patterns between problem states and the promising low-level heuristics, in which the problem states were represented by constraint density and tightness. The approach in [194] learned recurrent neural networks (RNNs) [78] to predict next suitable heuristic based on a set of promising heuristic sequences according to the final log return. The trained RNNs can be used later to generate a sequence of heuristics on an unseen problem.

4.2.2 Learn to accelerate evaluation

In addition to learning heuristic selection mechanism, there are also a few approaches aiming at speeding up the whole hyper-heuristic process with the help of ML techniques. Generally, they learn to evaluate the solutions by classification or regression methods.

- **Classification** As discussed in Sect. 4.1, evaluating solutions sometimes can be time consuming. Different from the approaches that use ML to speed up metaheuristics, accelerating hyper-heuristics can be treated as a classification problem, because hyper-heuristics include an acceptance mechanism that decides whether a solution is accepted or rejected. Therefore, a classifier can be trained to classify a solution into an acceptable one or an unacceptable one without explicitly computing the objective function. In [99], the authors focused on finding global hidden patterns in large data sets of heuristic sequence. The approach used NNs and logistic regression to classify the intermediate solutions generated by the hyper-heuristic into "good" solutions and "bad" solutions. Another method in [100] transformed a schedule as a pattern and trained a NN to learn if the pattern is good or not. Inspired by [100], another approach utilised a NN to learn if the relative changes in a schedule would improve the performance [174], which was faster than [100].
- **Regression** Another way to accelerate the evaluation for hyper-heuristics is to train a regression model to approximate computationally expensive objective function, which follows the same way that uses ML techniques to accelerate evaluation for metaheuristics and is also related to surrogate model. The approach in [159] proposed an evolvability metric estimation method based on a parallel perceptron, which accelerated the online heuristic selection process. In this study, the fitness value of an incoming solution was estimated by a single layer of perceptrons.

4.3 Machine learning for algorithm selection in optimisation

The algorithm selection in optimisation was first proposed by [142] aiming at answering the question: *which algorithm is likely to perform best for my (optimisation) problem?* This problem can be transformed into a learning task by mapping the features of problem instances to the best performing algorithm or algorithm performance. The algorithm selection in this section is related to the algorithm selection mechanism of hyper-heuristics. The former aims to perform algorithm selection before solving a problem while the latter executes algorithm selection during the iterative search process. However, the former can also be transformed into an online selection approach, which selects a suitable algorithm during the search, and solved by hyper-heuristics. However, it is noteworthy that the representation of the solutions generated by different algorithms during the search should be in the same form. For example, we cannot directly pass the solutions represented by a binary string of an EA to a PSO algorithm which represents the solutions by real numbers. This insight prevents most ML algorithm selection problems (Sect. 3.2) from being solved by hyper-heuristics, because most ML algorithms use different models to solve a problem. However, algorithm selection for clustering can be an exception, since the solutions generated by different clustering algorithms can be kept in the same form as clusters, which is discussed in Sect. 3.5.1. There has been a growing number of studies on algorithm selection in optimisation, recently. The approach in [154] extended the framework of algorithm selection in [142] and introduced a case study on graph colouring problem. Rather than only choosing the best performing algorithm, this method aimed to use the metadata to identify the strengths and weaknesses of algorithms, in which a few ML techniques, such as Naive Bayes classifiers, SVMs or NNs, were used based on the type of performance metrics. A recent work in [120] focused on algorithm selection on continuous black-box optimisation problem. This method first used exploratory landscape analysis to produce a set of instance features, then applied correlation analysis to select relevant features, finally used SVMs to learn the mapping between instance features and the best performing algorithm or some other algorithm performance measures. A case study on algorithm selection for the generalised assignment problem was provided by [42], in which Random Forest was used to learn the mapping between instance features and suitable algorithms. This work also discussed a few practical

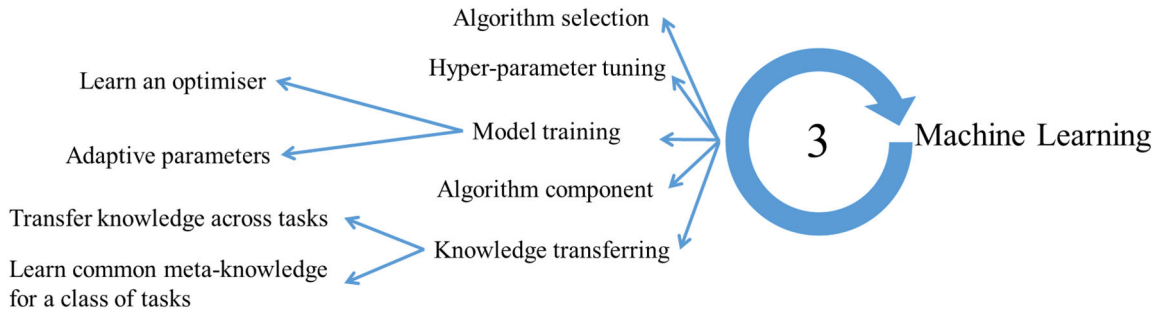

Fig. 4 Interaction 3: machine learning for machine learning

issues when applying algorithm selection, such as deciding whether to conduct algorithm selection or not. A latest survey provided an overview of algorithm selection on continuous and discrete optimisation problems with the help of ML techniques and also discussed several related problems, such as automated algorithm configuration and algorithm schedules [89].

5 Machine learning for machine learning

In Sect. 3, we have discussed how to make an ML approach more automatic from the point of view of introducing optimisation techniques. However, searching for a suitable ML algorithm and its corresponding hyper-parameters through optimisation techniques sometimes can be time consuming. Due to the powerful learning ability of ML techniques, researchers filled the gap by applying another ML algorithm at the meta-level to learn meta-knowledge from fulfilling base-level ML tasks and then using the learned meta-knowledge to guide unseen ones. Such approach is called meta-learning, also known as learning to learn. Several previous works provide overviews of meta-learning [5,24,97,175,177]. The most recent one in [175] categorised meta-learning techniques depending on the type of meta-data they leveraged, in which the meta-data were broadly classified into model evaluations, task properties and prior models. From another point of view, we provide a categorisation based on the type of meta-knowledge as shown in Fig. 4. This section presents an investigation on the recent progress in meta-learning. Most of these approaches focus on NNs, which are further discussed in the following subsections.

5.1 Learn to select algorithms

Rather than determination of an ML algorithm by experts, the meta-learning methods in this field work towards using ML techniques to learn how to automatically select appropriate algorithms for an ML task. This question can also be tackled by optimisation techniques as discussed in Sect. 3.2; however, using optimisation algorithms to search for a suitable algorithm can be time consuming, because we need to perform an ML algorithm during each optimisation iteration. To address the issue, some meta-learning approaches utilise ML algorithms to learn to select algorithms, in which the meta-model can efficiently choose suitable algorithms after training. Generally, these methods learned a classifier that determined a specific ML algorithm or a ranking of ML algorithms based on the characteristics of a task [25,141], or a regression model that captured the mapping between the characteristics of ML tasks and algorithm performance [62]. A recent approach took into account runtime and incorporated multi-objective measures that comprised of accuracy and runtime into two algorithm selection methods, average ranking and active testing, to accelerate the selection process [2].

5.2 Learn to tune hyper-parameters

The meta-learning approaches in this class concentrate on learning how to tune hyper-parameters for an ML algorithm. Different from the methods that use optimisation algorithms to search for promising hyper-parameters in Sect. 3.3, most of these approaches aim to learn the mapping between hyper-parameters and generalisation performance, which is usually represented by algorithm's performance on the validation datasets or the model parameters leading to good generalisation performance. Then hyper-parameter tuning can be conducted efficiently based on the learned mapping. Compared to the optimisation based methods that search among the hyper-parameter space, these meta-learning approaches are much more time-saving because they do not need to train a model for each set of hyper-parameters during searching, instead they evaluate a model by predicting the generalisation performance based on the learned mapping model. Even though training the mapping would take some time, hyper-parameter tuning can be fast after training. A recent approach in [105] introduced NNs to learn the mapping between hyper-parameters and the approximate optimal parameters of a base model. In their study, the optimal base model can be

directly given without time consuming training for each set of hyper-parameters, which can accelerate the hyper-parameter tuning. Furthermore, using NNs to approximate the black-box function between hyper-parameters and the optimal model parameters allowed gradient descent algorithms to be introduced to find promising hyper-parameters. Another two approaches trained regression models, such as support vector regression and NNs, to predict the performance or weights of NNs with respect to their architecture in order to speed up the search for the promising architecture of NNs [14,27].

5.3 Learning to train a model

These approaches focus on learning an efficient meta-learner that can be used to optimise the parameters of an ML model. Conventional ML algorithms typically use a manually designed optimisation algorithm, such as gradient-based methods, to train a model by building a loss function and minimising it with respect to the model parameters as discussed in Sect. 3.4. However, such process sometimes can be time consuming and stuck in local optima. To overcome these weaknesses, meta-learning methods in this class work towards learning an efficient optimisation algorithm for the sake of faster convergence and better performance. These approaches can be categorised into learning an optimiser and fast adaptive parameters based on whether the model parameters are iteratively updated.

- **Learning an optimiser** These approaches target at learning an optimiser that can be used to update the model parameters iteratively based on a sequence of gradients and objective values for a class of ML tasks. The two most representative approaches changed the design of an optimisation algorithm within an ML method into a learning problem [7,102]. The method in [7] trained a RNN as an optimiser to update the weights of a base ML model based on the gradient information. In [102], the authors transformed a parameter optimisation problem into a RL one, in which a policy was trained as an optimiser to update the model parameters iteratively. These approaches both outperformed the existing hand-designed optimisation algorithms on some ML problems, while they were not as stable as hand-designed algorithms due to the limited learning ability of the meta-learner. Also, they were restricted to solve a class of problems. To address the issue, the approach in [181] proposed learned optimisers that scale and generalise by introducing a hierarchical RNN architecture, which had high generalisation ability to new tasks.
- **Adaptive parameters** Rather than learning an optimiser that updates the model parameters iteratively, some approaches aim at learning to generate quick adaptive model parameters. Since these methods predict model parameters in a feedforward manner without the backward propagation of errors, they are much faster than those in the first branch while the quality of the generated parameters may be not guaranteed. Some methods in this branch focused on generating adaptive weights for different time steps in RNNs [73,149]. Different from conventional RNNs that share weights across all the time steps, these approaches meta-learned another RNN that predicted specific weights for each time step of the base RNN. The adaptive weights allowed the base RNN to process each element in the input sequence differently, which can improve the performance. Some other approaches focused on producing adaptive model parameters for different tasks. In [119], the model parameters were divided into slow weights and fast weights. Slow weights were generic weights across tasks, while fast weights were task-specific weights, which were outputted by a trained meta-learner.

5.4 Learning an algorithm component

An algorithm component can be seen as a discrete hyper-parameter of an ML algorithm, such as an activation function in NNs and a splitting criterion in DTs. Sections 3.3 and 5.2 focused on selecting such hyper-parameters from a pool of manually designed algorithm components. However, some meta-learning approaches aim to learn an algorithm component by embedding another ML model into a base one to represent the algorithm component and training them jointly. These approaches can achieve better performance because the algorithm components are automatically learned for a specific ML task while hand-designed ones are generic. A recent approach in [176] explored task-specific activation functions based on NNs rather than choosing a suitable hand-designed one. The experimental results showed that the explored new activation functions were different from the common ones and achieved better performance. Some other approaches worked towards exploring a promising policy or strategy in an ML algorithm, which is usually designed by experts. In [12], a meta-learning method was proposed to learn the strategy of selecting examples to label for active learning based on Matching Networks [178]. Another approach in [188] learned a splitting criterion for designing a DT based on a RNN, in which the learned RNN controller was used to predict the splitting feature at each non-leaf node. In [43], a strategy to control the magnitude of gradient updates was proposed for weakly labelled data. A confidence network was meta-trained to weight every update for the parameters of a base model in order to alleviate the influence of noisy data.

5.5 Learning to transfer knowledge

In ML, many different ML tasks are related to each other, such as handwriting recognition in different languages. Similarly, in the optimisation field, a lot of different optimisation problems are closely connected, for example different instances of the travelling salesman problem. These relevant tasks or problems contain some useful generic knowledge that can be utilised to assist other similar tasks or problems. In order to achieve faster convergence and better performance, some methods discussed in Sect. 4 focus on extracting common knowledge from solving a number of instances in a specific optimisation problem domain and apply the knowledge to solving unseen instances in the same domain. For the same purpose, in the ML field, some meta-learning approaches extract the meta-knowledge from one ML task and transfer it to a different but related task, or extract the common meta-knowledge across a particular class of tasks and apply it to unseen tasks in the same class. Even though the approaches in different fields solve different problems, they share the same goal of improving algorithms by transferring knowledge.

- **Transfer knowledge across tasks** Some approaches extract the meta-knowledge from one ML task and transfer it to another different but related task. In [180], the authors proposed a meta-learning method to model the minority class of imbalanced datasets. They transferred the knowledge extracted from data-rich classes to data-poor classes by learning the mapping between the parameters of many-shot model and those of few-shot model on data-rich classes and transferring it to data-poor classes. This approach presented a new idea of learning from imbalanced data. A comprehensive overview of this field was provided in [130]. This study focused on categorising and reviewing transfer learning for classification, regression and clustering problems.
- **Learn common meta-knowledge for a class of tasks** Some other approaches aim to extract the common meta-knowledge, which can be shared within a class of tasks. For example, the approach in [144] learned a set of basic function blocks that were shared across different tasks. These function blocks were iteratively selected by a trained router to construct a model for a specific task. A similar method in [57] learned a group of shared basic policies for a distribution of RL tasks, which can be quickly switched between by a trained master policy for unseen tasks. Another approach for RL learned a shared baseline reward function for a class of tasks [101]. The task-specific reward functions were estimated as a probabilistic distribution conditioned on the baseline reward function.

  Many other approaches in this research area concentrate on a specific challenging problem, few-shot learning, in which a concept need to be learned from only one or few examples. This problem cannot be tackled by conventional ML algorithms due to the limited number of examples. To address the issue, many meta-learning methods were proposed in this field. They aim to extract the common meta-knowledge across different few-shot learning tasks within a particular distribution and apply it to unseen tasks in the same distribution. These approaches can be roughly categorised into two classes. The first learns a common feature extractor for a distribution of few-shot learning tasks and do classification by measuring the similarity between the extracted features of examples [65,92,111,147,155,166,178]. These approaches usually chose convolutional neural networks (CNNs) [96] or long short-term memory (LSTM) [76] as a common feature extractor and used Euclidean distance, cosine distance or learned distance metric to measure similarity. The second class focuses on fast parameterisation of a base model by learning a common promising initialisation, parameter updating rules or adaptive parameters for a distribution of tasks [20,55,119,124,136,138]. The learned common initialisation or parameter updating rules allowed the model parameters to be quickly tuned to task-specific ones based on few examples.

6 Optimisation for optimisation

Similarly to meta-learning in ML, optimisation can also be improved by self-interaction. There are various ways how an optimisation algorithm can be used to improve itself or an other optimisation algorithm as shown in Fig. 5. Generally, these approaches perform search at a high-level and operate in an offline or online manner. Since there are extensive studies providing comprehensive reviews of the relevant well-established research fields ranging from automated algorithm generation to memetic computing [23,28,29,52,85,122,123,153,165], we briefly cover a few selected studies in this section to point out those different research areas.

Hyper-heuristics are search methods to select or generate (meta)heuristics; hence, they can be considered as optimisation methods improving individual performance of constituent heuristics for improved performance [28,30]. Automated generation of heuristics is of interest in many fields and Genetic Programming is one of the most commonly used tools as an optimisation technique to generate components of optimisation approaches [29]. The studies in [23,123] covered many approaches using Genetic Programming (hyper-heuristics) for solving various scheduling problems. As an example of a selection hyper-heuristic, the

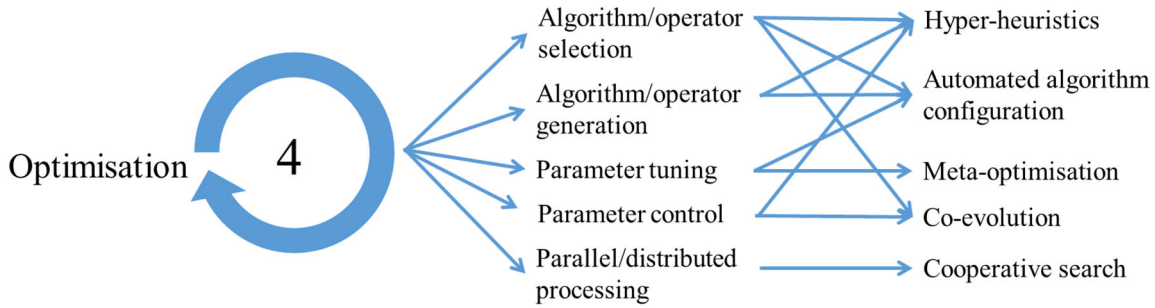

Fig. 5 Interaction 4: optimisation for optimisation

approach in [32] employed tabu search for detecting the best permutation of graph colouring heuristics to cooperatively construct near-optimal exam and course timetables. The results showed that mixing different heuristics rather than using one individually yield improved solutions. As a separate well-established field of research, there are many studies on cooperative search methods, that is, parallel/distributed implementation of multiple optimisation techniques improving upon the performance of each individual technique run on its own [52].

The majority of optimisation methods come with a set of algorithmic parameters influencing their performances and various algorithms might perform well on different instances and problems. Therefore, obtaining the best parameter setting of an optimisation algorithm can be formulated as another optimisation problem. Optimising the parameter settings as well as choosing the best algorithm for a problem can be performed by an optimisation algorithm instead of a ML technique as discussed in Sect. 4. Some approaches (e.g., meta-optimisation, automated algorithm configuration) aim to use an optimisation method to tune the parameters of another optimisation algorithm before the search or even configure the optimisation algorithm and its components [49,165]. For example, an EA can be used to search the best parameter settings for or configure another EA [49]. On the other hand, some other approaches, such as, co-evolutionary algorithms and hyper-heuristics often control and modify the algorithmic parameter settings adaptively during the search process for improved performance. An illustration of parameter control can be found in [128], which embedded the operator choices as well as their parameter settings of a memetic algorithm into the solution representation and co-evolved them adaptively along with the solutions to the problem instance in hand. As another example [112] presented an effective selection hyper-heuristic which not only chose the best operator to invoke but also set its parameter(s) adaptively at each decision point while solving a given problem instance. Both examples also show that the optimisation algorithms used in the optimisation-for-optimisation category can carry out multiple improvement tasks simultaneously.

7 Remarks and discussion

This paper has provided a global overview of the interactions between ML and optimisation as well as their self-interplays. Different from the existing review works that presented comprehensive investigations on an individual or dual interactions, we have aimed to draw a whole picture of the interplays between these two fields and link different interplays by comparing the similar approaches. Background and latest advances of both research area have been provided. In addition to investigating the representative methods, we have presented and discussed individual taxonomies for each interaction. We have also analysed the advantages and disadvantages of the approaches in different interactions that use different techniques to tackle the common tasks. Therefore, this paper serves as a useful tutorial for researchers in ML and optimisation fields to collaborate with each other for the sake of improvement. It has also provided the guidance for non-experts in these two fields on how to automatically design an algorithm without much experience.

Even though a lot of works have been done in the whole research area, there still remains some issues to be explored. Several guidelines on future research directions can be suggested.

- When a high-level ML or optimisation algorithm is introduced into a base-level ML or optimisation one, that is likely to introduce additional parameters to be tuned. Especially, when we use a high-level algorithm to design a base-level one, this would add the extra work of designing the high-level algorithm. Since our focus is on the base-level ML tasks, we usually do not pay much attention to the design of the high-level method and choose a regular configuration for the high-level algorithm to make it effective. A potential research direction would be simultaneous automated design of high-level and base-level algorithms.
- The existing approaches usually focus on a single interaction between ML and optimisation, while very few of them consider multiple interactions, which means using an improved ML or optimisation algorithm to further

enhance other ML or optimisation algorithms. It could be a promising research direction because multiple interactions mean multiple improvements. For example, the research work in [7] introduced ML techniques to learn a promising optimisation algorithm and the learned optimisation method is further used to quickly train an ML model. Auto-sklearn [54] introduced meta-learning techniques to be complementary to Bayesian optimisation for optimising an ML framework. Another research work in this direction can be found in [143].

– There are many software tools for ML and optimisation respectively, such as Weka 3 [182] and scikit-learn [132] for ML, HeuristicLab [179] and Opt4J [107] for optimisation. However, there is no API that enables them to smoothly interact with each other. Since more and more approaches about the interplays between ML and optimisation appear in the literature, the researchers would benefit from such an API that connects ML and optimisation software. In addition, there are several pieces of software for automating the design of ML approaches based on optimisation techniques, such as Auto-WEKA 2.0 [93], auto-sklearn [54] and TPOT [126]. They can automatically select suitable ML algorithms and tune corresponding hyper-parameters for a specific dataset, while they still cannot handle challenging ML problems, such as imbalanced classification or few-shot learning. Therefore, a more powerful tool that can combine ML and optimisation library and tackle challenging problems need to be developed in the future.
– With the rapid development of technologies, the real-world problems in both ML and optimisation field are becoming more and more complex and the scale of them are getting larger and larger. These problems include or generate massive data, which cannot be mined efficiently by conventional ML algorithms. However, very few existing approaches of the interactions between ML and optimisation have taken into account big data issues [129]. Therefore, more advanced and efficient ML techniques need to be introduced or developed to overcome the big data challenges for better interactions.
– Although ML techniques have been widely applied to optimisation in the literature, very few of them have covered the theoretical analysis of proposed approaches. They mainly focus on numerical experiments, which cannot guarantee the convergence and stability of the proposed algorithms. Therefore, the theoretical analysis of algorithm's convergence and stability need to be addressed as a future research direction.
– Many approaches in meta-learning can be trained end-to-end by embedding a high-level ML algorithm into a low-level one. However, when ML algorithms are introduced into optimisation methods to solve a combinatorial optimisation problem, most approaches separate the learning and optimisation process, because the decision variables of combinatorial optimisation are discrete, which is not consistent with the continuous parameters of most ML algorithms. They usually perform ML algorithms to extract knowledge first and then use it to do optimisation tasks. Such an approach is less efficient compared to end-to-end ones. It would be beneficial if we can train an ML model and solve an optimisation problem jointly. One potential way to do that is treating the parameters of an ML model as the decision variables of an optimisation problem and optimise them jointly in an end-to-end manner, so that an ML model can be trained during the optimisation process.
– The interactions that use ML techniques to improve optimisation mostly rely on conventional ML algorithms, such as NNs, SVMs and k-means etc. As more and more advanced ML methods have emerged recently, such as deep learning and the techniques that learn from very few examples, we could introduce these advanced techniques to further improve optimisation algorithms. In addition, algorithm selection is another issue when we introduce ML algorithms into optimisation since there are plenty of ML algorithms can be used. How to choose a suitable ML method for a specific optimisation problem need to be tackled.
– There are various ML algorithms which are embedded with human-designed algorithm components. Due to the higher learning ability of ML algorithms, they have been widely applied to automatically learning ML algorithm components that are designed by experts. The learned algorithm components achieved better performance than human-designed ones, such as the activation function of NNs, the splitting criterion of DTs, initialisation of an algorithm, optimisation algorithm for training a model, selection strategy for active learning, etc. A future area of research would be using meta-learning techniques to automatically learn those components, which is still underdeveloped.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the 21th International Conference on Machine Learning (ICML 2004), p. 1. Banff, Canada (2004)
2. Abdulrahman, S.M., Brazdil, P., van Rijn, J.N., Vanschoren, J.: Speeding up algorithm selection using average ranking and active

testing by introducing runtime. Mach. Learn. **107**(1), 79–108 (2018)
3. Alanazi, F., Lehre, P.K.: Limits to learning in reinforcement learning hyper-heuristics. In: European Conference on Evolutionary Computation in Combinatorial Optimization, pp. 170–185. Porto, Portugal (2016)
4. Alatas, B., Akin, E., Karci, A.: Modenar: multi-objective differential evolution algorithm for mining numeric association rules. Appl. Soft Comput. **8**(1), 646–656 (2008)
5. Anderson, M.L., Oates, T.: A review of recent research in metareasoning and metalearning. AI Mag. **28**(1), 12 (2007)
6. Andréasson, N., Evgrafov, A., Patriksson, M., Gustavsson, E., Önnheim, M.: An introduction to continuous optimization: foundations and fundamental algorithms, vol. 28. Studentlitteratur Lund (2005)
7. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: 30th Conference on Neural Information Processing Systems (NIPS 2016), pp. 3981–3989. Barcelona, Spain (2016)
8. Arabie, P., De Soete, G.: Clustering and Classification. World Scientific, Singapore (1996)
9. Asta, S., Özcan, E.: An apprenticeship learning hyper-heuristic for vehicle routing in hyflex. In: IEEE Symposium on Evolving and Autonomous Learning Systems (EALS), pp. 65–72. Orlando, USA (2014)
10. Asta, S., Özcan, E.: A tensor-based selection hyper-heuristic for cross-domain heuristic search. Inf. Sci. **299**, 412–432 (2015)
11. Asta, S., Özcan, E., Curtois, T.: A tensor based hyper-heuristic for nurse rostering. Knowl. Based Syst. **98**, 185–199 (2016)
12. Bachman, P., Sordoni, A., Trischler, A.: Learning algorithms for active learning. In: Proceedings of the 34th International Conference on Machine Learning (ICML 2017). Sydney, Australia (2017)
13. Back, T., Emmerich, M., Shir, O.: Evolutionary algorithms for real world applications [application notes]. IEEE Comput. Intell. Mag. **3**(1), 64–67 (2008)
14. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating neural architecture search using performance prediction. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
15. Bandler, J.W., Cheng, Q.S., Dakroury, S.A., Mohamed, A.S., Bakr, M.H., Madsen, K., Sondergaard, J.: Space mapping: the state of the art. IEEE Trans. Microw. Theory Tech. **52**(1), 337–361 (2004)
16. Barros, R.C., Basgalupp, M.P., De Carvalho, A.C., Freitas, A.A.: A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **42**(3), 291–312 (2012)
17. Barros, R.C., Basgalupp, M.P., Freitas, A.A., de Carvalho, A.C.: Evolutionary design of decision-tree algorithms tailored to microarray gene expression data sets. IEEE Trans. Evolut. Comput. **6**(18), 873–892 (2014)
18. Bennett, K.P., Parrado-Hernández, E.: The interplay of optimization and machine learning research. J. Mach. Learn. Res. **7**(Jul), 1265–1281 (2006)
19. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(Feb), 281–305 (2012)
20. Bertinetto, L., Henriques, J.F., Valmadre, J., Torr, P., Vedaldi, A.: Learning feed-forward one-shot learners. In: 30th Conference on Neural Information Processing Systems (NIPS 2016), pp. 523–531. Barcelona, Spain (2016)
21. Bishop, C.: Pattern Recognition and Machine Learning. Springer, New York (2006)
22. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, New York (2004)
23. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. **20**(1), 110–124 (2016)
24. Brazdil, P., Carrier, C.G., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer, Berlin (2008)
25. Brazdil, P.B., Soares, C., Da Costa, J.P.: Ranking learning algorithms: using ibl and meta-learning on accuracy and time results. Mach. Learn. **50**(3), 251–277 (2003)
26. Brighton, H., Mellish, C.: Advances in instance selection for instance-based learning algorithms. Data Min. Knowl. Discov. **6**(2), 153–172 (2002)
27. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Smash: one-shot model architecture search through hypernetworks. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
28. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
29. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: Computational Intelligence, pp. 177–201. Berlin, Heidelberg (2009)
30. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches: revisited. In: Handbook of Metaheuristics, pp. 453–477 (2019)
31. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. J. Heuristics. **9**(6), 451–470 (2003)
32. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for educational timetabling problems. Eur. J. Oper. Res. **176**(1), 177–192 (2007)
33. Calvet, L., de Armas, J., Masip, D., Juan, A.A.: Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. Open Math. **15**(1), 261–280 (2017)
34. Chandra, A., Yao, X.: Ensemble learning using multi-objective evolutionary algorithms. J. Math. Model. Algorithms. **5**(4), 417–445 (2006)
35. Chen, W.C., Tseng, L.Y., Wu, C.S.: A unified evolutionary training scheme for single and ensemble of feedforward neural network. Neurocomputing. **143**, 347–361 (2014)
36. Choong, S.S., Wong, L.P., Lim, C.P.: Automatic design of hyper-heuristic based on reinforcement learning. Inf. Sci. **436**, 89–107 (2018)
37. Christakopoulou, E., Karypis, G.: Local item-item models for top-n recommendation. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 67–74. Boston, USA (2016)
38. Cobos, C., Mendoza, M., León, E.: A hyper-heuristic approach to design and tuning heuristic methods for web document clustering. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1350–1358. New Orleans, USA (2011)
39. Corne, D., Dhaenens, C., Jourdan, L.: Synergies between operations research and data mining: the emerging use of multi-objective approaches. Eur. J. Oper. Res. **221**(3), 469–479 (2012)
40. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: International Conference on the Practice and Theory of Automated Timetabling, pp. 176–190. Konstanz, Germany (2000)
41. Deb, K., Hussein, R., Roy, P.C., Toscano, G.: A taxonomy for metamodeling frameworks for evolutionary multi-objective optimization. IEEE Trans. Evol. Comput. **23**, 104–116 (2018)
42. Degroote, H., González-Velarde, J.L., De Causmaecker, P.: Applying algorithm selection-a case study for the generalised assignment problem. Electr. Notes Discrete Math. **69**, 205–212 (2018)
43. Dehghani, M., Severyn, A., Rothe, S., Kamps, J.: Learning to learn from weak supervision by full supervision. In: 31st Conference on

Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
44. del Jesus, M.J., Gamez, J.A., Gonzalez, P., Puerta, J.M.: On the discovery of association rules by means of evolutionary algorithms. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **1**(5), 397–415 (2011)
45. Deng, Y., Liu, Y., Zhou, D.: An improved genetic algorithm with initial population strategy for symmetric TSP. Math. Prob. Eng. **2015** (2015)
46. de Sá, A.G., Pappa, G.L.: A hyper-heuristic evolutionary algorithm for learning bayesian network classifiers. In: Ibero-American Conference on Artificial Intelligence, pp. 430–442. Santiago de Chile, Chile (2014)
47. Di Gaspero, L., Urli, T.: Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In: International Conference on Learning and Intelligent Optimization, pp. 384–389. Paris, France (2012)
48. Díaz-Manríquez, A., Toscano, G., Coello, C.A.C.: Comparison of metamodeling techniques in evolutionary algorithms. Soft Comput. **21**(19), 5647–5663 (2017)
49. Dioşan, L., Oltean, M.: Evolutionary design of evolutionary algorithms. Genet. Program. Evol. Mach. **10**(3), 263–306 (2009)
50. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
51. Drake, J.H., Özcan, E., Burke, E.K.: An improved choice function heuristic selection for cross domain heuristic search. In: International Conference on Parallel Problem Solving from Nature, pp. 307–316. Taormina, Italy (2012)
52. El-Abd, M., Kamel, M.: A taxonomy of cooperative search algorithms. In: International Workshop on Hybrid Metaheuristics, pp. 32–41. Berlin, Heidelberg (2005)
53. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: The kdd process for extracting useful knowledge from volumes of data. Commun. ACM. **39**(11), 27–34 (1996)
54. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: 29th Conference on Neural Information Processing Systems (NIPS 2015), pp. 2962–2970. Montreal, Canada (2015)
55. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning (ICML 2017). Sydney, Australia (2017)
56. Fong, S., Deb, S., Yang, X.S.: How meta-heuristic algorithms contribute to deep learning in the hype of big data analytics. In: Progress in Intelligent Computing Techniques: theory, Practice, and Applications, pp. 3–25. Springer (2018)
57. Frans, K., Ho, J., Chen, X., Abbeel, P., Schulman, J.: Meta learning shared hierarchies. In: International Conference on the 6th Learning Representations (ICLR 2018). Vancouver, Canada (2018)
58. Freitas, A.A.: A review of evolutionary algorithms for data mining. In: Soft Computing for Knowledge Discovery and Data Mining, pp. 79–111. Springer, Boston, USA (2008)
59. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing. **64**, 107–117 (2005)
60. Gaber, M.M.: Advances in data stream mining. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **2**(1), 79–85 (2012)
61. Gagné, C., Sebag, M., Schoenauer, M., Tomassini, M.: Ensemble learning for free with evolutionary algorithms? In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 1782–1789. London, UK (2007)
62. Gama, J., Brazdil, P.: Characterization of classification algorithms. In: Portuguese Conference on Artificial Intelligence, pp. 189–200. Springer, Berlin (1995)
63. García, S., Herrera, F.: Evolutionary undersampling for classification with imbalanced datasets: proposals and taxonomy. Evolut. Comput. **17**(3), 275–306 (2009)
64. Garcia, S., Luengo, J., Sáez, J.A., Lopez, V., Herrera, F.: A survey of discretization techniques: taxonomy and empirical analysis in supervised learning. IEEE Trans. Knowl. Data Eng. **25**(4), 734–750 (2013)
65. Garcia, V., Bruna, J.: Few-shot learning with graph neural networks. In: International Conference on the 6th Learning Representations (ICLR 2018). Vancouver, Canada (2018)
66. Ghorban, F., Milani, N., Schugk, D., Roese-Koerner, L., Su, Y., Müller, D., Kummert, A.: Conditional multichannel generative adversarial networks with an application to traffic signs representation learning. In: Progress in Artificial Intelligence, pp. 1–10 (2018)
67. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587. Washington, USA (2014)
68. Glover, F.: Tabu search–part i. ORSA J. Comput. **1**(3), 190–206 (1989)
69. Güngör, Z., Ünler, A.: K-harmonic means data clustering with simulated annealing heuristic. Appl. Math. Comput. **184**(2), 199–209 (2007)
70. Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macia, N., Ray, B., Saeed, M., Statnikov, A., et al.: Design of the 2015 chalearn automl challenge. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. Budapest, Hungary (2015)
71. Guyon, I., Chaabane, I., Escalante, H.J., Escalera, S., Jajetic, D., Lloyd, J.R., Macià, N., Ray, B., Romaszko, L., Sebag, M., et al.: A brief review of the chalearn automl challenge: any-time any-dataset learning without human intervention. In: Proceedings of the 33th International Conference on Machine Learning (ICML 2016), pp. 1842–1850. New York, USA (2016)
72. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**(Mar), 1157–1182 (2003)
73. Ha, D., Dai, A., Le, Q.V.: Hypernetworks. In: International Conference on the 4th Learning Representations (ICLR 2016). San Juan, Puerto Rico (2016)
74. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. **10**, 993–1001 (1990)
75. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **9**, 1263–1284 (2008)
76. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
77. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press, Cambridge (1992)
78. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. **79**(8), 2554–2558 (1982)
79. Hruschka, E.R., Campello, R.J., Freitas, A.A., et al.: A survey of evolutionary algorithms for clustering. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **39**(2), 133–155 (2009)
80. Hunger, J., Huttner, G.: Optimization and analysis of force field parameters by combination of genetic algorithms and neural networks. J. Comput. Chem. **20**(4), 455–471 (1999)
81. Hutter, F., Kotthoff, L., Vanschoren, J.: Automatic Machine Learning: Methods, Systems. Challenges. Springer, Berlin (2019)
82. Jain, P., Kar, P., et al.: Non-convex optimization for machine learning. Found. Trends Mach. Learn. **10**(3–4), 142–336 (2017)
83. Jin, Y.: Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evolut. Comput. **1**(2), 61–70 (2011)

84. Jourdan, L., Dhaenens, C., Talbi, E.G.: Using datamining techniques to help metaheuristics: a short survey. In: Proceedings of the Third international workshop on Hybrid Metaheuristics, pp. 57–69. Gran Canaria, Spain (2006)
85. Karafotias, G., Hoogendoorn, M., Eiben, Á.E.: Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans. Evol. Comput. **19**(2), 167–187 (2015)
86. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis, vol. 344. Wiley, Hoboken (2009)
87. Kennedy, J.: Stereotyping: improving particle swarm performance with cluster analysis. In: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 2, pp. 1507–1512. La Jolla, USA (2000)
88. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95—International Conference on Neural Networks, vol. 4, pp. 1942–1948. Perth, Australia (1995)
89. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: survey and perspectives. Evolut. Comput. **27**(1), 3–45 (2019)
90. Kim, H.S., Cho, S.B.: An efficient genetic algorithm with less fitness evaluation by clustering. In: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 2, pp. 887–894. Seoul, South Korea (2001)
91. Kingma, D.P., Ba, J.L.: Adam: a method for stochastic optimization. In: Proceeding of the 2nd International Conference on Learning Representations (ICLR 2014). Banff, Canada (2014)
92. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: Proceedings of the 32th International Conference on Machine Learning (ICML 2015). Lille, France (2015)
93. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-weka 2.0: automatic model selection and hyperparameter optimization in weka. J. Mach. Learn. Res. **18**(1), 826–830 (2017)
94. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**(2), 87–112 (1994)
95. Kumari, A.C., Srinivas, K., Gupta, M.: Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In: IEEE 3rd International Advance Computing Conference (IACC), pp. 813–818. Ghaziabad, India (2013)
96. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. The Handbook of Brain Theory and Neural Networks, vol. 3361(10) (1995)
97. Lemke, C., Budka, M., Gabrys, B.: Metalearning: a survey of trends and technologies. Artif. Intell. Rev. **44**(1), 117–130 (2015)
98. Li, F.F., Rob, F., Pietro, P.: One-shot learning of object categories. IEEE Trans. Pattern Anal. Mach. Intell. **28**(4), 594–611 (2006)
99. Li, J., Burke, E.K., Qu, R.: Integrating neural networks and logistic regression to underpin hyper-heuristic search. Knowl. Based Syst. **24**(2), 322–330 (2011)
100. Li, J., Burke, E.K., Qu, R.: A pattern recognition based intelligent search method and two assignment problem case studies. Appl. Intell. **36**(2), 442–453 (2012)
101. Li, K., Burdick, J.W.: Meta inverse reinforcement learning via maximum reward sharing for human motion analysis. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
102. Li, K., Malik, J.: Learning to optimize. In: Proceedings of the 5th International Conference on Learning Representations (ICLR 2017). Toulon, France (2017)
103. Lopes, L., Smith-Miles, K.: Generating applicable synthetic instances for branch problems. Oper. Res. **61**(3), 563–577 (2013)
104. López Jaimes, A., Coello Coello, C.A., Chakraborty, D.: Objective reduction using a feature selection technique. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 673–680. Atlanta, USA (2008)
105. Lorraine, J., Duvenaud, D.: Hyperparameter optimization with hypernets. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
106. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of Metaheuristics, pp. 320–353. Kluwer Academic Publishers, Boston (2003)
107. Lukasiewycz, M., Glaß, M., Reimann, F., Teich, J.: Opt4j: a modular framework for meta-heuristic optimization. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 1723–1730. Dublin, Ireland (2011)
108. Mahi, M., Baykan, Ö.K., Kodaz, H.: A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. Appl. Soft Comput. **30**, 484–490 (2015)
109. Martínez-Estudillo, A.C., Hervás-Martínez, C., Martínez-Estudillo, F.J., García-Pedrajas, N.: Hybridization of evolutionary algorithms and local search by means of a clustering method. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **36**(3), 534–545 (2005)
110. McClymont, K., Keedwell, E.C.: Markov chain hyper-heuristic (mchh): an online selective hyper-heuristic for multi-objective continuous problems. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 2003–2010. Dublin, Ireland (2011)
111. Mishra, N., Rohaninejad, M., Chen, X., Abbeel, P.: A simple neural attentive meta-learner. In: International Conference on the 6th Learning Representations (ICLR 2018). Vancouver, Canada (2018)
112. Mısır, M., Verbeeck, K., De Causmaecker, P., Berghe, G.V.: An intelligent hyper-heuristic framework for chesc 2011. In: Learning and Intelligent Optimization, pp. 461–466. Springer, Berlin (2012)
113. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
114. Mockus, J.: Bayesian Approach to Global Optimization: Theory and Applications, vol. 37. Springer, Amsterdam (2012)
115. Montazeri, M.: Hhfs: hyper-heuristic feature selection. Intell. Data Anal. **20**(4), 953–974 (2016)
116. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S.: A survey of multiobjective evolutionary clustering. ACM Comput. Surv. (CSUR). **47**(4), 61 (2015)
117. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S., Coello, C.A.C.: A survey of multiobjective evolutionary algorithms for data mining: part I. IEEE Trans. Evol. Comput. **18**(1), 4–19 (2014)
118. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S., Coello, C.A.C.: Survey of multiobjective evolutionary algorithms for data mining: part II. IEEE Trans. Evol. Comput. **18**(1), 20–35 (2014)
119. Munkhdalai, T., Yu, H.: Meta networks. In: Proceedings of the 34th International Conference on Machine Learning (ICML 2017). Sydney, Australia (2017)
120. Muñoz, M.A., Smith-Miles, K.A.: Performance analysis of continuous black-box optimization algorithms via footprints in instance space. Evolut. Comput. **25**(4), 529–554 (2017)
121. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: Metaheuristics: Computer Decision-Making, pp. 523–544. Springer, Boston (2003)
122. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: a literature review. Swarm Evolut. Comput. **2**, 1–14 (2012)
123. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. **3**(1), 41–66 (2017)
124. Nichol, A., Achiam, J., Schulman, J.: On first-order meta-learning algorithms. CoRR arXiv:1803.02999 (2018)

125. Ojha, V.K., Abraham, A., Snášel, V.: Metaheuristic design of feedforward neural networks: a review of two decades of research. Eng. Appl. Artif. Intell. **60**, 97–116 (2017)
126. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, 8, pp. 485–492. Denver, Colorado, USA (2016)
127. Ortiz-Bayliss, J.C., Terashima-Marín, H., Conant-Pablos, S.E.: A neuro-evolutionary hyper-heuristic approach for constraint satisfaction problems. Cognit. Comput. **8**(3), 429–441 (2016)
128. Özcan, E., Drake, J.H., Altıntaş, C., Asta, S.: A self-adaptive multimeme memetic algorithm co-evolving utility scores to control genetic operators and their parameter settings. Appl. Soft Comput. **49**, 81–93 (2016)
129. Pacheco, A.D., Reyes-García, C.A.: Full model selection in huge datasets through a meta-learning approach. International Conferences Big Data Analytics, Data Mining and Computational Intelligence **2018**, 19–26 (2018)
130. Pan, S.J., Yang, Q., et al.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
131. Park, J., Kim, K.Y.: Meta-modeling using generalized regression neural network and particle swarm optimization. Appl. Soft Comput. **51**, 354–369 (2017)
132. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2011)
133. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. **27**(8), 1226–1238 (2005)
134. Platt, J.: Sequential minimal optimization: a fast algorithm for training support vector machines (1998)
135. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural Netw. **12**(1), 145–151 (1999)
136. Qiao, S., Liu, C., Shen, W., Yuille, A.L.: Few-shot image recognition by predicting parameters from activations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Salt Lake City, USA (2018)
137. Ramírez-Gallego, S., García, S., Benítez, J.M., Herrera, F.: Multivariate discretization based on evolutionary cut points selection for classification. IEEE Trans. Cybern. **46**(3), 595–608 (2016)
138. Ravi, S., Larochelle, H.: Optimization as a model for few shot learning. In: Proceedings of the 5th International Conference on Learning Representations (ICLR 2017). Toulon, France (2017)
139. Reed, M., Yiannakou, A., Evering, R.: An ant colony algorithm for the multi-compartment vehicle routing problem. Appl. Soft Comput. **15**, 169–176 (2014)
140. Ren, Y., Zhang, L., Suganthan, P.N.: Ensemble classification and regression-recent developments, applications and future directions. IEEE Comput. Intell. Mag. **11**(1), 41–53 (2016)
141. Rendell, L., Cho, H.: Empirical learning as a function of concept character. Mach. Learn. **5**(3), 267–298 (1990)
142. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
143. Rosales-Pérez, A., Gonzalez, J.A., Coello, C.A.C., Escalante, H.J., Reyes-Garcia, C.A.: Surrogate-assisted multi-objective model selection for support vector machines. Neurocomputing. **150**, 163–172 (2015)
144. Rosenbaum, C., Klinger, T., Riemer, M.: Routing networks: adaptive selection of non-linear functions for multi-task learning. In: International Conference on the 6th Learning Representations (ICLR 2018). Vancouver, Canada (2018)
145. Rubio-Largo, Á., Vanneschi, L., Castelli, M., Vega-Rodríguez, M.A.: Multiobjective metaheuristic to design rna sequences. IEEE Trans. Evol. Comput. **23**(1), 156–169 (2018)
146. Ruder, S.: An overview of gradient descent optimization algorithms. CoRR arXiv:1609.04747 (2016)
147. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: Meta-learning with memory-augmented neural networks. In: Proceedings of the 33th International Conference on Machine Learning (ICML 2016), pp. 1842–1850. New York, USA (2016)
148. Saxena, D.K., Duro, J.A., Tiwari, A., Deb, K., Zhang, Q.: Objective reduction in many-objective optimization: linear and nonlinear algorithms. IEEE Trans. Evol. Comput. **17**(1), 77–99 (2013)
149. Schlag, I., Schmidhuber, J.: Gated fast weights for on-the-fly neural program generation. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
150. Shahvari, O., Logendran, R.: An enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. Comput. Oper. Res. **77**, 154–176 (2017)
151. Shboul, B., Myaeng, S.H.: Initializing k-means using genetic algorithms. In: International Conference on Computational Intelligence and Cognitive Informatics (ICCICI 09), pp. 114–118 (2009)
152. Shelokar, P., Jayaraman, V.K., Kulkarni, B.D.: An ant colony approach for clustering. Anal. Chim. Acta. **509**(2), 187–195 (2004)
153. Smith, J.: Co-evolving memetic algorithms: initial investigations. In: International Conference on Parallel Problem Solving from Nature, pp. 537–546. Berlin, Heidelberg (2002)
154. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Comput. Oper. Res. **45**, 12–24 (2014)
155. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: 31th Conference on Neural Information Processing Systems (NIPS 2017), pp. 4080–4090. Long Beach, USA (2017)
156. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: 26th Conference on Neural Information Processing Systems, pp. 2951–2959. Sierra Nevada, USA (2012)
157. Sochi, T.: Introduction to tensor calculus. CoRR arXiv:1603.01660 (2016)
158. Sörensen, K., Glover, F.W.: Metaheuristics. In: Encyclopedia of Operations Research and Management Science, pp. 960–970. Springer (2013)
159. Soria-Alcaraz, J.A., Espinal, A., Sotelo-Figueroa, M.A.: Evolvability metric estimation by a parallel perceptron for on-line selection hyper-heuristics. IEEE Access. **5**, 7055–7063 (2017)
160. Soria-Alcaraz, J.A., Özcan, E., Swan, J., Kendall, G., Carpio, M.: Iterated local search using an add and delete hyper-heuristic for university course timetabling. Appl. Soft Comput. **40**, 581–593 (2016)
161. Sousa, R., Gama, J.: Multi-label classification from high-speed data streams with adaptive model rules and random rules. Progress in Artificial Intelligence. pp. 1–11 (2018)
162. Sra, S., Nowozin, S., Wright, S.J.: Optimization for Machine Learning. MIT Press, London (2012)
163. Sriwanna, K., Boongoen, T., Iam-On, N.: An evolutionary cut points search for graph clustering-based discretization. In: 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 1–6 (2016)
164. Streichert, F., Stein, G., Ulmer, H., Zell, A.: A clustering based niching method for evolutionary algorithms. In: International

Conference on Artificial Evolution (Evolution Artificielle), pp. 644–645. Marseille, France (2003)
165. Stützle, T.: Automated algorithm configuration: advances and prospects. In: Intelligent Distributed Computing VIII, p. 5. Cham (2015)
166. Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P.H., Hospedales, T.M.: Learning to compare: relation network for few-shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Salt Lake City, USA (2018)
167. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
168. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 847–855. Chicago, USA (2013)
169. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al.: Stanley: the robot that won the darpa grand challenge. J. Field Robot. **23**(9), 661–692 (2006)
170. Thrun, S., Pratt, L.: Learning to Learn. Springer, New York (2012)
171. Triguero, I., García, S., Herrera, F.: Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. Pattern Recognit. **44**(4), 901–916 (2011)
172. Tsai, C.W., Song, H.J., Chiang, M.C.: A hyper-heuristic clustering algorithm. In: IEEE International Conference on Systems. Man, and Cybernetics (SMC), pp. 2839–2844. Seoul, South Korea (2012)
173. Tyasnurita, R., Özcan, E., John, R.: Learning heuristic selection using a time delay neural network for open vehicle routing. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 1474–1481. San Sebastian, Spain (2017)
174. Václavík, R., Šůcha, P., Hanzálek, Z.: Roster evaluation based on classifiers for the nurse rostering problem. J. Heuristics. **22**(5), 667–697 (2016)
175. Vanschoren, J.: Meta-learning: a survey. CoRR arXiv:1810.03548 (2018)
176. Vercellino, C.J., Wang, W.Y.: Hyperactivations for activation function exploration. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
177. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. **18**(2), 77–95 (2002)
178. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In: 30th Conference on Neural Information Processing Systems (NIPS 2016), pp. 3630–3638. Barcelona, Spain (2016)
179. Wagner, S., Affenzeller, M.: Heuristiclab: a generic and extensible optimization environment. In: Adaptive and Natural Computing Algorithms, pp. 538–541 (2005)
180. Wang, Y.X., Ramanan, D., Hebert, M.: Learning to model the tail. In: 31th Conference on Neural Information Processing Systems (NIPS 2017), pp. 7032–7042. Long Beach, USA (2017)
181. Wichrowska, O., Maheswaranathan, N., Hoffman, M.W., Colmenarejo, S.G., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: Proceedings of the 34th International Conference on Machine Learning (ICML 2017). Sydney, Australia (2017)
182. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, Burlington (2016)
183. Wong, L.P., Choong, S.S.: A bee colony optimization algorithm with frequent-closed-pattern-based pruning strategy for traveling salesman problem. In: 2015 Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp. 308–314. Tainan, Taiwan (2015)
184. Wong, L.P., Low, M.Y.H., Chong, C.S.: Bee colony optimization with local search for traveling salesman problem. Int. J. Artif. Intell. Tools. **19**(03), 305–334 (2010)
185. Wong, L.P., Low, M.Y.H., Chong, C.S.: A generic bee colony optimization framework for combinatorial optimization problems. In: 2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation, pp. 144–151 (2010)
186. Wu, X., Zhu, X., Wu, G.Q., Ding, W.: Data mining with big data. IEEE Trans. Knowl. Data Eng. **26**(1), 97–107 (2014)
187. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: bridging the gap between human and machine translation. CoRR arXiv:1609.08144 (2016)
188. Xiong, Z., Zhang, W., Zhu, W.: Learning decision trees with reinforcement learning. In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Workshop on Meta-learning. Long Beach, USA (2017)
189. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. **20**(4), 606–626 (2016)
190. Yalcinoz, T., Altun, H.: Power economic dispatch using a hybrid genetic algorithm. IEEE Power Eng. Rev. **21**(3), 59–60 (2001)
191. Yan, X., Wu, M., Sun, B.: An adaptive ls-svm based differential evolution algorithm. In: International Conference on Signal Processing Systems, pp. 406–409. Singapore (2009)
192. Yao, Q., Wang, M., Jair, E.H., Guyon, I., Hu, Y., Li, Y., Tu, W., Yang, Q., Yu, Y.: Taking human out of learning applications: a survey on automated machine learning. CoRR arXiv:1810.13306 (2018)
193. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE. **87**(9), 1423–1447 (1999)
194. Yates, W.B., Keedwell, E.C.: Offline learning for selection hyper-heuristics with elman networks. In: International Conference on Artificial Evolution (Evolution Artificielle), pp. 217–230. Paris, France (2017)
195. Zhang, H., Lu, J.: Adaptive evolutionary programming based on reinforcement learning. Inf. Sci. **178**(4), 971–984 (2008)
196. Zhang, J., Chung, H.S.H., Lo, W.L.: Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. IEEE Trans. Evol. Comput. **11**(3), 326–335 (2007)
197. Zhang, J., Zhan, Z.H., Lin, Y., Chen, N., Gong, Y.J., Zhong, J.H., Chung, H.S., Li, Y., Shi, Y.H.: Evolutionary computation meets machine learning: a survey. IEEE Comput. Intell. Mag. **6**(4), 68–75 (2011)
198. Zhang, X., Tian, Y., Cheng, R., Jin, Y.: A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization. IEEE Trans. Evol. Comput. **22**(1), 97–112 (2018)