Review

Ilias Mitrai* and Prodromos Daoutidis*

Accelerating process control and optimization via machine learning: a review

https://doi.org/10.1515/revce-2024-0060
Received August 13, 2024; accepted February 12, 2025;
published online March 12, 2025

Abstract: Process control and optimization have been widely used to solve decision-making problems in chemical engineering applications. However, identifying and tuning the best solution algorithm is challenging and time-consuming. Machine learning tools can be used to automate these steps by learning the behavior of a numerical solver from data. In this paper, we discuss recent advances in (i) the representation of decision-making problems for machine learning tasks, (ii) algorithm selection, and (iii) algorithm configuration for monolithic and decomposition-based algorithms. Finally, we discuss open problems related to the application of machine learning for accelerating process optimization and control.

Keywords: process control; process optimization; machine learning

1 Introduction

The design and operation of chemical processes depend on decisions spanning a wide range of scales, from the molecular up to the enterprise-wide, and constrained by multiple physical and chemical phenomena (Daoutidis et al. 2018; Grossmann 2012; Hanselman and Gounaris 2016; Pistikopoulos et al. 2021). Process control and optimization methods provide a systematic framework to identify the best possible decisions in designing and operating a process, subject to constraints that emerge from physics or design and operational considerations. Over the last few decades, there have been significant advances in both theory and algorithm development regarding the control of nonlinear and constrained process systems (Christofides et al. 2013; Daoutidis et al. 2023b; Ellis et al. 2014; McAllister and Rawlings 2022; Mesbah 2016; Shin et al. 2019), as well as the solution of broad classes of optimization problems (Biegler 2024; Boukouvala et al. 2016; Grossmann et al. 2016; Tawarmalani and Sahinidis 2005; Wächter and Biegler 2006).

Despite these advances, control and optimization problems that challenge the computational performance of state-of-the-art algorithms continue to emerge. Some examples of application domains where such problems occur include the real-time operation of chemical processes interacting with renewable energy resources, the decarbonization of the energy sector, and the design of resilient, sustainable and circular supply chain networks (National Academies of Sciences Engineering and Medicine 2022). The scale and complexity in these systems and the multiple spatial and temporal scales that are often present make the solution of the corresponding control and optimization problems challenging. Different approaches have been followed to improve the tractability of such problems. For example, one can potentially reduce the computational complexity by reformulating the problem (Liberti and Pantelides 2006; Raghunathan and Biegler 2003). However, finding a suitable exact reformulation is generally not possible. Data-driven approaches namely surrogate and hybrid modeling, have also been developed to learn a surrogate model with lower computational complexity (Bhosekar and Ierapetritou 2018; Bradley et al. 2022; Cozad et al. 2014; Misener and Biegler 2023; Sansana et al. 2021). Although this approach has received significant attention, the solution returned is (inherently) approximate.

An alternative approach is to accelerate the solution process itself by (1) selecting a solution strategy (algorithm selection) and (2) tuning it (algorithm configuration) such that a desired performance function like solution time is minimized. The acceleration is usually achieved by exploiting some underlying property of the decision-making problem. An example is the case of structured decision-making problems, where the structure can be used as the basis of decomposition-based optimization algorithms, which are usually faster than monolithic algorithms for large-scale problems (Conejo et al. 2006). Although this approach does

***Corresponding authors: Ilias Mitrai**, McKetta Department of Chemical Engineering, The University of Texas at Austin, Austin, TX 78712, USA, E-mail: imitrai@che.utexas.edu; and **Prodromos Daoutidis**, Department of Chemical Engineering and Materials Science, University of Minnesota, Minneapolis, MN 55455, USA, E-mail: daout001@umn.edu

not compromise solution quality, selecting and tuning a solution algorithm is nontrivial. Current state-of-the-art algorithms or solvers, especially commercial ones, are complex systems with many algorithmic steps, each one potentially having a set of hyperparameters. Furthermore, the quantitative effect of the problem formulation on the performance of an algorithm, such as solution time, is not known a-priory, i.e., the selection and tuning of the algorithm are black-box optimization problems since the solution time or quality (for local solvers) cannot be determined a-priory.

To this end, machine learning (ML) can be used to learn the behavior of an algorithm for a class of decision-making problems from data. ML has been widely used in chemical engineering for modeling chemical and physical systems and developing data-driven optimization and control algorithms (Bhosekar and Ierapetritou 2018; Bradley et al. 2022; Cozad et al. 2014; Daoutidis et al. 2023a; Sansana et al. 2021; Schweidtmann et al. 2021; Tang and Daoutidis 2022). Usually, data is used to learn a system's chemical, physical, or control-relevant properties. In the context of algorithm selection and configuration, the data is used to learn the effect of the problem formulation on the computational performance of an algorithm.

The application of ML for accelerating an algorithm has recently received significant attention in the operations research and computer science communities and has shown the potential for significant computational savings (Bengio et al. 2021). This approach has received less attention in the chemical engineering literature, where the emphasis has been on improving the problem formulation and developing new optimization algorithms with well-characterized optimality properties. ML has been mainly used to analyze the solution time for production scheduling optimization problems (Kim and Maravelias 2022) and accelerate decomposition-based algorithms for the solution of mixed integer model predictive control (Mitrai and Daoutidis 2024a,d), supply chain optimization (Triantafyllou and Papathanasiou 2024), and capacity expansion problems (Allen et al. 2023). Decision-making problems that arise in chemical engineering have certain features, such as nonlinearity in the form of bilinear terms (flowrate multiplied by concentration) or exponentials with continuous variables ($e^{-\frac{E}{RT}}c$) and certain structure in the constraints, such as tri-diagonal structure which arises in model predictive control applications. *We posit that developing ML-based methods for accelerating general-purpose solvers as well as decomposition-based solution algorithms is a fitting approach to improve the tractability of complex decision-making problems in chemical engineering*.

In this paper, we aim to review the algorithm selection and configuration problems, review recent advances in using ML to accelerate the solution of decision-making problems and discuss open problems and future directions for applying this approach to chemical engineering problems. In Section 2, we formally introduce the algorithm selection and configuration problems. In Section 3, the representation of an optimization problem in a format that can be used as input to standard ML models is discussed. In Sections 4–6, we present the application of ML for selecting and tuning an algorithm, and finally, in Section 7, we discuss open problems and opportunities related to the acceleration of numerical algorithms using machine learning.

2 The algorithm configuration and selection problems

2.1 The algorithm selection problem

Consider a general decision-making problem

$$\begin{aligned} P(p) \coloneqq \underset{x}{minimize} \quad & f(x;p) \\ \text{subject to} \quad & g_i(x;p) \le 0 \quad \forall\, i = 1,\dots,m_{\text{in}} \\ & h_j(x;p) = 0 \quad \forall\, j = 1,\dots,m_{\text{eq}} \\ & x \in \mathbb{R}^{N_x^c} \times \mathbb{Z}^{N_x^d}, \end{aligned} \tag{1}$$

where $N_x^c + N_x^d = N$, $m_{\text{in}} + m_{\text{eq}} = M$, and p are the parameters of the problem (which can be both continuous and integer), x are the decision variables and the objective f as well as the constraints g_i, h_j can be convex (linear or nonlinear) or nonconvex. The first question that arises during the solution of a decision-making problem is which algorithm to select for the solution of the problem. In general, finding or developing an algorithm that performs well for any decision-making problem is not possible (Hutter et al. 2014; Markót and Schichl 2011; Smith-Miles and Lopes 2012). Thus, for a given problem, one must find the most suitable algorithm or solution strategy. This is formally known as the algorithm selection problem and is stated as follows (Rice 1976):

Algorithm selection. Given an optimization problem $P(p)$ and a set of algorithms $\mathscr{A} = \{a_1, \dots, a_{|\mathscr{A}|}\}$, determine which algorithm a^* should be used to solve the problem such that a desired performance function $m : \mathscr{P} \times \mathscr{A} \to \mathscr{M}$ is optimized.

The performance function m is a metric used to compare two algorithms. Typical performance functions can be the computational time or the solution quality for a given computational budget. The choice of the performance function depends on the application. For example, solution time might be more important for an online application, whereas solution quality and feasibility might be better for a design or safety-critical application.

Given a decision-making problem, the set of available algorithms, and a performance function, algorithm selection can be posed as an optimization problem as follows

$$\alpha^* \in \arg\min_{\alpha \in \mathscr{A}} m(P(p), \alpha). \tag{2}$$

This problem is also known as per-instance algorithm selection since it considers only a specific decision-making problem. However, it can be easily extended to identify the best algorithm for a class of decision-making problems (Kerschke et al. 2019).

The algorithm selection problem is a black-box optimization problem since the performance function m is not known explicitly, and evaluating an algorithm for a given problem can require significant computational resources. The standard approach to solving this problem relies on ML, where data are used to approximate the performance function, and the best algorithm is selected based on the predictions of the learned model.

2.2 Algorithm configuration

Once an algorithm is selected, the next step is tuning of the algorithm. Let's consider the case where an algorithm α with parameters π_α is available to solve a decision-making problem $P(p)$ (Eq. (1)). We will refer to the parameters of the algorithm π_α as hyperparameters in order to distinguish them from the parameters of the decision-making problem p. The values of the hyperparameters π_α, also known as tuning or configuration, have a significant effect on the computational performance of the algorithm. Usually, these hyperparameters are selected by considering the average performance of the algorithm over a set of instances. However, one can exploit specific features of a problem and find a tuning that is optimal for the specific instance. This problem is formally known as the per-instance algorithm configuration problem and is stated as follows (Eggensperger et al. 2019; Schede et al. 2022):

Algorithm configuration. Given a decision-making problem $P(p)$, and an algorithm α with hyperparameters $\pi_\alpha \in \Pi_\alpha$ find hyperparameters π_α^* such that a performance function $m_{\text{conf}}^\alpha : \mathscr{P} \times \Pi_\alpha \mapsto \mathscr{M}$ is optimized.

The algorithm configuration problem has three components. The first is the decision-making problem $P(p)$, which is given. The second is the space of possible configurations Π_α, which is algorithm dependent. For example, in gradient descent algorithms, a common hyperparameter is the step size (or learning rate), which is a positive number, i.e., $\Pi_\alpha = \mathbb{R}_+$. The last component is the performance function m_{conf}^α, a metric used to compare two configurations of the algorithm α for a given problem. Similar to the algorithm selection problem, based on the application considered, different performance functions can be used such as solution time or solution quality. These components lead to the following formulation of the algorithm configuration problem

$$\pi_\alpha^* \in \arg\min_{\pi_\alpha \in \Pi_\alpha} m_{\text{conf}}^\alpha(P, \pi_\alpha). \tag{3}$$

The solution to the algorithm configuration problem is challenging. First, the performance function m_{conf}^α is not known explicitly, i.e., algorithm configuration is a black-box optimization problem. Also, evaluating the performance of a configuration for a given problem can be computationally expensive for large-scale decision-making problems. Finally, the search space of possible algorithm configurations can be very large.

The first approach to solving the algorithm configuration problem is to rely on sampling-based black-box optimization algorithms. Although this approach has been extensively used in the literature (Chen et al. 2011; Hutter et al. 2009, 2010, 2011; Liu et al. 2019), it can be slow for online applications, where given a decision-making problem, one must quickly find the best configuration of the algorithm and implement it. In such cases, ML can be used to learn (or approximate) offline either the performance function m_{conf}^α using a surrogate $\hat{m}_{\text{conf}}^\alpha$ or the solution of the algorithm configuration problem itself. Once these models are learned, then they are used online to find the best configuration.

2.3 Relation between algorithm selection and configuration

The algorithm selection and configuration problems share several characteristics. First, algorithm configuration can be considered as a special case of algorithm selection. Specifically, each configuration of an algorithm can be considered as a different algorithm, and thus, identifying the best possible configuration is equivalent to selecting the best algorithm. This can be considered as a simultaneous algorithm selection and configuration approach since one must consider simultaneously all the possible combinations of algorithms and configurations. In general, algorithm configuration is usually more challenging than algorithm selection since the search space is much larger. The algorithm selection problem has one degree of freedom, the algorithm to be used, and the number of available algorithms is usually small. However, in the algorithm configuration problem, the degrees of freedom are equal to the number of hyperparameters, and the possible number of configurations can be very large.

Both problems can be solved either via black-box optimization or ML-based approaches. In general, black-box methods have been used for offline applications where a solver is tuned to perform well on average for a given set of instances. Black-box optimization methods require a function evaluation, i.e., computing the performance function m for a given problem. In the context of algorithm selection and configuration, this translates to solving the decision-making problem $P(p)$ to optimality and obtaining the value of the performance function. This approach cannot be applied in an online setting where one has to identify the best algorithm or tuning without solving the problem. In such cases, one must learn a surrogate model from data offline and then use it for inference online.

Finally, the tasks of selecting and tuning an algorithm, as presented in Sections 2.1 and 2.2, can be considered static problems since they are solved only once. In general, algorithm selection and configuration can be performed multiple times when solving a decision-making problem, leading to dynamic algorithm selection and configuration problems. Consider, for example, branch and bound-based algorithms where different solvers can be used at different nodes in the tree (Markót and Schichl 2011) or even different solver tuning. This difference (static or dynamic) motivates the adoption of different solution strategies (see Figure 1). The static case is a one-step decision-making problem since, given an optimization problem (Eq. (1)), the ML model is used to identify the best algorithm or tuning. On the contrary, the dynamic case requires constant interaction between an ML model and the algorithm. Given the decision-making problem (Eq. (1)) and the state of the solution process, the ML model determines the best configuration for the algorithm; this is a multi-step decision-making problem. This difference motivates the application of supervised and reinforcement learning for algorithm selection and configuration as presented in the next sections.

3 Decision-making problems as inputs to ML models

Let's consider the case where an ML model $\hat{m}$, such as a feedforward neural network, is used to predict the solution time t of a given algorithm a for an instance $P(p)$, i.e., $t = \hat{m}(P(p), a)$. A major limitation in developing such a model is that the optimization problem in Eq. (1) cannot be used as the input to standard ML models, such as a neural network, decision tree, random forest, etc. A decision-making problem cannot be considered as a tabular or Euclidian data point since it has variables, constraints, and an objective (or multiple ones in the case of multi-objective problems). Also, the number of variables and constraints can vary for different problems or instances. Therefore, a transformation step is necessary to represent a decision-making problem in a form that can be used as the input to an ML model. This representation should (1) capture essential information about the problem, (2) be amenable to use for different problem sizes, i.e., varying number of variables and constraints, (3) not be affected by the ordering of the variables and constraints, and (4) be computed/constructed efficiently.

3.1 Vectorial feature representation

The standard approach to achieve the above requirements is to extract a set of easily computable features $\nu(P(p)) \in \mathbb{R}^{N_{\text{features}}}$ from the problem formulation and use them as inputs to an ML model (see Figure 2). We will refer to this as the vectorial feature representation. Examples of these features include the number of continuous and discrete variables, the number of constraints, the number of nonconvex terms, the convexity of the objective, etc. We refer the reader to (Hutter

Optimization problem Vectorial representation

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b \\ & x \in \mathbb{Z}^{n_x^d} \times \mathbb{R}^{n_x^c} \end{aligned} \qquad \nu = \begin{bmatrix} \text{Number of cont. variables} \\ \text{Number of discrete variables} \\ \vdots \\ \text{Condition number of } A \end{bmatrix}$$

Figure 2: Vectorial feature representation of an optimization problem.

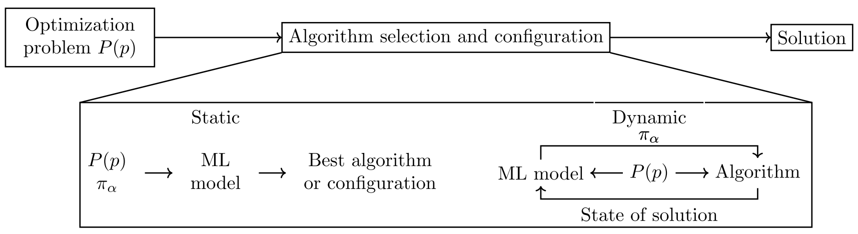


Figure 1: High level overview of ML-based solution approaches for algorithm selection and configuration.

et al. 2014; Schede et al. 2022; Smith-Miles and Lopes 2012) for an extended list. Although this approach has been extensively used to predict the solution time of mixed integer optimization problems, it has two limitations. First, significant effort and domain knowledge are required to identify the most informative features for a given class of problems. Secondly, the vectorial representation does not account for the exact interaction pattern among the variables and constraints.

3.2 Graph representation

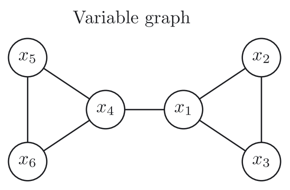
An alternative approach to represent a decision-making problem is a graph that can capture the interaction pattern between the variables and constraints. A graph $\mathscr{G}$ is a mathematical object that captures the interaction between a set of objects called nodes or vertices. We define node i as v_i and $V = \{v_i\}_{i=1}^N$ the set of nodes. The interaction pattern is captured via the edges $E = \{e_{ij}\}_{i\in V, j\in V}$ where $e_{ij} = 1$ if node i is connected with node j. A graph can also be represented by the adjacency matrix $A \in \mathbb{R}^{N\times N}$ where $A_{ij} = 1$ if an edge exists between node i and j, i.e., if $e_{ij} = 1$.

Three graphs can be used to represent a decision-making problem (Allman et al. 2019). The first and most generic one is the bipartite variable-constraint graph $\mathscr{G}_b(V_n, V_m, E)$ ($|V_n| = n, |V_m| = m$) with adjacency matrix A_b. This graph has two sets of nodes, one representing the constraints V_m and the other the variables V_n. The edges E capture the presence of a variable in a constraint. The second type of graph is a constraint graph $\mathscr{G}_c(V_m, E_m)$, where the nodes V_m are the constraints of the problem and the edges E_m represent the variables that couple two constraints. In this case, an edge between two nodes i and j can have a weight $w_{ij} \in \mathbb{Z}_+$, which denotes the number of variables that couple two constraints. The third type is the variable graph $\mathscr{G}_n(V_n, E_n)$ where the nodes V_n are the variables of the problem, the edges E_m represent whether two variables are coupled by appearing together in one or more constraints, and each edge has a weight that denotes the number of constraints that couple two variables. An example of a variable graph is presented in Figure 3.

The graph representation captures the structural coupling between the constraint and the variables, i.e., the presence or not of a variable in a constraint, as reflected in the adjacency matrix, as well as the strength of interaction captured via the edge weights. Such representations have been used extensively for developing control architectures, as well as implementing decomposition-based optimization and control algorithms (Allman et al. 2019; Aykanat et al.

Optimization problem

$$\begin{array}{rl}
\min & f(x_1, ..., x_6) \\
\text{s.t.} & c_1(x_1, x_3) \leq 0 \\
& c_2(x_1, x_2, x_3) \leq 0 \\
& c_3(x_1) \leq 0 \\
& c_4(x_1, x_4) \leq 0 \\
& c_5(x_4, x_5) \leq 0 \\
& c_6(x_4, x_5, x_6) \leq 0 \\
& x_1 \in [0, 10] \\
& x_2 \in \{0, 1\} \\
& x_3 \in \mathbb{Z}_+ \\
& x_4 \in [-10, 10] \\
& x_5 \in \{1, 2, 3, 4, 5\} \\
& x_6 \in (-\infty, \infty)
\end{array}$$



Graph representation $\mathcal{G}(A)$

$$\begin{array}{c|cccccc}
 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\
\hline
x_1 & 0 & 1 & 1 & 1 & 0 & 0 \\
x_2 & 1 & 0 & 1 & 0 & 0 & 0 \\
x_3 & 1 & 1 & 0 & 0 & 0 & 0 \\
x_4 & 1 & 0 & 0 & 0 & 1 & 0 \\
x_5 & 0 & 0 & 0 & 1 & 0 & 1 \\
x_6 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}$$

Figure 3: Graph representation of an optimization problem.

2004; Bergner et al. 2015; Ferris and Horn 1998; Jalving et al. 2022; Jogwar and Daoutidis 2017; Khaniyev et al. 2018; Michelena and Papalambros 1997; Moharir et al. 2017; Rio-Chanona et al. 2016; Shin et al. 2021; Tang et al. 2018; Wang and Ralphs 2013).

Under this representation, a decision-making problem, and in general a system of equations, is represented by a graph $\mathscr{G}$ with adjacency matrix A. Note that the graph and the adjacency matrices depend on the decision-making problem $P(p)$, i.e., $G(P(p))$ and $A(P(p))$.

3.3 Graph representation with nodal and edge features

Although the graph representation captures the structure of the problem, it does not account for the domain of the variables and the functional form in which they appear in the constraints. To achieve this, a set of features can be associated with each node and edge in the graph. For example, in the bipartite graph representation, a set of features ϕ_v^i can be used for each variable i, ϕ_c^j for each constraint j, and ϕ_e^{ij} for an edge between variable i and constraint j. Concatenation of these features form the feature matrices F_v, F_c, F_e, and a decision-making problem (Eq. (1)) can be represented by four matrices, the adjacency matrix A, the variable feature matric F_v, the constraint feature matrix F_c, and the edge feature matrix F_e (see Figure 4 for an example).

This representation has been extensively used for Mixed Integer Linear Programming problems (Ding et al. 2020; Gasse et al. 2019; Gupta et al. 2020, 2022; Li et al. 2022; Liu et al. 2022; Nair et al. 2020; Paulus et al. 2022). Some examples of features include the domain of the variables for the variable nodes, the type of constraint (equality or inequality) for the constraint nodes, and the coefficient of a variable in an edge for the edges. The ability of this representation to distinguish

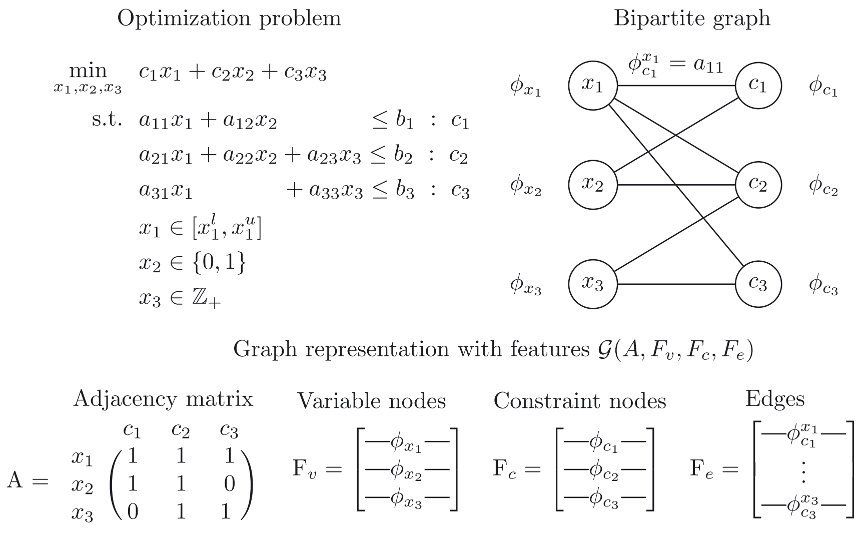

Optimization problem
$\min_{x_1,x_2,x_3} c_1x_1 + c_2x_2 + c_3x_3$
s.t. $a_{11}x_1 + a_{12}x_2 \le b_1 : c_1$
$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \le b_2 : c_2$
$a_{31}x_1 + a_{33}x_3 \le b_3 : c_3$
$x_1 \in [x_1^l, x_1^u]$
$x_2 \in \{0,1\}$
$x_3 \in \mathbb{Z}_+$
Bipartite graph
$\phi_{c_1}^{x_1} = a_{11}$
ϕ_{x_1} ϕ_{x_2} ϕ_{x_3} ϕ_{c_1} ϕ_{c_2} ϕ_{c_3}
Graph representation with features $\mathcal{G}(A, F_v, F_c, F_e)$
Adjacency matrix
Variable nodes
Constraint nodes
Edges
$A = \begin{array}{c|ccc} & c_1 & c_2 & c_3 \\ \hline x_1 & 1 & 1 & 1 \\ x_2 & 1 & 1 & 0 \\ x_3 & 0 & 1 & 1 \end{array}$
$F_v = [\phi_{x_1}; \phi_{x_2}; \phi_{x_3}]$
$F_c = [\phi_{c_1}; \phi_{c_2}; \phi_{c_3}]$
$F_e = [\phi_{c_1}^{x_1}; \vdots; \phi_{c_3}^{x_3}]$

Figure 4: Graph representation with features of a mixed integer linear optimization problem.

between different optimization problems has been proven rigorously for specific classes of LP and MILP problems and for specific tasks such as predicting optimal solution and feasibility (Chen et al. 2022a,b).

Remark 3.1. The representations presented in this section can be used as inputs to a surrogate model that predicts the computational performance of an algorithm. To this end, the following question arises: *Which representation should be used?* The chosen representation should be able to represent the key characteristics of a problem that affect the computational performance of a solver. Furthermore, the selection of the representation will determine the class of ML models that can be used. The vectorial representation can be used with interpretable models, such as decision trees and linear regression, as well as noninterpretable models, such as neural networks, random forests, gaussian processes, etc. The graph representation requires geometric deep learning models (Bronstein et al. 2017, 2021), such as graph neural networks, which are not inherently interpretable. This selection affects our ability to understand the computational performance of a solver (see Section 7.2 for a detailed discussion on this).

4 Learning to select a solution strategy

Given the aforementioned representations, first, we focus on the application of ML techniques for algorithm selection. One approach relies on regression to predict the value of the performance function for a given problem and then select the best algorithm. For each available algorithm α, data are generated to approximate the performance function m_α with a surrogate $\widehat{m}_\alpha$ where the input is a representation of the decision-making problem and the label is the value of the performance function of algorithm α. In this data generation process, the tuning of each algorithm α can either be the default one or the best possible one for the given instance. This approach has mainly exploited the vectorial feature representation of a decision-making problem (Hutter et al. 2014; Leyton-Brown et al. 2009) to predict the solution time of algorithms using neural networks (Eggensperger et al. 2017; Smith-Miles and Hemert 2011), decision trees (Smith-Miles and Hemert 2011), Gaussian processes (Bartz-Beielstein and Markon 2004), and sparse polynomial regression (Huang et al. 2010). Some applications include determining if dynamic programming or branch and search should be used for solving a knapsack problem (Hall and Posner 2007) and selecting a heuristic for constraint programming (Allen and Minton 1996).

An alternative approach is to approximate the solution of the algorithm selection problem itself, i.e., approximate the mapping $\mathscr{C}$ between the decision-making problem $P(p)$ and the best algorithm with a surrogate one $\widehat{\mathscr{C}}$, i.e., $\alpha^* = \widehat{\mathscr{C}}(P(p))$. In this approach, the output of the approximate map $\widehat{\mathscr{C}}$ is one of the available algorithms. Thus, the algorithm selection problem can be posed as a multi-class classification problem where a classifier will predict the solver that has the highest probability of being the solution to the algorithm selection problem. This approach has been used to determine the best solution strategy for traveling salesman problems (Pihera and Musliu 2014), select local nonlinear solver during branch and bound for mixed integer nonlinear optimization problems (Markót and Schichl 2011), determine whether Dantzig–Wolfe decomposition should be used for the solution of mixed integer linear optimization problems

(Kruber et al. 2017), determine whether a convex mixed integer nonlinear optimization should be solved with branch and bound or the outer approximation algorithm (Mitrai and Daoutidis 2024c), and determine whether a mixed integer quadratic optimization problem should be linearized or not (Bonami et al. 2022).

Yet another solution approach is based on case-based reasoning, an artificial intelligence approach where a task is solved based on the solution of other similar tasks (Kolodner 1992). In the context of algorithm selection, for a given problem, an algorithm α is selected based on its performance in similar instances. Case-based reasoning has been used to select whether a constraint programming or mixed integer programming approach should be used to solve a bid evaluation problem in combinatorial auctions using as features some properties of the graph representation of the problem, such as graph density, node degree, etc. (Guerri and Milano 2004).

5 Learning to configure an algorithm

The problem of learning to configure an algorithm has received significant attention from the operations research, computer science, and ML/AI communities. We hereby focus only on the acceleration of optimization algorithms for the solution of linear, mixed integer linear, and mixed integer nonlinear optimization problems. A decision-making problem can be solved either monolithically, where an algorithm considers all the variables simultaneously, or using a decomposition-based algorithm, where the problem is decomposed into a number of subproblems that are solved iteratively. Given the different nature of monolithic and decomposition-based algorithms, different algorithm configuration tasks arise. Thus, we consider the configuration of these algorithms separately.

5.1 Configuring monolithic solvers

5.1.1 Initialization

Initialization of an optimization algorithm is not usually considered a hyperparameter, yet it can have a significant effect on its computational performance. Usually, intuition and heuristics are used to identify a good feasible solution. However, the development of such initialization approaches is time consuming.

ML has been used to predict the optimal solution of a class of decision-making problems and use the prediction either as an initial guess or to fix some of the variables of the problem. This approach relies on input–output data $\mathscr{D} = \left\{P(p_i), x_i^*\right\}_{i=1}^{N_{\text{data}}}$, where the features are some representation of the decision making problem, as discussed in Section 3, and the label is the optimal solution x_i^* or part thereof. Given such data sets, supervised learning is used to train regression and classification models. Usually, regression is used for predicting the values of continuous variables, whereas classification is used for integer variables. This approach has been extensively used to accelerate the solution of decision making problems which are solved repeatedly online. Typical examples include model predictive control (MPC), where ML models predict the control action (Kumar et al. 2021; Vaupel et al. 2020), the values of the integer variables (Cauligi et al. 2022; Masti et al. 2020; Russo et al. 2023; Zhu and Martius 2020) (for mixed integer MPC), active constraints (Bertsimas and Stellato 2022; Cauligi et al. 2021; Klaučo et al. 2019; Misra et al. 2022), optimal power flow problems (Park and Van Hentenryck 2023), and facility location problems (Triantafyllou and Papathanasiou 2024).

An alternative approach is to approximate the iterative nature of optimization algorithms via ML models, i.e., emulate the evolution of the variables' values during the solution process. This approach has been used to emulate interior point solvers for predicting the solution of optimal power flow problems (Baker 2022) using the feature representation and general linear optimization problems (Qian et al. 2024) using the graph representation of the problem with features. These initialization approaches are based on the assumption that an initial guess close to the optimal solution will reduce the computational time. The main limitation of these initialization approaches is that the prediction is not necessarily feasible. Therefore, a feasibility restoration step is required to construct a feasible solution (Chen et al. 2024; Kotary et al. 2021). Alternatively, one can develop/compute rigorous bounds on the output of the ML model (Hertneck et al. 2018; Paulson and Mesbah 2020) to guarantee constraint satisfaction.

5.1.2 ML for preprocessing

Another key component of modern optimization solvers is preprocessing, a set of techniques used to reformulate the optimization problem and usually strengthen its relaxation (Achterberg et al. 2020). An example of a preprocessing procedure is bound tightening, where given a decision-making problem, the bounds of the variables are updated based on optimality and feasibility arguments. The former is known as optimality based bound tightening (OBBT), where given a decision-making problem, first the problem is convexified, and then the maximum and minimum value that a variable can take is found. This approach has been shown to lead to a reduction in solution time; however, it requires the solution of two optimization problems for each variable. ML has been used to determine the variables for which OBBT should be applied (Cengil et al. 2022). This approach has been applied to the solution of optimal power flow problems where the ML model takes as input a vectorial representation of the parameters of the optimization problem and predicts the variables for which application of OBBT leads to the best bound. Finally, we note that a similar approach has been developed for the case of feasibility based bound tightening (FBBT) (Nannicini et al. 2011) where the goal is to compute updated (tighter) bounds for all the variables while satisfying the constraints.

5.1.3 ML for branch and bound

Branch and bound is the backbone of mixed-integer optimization solvers. In this approach, given a mixed-integer linear optimization problem of the form

$$\begin{aligned} \underset{x,y}{minimize} \quad & c_1^\top x + c_2^\top y \\ \text{subject to} \quad & A_1 x + A_2 y \le b \\ & x \in \{0,1\}^{N_d}, y \in \mathbb{R}^{N_c}, \end{aligned} \tag{4}$$

branch and bound starts by solving the continuous relaxation, i.e., setting $x \in [0,1]^{N_d}$ in Eq. (4). The solution to this problem is usually fractional, i.e., the values of the x variables are not integers. In this case, first, a variable x_i is selected, and two new problems are created: one where x_i is fixed to zero and one where x_i is fixed to one (see Figure 5). The procedure of selecting a variable to branch is known as variable selection. Once these two problems are generated, one has to select which one to solve (node P_1, P_3 or P_4 in Figure 5); this is known as node selection. Overall, the variable and node selection strategies determine the

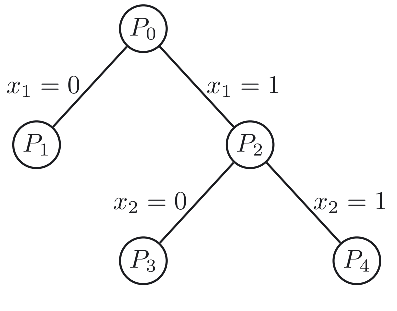


Figure 5: Branch and bound tree for a mixed integer linear optimization problem with two binary variables.

computational efficiency of the branch and bound algorithm (Lodi and Zarpellon 2017). Different variable selection rules have been proposed. A typical example is strong branching, where both branches corresponding to $x_i = 0$ and $x_i = 1$ are solved for each and every variable x_i with fractional value, and the one providing the best bound is selected. Although this approach leads to smaller branch and bound trees, i.e., fewer nodes are explored, it is computationally expensive. Identifying the best variable and node strategy is an algorithm configuration problem.

Several ML-based approaches have recently been proposed to automate and reduce the computational effort related to making optimal decisions during the branch and bound solution process for mixed integer linear optimization problems (Lodi and Zarpellon 2017). For variable selection, most approaches rely on the concept of imitation learning, where an ML model tries to copy the behavior of an expert, such as strong branching for the case of variable selection. This approach relies either on the vectorial representation of the problem (Alvarez et al. 2017; Khalil et al. 2016) or the graph representation with features (Gasse et al. 2019). An alternative approach is to exploit the sequential nature of variable selection and use reinforcement learning to find the variable to branch (Etheve et al. 2020; Huang et al. 2022; Parsonson et al. 2023; Scavuzzo et al. 2022). Finally, based on the relation between algorithm configuration and selection, the selection of a branching strategy has also been posed as an algorithm selection task for mixed integer linear problems (Di Liberto et al. 2016) and for spatial branching for polynomial optimization problems (Ghaddar et al. 2023).

Regarding node selection, two approaches have been proposed. In the first, node selection is posed as a Markov decision process and a policy is learned to determine which node to solve using imitation learning (He et al. 2014; Labassi et al. 2022). The alternative is to pose node selection as a multiarm bandit problem, where given a set of options, one must select an option that will lead to the highest reward. In the context of node selection, the options correspond to the available nodes to explore, and the reward can be either the solution time or the size of the branch and bound tree to be explored (Sabharwal et al. 2012).

5.1.4 ML for cutting planes

An important component of mixed-integer optimization algorithms/solvers is cutting planes (Dey and Molinaro 2018). These are usually linear inequalities that reduce the search space without affecting optimality. However, selecting which cutting plane to add is nontrivial since multiple types of cutting planes can be generated, and different numbers of cutting planes can be added during branch and bound.

Mixed-integer optimization solvers create a pool of cuts and add them based on heuristics.

Similar to learning to branch, ML can be used to select which cuts to add. These approaches usually learn a model that approximates the outcome of an expert, i.e., a rule or heuristic, that identifies the best possible cut (Baltean-Lugojan et al. 2019; Marousi and Kokossis 2022; Paulus et al. 2022). The addition of cutting planes can also be considered as a multistep process since cuts can be added to the root node as well as the other nodes that are explored during branch and bound. This has been considered in (Berthold et al. 2022), where a regression model is used to determine whether using local cuts at a node of the branch and bound tree can lead to a reduction in solution time. The alternative is to rely on reinforcement learning to determine which cuts to add in each node of the branch and bound tree (Tang et al. 2020; Wang et al. 2023).

5.2 Configuring all the parameters of a solver simultaneously

The ML approaches for algorithm configuration consider a specific aspect of the algorithm. One could consider all the parameters of a solver simultaneously. In this case, supervised, unsupervised, and reinforcement learning can be potentially used to identify the best configuration. Such approaches have been proposed for tuning mixed integer optimization solvers (Bartz-Beielstein and Markon 2004; Hutter et al. 2006, 2009, 2010; Iommazzo et al. 2020; Xu et al. 2011).

This approach can in principle exploit synergies between different parts of an algorithm or a solver. However, it leads to a significant increase in the complexity of the configuration and, subsequently, learning tasks. Furthermore, new architectures might be necessary to capture detailed information about the decision-making problem and the algorithm. For example, the graph representation with features and graph neural networks can guide the variable selection search during branch and bound. The algorithm, however, is usually represented as a vector, and each entry denotes the value of a hyperparameter. Therefore, new architectures and representations might be necessary to simultaneously capture information about the problem formulation and the algorithm configuration. Finally, we note that these ML-based approaches usually cannot provide guarantees regarding the performance of a solver or a configuration. This has motivated systematic analysis and design of numerical algorithms using data-driven (Balcan et al. 2024; Dietrich et al. 2020; Doncevic et al. 2024; Sambharya and Stellato 2024) and mathematical programming approaches (Das Gupta et al. 2024; Drori and Teboulle 2014; Mitsos et al. 2018). A list with the application of different ML approaches and the associated references for algorithm selection and configuration can be found in Table 1.

Table 1: Algorithm selection and configuration for different classes of optimization problems.

Task		**Optimization problem class**		
		Continuous (linear and nonlinear)	**Mixed integer linear**	**Mixed integer nonlinear**
Algorithm selection		Markót and Schichl (2011)	Guerri and Milano (2004), Hall and Posner (2007), Leyton-Brown et al. (2009), Smith-Miles and Hemert (2011), Pihera and Musliu (2014), and Kruber et al. (2017)	Bonami et al. (2022) and Mitrai and Daoutidis (2024c)
Algorithm configuration	Initialization	Vaupel et al. (2020), Kumar et al. (2021), Baker (2022), Park and Van Hentenryck (2023), and Qian et al. (2024)	Masti et al. (2020), Zhu and Martius (2020), Cauligi et al. (2022), Russo et al. (2023), Klaučo et al. (2019), Cauligi et al. (2021), Misra et al. (2022), Bertsimas and Stellato (2022), and Triantafyllou and Papathanasiou (2024)	
	Preprocessing	Cengil et al. (2022)		Nannicini et al. (2011)
	Branching priority	Ghaddar et al. (2023)	Khalil et al. (2016), Alvarez et al. (2017), Gasse et al. (2019), Etheve et al. (2020), Scavuzzo et al. (2022), Huang et al. (2022), and Parsonson et al. (2023)	
	Node selection		He et al. (2014), Labassi et al. (2022), and Sabharwal et al. (2012)	
	Cutting planes		Paulus et al. (2022), Baltean-Lugojan et al. (2019), Marousi and Kokossis (2022), Berthold et al. (2022), Tang et al. (2020), and Wang et al. (2023)	
	Multiple parameters	Chen et al. (2011)	Hutter et al. (2010), Hutter et al. (2009), Bartz-Beielstein and Markon (2004), Xu et al. (2011), Hutter et al. (2006), and Iommazzo et al. (2020)	Liu et al. (2019)

6 Learning to configure decomposition-based algorithms

Decomposition-based optimization algorithms have been extensively used to solve large-scale decision-making problems. Unlike monolithic approaches where all the variables are considered simultaneously, decomposition-based algorithms decompose the variables (and constraints) into a number of subproblems that are solved repeatedly. Most decomposition-based algorithms can be classified either as distributed or hierarchical. The main difference lies in the sequence upon which the subproblems are solved. In distributed algorithms, all the subproblems are solved in parallel and are coordinated via dual information, whereas in hierarchical algorithms, the subproblems are solved sequentially and are coordinated either via dual information or cuts (for the case of cutting plane-based algorithms). In general, the solution of a decision-making problem with a decomposition-based algorithm has three steps: (1) problem decomposition, (2) selection of coordination scheme, and (3) configuration. These steps can be considered as hyperparameters of a decomposition-based algorithm; therefore, several algorithm configuration problems must be solved prior to the implementation of decomposition-based algorithms. A list with the various ML-based approaches and corresponding references for accelerating decomposition-based algorithms can be found in Table 2.

6.1 Learning the structure of an optimization problem

The decomposition of an optimization problem is the basis for the application of a decomposition-based optimization algorithm and can have a significant effect on the computational performance of the algorithm. Traditionally, a decomposition was obtained using intuition about the coupling (structure) among the variables and constraints. Although this approach has been applied extensively, identifying the underlying structure of a problem is time-consuming and may not even be possible using only intuition.

Several automated structure detection methods have been proposed in the literature. These approaches rely on the graph representation of an optimization problem as represented in Section 3.2. Given the graph of a decision-making problem, graph partitioning algorithms are used to decompose the graph, i.e., the decision-making problem, into a number of subproblems. Typical algorithms include hypergraph partitioning (Aykanat et al. 2004; Ferris and Horn 1998; Jalving et al. 2022; Michelena and Papalambros 1997; Wang and Ralphs 2013) and community detection (Allman et al. 2019; Bergner et al. 2015; Khaniyev et al. 2018; Mitrai and Daoutidis 2020; Rio-Chanona et al. 2016; Tang et al. 2018). These graph partitioning methods usually make a-priory assumptions about the number of subproblems and the interaction patterns among them. To overcome these limitations, we have recently proposed the application of stochastic block modeling and Bayesian inference for estimating the structure of an optimization problem (Mitrai and Daoutidis 2021; Mitrai et al. 2022). This approach assumes that the graph of an optimization problem is generated by a probabilistic model with parameters b that capture information about the partition of the nodes into blocks and ω which captures interaction pattern between the blocks. The parameter b is a vector where the ith entry denotes the block membership of node i in the partition of the graph. For the variable graph, this parameter denotes the block membership of each variable, whereas in the

Table 2: Algorithm selection and configuration for decomposition-based optimization algorithms.

Task	Decomposition-based algorithm		
	Benders and generalized Benders decomposition	**Column generation**	**Lagrangean decomposition**
Structure detection	Michelena and Papalambros (1997), Wang and Ralphs (2013), Ferris and Horn (1998), Aykanat et al. (2004), Jalving et al. (2022), Bergner et al. (2015), Khaniyev et al. (2018), Tang et al. (2018), Allman et al. (2019), Rio-Chanona et al. (2016), Mitrai and Daoutidis (2020), Mitrai et al. (2022), Mitrai and Daoutidis (2021), Tang et al. (2018), and Basso et al. (2020)		
Initialization	Mitrai and Daoutidis (2024b,c)		Demelas et al. (2023) and Biagioni et al. (2020)
Coordination via cutting planes	Jia and Shen (2021) and Lee et al. (2020)	Morabit et al. (2021) and Chi et al. (2022)	

constraint graph, the block membership of each constraint. Given the graph of a decision-making problem, the parameters b are estimated or 'learned' via Bayesian inference. The estimated structure can be used as the basis for the application of distributed and hierarchical decomposition-based algorithms.

Finally, we note that regarding problem decomposition, the aforementioned approaches rely on the assumption that decomposing a decision-making problem based on the underlying structure leads to good computational performance. Although this has been shown to be a good assumption for a large class of problems (Basso et al. 2020; Tang et al. 2018), it is not guaranteed that a structure-based decomposition is the best possible one. An example is the case of the solution of two-stage stochastic optimization problems using Benders decomposition. Traditionally, the original problem is decomposed into a master problem, which considers the first stage decisions and a set of independent subproblems, each one representing a scenario. Recently it has been shown that adding some scenarios (or subproblems) to the master problem can lead to a reduction in the solution time (Crainic et al. 2021). In general, finding the best possible decomposition is an open problem.

6.2 Learning to warm-start decomposition-based optimization algorithms

Similar to the initialization of monolithic algorithms, the initialization of a decomposition-based algorithm can significantly affect its computational performance. However, predicting only the values of the variables is not enough since it does not account for the coordination aspect of a decomposition-based algorithm.

6.2.1 Initialization of distributed algorithms

For distributed-based algorithms, the coordination is achieved using Lagrangean multipliers. Therefore, an initialization requires an estimate of the values of the variables of the problem as well as the Lagrangean multipliers. This increases the complexity of the learning task compared to the initialization of monolithic solvers. This approach has been used to initialize the Lagrangean relaxation algorithm (a distributed decomposition-based algorithm) to solve network design and facility location problems (Demelas et al. 2023). This is achieved using an encoder–decoder architecture, where the input is the graph representation of the problem with features and the solution of the linear relaxation, and the output is an estimate of the multipliers. A similar approach has been developed for accelerating the alternating direction methods of multipliers (ADMM) using a recurrent neural network to predict the values of the Lagrangean multipliers and the complicating variables for the solution of optimal power flow problems (Biagioni et al. 2020).

6.2.2 Cutting plane-based hierarchical decomposition algorithms

Initialization is more challenging for cutting plane-based decomposition algorithms. In these methods, a decision-making problem is usually decomposed into a master problem, which contains all the integer variables and potentially some continuous, and a subproblem, which considers only continuous variables. The solution of the subproblem depends on the values of the variables of the master problem, which are called complicating variables. The master problem and subproblem are solved sequentially and are coordinated via cutting planes, i.e., linear inequalities that inform the master problem about the effect of the complicating variables on the subproblem. Usually, in the first iteration, the master problem is solved without cuts; the cuts are added iteratively based on the solution of the master problem. Adding an initial set of cuts can lead to better bounds and, thus, convergence in fewer iterations. However, similar to the cutting plane methods for branch and bround, determining which cuts to add as a warm start for decomposition-based methods is nontrivial. First, the number of potential cuts can be very large, and selecting which ones to add is a complex combinatorial problem. The second issue is related to the validity of the cuts for different instances. For cases where the parameters of the subproblem do not change, the cuts can be evaluated only once and added to the master problem every time a new instance must be solved. However, if the parameters of the subproblem change, then the previously evaluated cuts are not valid. Thus, one has to evaluate them, i.e., solve the subproblem, before adding them to the master problem.

In recent work, we have proposed several ML-based approaches to learn to initialize Benders decomposition by adding an initial set of cuts in the master problem for the solution of mixed integer model predictive control problems. For cases where the parameters of the subproblem do not change, and the complicating variables are continuous, we posed the cut selection problem as an algorithm configuration problem (Mitrai and Daoutidis 2024d). The number of cuts corresponds to the number of points used to discretize the domain of the complicating variables, and the performance function was the solution time, which was learned via active and supervised learning.

For the case where the parameters of the subproblem do not change, and the complicating variables are both discrete and continuous, the cut selection process has two steps. First, an ML-based branch and check Benders decomposition algorithm is used to obtain an approximate integer feasible solution and a set of integer feasible solutions, which are explored during branch and check (Mitrai and Daoutidis 2024a). The cuts related to these integer feasible solutions are added to the master problem, and then Benders decomposition is implemented to obtain the solution of the problem (Mitrai and Daoutidis 2024b). The integer feasible solutions guide the selection of the cuts to be added to the master problem. Finally, in the most generic case where the parameters of the subproblem change and the complicating variables are both continuous and discrete, a similar approach can be followed, where first, a set of integer feasible solutions is obtained by the ML-based branch and check. However, since the parameters of the subproblem change, the cuts related to these integer feasible solutions are first evaluated by solving the subproblem and then added to the master problem.

6.3 Learning to coordinate cutting plane-based decomposition algorithms

Once a decomposition is decided and an initialization is selected, the next step is the implementation of the algorithm. As discussed in Sections 6.2.2, for cutting plane-based algorithms, the steps are (1) solve the master problem and obtain the values of the complicating variables, (2) solve the subproblem, and (3) incorporate in the master problem information on the subproblems in the form of cuts. These three steps are repeated until the algorithm converges. Selecting which cutting planes to add during the solution process is an algorithm configuration problem.

In certain classes of problems, multiple subproblems can exist, and in each iteration, multiple cuts can be generated and added to the master problem. Although this strategy seems reasonable at first since a cut contains information about the subproblem, it can also significantly increase the computational complexity of the master problem. This has led to the development of ML-based architectures to determine which cuts to generate and add to the master problem during the solution. Two approaches have been developed to achieve this.

In the first, a classifier is used to predict whether a cut is valuable and should be added to the master problem. Different metrics are proposed to deem a cut valuable. The most commonly used one is the improvement in the bounds. This approach has been applied for the solution of two-stage stochastic optimization problems using Benders and generalized Benders decomposition (Jia and Shen 2021; Lee et al. 2020) as well as the solution of multistage stochastic optimization problems (Borozan et al. 2023). We note that a similar approach has been proposed for column generation where an ML model predicts if a column can lead to improvements in the bounds (Morabit et al. 2021).

The second approach exploits the iterative nature of decomposition-based algorithms and poses the cut selection problem as a reinforcement learning problem (Chi et al. 2022). Specifically, the solution of a decision-making problem with a decomposition-based algorithm is modeled as a Markov decision process, and the goal is to train a reinforcement learning agent which given a candidate set of cuts (obtained from the solution of the master problem) selects the cuts that should be added such that the number of steps (iterations) required to solve the problem is minimized.

7 Open problems and conclusions

In this section, we discuss open problems and new opportunities for applying ML to enhance the computational performance of algorithms for executing computational tasks in chemical engineering.

7.1 Application to general numerical tasks

The concepts discussed in this paper, as well as the ML-based solution strategies, can be applied to generic computational tasks that arise in chemical engineering. Typical examples include steady-state and dynamic process simulation. In such cases, one must solve a system of equations using an iterative numerical algorithm that has hyperparameters. Hence, algorithm selection and configuration approaches can be used to select the best simulation algorithm and tune it for the specific computational task. Some examples include the tuning of the successive over-relaxation algorithm for the solution of linear systems of equations (Khodak et al. 2023), selecting solvers for the solution of linear systems of equations (Bhowmick et al. 2009; Demmel et al. 2005; Dufrechou et al. 2021) and for the solution of initial value problems (Kamel et al. 1993). These results show that ML, in tandem with appropriate representations, might be able to accelerate process simulation, especially for large-scale and nonlinear systems, which are common in chemical engineering applications.

7.2 Can ML generate new insights?

All the aforementioned ML-based algorithm selection and configuration approaches answer the question of which algorithm to use and how to tune it. The next question is why is an algorithm (or configuration) able to solve a given problem instance efficiently? In other words, can ML generate new insights regarding the efficiency of a given algorithm for a class of decision-making problems? This question is relevant not only in the context of optimization algorithms but for the execution of numerical tasks in general (Kotthoff 2016). An approach to understanding the difficulty of solving a problem is to approximate the performance functions with interpretable models, such as decision trees, linear regression, and symbolic regression. However, these models usually have low accuracy, and more accurate models usually rely on deep learning (graph neural networks, feedforward neural networks, etc.), which is not inherently interpretable. This necessitates the utilization of explainable artificial intelligence tools for analyzing the outputs of deep learning models and potentially developing new interpretable deep learning architectures (Li et al. 2018; Rudin 2019). Overall, explaining and understanding the computational performance of an algorithm for a given decision-making problem is an open research problem.

7.3 Data availability

The data generation process is usually the most time consuming step in the development of an automated algorithm selection or configuration framework since a large number of decision-making problems must be solved, usually to optimality. Although parallel computing can be used to generate such datasets, this still requires significant computational resources. This computational cost can be potentially reduced using active, semi-, self-, and transfer learning approaches.

Active learning is a commonly used approach for cases where obtaining the labels of a data point is expensive (Settles 2009). This approach has been used to learn to initialize generalized Benders decomposition for the solution of mixed-integer model predictive control problems (Mitrai and Daoutidis 2024d). In this setting, a pool of data points is available, but only the features are known (e.g., some representation of the decision-making problems and the tuning) and obtaining the label requires the solution of the decision-making problem. The selection of the data point to be labeled is guided by the uncertainty of the prediction, i.e., we label the data point (combination of decision-making problem and tuning) for which the prediction of the solution time is the least certain. This approach still requires the labeling of data points.

Semi-supervised learning uses simultaneously labeled (usually few) and unlabeled data to train an ML model (Van Engelen and Hoos 2020). An example is wrapper methods where a first model is trained using the labeled data (initial training set). The model is subsequently used to general pseudo-labels for the unlabeled data which are added to the training data set and the model is retrained. Self-supervised learning uses the available unlabeled data to learn representations that can be useful for subsequent tasks such as classification and regression. Finally, transfer learning can be used to reduce the size of the training dataset by exploiting ML models trained for similar tasks (Weiss et al. 2016), such as branching in mixed integer linear and mixed integer nonlinear optimization problems.

7.4 Generative artificial intelligence

All the ML-based methods discussed so far are based on predictive machine learning/artificial intelligence techniques, namely supervised, unsupervised, and reinforcement learning. Recently generative artificial intelligence has made significant progress in developing AI-based systems capable of generating new content, such as video, image, and text. Given this remarkable progress, it is natural to wonder whether generative AI can be used to accelerate the solution of a decision-making problem.

The first application of generative AI is problem formulation from a natural language description of a decision-making problem. Preliminary results show that large language models (LLMs) can successfully formulate an optimization problem when the number of parameters, variables, and constraints is small (Ramamonjison et al. 2022, 2023). The natural language description has also been used to analyze infeasibility in a decision-making problem by making the LLM model interact with an optimization solver (Chen et al. 2023). LLMs have also been used to learn or discover new algorithms (Romera-Paredes et al. 2024) by coupling an LLM with a genetic programming framework, where the LLM provides new candidate algorithms which are evaluated and subsequently mutated by the LLM. The last application is that of generating optimization instances. This is achieved using the graph representation, with node and edge features, of a decision-making problem and developing a model that generates new graphs, i.e., optimization problems (Geng et al. 2024).

Overall, generative AI can be conceptually used for problem formulation, explaining the solution of a computational task, discovering new algorithms, and reformulating a decision-making problem. However, the capability of current transformer-based deep learning architectures (both ones depending on natural language and graph-based) to perform these tasks is an open problem.

Research ethics: Not applicable.
Informed consent: Not applicable.
Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.
Use of Large Language Models, AI and Machine Learning Tools: None declared.
Conflict of interest: The authors state no conflict of interest.
Research funding: McKetta Department of Chemical Engineering for IM and NSF-CBET (award no. 2313289) for PD.
Data availability: Not applicable.

References

Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., and Weninger, D. (2020). Presolve reductions in mixed integer programming. *INFORMS J. Comput.* 32: 473–506.

Allen, J.A. and Minton, S. (1996). Selecting the right heuristic algorithm: runtime performance predictors. In: *Advances in artifical intelligence: 11th biennial conference of the Canadian society for computational studies of intelligence, AI'96 Toronto, Ontario, Canada, May 21–24, 1996 proceedings 11*. Springer, pp. 41–53.

Allen, R.C., Iseri, F., Demirhan, C.D., Pappas, I., and Pistikopoulos, E.N. (2023). Improvements for decomposition based methods utilized in the development of multi-scale energy systems. *Comput. Chem. Eng.* 170: 108135.

Allman, A., Tang, W., and Daoutidis, P. (2019). DeCODe: a community-based algorithm for generating high-quality decompositions of optimization problems. *Opt. Eng.* 20: 1067–1084.

Alvarez, A.M., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS J. Comput.* 29: 185–195.

Aykanat, C., Pinar, A., and Çatalyürek, Ü.V. (2004). Permuting sparse rectangular matrices into block-diagonal form. *SIAM J. Sci. Comput.* 25: 1860–1879.

Baker, K. (2022). Emulating ac opf solvers with neural networks. *IEEE Trans. Power Syst.* 37: 4950–4953.

Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. (2024). Learning to branch: generalization guarantees and limits of data-independent discretization. *J. ACM* 71: 1–73.

Baltean-Lugojan, R., Bonami, P., Misener, R., and Tramontani, A. (2019). Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. *preprint*, Available at: http://www.optimization-online.org/DB_HTML/2018/11/6943.html.

Bartz-Beielstein, T. and Markon, S. (2004). Tuning search algorithms for real-world applications: a regression tree based approach. In: *Proceedings of the 2004 congress on evolutionary computation (IEEE cat. no. 04TH8753)*, Vol. 1. IEEE, pp. 1111–1118.

Basso, S., Ceselli, A., and Tettamanzi, A. (2020). Random sampling and machine learning to understand good decompositions. *Ann. Oper. Res.* 284: 501–526.

Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *Eur. J. Oper. Res.* 290: 405–421.

Bergner, M., Caprara, A., Ceselli, A., Furini, F., Lübbecke, M.E., Malaguti, E., and Traversi, E. (2015). Automatic Dantzig-Wolfe reformulation of mixed integer programs. *Math. Prog.* 149: 391–424.

Berthold, T., Francobaldi, M., and Hendel, G. (2022). Learning to use local cuts. *arXiv preprint arXiv:2206.11618*.

Bertsimas, D. and Stellato, B. (2022). Online mixed-integer optimization in milliseconds. *INFORMS J. Comput.* 34: 2229–2248.

Bhosekar, A. and Ierapetritou, M.G. (2018). Advances in surrogate based modeling, feasibility analysis, and optimization: a review. *Comput. Chem. Eng.* 108: 250–267.

Bhowmick, S., Toth, B., and Raghavan, P. (2009). Towards low-cost, high-accuracy classifiers for linear solver selection. In: *Computational science–ICCS 2009: 9th international conference Baton Rouge, LA, USA, May 25–27, 2009 proceedings, part I*. Springer, pp. 463–472.

Biagioni, D., Graf, P., Zhang, X., Zamzam, A.S., Baker, K., and King, J. (2020). Learning-accelerated ADMM for distributed DC optimal power flow. *IEEE Control Syst. Lett.* 6: 1–6.

Biegler, L.T. (2024). Multi-level optimization strategies for large-scale nonlinear process systems. *Comput. Chem. Eng.* 185: 108657.

Bonami, P., Lodi, A., and Zarpellon, G. (2022). A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX. *Oper. Res.* 70: 3303–3320.

Borozan, S., Giannelos, S., Falugi, P., Moreira, A., and Strbac, G. (2023). A machine learning-enhanced benders decomposition approach to solve the transmission expansion planning problem under uncertainty. *arXiv preprint arXiv:2304.07534*.

Boukouvala, F., Misener, R., and Floudas, C.A. (2016). Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. *Eur. J. Oper. Res.* 252: 701–727.

Bradley, W., Kim, J., Kilwein, Z., Blakely, L., Eydenberg, M., Jalvin, J., Laird, C., and Boukouvala, F. (2022). Perspectives on the integration between first-principles and data-driven modeling. *Comput. Chem. Eng.*: 107898, https://doi.org/10.1016/j.compchemeng.2022.107898.

Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Process. Mag.* 34: 18–42.

Bronstein, M.M., Bruna, J., Cohen, T., and Veličković, P. (2021). Geometric deep learning: grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*.

Cauligi, A., Culbertson, P., Schmerling, E., Schwager, M., Stellato, B., and Pavone, M. (2021). Coco: online mixed-integer control via supervised learning. *IEEE Robot. Autom. Lett.* 7: 1447–1454.

Cauligi, A., Chakrabarty, A., Di Cairano, S., and Quirynen, R. (2022). PRISM: recurrent neural networks and presolve methods for fast mixed-integer optimal control. In: *Learning for dynamics and control conference*. PMLR, pp. 34–46.

Cengil, F., Nagarajan, H., Bent, R., Eksioglu, S., and Eksioglu, B. (2022). Learning to accelerate globally optimal solutions to the AC optimal power flow problem. *Electr. Power Syst. Res.* 212: 108275.

Chen, W., Shao, Z., Wang, K., Chen, X., and Biegler, L.T. (2011). Random sampling-based automatic parameter tuning for nonlinear programming solvers. *Ind. Eng. Chem. Res.* 50: 3907–3918.

Chen, Z., Liu, J., Wang, X., Lu, J., and Yin, W. (2022a). On representing linear programs by graph neural networks. *arXiv preprint arXiv:2209.12288*.

Chen, Z., Liu, J., Wang, X., Lu, J., and Yin, W. (2022b). On representing mixed-integer linear programs by graph neural networks. *arXiv preprint arXiv: 2210.10759*.

Chen, H., Constante-Flores, G.E., and Li, C. (2023). Diagnosing infeasible optimization problems using large language models. *arXiv preprint arXiv:2308.12923*.

Chen, H., Constant-Flores, G.E., and Li, C. (2024). Diagnosing infeasible optimization problems using large language models. *Inf. Sys. Oper. Res.* 62: 573–587.

Chi, C., Aboussalah, A., Khalil, E., Wang, J., and Sherkat-Masoumi, Z. (2022). A deep reinforcement learning framework for column generation. *Adv. Neural Inf. Process. Syst.* 35: 9633–9644.

Christofides, P.D., Scattolini, R., Muñoz de la Peña, D., and Liu, J. (2013). Distributed model predictive control: a tutorial review and future research directions. *Comput. Chem. Eng.* 51: 21–41.

Conejo, A.J., Castillo, E., Minguez, R., and Garcia-Bertrand, R. (2006). *Decomposition techniques in mathematical programming: engineering and science applications*. Springer Science & Business Media, Berlin.

Cozad, A., Sahinidis, N.V., and Miller, D.C. (2014). Learning surrogate models for simulation-based optimization. *AIChE J.* 60: 2211–2227.

Crainic, T.G., Hewitt, M., Maggioni, F., and Rei, W. (2021). Partial benders decomposition: general methodology and application to stochastic network design. *Transp. Sci.* 55: 414–435.

Daoutidis, P., Lee, J.H., Harjunkoski, I., Skogestad, S., Baldea, M., and Georgakis, C. (2018). Integrating operations and control: a perspective and roadmap for future research. *Comput. Chem. Eng.* 115: 179–184.

Daoutidis, P., Lee, J.H., Rangarajan, S., Chiang, L., Gopaluni, B., Schweidtmann, A.M., Harjunkoski, I., Mercangöz, M., Mesbah, A., Boukouvala, F., et al. (2023a). Machine learning in process systems engineering: challenges and opportunities. *Comput. Chem. Eng.*: 108523, https://doi.org/10.1016/j.compchemeng.2023.108523.

Daoutidis, P., Megan, L., and Tang, W. (2023b). The future of control of process systems. *Comput. Chem. Eng.* 178: 108365.

Das Gupta, S., Van Parys, B.P., and Ryu, E.K. (2024). Branch-and-bound performance estimation programming: a unified methodology for constructing optimal optimization methods. *Math. Program.* 204: 567–639.

Demelas, F., Roux, J.L., Lacroix, M., and Parmentier, A. (2023). Predicting accurate Lagrangian multipliers for mixed integer linear programs. *arXiv preprint arXiv:2310.14659*.

Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, R.C., and Yelick, K. (2005). Self-adapting linear algebra algorithms and software. *Proc. IEEE* 93: 293–312.

Dey, S.S. and Molinaro, M. (2018). Theoretical challenges towards cutting-plane selection. *Math. Program.* 170: 237–266.

Di Liberto, G., Kadioglu, S., Leo, K., and Malitsky, Y. (2016). Dash: dynamic approach for switching heuristics. *Eur. J. Oper. Res.* 248: 943–953.

Dietrich, F., Thiem, T.N., and Kevrekidis, I.G. (2020). On the Koopman operator of algorithms. *SIAM J. Appl. Dyn. Syst.* 19: 860–885.

Ding, J.-Y., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., and Song, L. (2020). Accelerating primal solution findings for mixed integer programs based on solution prediction. *Proc. AAAI Conf. Artif. Intell.* 34: 1452–1459.

Doncevic, D.T., Mitsos, A., Guo, Y., Li, Q., Dietrich, F., Dahmen, M., and Kevrekidis, I.G. (2024). A recursively recurrent neural network (r2n2) architecture for learning iterative algorithms. *SIAM J. Sci. Comput.* 46: A719–A743.

Drori, Y. and Teboulle, M. (2014). Performance of first-order methods for smooth convex minimization: a novel approach. *Math. Program.* 145: 451–482.

Dufrechou, E., Ezzatti, P., Freire, M., and Quintana-Ortí, E.S. (2021). Machine learning for optimal selection of sparse triangular system solvers on GPUs. *J. Parallel Distr. Comput.* 158: 47–55.

Eggensperger, K., Lindauer, M., and Hutter, F. (2017). Neural networks for predicting algorithm runtime distributions. *arXiv preprint arXiv: 1709.07615*.

Eggensperger, K., Lindauer, M., and Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *J. Artif. Intell. Res.* 64: 861–893.

Ellis, M., Durand, H., and Christofides, P.D. (2014). A tutorial review of economic model predictive control methods. *J. Process Control* 24: 1156–1178.

Etheve, M., Alès, Z., Bissuel, C., Juan, O., and Kedad-Sidhoum, S. (2020). Reinforcement learning for variable selection in a branch and bound algorithm. In: *International conference on integration of constraint programming, artificial intelligence, and operations research*. Springer, pp. 176–185.

Ferris, M.C. and Horn, J.D. (1998). Partitioning mathematical programs for parallel solution. *Math. Program.* 80: 35–61.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Adv. Neural Inf. Process. Syst.* 32.

Geng, Z., Li, X., Wang, J., Li, X., Zhang, Y., and Wu, F. (2024). A deep instance generative framework for milp solvers under limited data availability. *Adv. Neural Inf. Process. Syst.* 36.

Ghaddar, B., Gómez-Casares, I., González-Díaz, J., González-Rodríguez, B., Pateiro-López, B., and Rodríguez-Ballesteros, S. (2023). Learning for spatial branching: an algorithm selection approach. *INFORMS J. Comput.* 35: 1024–1043.

Grossmann, I.E. (2012). Advances in mathematical programming models for enterprise-wide optimization. *Comput. Chem. Eng.* 47: 2–18.

Grossmann, I.E., Apap, R.M., Calfa, B.A., García-Herreros, P., and Zhang, Q. (2016). Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty. *Comput. Chem. Eng.* 91: 3–14.

Guerri, A. and Milano, M. (2004). Learning techniques for automatic algorithm portfolio selection. *ECAI* 16: 475.

Gupta, P., Gasse, M., Khalil, E., Mudigonda, P., Lodi, A., and Bengio, Y. (2020). Hybrid models for learning to branch. *Adv. Neural Inf. Process. Syst.* 33: 18087–18097.

Gupta, P., Khalil, E.B., Chetélat, D., Gasse, M., Bengio, Y., Lodi, A., and Kumar, M.P. (2022). Lookback for learning to branch. *arXiv preprint arXiv:2206.14987*.

Hall, N.G. and Posner, M.E. (2007). Performance prediction and preselection for optimization and heuristic solution procedures. *Oper. Res.* 55: 703–716.

Hanselman, C.L. and Gounaris, C.E. (2016). A mathematical optimization framework for the design of nanopatterned surfaces. *AIChE J.* 62: 3250–3263.

He, H., Daume, H., III, and Eisner, J.M. (2014). Learning to search in branch and bound algorithms. *Adv. Neural Inf. Process. Syst.* 27.

Hertneck, M., Köhler, J., Trimpe, S., and Allgöwer, F. (2018). Learning an approximate model predictive controller with guarantees. *IEEE Control Syst. Lett.* 2: 543–548.

Huang, L., Jia, J., Yu, B., Chun, B.-G., Maniatis, P., and Naik, M. (2010). Predicting execution time of computer programs using sparse polynomial regression. *Adv. Neural Inf. Process. Syst.* 23.

Huang, Z., Chen, W., Zhang, W., Shi, C., Liu, F., Zhen, H.-L., Yuan, M., Hao, J., Yu, Y., and Wang, J. (2022). Branch ranking for efficient mixed-integer programming via offline ranking-based policy learning. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer, pp. 377–392.

Hutter, F., Hamadi, Y., Hoos, H.H., and Leyton-Brown, K. (2006). Performance prediction and automated tuning of randomized and parametric algorithms. In: *International conference on principles and practice of constraint programming*. Springer, pp. 213–228.

Hutter, F., Hoos, H.H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *J. Artif. Intell. Res.* 36: 267–306.

Hutter, F., Hoos, H.H., and Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In: *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*. Springer, pp. 186–202.

Hutter, F., Hoos, H.H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In: *International conference on learning and intelligent optimization*. Springer, pp. 507–523.

Hutter, F., Xu, L., Hoos, H.H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: methods & evaluation. *Artif. Intell.* 206: 79–111.

Iommazzo, G., d'Ambrosio, C., Frangioni, A., and Liberti, L. (2020). Learning to configure mathematical programming solvers by mathematical programming. In: *International conference on learning and intelligent optimization*. Springer, pp. 377–389.

Jalving, J., Shin, S., and Zavala, V.M. (2022). A graph-based modeling abstraction for optimization: concepts and implementation in plasmo.jl. *Math. Program. Comput.* 14: 699–747.

Jia, H. and Shen, S. (2021). Benders cut classification via support vector machines for solving two-stage stochastic programs. *INFORMS J. Optim.* 3: 278–297.

Jogwar, S.S. and Daoutidis, P. (2017). Community-based synthesis of distributed control architectures for integrated process networks. *Chem. Eng. Sci.* 172: 434–443.

Kamel, M.S., Enright, W.H., and Ma, K. (1993). ODEXPERT: an expert system to select numerical solvers for initial value ODE systems. *ACM Trans. Math. Software* 19: 44–62.

Kerschke, P., Hoos, H.H., Neumann, F., and Trautmann, H. (2019). Automated algorithm selection: survey and perspectives. *Evol. Comput.* 27: 3–45.

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. *Proc. AAAI Conf. Artif. Intell.* 30, https://doi.org/10.1609/aaai.v30i1.10080.

Khaniyev, T., Elhedhli, S., and Erenay, F.S. (2018). Structure detection in mixed-integer programs. *INFORMS J. Comput.* 30: 570–587.

Khodak, M., Chow, E., Balcan, M.-F., and Talwalkar, A. (2023). Learning to relax: setting solver parameters across a sequence of linear system instances. *arXiv preprint arXiv:2310.02246*.

Kim, B. and Maravelias, C.T. (2022). Supervised machine learning for understanding and improving the computational performance of chemical production scheduling mip models. *Ind. Eng. Chem. Res.* 61: 17124–17136.

Klaučo, M., Kalúz, M., and Kvasnica, M. (2019). Machine learning-based warm starting of active set methods in embedded model predictive control. *Eng. Appl. Artif. Intell.* 77: 1–8.

Kolodner, J.L. (1992). An introduction to case-based reasoning. *Artif. Intell. Rev.* 6: 3–34.

Kotary, J., Fioretto, F., Van Hentenryck, P., and Wilder, B. (2021). End-to-end constrained optimization learning: a survey. *arXiv preprint arXiv: 2103.16378*, https://doi.org/10.24963/ijcai.2021/610.

Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: a survey. In: *Data mining and constraint programming*. Springer, Switzerland, pp. 149–190.

Kruber, M., Lübbecke, M.E., and Parmentier, A. (2017). Learning when to use a decomposition. In: *International conference on AI and OR techniques in constraint programming for combinatorial optimization problems*. Springer, pp. 202–210.

Kumar, P., Rawlings, J.B., and Wright, S.J. (2021). Industrial, large-scale model predictive control with structured neural networks. *Comput. Chem. Eng.* 150: 107291.

Labassi, A.G., Chételat, D., and Lodi, A. (2022). Learning to compare nodes in branch and bound with graph neural networks. *Adv. Neural Inf. Process. Syst.* 35: 32000–32010.

Lee, M., Ma, N., Yu, G., and Dai, H. (2020). Accelerating generalized benders decomposition for wireless resource allocation. *IEEE Wireless Commun.* 20: 1233–1247.

Leyton-Brown, K., Nudelman, E., and Shoham, Y. (2009). Empirical hardness models: methodology and a case study on combinatorial auctions. *J. ACM* 56: 1–52.

Li, O., Liu, H., Chen, C., and Rudin, C. (2018). Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions. *Proc. AAAI Conf. Artif. Intell.* 32, https://doi.org/10.1609/aaai.v32i1.11771.

Li, X., Qu, Q., Zhu, F., Zeng, J., Yuan, M., Mao, K., and Wang, J. (2022). Learning to reformulate for linear programming. *arXiv preprint arXiv: 2201.06216*.

Liberti, L. and Pantelides, C.C. (2006). An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *J. Glob. Optim.* 36: 161–189.

Liu, J., Ploskas, N., and Sahinidis, N.V. (2019). Tuning BARON using derivative-free optimization algorithms. *J. Glob. Optim.* 74: 611–637.

Liu, D., Fischetti, M., and Lodi, A. (2022). Learning to search in local branching. *Proc. AAAI Conf. Artif. Intell.* 36: 3796–3803.

Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *Top* 25: 207–236.

Markót, M.C. and Schichl, H. (2011). Comparison and automated selection of local optimization solvers for interval global optimization methods. *SIAM J. Optim.* 21: 1371–1391.

Marousi, A. and Kokossis, A. (2022). On the acceleration of global optimization algorithms by coupling cutting plane decomposition algorithms with machine learning and advanced data analytics. *Comput. Chem. Eng.* 163: 107820.

Masti, D., Pippia, T., Bemporad, A., and De Schutter, B. (2020). Learning approximate semi-explicit hybrid MPC with an application to microgrids. *IFAC-PapersOnLine* 53: 5207–5212.

McAllister, R.D. and Rawlings, J.B. (2022). Advances in mixed-integer model predictive control. In: *2022 American control conference (ACC)*. IEEE, pp. 364–369.

Mesbah, A. (2016). Stochastic model predictive control: an overview and perspectives for future research. *IEEE Control Syst. Mag.* 36: 30–44.

Michelena, N.F. and Papalambros, P.Y. (1997). A hypergraph framework for optimal model-based decomposition of design problems. *Comput. Optim. Appl.* 8: 173–196.

Misener, R. and Biegler, L. (2023). Formulating data-driven surrogate models for process optimization. *Comput. Chem. Eng.* 179: 108411.

Misra, S., Roald, L., and Ng, Y. (2022). Learning for constrained optimization: identifying optimal active constraint sets. *INFORMS J. Comput.* 34: 463–480.

Mitrai, I. and Daoutidis, P. (2020). Decomposition of integrated scheduling and dynamic optimization problems using community detection. *J. Process Control* 90: 63–74.

Mitrai, I. and Daoutidis, P. (2021). Efficient solution of enterprise-wide optimization problems using nested stochastic blockmodeling. *Ind. Eng. Chem. Res.* 60: 14476–14494.

Mitrai, I. and Daoutidis, P. (2024a). Computationally efficient solution of mixed integer model predictive control problems via machine learning aided benders decomposition. *J. Process Control* 137: 103207.

Mitrai, I. and Daoutidis, P. (2024b). Learning to recycle benders cuts for mixed integer model predictive control. In: *Computer aided chemical engineering*, Vol. 53. The Netherlands: Elsevier, pp. 1663–1668.

Mitrai, I. and Daoutidis, P. (2024c). Taking the human out of decomposition-based optimization via artificial intelligence, part I: learning when to decompose. *Comput. Chem. Eng.* 186: 108688.

Mitrai, I. and Daoutidis, P. (2024d). Taking the human out of decomposition-based optimization via artificial intelligence, part II: learning to initialize. *Comput. Chem. Eng.* 186: 108686.

Mitrai, I., Tang, W., and Daoutidis, P. (2022). Stochastic blockmodeling for learning the structure of optimization problems. *AIChE J.* 68: e17415.

Mitsos, A., Najman, J., and Kevrekidis, I.G. (2018). Optimal deterministic algorithm generation. *J. Glob. Optim.* 71: 891–913.

Moharir, M., Kang, L., Daoutidis, P., and Almansoori, A. (2017). Graph representation and decomposition of ODE/hyperbolic PDE systems. *Comput. Chem. Eng.* 106: 532–543.

Morabit, M., Desaulniers, G., and Lodi, A. (2021). Machine-learning–based column selection for column generation. *Transp. Sci.* 55: 815–831.

Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al. (2020). Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*.

Nannicini, G., Belotti, P., Lee, J., Linderoth, J., Margot, F., and Wächter, A. (2011). A probing algorithm for MINLP with failure prediction by SVM. In: *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems: 8th international conference, CPAIOR 2011, Berlin, Germany, May 23–27, 2011. Proceedings 8*. Springer, pp. 154–169.

National Academies of Sciences, Engineering, and Medicine (2022). New Directions for Chemical Engineering. Washington D.C.: National Academies of Sciences, Engineering, and Medicine.

Park, S. and Van Hentenryck, P. (2023). Self-supervised primal-dual learning for constrained optimization. *Proc. AAAI Conf. Artif. Intell.* 37: 4052–4060.

Parsonson, C.W., Laterre, A., and Barrett, T.D. (2023). Reinforcement learning for branch-and-bound optimisation using retrospective trajectories. *Proc. AAAI Conf. Artif. Intell.* 37: 4061–4069.

Paulson, J.A. and Mesbah, A. (2020). Approximate closed-loop robust model predictive control with guaranteed stability and constraint satisfaction. *IEEE Control Syst. Lett.* 4: 719–724.

Paulus, M.B., Zarpellon, G., Krause, A., Charlin, L., and Maddison, C. (2022). Learning to cut by looking ahead: cutting plane selection via imitation learning. In: *International conference on machine learning*. PMLR, pp. 17584–17600.

Pihera, J. and Musliu, N. (2014). Application of machine learning to algorithm selection for TSP. In: *2014 IEEE 26th international conference on tools with artificial intelligence*. IEEE, pp. 47–54.

Pistikopoulos, E.N., Barbosa-Povoa, A., Lee, J.H., Misener, R., Mitsos, A., Reklaitis, G.V., Venkatasubramanian, V., You, F., and Gani, R. (2021). Process systems engineering–the generation next? *Comput. Chem. Eng.* 147: 107252.

Qian, C., Chételat, D., and Morris, C. (2024). Exploring the power of graph neural networks in solving linear optimization problems. In: *International conference on artificial intelligence and statistics*. PMLR, pp. 1432–1440.

Raghunathan, A.U. and Biegler, L.T. (2003). Mathematical programs with equilibrium constraints (MPECs) in process engineering. *Comput. Chem. Eng.* 27: 1381–1392.

Ramamonjison, R., Li, H., Yu, T.T., He, S., Rengan, V., Banitalebi-Dehkordi, A., Zhou, Z., and Zhang, Y. (2022). Augmenting operations research with auto-formulation of optimization models from problem descriptions. *arXiv preprint arXiv:2209.15565*.

Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., et al. (2023). Nl4opt competition: formulating optimization problems based on their natural language descriptions. In: *NeurIPS 2022 competition track*. PMLR, pp. 189–203.

Rice, J.R. (1976). The algorithm selection problem. *Adv. Comput.* 15: 65–118.

Rio-Chanona, E.A.D., Fiorelli, F., and Vassiliadis, V.S. (2016). Automated structure detection for distributed process optimization. *Comput. Chem. Eng.* 89: 135–148.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J., Ellenberg, J.S., Wang, P., Fawzi, O., et al. (2024). Mathematical discoveries from program search with large language models. *Nature* 625: 468–475.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1: 206–215.

Russo, L., Nair, S.H., Glielmo, L., and Borrelli, F. (2023). Learning for online mixed-integer model predictive control with parametric optimality certificates. *IEEE Control Syst. Lett.* 7: 2215–2220.

Sabharwal, A., Samulowitz, H., and Reddy, C. (2012). Guiding combinatorial optimization with UCT. In: *Integration of AI and OR Techniques in contraint programming for combinatorial optimzation problems: 9th international conference, CPAIOR 2012, Nantes, France, May 28–June 1, 2012. Proceedings 9*. Springer, pp. 356–361.

Sambharya, R. and Stellato, B. (2024). Data-driven performance guarantees for classical and learned optimizers. *arXiv preprint arXiv:2404.13831*.

Sansana, J., Joswiak, M.N., Castillo, I., Wang, Z., Rendall, R., Chiang, L.H., and Reis, M.S. (2021). Recent trends on hybrid modeling for industry 4.0. *Comput. Chem. Eng.* 151: 107365.

Scavuzzo, L., Chen, F., Chételat, D., Gasse, M., Lodi, A., Yorke-Smith, N., and Aardal, K. (2022). Learning to branch with tree mdps. *Adv. Neural Inf. Process. Syst.* 35: 18514–18526.

Schede, E., Brandt, J., Tornede, A., Wever, M., Bengs, V., Hüllermeier, E., and Tierney, K. (2022). A survey of methods for automated algorithm configuration. *J. Artif. Intell. Res.* 75: 425–487.

Schweidtmann, A.M., Esche, E., Fischer, A., Kloft, M., Repke, J.-U., Sager, S., and Mitsos, A. (2021). Machine learning in chemical engineering: a perspective. *Chem. Ing. Tech.* 93: 2029–2039.

Settles, B. (2009). *Active learning literature survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison.

Shin, J., Badgwell, T.A., Liu, K.-H., and Lee, J.H. (2019). Reinforcement learning–overview of recent progress and implications for process control. *Comput. Chem. Eng.* 127: 282–294.

Shin, S., Coffrin, C., Sundar, K., and Zavala, V.M. (2021). Graph-based modeling and decomposition of energy infrastructures. *IFAC-PapersOnLine* 54: 693–698.

Smith-Miles, K. and van Hemert, J. (2011). Discovering the suitability of optimisation algorithms by learning from evolved instances. *Ann. Math. Artif. Intell.* 61: 87–104.

Smith-Miles, K. and Lopes, L. (2012). Measuring instance difficulty for combinatorial optimization problems. *Comput. Oper. Res.* 39: 875–889.

Tang, W. and Daoutidis, P. (2022). Data-driven control: overview and perspectives. In: *2022 American control conference (ACC)*. IEEE, pp. 1048–1064.

Tang, W., Allman, A., Pourkargar, D.B., and Daoutidis, P. (2018). Optimal decomposition for distributed optimization in nonlinear model predictive control through community detection. *Comput. Chem. Eng.* 111: 43–54.

Tang, Y., Agrawal, S., and Faenza, Y. (2020). Reinforcement learning for integer programming: learning to cut. In: *International conference on machine learning*. PMLR, pp. 9367–9376.

Tawarmalani, M. and Sahinidis, N.V. (2005). A polyhedral branch-and-cut approach to global optimization. *Math. Program.* 103: 225–249.

Triantafyllou, N. and Papathanasiou, M.M. (2024). Deep learning enhanced mixed integer optimization: learning to reduce model dimensionality. *arXiv preprint arXiv:2401.09556*.

Van Engelen, J.E. and Hoos, H.H. (2020). A survey on semi-supervised learning. *Mach. Learn.* 109: 373–440.

Vaupel, Y., Hamacher, N.C., Caspari, A., Mhamdi, A., Kevrekidis, I.G., and Mitsos, A. (2020). Accelerating nonlinear model predictive control through machine learning. *J. Process Control* 92: 261–270.

Wächter, A. and Biegler, L.T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Prog.* 1: 25–57.

Wang, J. and Ralphs, T. (2013). Computational experience with hypergraph-based methods for automatic decomposition in discrete optimization. In: *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems: 10th international conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18–22, 2013. Proceedings 10*. Springer, pp. 394–402.

Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. (2023). Learning cut selection for mixed-integer linear programming via hierarchical sequence model. *arXiv preprint arXiv:2302.00244*.

Weiss, K., Khoshgoftaar, T.M., and Wang, D. (2016). A survey of transfer learning. *J. Big Data* 3: 1–40.

Xu, L., Hutter, F., Hoos, H.H., and Leyton-Brown, K. (2011). Hydra-MIP: automated algorithm configuration and selection for mixed integer programming. In: *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pp. 16–30.

Zhu, J.-J. and Martius, G. (2020). Fast non-parametric learning to accelerate mixed-integer programming for hybrid model predictive control. *IFAC-PapersOnLine* 53: 5239–5245.