Applied Intelligence (2024) 54:1020–1046
https://doi.org/10.1007/s10489-023-05240-w

Beyond games: a systematic review of neural Monte Carlo tree search applications

Marco Kemmerling[1] · Daniel Lütticke[1] · Robert H. Schmitt[1]

Accepted: 12 December 2023 / Published online: 28 December 2023

Abstract

The advent of AlphaGo and its successors marked the beginning of a new paradigm in playing games using artificial intelligence. This was achieved by combining Monte Carlo tree search, a planning procedure, and deep learning. While the impact on the domain of games has been undeniable, it is less clear how useful similar approaches are in applications beyond games and how they need to be adapted from the original methodology. We perform a systematic literature review of peer-reviewed articles detailing the application of neural Monte Carlo tree search methods in domains other than games. Our goal is to systematically assess how such methods are structured in practice and if their success can be extended to other domains. We find applications in a variety of domains, many distinct ways of guiding the tree search using learned policy and value functions, and various training methods. Our review maps the current landscape of algorithms in the family of neural monte carlo tree search as they are applied to practical problems, which is a first step towards a more principled way of designing such algorithms for specific problems and their requirements.

Keywords Monte carlo tree search · MCTS · Neural monte carlo tree search · Reinforcement learning · Model-based reinforcement learning · Decision-time planning

1 Introduction

The combination of Monte Carlo Tree Search (MCTS) and deep learning led to the historical event of the computer program AlphaGo beating a human champion in the game of Go [99], which had been considered beyond the capabilities of computational approaches for a long time. Since then, such approaches, which we term *neural MCTS* in this review, have enjoyed a huge amount of popularity. They have been applied to many other games and yield promising results in the field of general game playing [89, 100].

While the effectiveness of neural MCTS in game contexts has been clearly established, the transfer of such approaches to non-game-playing applications is still a fairly recent development and hence less well understood. This transfer to other applications may create significant value, as previously computationally intractable problems may become tractable, just as playing the game of Go became tractable with the introduction of AlphaGo. More generally, neural MCTS methods may be able to find higher quality solutions than previous approaches and occupy a unique niche by balancing the trade-off between computational cost and solution quality.

In principle, neural MCTS methods are applicable to any (discrete) problem that can be addressed by traditional model-free reinforcement learning methods. The promise of neural MCTS approaches is in spending additional computational budget to increase decision quality compared to model-free reinforcement learning. This additional computational budget is spent on a planning procedure guided by neural networks, therefore potentially combining the advantages of forward planning and generalising from past experience.

Since games have different characteristics than many other problems neural MCTS approaches can be applied to, a direct transfer of algorithms like AlphaZero without any modifications to problems other than games is often not possible. This leads researchers to tailor neural MCTS methods to their specific applications and hence creates a considerable amount

✉ Marco Kemmerling
marco.kemmerling@ima.rwth-aachen.de

1 Information Management in Mechanical Engineering (WZL-MQ/IMA), RWTH Aachen University, Aachen, Germany

of algorithmic variation in the field. An overarching understanding of why certain variants are especially suitable for certain problem settings has not been explicitly developed in the literature yet. Further, the existing algorithmic variation is not well documented, as different variants continue to be introduced in individual publications, but little attention has been devoted to creating a clear view of the bigger picture. The first step towards such a view of the bigger picture is in reviewing the current state neural MCTS approaches and applications.

Some surveys and reviews of MCTS approaches have been published in the past, e.g. Mandziuk [68] provides a survey of selected MCTS applications on non-game-playing problems. However, it only examines two different applications, neither of which feature any neural guidance.

A more extensive survey is provided in [161], which also features a brief section on the combination of MCTS and deep learning. However, their focus is on games rather than other applications.

We are not aware of any reasonably extensive review of applications of neural MCTS methods in non-game-playing domains. We believe that such a review can shed light on the extent to which such methods can be transferred to more practical problems and how neural MCTS methods can be designed to cope with the requirements of different use cases. To gain an understanding of the kinds of problems for which neural MCTS is suitable, we first review problem settings of already existing neural MCTS applications. In a second step, we review the existing variation in the design of neural MCTS algorithms. Our intention behind this review is two-fold: First, we aim to give practitioners an overview to evaluate whether their applications are suitable for the use of neural MCTS and what algorithmic design choices are available to them. Second, we hope to contribute towards a more thorough understanding of the effect of different design choices and towards a more principled guide to create neural MCTS algorithms for specific problem settings.

The following research questions guide our review:

1. In which disciplines, domains, and application areas is neural MCTS used? What are the commonalities and differences in the observed applications?
2. What differences in the design of neural MCTS methods can be observed compared to applications in games?
3. Where and how can neural guidance be used during the tree search?

To address these questions, we perform a systematic literature review and analyze the resulting literature. Starting with a keyword search in multiple databases, we filter articles for relevance, perform additional forward and backward searches, and extract a set of predefined data items from each article included in the review. The detailed review process is described in Section 3. Before, we provide a brief introduction to the concepts of reinforcement learning, MCTS, and AlphaZero in the next section. The remaining sections begin with a focus on the problems described in the surveyed publications in Section 4, and continue with an examination of the employed methods in Section 5. We end our review with a brief discussion in Section 6.

2 Reinforcement learning & neural MCTS

2.1 Reinforcement learning

Reinforcement Learning (RL) is a paradigm of machine learning, in which *agents* learn from experience collected from an *environment*. To do so, an agent observes the *state* s of the environment and executes an *action* a based on this state. Upon acting, agents receive a *reward* r and observe a new state s'. A problem which follows this kind of formulation is called a Markov decision process (MDP) if the new state s' only depends on the state s immediately preceding it and the action a of the agent. The agent's goal in such an MDP is to maximize the return, i.e. the expected long-term cumulative reward, by learning an appropriate *policy* π, i.e. a behavioural strategy that prescribes an action, or a probability distribution over actions, for a given state [108].

Such a policy can be learned directly from experience, e.g. through policy gradient methods, or it can be derived from a learned action-value function. An action-value function $Q^{\pi}(s, a)$ estimates the value, i.e. the expected return, of taking action a in state s. From such a learned action-value function, a deterministic policy can be derived by greedily choosing the best action, while a stochastic policy can be derived by sampling actions proportionally to their value. In addition to the policy- and value-based approaches described so far, hybrid approaches which synergistically learn both policy and value functions are often employed as well. Such methods are called actor-critic approaches and often learn the state-value function $V^{\pi}(s)$ instead of the action-value function $Q^{\pi}(s, a)$. The former computes the expected return of state s when following policy π, while the latter computes the expected return of state s when first executing action a and following π in subsequent steps [108]. The superscript π is often omitted for more concise notation.

In contrast to model-free approaches, in model-based RL, a *model* of the environment is used for *planning*. A model simulates the dynamics of the environment either exactly or approximately. Planning simply refers to the simulation of experience using a model and planning approaches can be categorized into *background planning* and *decision-time planning*. In the former, the training data consisting of real experiences collected from the environment is augmented with imagined experience generated from a model. In the latter, the action selection at a given time-step is dependent on planning (ahead) using a model, i.e. the consequences of different choices of actions are imagined to improve the policy for the current state [73]. MCTS, further explained in the following, can be considered a form of decision-time planning.

2.2 The connection between RL and MCTS

MCTS arose as a heuristic search method to play combinatorial games by performing a type of tree search based on random sampling. In such games, a player has to decide which action to perform in a given state to maximize an outcome z at the terminal state of the game. While MCTS has not been traditionally thought of as a type of reinforcement learning, the scenario described here bears strong similarities to the formulation of reinforcement learning problems given earlier and some authors have explored this connection in detail [116]. To avoid ambiguity, we will not use the term reinforcement learning to refer to MCTS in this article. Similarly to RL, MCTS also produces a policy π_{MCTS} and a value estimate v_{MCTS}. In MCTS, the policy is produced for a given state s by a multi-step look-ahead search, i.e. by considering future scenarios and determining which sequence of actions will lead to favourable outcomes starting from s. This policy is produced anew for every encountered state, i.e. the determined policy does not generalize to states other than the one currently encountered. In contrast, traditional RL produces policies by learning from past experience that aim to generalize to unseen situations. At decision-time, no forward search is performed and an action is simply chosen based on the policy learned from past experience. In a sense, MCTS looks into the future, while traditional RL looks back to the past to determine actions. As a consequence, RL requires computationally expensive upfront training but incurs negligible computational cost at decision time, while MCTS requires no training, but performs computationally expensive planning at decision time.

2.3 MCTS

The general idea of MCTS is to iteratively build up a search tree of the solution space by balancing the exploration of infrequently visited tree branches with the exploitation of known, promising tree branches. This is accomplished by the repeated execution of four different phases: selection, expansion, evaluation, and back-propagation. In the selection phase, starting from the root node, actions are chosen until a leaf node s_L is encountered. New children are then added to this leaf node in the expansion phase and their value is estimated in the evaluation phase. Finally, the values of the newly added nodes are back-propagated up the tree to update the values of nodes along the path to s_L. In the following, we describe each of these phases in more detail.

Selection In the selection phase, starting from the root node, an action is chosen according to some mechanism. This leads to a new state, in which the selection mechanism is applied again. The process is repeated until a leaf node is encountered.

The mechanism of action selection is referred to as the *tree policy*. While different mechanisms exist, the UCB1 [4] formula is a popular choice. When MCTS is used with the UCB1 formula, the resulting algorithm is called Upper Confidence Bound for Trees (UCT) [53]. In UCT, the action selection is defined as follows:

$$a = \underset{a}{\operatorname{argmax}} \frac{W(s,a)}{N(s,a)} + c\sqrt{\frac{ln\ N(s)}{N(s,a)}} \tag{1}$$

where $W(s,a)$ represents the number of wins encountered in the search up to this point when choosing action a in state s, $N(s,a)$ the number of times a has been selected in s, and $N(s)$ the number of times s has been visited. The left part of the sum encourages exploitation of actions known to lead to favourable results where the fraction $\frac{W(s,a)}{N(s,a)}$ can be seen as an approximation of $Q(s,a)$. The right part of the sum encourages exploration by giving a higher weight to actions that have been visited less often compared to the total visit count of the state. Exploration and exploitation are balanced by the exploration constant c. For game outcomes $z \in [0, 1]$, the optimal choice of c is $c = \frac{1}{\sqrt{2}}$ [54], but for rewards outside this range, c may have to be adjusted [8].

Expansion After repeated application of the selection step, the search may arrive at a node with unexpanded potential children. Once this happens, one or more children of the node will be expanded. There are some possible variations in this phase. In some cases, all possible children are expanded when a leaf node (a node with *no* children) is encountered. In other cases, a single child is expanded when an expandable node (a node with *some* as of yet unexpanded children) is encountered. Expanding all children right away may lead to undesirable tree growth depending on the application.

In some literature, expandable nodes are also called leaf nodes. For clarity, we will only use the term leaf node to refer to true leaf nodes without any children in this article. Note that a leaf node is not the same as a terminal node, with the former merely being the current end of a tree branch, while the latter is a node that represents an end state of the game (see Fig. 1).

Evaluation Once a node has been expanded, it is evaluated to initialize $W(s,a)$ and $N(s,a)$. This evaluation is sometimes also called *simulation*, *roll-out*, or *play-out* and consists of playing the game starting from the newly expanded node until

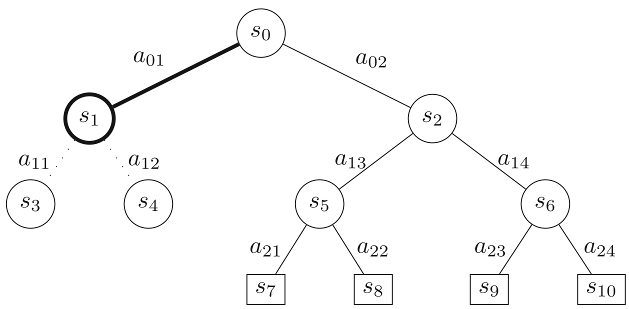

Fig. 1 Search tree as seen at a given time during the search. The current node is indicated with a thick border, as are the edges that were traversed in the current iteration of the search. s_1 is currently a leaf node and its potential children s_3 and s_4 are considered for expansion, as their dotted edges signify. The terminal nodes s_7, s_8, s_9, and s_10 are represented by rectangular nodes

a terminal state is encountered. The outcome z at the terminal state is the result of the evaluation. The game is played according to a *default policy*, which determines the sequence of actions between the newly expanded node and the terminal one. In the simplest case, the default policy samples actions uniformly randomly [54].

Instead of evaluating newly expanded nodes, it is also possible to only evaluate leaf nodes and and leave the evaluation of the newly expanded nodes for a later point in the search, when they are again encountered as leaf nodes themselves.

Back-propagation The outcome z from the evaluation phase is propagated up the tree to update $W(s, a)$ among the preceding nodes. The visit counts of all selected nodes are incremented as well.

Once the back-propagation phase is finished, the process starts anew from the selection phase until a predefined simulated budget is reached.

2.4 AlphaZero

The program known as *AlphaGo* drew attention for being the first computer program to beat a professional human player in a full-size game of Go [99] by combining deep learning and MCTS. While AlphaGo relied on supervised pre-training on human expert moves prior to reinforcement learning, its successor, AlphaGo Zero, was only trained using reinforcement learning by self-play. It further simplified the training by reducing the number of employed neural networks. AlphaGo Zero was still developed specifically for the board game Go and incorporated some game-specific mechanisms. In contrast, the next iteration of the AlphaGo family, AlphaZero, is more generic and can be applied to a variety of board games.

The algorithms introduced in this subsection are all examples of neural MCTS, i.e. MCTS guided by neural networks. While we focus on the AlphaZero family here due to its popularity, similar ideas were independently proposed under the name of *Expert Iteration* [2]. In the following, we provide more details on AlphaZero as one representative of neural MCTS methods utilized for games.

Like regular MCTS, AlphaZero follows the four phases of selection, expansion, evaluation, and back-propagation. Some of the phases are assisted by a neural network f_θ, which, given a state, produces a policy vector $\mathbf{p}$, i.e. a probability distribution over all actions, and an estimate v of the state value.

The selection phase in AlphaZero uses a variant of the Predictor + UCT (PUCT) formula [83]:

$$a = \underset{a}{\operatorname{argmax}}\ Q(s, a) + c\ P(s, a)\ \frac{\sqrt{N(s)}}{1 + N(s, a)} \tag{2}$$

where $P(s, a)$ denotes a prior probability of choosing action a in state s given by f_θ [100].

Once the selection phase reaches a leaf node s_L, it is evaluated by the neural network $(\mathbf{p}, v) = f_\theta(s_L)$. The leaf node is then fully expanded and its children initialized with $N(s_L, a) = 0$, $W(s_L, a) = 0$, $Q(s_L, a) = 0$, $P(s_L, a) = p_a$. In the back-propagation step, the statistics of each node including and preceding s_L are then updated as: $N(s_t, a_t) = N(s_t, a_t) + 1$, $W(s_t, a_t) = W(s_t, a_t) + v$, $Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$ [100].

Note that the expansion and evaluation phases are interwoven here to some degree and do not strictly follow the order of the phases in standard MCTS. In this review, we are generally not overly concerned with the MCTS phases as a strictly ordered set of algorithmic steps, but more with the function each phase fulfills in the tree search.

Once the search budget is exhausted, an improved policy π_{MCTS} is derived from the visit counts $N(s, a)$ in the tree and a corresponding value estimate v_{MCTS} is extracted. The produced policy π_{MCTS} and value estimate v_{MCTS} are then used as training targets to further improve f_θ. Since AlphaZero plays two-player games, some mechanism is required to determine the actions of the second player. In what is called *self-play*, the actions for the second player are chosen by (some version of) the same policy currently being trained for the first player [100].

As a model-based RL algorithm, AlphaZero needs a model of the environment to perform the tree search. This model is simply assumed to be given, although further extensions such as MuZero [89] demonstrate that such a model can be learned from collected experience during the search.

To recapitulate, the neural guidance in AlphaZero, consists of evaluating nodes by using f_θ to compute v and $\mathbf{p}$, which are then used in the selection phase. The way neural guidance is used in AlphaZero is not the only possible form of neural guidance. Other possibilities to guide the search exist, as will become apparent in Section 5.

3 Research methodology

We perform a systematic literature review, i.e. our review follows a structured, explicit, and reproducible method to identify and evaluate a body of literature relevant to our research questions [117]. Our approach is sequential, meaning that we follow a series of pre-defined steps in a given sequence consisting of a keyword search in multiple databases, a screening process to filter for relevant articles, a forward and backward search, data extraction from all included articles, followed by analysis and synthesis of the results.

While we aim to take a neutral position and hence do not want to limit the collected literature on arbitrary grounds, a comprehensive literature search attempting to capture *all* the relevant literature is infeasible due to incurred time-requirements. Instead, we aim to balance feasibility and coverage by collecting a representative sample of the existing literature by limiting ourselves to a keyword search with a defined set of keywords in a limited number of databases. Both the set of keywords as well as the set of databases could be enlarged to arrive at more comprehensive results.

3.1 Search query & databases

To find relevant publications, we derive three types of keywords:

1. Based on neural MCTS being a combination of MCTS and neural networks or traditional reinforcement learning:
   - "reinforcement learning" AND "monte carlo tree search"
   - "neural monte carlo tree search"
   - "neural MCTS"
2. Based on MCTS providing the ability to perform decision-time planning in a model-based reinforcement learning setting:
   - "decision-time planning" AND "reinforcement learning"
3. Based on the names given to algorithms in the AlphaGo family:
   - AlphaGo
   - AlphaZero
   - MuZero

Each of the partial search strings expressed after a bullet point above is connected with an OR operator to arrive at one overall search query.

We use this query to search for publications in the databases Web of Science, IEEExplore, Scopus, ScienceDirect, and PubMed. The search query is applied to the abstract, title, and keywords.

3.2 Eligibility criteria & screening

To be included in the review, a given publication must fulfill a predefined set of eligibility criteria:

1. **Must feature an application of MCTS guided by a neural network.** This excludes publications which are purely reviews or surveys. By guidance, we mean that a learned policy or value function is used in at least one of the phases of the tree search. These functions can of course be learned by other means than neural networks. We explicitly only consider neural network based approaches here because preliminary searches showed that attempting to include other methods leads to many more irrelevant search results while providing relatively little additional value.
2. **The publication must contain at least some amount of validation of the presented approach.** Purely conceptual articles are not considered.
3. **The problem to which neural MCTS is applied must not be a game.** While many impressive results have been achieved using neural MCTS in game playing, we are interested in determining whether such approaches transfer to other applications as well. We do consider applications that are not typically considered a game, but have been modelled as a game to facilitate the use of neural MCTS.
4. **Publication language must be English.**

For each of the publications retrieved during the keyword search described in the previous section, we assess its eligibility according to the above criteria. After the removal of duplicates, the screening process is conducted in two phases. In the first phase, we only examine the abstract of each article and discard it if it is clear that at least one of the above criteria is not fulfilled. If there is any ambiguity, we reexamine the article in a second phase, where we repeat the process using the full text of the article. Any article that is not discarded in this second phase will be included in our review.

3.3 Forward and backward search

After the abstract and full-text screening described above, we perform a forward and backward search [130] based on the set of publications which has passed the screening process. That is, for every article, we check its references for further relevant publications and also look for publications that in turn reference the articles which passed our screening.

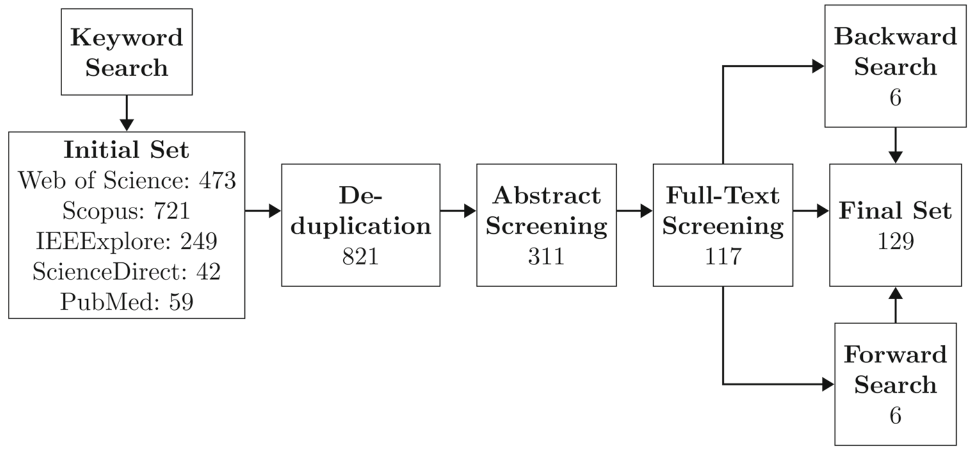

Fig. 2 The literature search and screening process starting from a set of keywords to the final set of publications to be included in the review

In this step, we notice that among the additionally identified literature, many simply cite an approach already included in our review for a similar application without performing any modifications or providing additional details. We do not include such publications in our review since they do not provide any benefit in addressing our research questions. The eligibility criteria described above apply to the results from forward and backward search as well.

The full search process from keyword search to our final set of publications is visualized in Fig. 2.

3.4 Data extraction

For every article resulting from the search process described above, we extract information along a set of predefined categories. These are given in Table 1, where *problem* refers to a short description of the examined problem, *time* is either continuous or discrete, *horizon* either finite or infinite, *transitions* either deterministic or stochastic.

Since it can be difficult to extract information from works originating from different disciplines with differing terminology and varying descriptions of details, we generally err on the side of providing incomplete rather than wrong information.

Some of the collected information did not lead to notable insights and will hence not be discussed in this review. This includes the activation functions, the author affiliations, and comparisons of the neural MCTS approach with model-free RL or MCTS without neural guidance. In such comparisons, neural MCTS tends to find solutions with superior quality, but this could simply be due to positive-results publication bias.

4 Neural MCTS applications

4.1 Application fields

To determine the applicability of neural MCTS outside of game-playing, we survey the areas of application to which neural MCTS has been transferred. The algorithmic details of individual approaches are analyzed in Section 5. Here, we simply outline where such approaches are applied. We find applications in a wide variety of domains including chemistry, medicine, production, electrical engineering, and computer science. In the following, we assign each publication to a specific application area. Note that his merely serves the purpose of creating an overview of the research landscape. Many articles could be assigned to more than one category, and the choice of categories itself could have been made in many different ways.

Chemistry In the chemical literature in particular, neural MCTS has received considerable attention. It has been used to perform synthesis planning, de novo molecular design,

Table 1 Information extracted from each article after screening

Application	Neural MCTS	Implementation	Other
Problem	Selection Details	Network Type	Country
Time	Expansion Details	Activation Functions	Organization Type
Reward Scale	Evaluation Details	Policy Loss	Comparison MCTS
Horizon	Use of Self-play	Value Loss	Comparison Model-free
Transitions	Training Method	MCTS Hyperparameters	Comparison Baselines
Action space	Learned Model	Code Availability	Used Hardware

Table 2 Applications in chemistry

Source	Application
[23]	Protein Folding
[25]	Lead Generation (Drug Discovery)
[67]	RNA folding
[105]	Reconstructing Molecular Structure from Nuclear Magnetic Resonance Spectra
[109]	Optimization of De Novo Stable Organic Radicals for Aqueous Redox Flow Batteries
[36]	Synthesis Planning
[48]	Synthesis Planning
[90]	Synthesis Planning
[91]	Synthesis Planning
[112]	Synthesis Planning
[126]	Synthesis Planning
[148]	Synthesis Planning
[63]	De Novo Molecule Design
[65]	De Novo Molecule Design
[74]	De Novo Molecule Design
[78]	De Novo Molecule Design
[106]	De Novo Molecule Design
[145]	De Novo Molecule Design

protein folding, and more (see Table 2). In many cases, states are represented by a simplified molecular-input line-entry system (SMILES) string, a notation allowing for the representation of molecular structure [132]. Such a string can then be iteratively constructed during the MCTS process until a viable molecule is found or the attempt is discarded.

Molecular applications appear to be a comparatively mature branch of neural MCTS research, as evidenced by the fact that authors are building on each others work and by the existence of standardized, commonly used implementations such as ChemTS [145]. This is an exception rather than the norm, as most literature is more disjointed and most other works either do not specify any implementation or use a custom one (see Appendix A for a detailed list of observed implementations).

Table 3 Applications in material science

Source	Application
[24]	Depth-graded Multilayer Structure Design for X-ray Optics
[122]	Constitutive Model Generation
[152]	Inverse Material Design of Ionic Liquids for CO2 Capture
[110]	Design of Metal-Organic Frameworks
[135]	

Table 4 Applications in electronic design

Source	Application
[15]	Electronic Circuit Routing
[41]	
[80]	
[111]	Memory Chip Redundancy Analysis

Material science Closely related is the domain of material science (see Table 3). In some cases, the SMILES representation is used here as well, such as in the design of metal-organic frameworks [110, 135], where metal ions and organic ligands are combined to create structures of various shapes.

In other cases, neural MCTS is used to optimize the thickness of alternating layers of two materials in a multilayer structure such that certain desired properties are achieved [24] and to generate models which describe the mechanical behaviour of materials in various circumstances [122].

Electronics design In the design of electronic circuits, neural MCTS is applied to solve routing problems in multiple cases [15, 41, 80] (see Table 4), where it can outperform e.g. traditional A* based approaches [41]. Thacker et al. [111] further explore performing redundancy analysis in memory chips using neural MCTS.

Energy systems Neural MCTS finds many applications in the operation of energy systems, especially in innovative grid concepts which aim to enable a more sustainable energy supply (see Table 5). This includes optimizing the operation of residential microgrids in an online fashion [96] as well as non-intrusive load monitoring and identification [50].

Some attention is also directed towards managing a grid consisting of renewable energy sources and battery systems,

Table 5 Applications in energy systems

Source	Application
[1]	Voltage Regulation in Distributed Energy Systems
[50]	Load Monitoring and Identification in Smart Grids
[96]	Residential Microgrid Scheduling
[107]	Electrical Transmission Network Self-Healing
[114]	Artificial Dispatcher Intelligent Control System in Electric Networks
[133]	Predictive Maintenance Management for Battery Energy Storage Systems

Table 6 Applications in production systems

Source	Application
[37]	Production Line Buffer Planning
[38]	Dynamically Interconnected Assembly Systems Scheduling
[58]	Industrial Mining Production Scheduling
[81]	Sheet Metal Production Scheduling
[121]	Parallel Machine Workshop Scheduling
[147]	Collaborative Assembly

which absorb the former's fluctuations in power output. For instance, Al-Saffar et al. [1] devise a system to coordinate voltage regulation in a distributed energy network with battery systems at multiple locations, while [133] use neural MCTS to address predictive maintenance problems in such systems.

Production Applications in production systems mainly concern themselves with various kinds of scheduling approaches. Here, the processing sequence of jobs or operations on different machines is to be determined to e.g. minimize the total time until all jobs have finished processing (see Table 6). Traditional RL approaches are also increasingly being investigated for these types of problems [34, 51, 86, 149]. A closer examination of the advantages and disadvantages of traditional RL and neural MCTS methods for scheduling approaches may be an interesting line of future research.

Further applications in production include line buffer planning in car manufacturing [37] as well as assembly planning in collaborative human-robot scenarios [147].

Combinatorial optimization While the scheduling problems described above are problems from the field of combinatorial optimization, the authors approach them from a production perspective and pay close attention to the details of their individual use cases. A second group of combinatorial optimization applications can be found in Table 7. Here, the problems are more abstract and investigated from a computer science lens.

Table 7 Applications in combinatorial optimization problems

Source	Application
[46]	Graph Coloring
[52]	PBQP-based register allocation
[75]	Machine Scheduling, Vehicle Routing
[134]	Bin Packing
[136]	Traveling Salesman Problem
[137]	
[138]	Highest Safe Rung Problem
[139]	
[141]	Quantified Boolean Formula Satisfaction

Table 8 Applications in cloud and edge computing

Source	Application
[11]	Task Offloading for UAV Edge Computing
[12]	Resource Allocation in Mobile Edge Computing
[19]	Directed Acyclic Graph Task Scheduling
[45]	
[77]	Cloud Workflow Scheduling

Combinatorial optimization problems share many similarities with combinatorial board games. In a reinforcement learning context, they are typically solved constructively by building a solution iteratively from scratch, or they are solved by improvement, i.e. by iteratively improving some existing solution. In both cases, the problem features inherently discrete time steps and an inherently discrete action space. Differences can be observed in that there is no obvious notion of winning or losing, but rather a sense of relative performance. In addition, combinatorial games feature a fixed board and a fixed set of game pieces. In, e.g. a traveling salesman problem, the equivalent of a board may be considered a graph with weighted edges connecting different cities. Such a graph, however, will vary, with each problem instance consisting of different cities to be traveled through. In machine scheduling problems, operations may be considered the game pieces. Depending on the exact problem formulation, each operation needs to be processed on a specific machine for a specific duration. An operation could therefore be described as a game piece which can be freely parameterized by properties such as the duration, contrary to pieces in typical games.

Cloud & edge computing As before in the production domain, a primary application in cloud and edge computing concerns scheduling problems (see Table 8). Again, the scheduling problems are combinatorial optimization problems, but the authors' interests arise from the domain of cloud computing itself and the presented problems are less abstract.

Graph navigation Navigating a graph from a given node to a target node is a relevant task in many settings, but is gaining attention, particularly in knowledge graph research. Here, a common task is knowledge graph completion, which

Table 9 Applications on graphs

Source	Application
[40]	Knowledge Graph Navigation
[95]	(Knowledge) Graph Navigation
[123]	Graph Navigation
[128]	Knowledge Graph Completion

Table 10 Applications in networking and communications

Source	Application
[30]	Service Function Chain Deployment Problem in Network Function Virtualization
[32]	Intrusion Defense in Software Defined Networking
[57]	Network Function Virtualization Mapping and Scheduling
[118]	Virtual Network Embedding
[127]	Resource Assignment in OpenRAN Networks
[69]	Layout Optimization of Mobile Ad Hoc Networks
[120]	
[144]	
[159]	
[160]	
[146]	
[10]	Dynamic Spectrum Sharing of LTE and NR
[87]	Radio Resource Scheduling
[92]	Wireless Communication Scheduling
[142]	Multi-RAT Access
[71]	MIMO Detection
[13]	Pilot-power Allocation for MIMO Systems

involves the prediction of missing relations between individual entities [128]. Graph navigation is an important sub-task of knowledge graph completion [95], for which neural MCTS has been investigated (see Table 9) and been shown to outperform existing baselines [95].

Networking & communications Applications in networking and communications (see Table 10) range from network function virtualization [30, 57], to network topology optimization [69, 120, 144, 146, 159, 160], to spectrum sharing in mobile networks with multiple radio access technologies [10, 142].

One notable example here is the use of neural MCTS for intrusion defense in software-defined networking scenarios [32]. Here, the defense problem is actually modelled as a two-player game, which is an exception among mostly single player scenarios within the surveyed literature.

Autonomous driving & motion planning Autonomous driving applications make up a comparatively large group (Table 11), including general motion planning tasks [39, 61, 76, 131], motion planning tasks in autonomous parking scenarios [104, 150], and motion planning tasks in multi-agent settings [82, 103]. More specialised tasks such as lane keeping [56], overtaking [70], and higher-level decision making during autonomous driving [44] are considered as well.

Such problems are often fundamentally different from combinatorial games. For instance, Weingertner et al. [131] consider a motion planning problem, in which the acceleration of a vehicle is controlled along a predetermined

Table 11 Applications in autonomous driving, as well as path and motion planning

Source	Application
[104]	Motion Planning in Autonomous Parking
[150]	
[39]	Motion Planning in Autonomous Driving
[61]	
[76]	
[131]	
[14]	Driving Maneuver Prediction
[44]	Tactical Decision Making
[56]	Lane Keeping Tasks
[70]	Overtaking Tasks
[82]	Multi-Robot Motion and Path Planning
[103]	Multi-Vehicle Motion and Path Planning

path. In its natural formulation, such a problem requires selecting continuous actions in continuous time. To apply neural MCTS, both time and action space are discretized. The resulting solution demonstrates good performance and outperforms A* search, pure deep learning approaches, and model predictive control.

Natural language processing In Table 12, several applications of conversational agents are shown, in which agents assist users in completing tasks [125], negotiate with users to divide a given set of resources [49], and try to convince users of a certain view by framing messages in different ways [9]. While humans are difficult to simulate as conversational partners explicitly, models that approximate narrow conversational behaviour of humans can be trained on historical data and then utilized as part of the tree search [9, 125].

Natural language processing is itself a diverse field, in which topics such as sentiment analysis [20] and named entity recognition [59] are being addressed with neural MCTS.

Machine learning MCTS guided by machine learning models can in turn be used in certain machine learning tasks (see Table 13). For instance [47, 129] apply neural MCTS

Table 12 Applications in Natural Language Processing

Source	Application
[9]	Personalized Messaging
[49]	Negotiation Dialogues
[125]	Task-Completion Dialogues
[20]	Sentiment Analysis
[42]	Text Matching
[59]	Named Entity Recognition

Table 13 Applications in machine learning

Source	Application
[47]	Neural Network Distillation
[129]	CNN Filter Pruning
[17]	Costly Feature Classification
[64]	Symbolic Regression
[158]	Recommender Systems

to reduce the size of neural networks by network distillation in the former and convolutional neural network (CNN) filter pruning in the latter case.

Lu et al. [64] further approach the task of symbolic regression with neural MCTS. Here, instead of solving regression tasks by adjusting the coefficients of a e.g. a linear or polynomial function, the terms of a function themselves (e.g. sinusoids, square operations, constants) are determined and connected through mathematical operators such as addition and divison. In MCTS, the full expression of a function can be built up step by step.

Computer science Computer Science offers a wide range of opportunities for the application of neural MCTS (see Table 14), many of which are presented in separate sections. Others do not warrant their own section due the small number of publications in their specific niche, but are nevertheless interesting. The number of publications this applies to demonstrates the wide applicability of neural MCTS.

One notable example is AlphaTensor [26], where neural MCTS is used to find efficient algorithms for matrix multiplication. Others include the optimization of database queries [151], the recovery of sparse signals [18, 155], and various applications in quantum computing [16, 22, 102].

Table 14 Applications in computer science

Source	Application
[26]	Matrix Multiplication Algorithm Discovery
[94]	Sequence Discovery
[140]	Model Checking
[151]	Database Query Optimization
[153]	Low-Density Parity-Check Code Construction
[18]	Sparse Signal Recovery
[155]	
[35]	Automatic Theorem Proving
[156]	
[157]	
[16]	Quantum Annealing Schedule Optimization
[22]	Quantum Dynamics Optimization
[102]	Qubit Routing

Table 15 Applications in various other fields

Source	Application
[143]	Active Space Multi-Debris Removal
[154]	Configuration of Cellular Satellites
[31]	Cognitive Radar Task Scheduling
[93]	
[5]	Object Rearrangement
[119]	Scene Arrangement Planning
[98]	User Interface Optimization
[113]	
[84]	Radiotherapy Beam Orientation Selection
[85]	
[3]	Fluid Structure Topology Optimization
[80]	Truss Design
[21]	Mobile Crowdsensing
[27]	Electromagnetic Situation Analysis
[28]	Robotic Manipulation
[33]	Wildfire Spread Prediction
[55]	Pneumatic Actuator Control
[62]	Document Style Reverse Engineering
[124]	Railway Timetable Rescheduling

Finally, Table 15 shows applications in various other fields that do not fit into any of the previous categories. These feature a diverse set of problems including the optimization of user interfaces [98, 113], control of a pneumatic actuator [55], as well as design tasks for trusses [80] and fluid structures [3]. As in many examples here, similar design tasks are also being approached with traditional RL [29]. In future studies, direct comparisons of traditional RL and neural MCTS methods on specific problems may help to decide what approach is preferable under which conditions.

In summary, the applications described in the above sections originate in a variety of different disciplines including chemistry, medicine, computer science, mathematics, and electrical engineering. The types of problems include optimization tasks of various kinds, control problems, generative design tasks, and many others. Clearly, neural MCTS shows wide applicability beyond combinatorial games, to problems which in part share and do not share the properties of games.

4.2 Application characteristics

Like games, the problems surveyed here can be formulated as a MDP. Playing combinatorial games involves choosing discrete actions at discrete time-steps in a finite horizon setting, i.e. games are episodic with well-defined terminal conditions. Rewards are typically sparse and correspond to a small set of possible game outcomes: loss (-1), draw (0), and win (1). While the state transitions of each individual player are typically deterministic, the presence of a second player introduces uncertainty about the states which will be encountered at the next turn. While many of the applications surveyed here share many of these properties, neural MCTS is also applied to applications which differ from combinatorial games in one or multiple dimensions.

Time Many settings do not have a turn-based nature, but allow for the execution of actions at arbitrary points in time, i.e. time often has a continuous nature. This does not appear to hinder the application of neural MCTS, as many authors simply discretize the time dimension in their problem formulation [5, 28, 33, 39, 47, 55, 56, 59, 76, 104, 131, 154].

Finite & infinite horizons Like combinatorial games, most of the applications surveyed here consist of a finite horizon problem, i.e. the problem is solved in episodes of finite length. In some cases, the natural formulation of the problem features an infinite horizon. To apply neural MCTS, episodes can then be created artificially by setting a maximum number of steps after which the episode always terminates, as is done in [58, 96, 113].

Transitions Many of the surveyed problems are of a completely deterministic nature, which is a fundamental difference compared to the combinatorial games domain. In such cases, the tree search may be modified to take advantage of the deterministic transitions (see Section 5.3 for more details).

Nevertheless, some problem formulations with stochastic state transitions can be observed [32, 38, 58, 69, 92, 113, 125, 142].

Rewards The reward structure of typical problem settings often does not share the simplicity of the reward function present in games. Instead of a set with two or three distinct reward values, rewards are typically given on a continuum corresponding to the quality of the obtained solutions. Often, the rewards are not even clearly bounded on one or both sides (see e.g. [5, 45, 67, 90, 96, 125, 147, 154]).

In some cases, the reward is transformed by self-play inspired mechanisms. This will be investigated in more detail in Section 5.2.

While the majority of surveyed problems feature some kind of sparse reward at the end of an episode, in some cases, more fine-grained rewards after each action are incorporated into the tree search [26, 44, 105].

Action spaces MCTS naturally lends itself well to discrete action spaces, as is the case in combinatorial board games. While modifications of (neural) MCTS for continuous action spaces exist [72], the vast majority of applications surveyed here exhibit discrete action spaces. Notable exceptions are the approaches of Lei et al. [61] and Paxton et al. [76].

Further, Raina et al. [80] apply a hierarchical reinforcement learning approach, in which neural MCTS is used for an overarching set of discrete actions while subsequent, continuous actions are determined by another mechanism.

Finally, it is always possible to discretize a naturally continuous action space. While this reduces the amount of precision with which actions can be chosen, some applications can nevertheless be successfully approached in this manner [26, 96].

State spaces While the exact characteristics of state spaces depend not only on the underlying problem, but also on how the problem is modelled, games such as Go have a well-defined, regular board, which is helpful in formulating a state space. In Go, the board of fixed size consisting of cells which are positioned in spatial relation to each other lends itself well to processing by a CNN. The problems surveyed here feature a diverse range of state spaces which are processed by different kinds of neural networks. Next to CNNs, the employed neural networks include recursive neural networks such as long short-term memory networks [5, 25, 28, 35, 40, 42, 46, 96, 113, 119] and gated recurrent units [20, 33, 49, 59, 65, 106, 110, 122, 123, 141, 145, 158], as well as graph neural networks [48, 52, 61, 63, 75, 77, 102, 105, 109, 123, 136, 137, 141, 157]. Less frequent types include transformers [26, 78, 87] and the DeepSet architecture [146]. In many cases, a simple multi-layer perceptron is sufficient [10–13, 16, 22, 32, 36–38, 41, 45, 55, 56, 58, 76, 81, 90, 91, 97, 112, 125, 126, 131, 140, 150, 151, 155].

The chosen architecture and its depth will to some degree determine what kind of hardware is required to train a neural MCTS approach.

4.3 Hardware requirements

The training of AlphaZero involved more than 5000 tensor processing units [100]. One might hence question whether the application of neural MCTS is a viable option for researchers and practitioners who do not have access to resources of that magnitude.

Some of the publications we review do report the usage of significant resources. For example, Huang et al. [46] use up to 300 NVIDIA 1080Ti and 2080Ti GPUs during training. The majority of the reported hardware, however, is not out of reach for typical organizations and even private individuals. Genheden et al. [36] report that they use a single Intel Xeon CPU with a single NVIDIA 2080Ti GPU on a machine with 64 GB memory. Many others use a single high-end consumer CPU and GPU [57, 58, 84, 94, 155].

On the lower end, some researchers even use consumer notebooks to train neural MCTS methods [45, 141, 143].

Clearly, hardware requirements vary by application and the complexity of the employed neural networks. While it is difficult to predict what level of hardware is required for a given application and desired solution quality, it is clear that moderately powerful hardware can be successfully utilized in many applications.

5 Neural MCTS methodologies

After gaining an overview of the breadth of possible neural MCTS applications in the previous section, we now turn our attention to the design of neural MCTS approaches as they were encountered during the review.

5.1 Guidance network training

Before delving into the inner mechanisms of neural MCTS, we content ourselves with the knowledge that learned policy and value networks are used to guide the tree search in some way. In the following, we first dedicate some attention to the training procedure used in AlphaZero [100], before discussing alternatives found during the review.

Policy improvement by MCTS In AlphaZero [100], a learned policy is iteratively improved by guiding an MCTS search and in turn using the search results to improve the learned policy (see Fig. 3). We refer to this procedure as *policy improvement by MCTS*. More concretely, the learned policy π_θ guides the tree search in one or multiple of the search phases and a new policy π_{MCTS} for the state under consideration is obtained after a given number of MCTS simulations. Since this new policy is typically stronger than the initial,

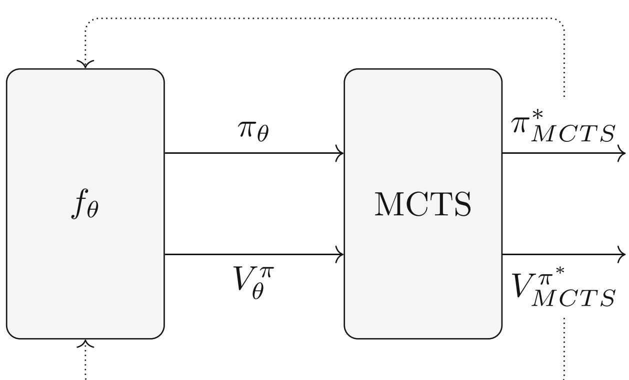

Fig. 3 MCTS as a policy improvement operator. The learned policy and value function are used to guide the tree search, which then produces an improved policy π^*_{MCTS} and value estimate $V^{\pi^*}_{MCTS}$ for a given state. As visualized by the dotted lines, π^*_{MCTS} and $V^{\pi^*}_{MCTS}$ can then also be used as training targets for the neural network

learned one, it can be used as a training target for the policy network. More precisely, the policy network and MCTS produce policy probability vectors $\mathbf{p}_\theta$ and $\mathbf{p}_{MCTS}$ for a given state, where the former can be seen as the actual prediction and the latter as the prediction target. These can then be used in a cross-entropy loss function to train the policy network: $L_{CE} = -\,\mathbf{p}^T_{MCTS}\,\log\,\mathbf{p}_\theta$. Accordingly, if a value function is learned alongside the policy, its value estimates are adjusted in the direction of those found by MCTS by using the mean squared error (MSE) as a loss function: $L_{MSE} = (v_{MCTS} - v)^2$. Both terms are typically combined with a regularization term into a single loss function

$$L = (v_{MCTS} - v)^2 - \,\mathbf{p}^T_{MCTS}\,\log\,\mathbf{p}_\theta + c\,\|\theta\|^2 \tag{3}$$

The vast majority of the articles we reviewed that improve a learned policy by MCTS use the loss function given in (3). We find two exceptions to this, one in which the Kullback-Leibler divergence is used instead of the cross-entropy [17] and one in which the Kullback-Leibler divergence is also used instead of the cross-entropy, but a quantile regression distributional loss is additionally used instead of the MSE [26]. It may be worth investigating what effect these loss functions have on training, but the combination of cross-entropy and MSE clearly emerges as the default choice during our review.

Alternative training approaches We find two broad groups of training approaches during our review: (1) training before the networks are used to guide the tree search and (2) training during the tree search, i.e. iteratively performing tree search and using its results for training. In group (1), training is facilitated either by supervised learning on labelled examples or by classical reinforcement learning algorithms in the policy-based, value-based, and actor-critic families. In group (2), the dominant approach is policy improvement by MCTS, but some variations on this exist. Finally, both approaches can be combined by first performing what can be considered a pre-training and then training further during the tree search. This is sometimes referred to as a warm-start.

Figure 4 shows the distribution of these different approaches as found in the surveyed publications and which specific methods are employed in each approach. A large portion of authors train their networks during the search by using policy improvement by MCTS. In some cases, training during the search occurs by other methods such as Q-Learning [1, 37, 95, 123] and Maximum Entropy RL [157] on the MCTS trajectories. Instead of improving the policy (alongside the value function) by MCTS, it is also feasible to refine the value function individually [62, 137], without any learned policy. We term this approach *value function refinement by MCTS*

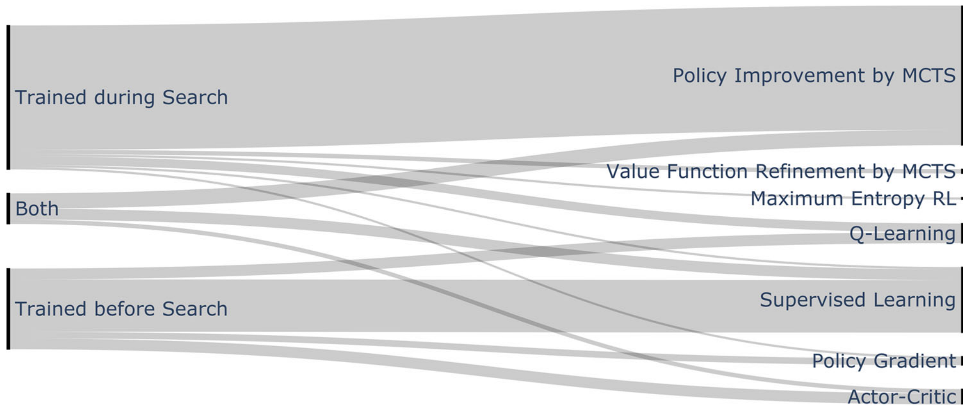

Fig. 4 Depending on the approach, policy and value networks are trained before the search, during the search, or both. Depending on this choice, different training methods are employed. Policy gradient and Actor-Critic are families of algorithms that encompass multiple specific algorithms

here. Likewise, while policy improvement by MCTS typically trains both a policy and a value function, sometimes the policy is also trained in isolation [93].

When trained before the tree search, networks are most often trained by supervised learning on labelled demonstrations. Q-Learning [75, 76, 119, 125, 131], policy gradient [24, 45, 55, 57, 64, 77] and actor-critic methods [17, 41, 121, 129] are also employed.

When both training phases are performed, the most common approach is to combine supervised pre-training with policy improvement by MCTS [16, 81, 107, 126, 154].

Overall, there is some variety in the employed training approaches, but the dominant strategies are supervised training before the search and policy improvement by MCTS, sometimes combined in one approach as in the original AlphaGo publication [99]. In a combinatorial game setting, policy improvement by MCTS requires some mechanism by which the opponent's moves are generated. In the following, such a mechanism and its relevance for applications beyond games are discussed.

5.2 Self-play beyond games

One of the components leading to the success of AlphaGo is the concept of self-play [99]. In a self-play setting, opponents in a multi-player game are controlled by (some version of) the same policy, i.e. the policy plays against itself [43]. Learning in such a scenario has the advantage that the policy always faces an opponent of comparable skill, which evolves as the training progresses. However, since many non-game-playing applications have an inherently single-player nature, the role of self-play beyond games is not obvious.

To gain a clearer understanding of the applicability of self-play in such cases, we surveyed its usage among the publications included in our review. During this process, it became apparent that many authors use the term self-play, but that the meaning of the term varies. This may be due to the lack of an accepted, standardized definition. In the following, we first delineate different meanings of the term we encountered and then report the usages of different versions of self-play in our review.

In a two-player setting, the term self-play can be understood intuitively. As used in the original AlphaGo publication [99], self-play entails that the policy currently being learned plays a game against some older version of itself in a two-player turn-based setting. This means that this other version of the current policy is being used to generate new states by playing every other turn of the game, as well as to obtain the final reward of the game.

In single-player settings, the term self-play is also often used, but its meaning is less obvious. In a single-player setting, the *state generating* property described above is not applicable, since the state transitions do not depend on the actions of another player. Generating a new state requires only the current state and the agent's action. The *reward generating* property described above, however, is applicable if the reward function is designed accordingly. If the reward is not simply dependent on the performance of the current policy, but on the relative performance compared to some prior version of the policy, the reward generating property of self-play is transferred to the single-player setting. In other words, the process of a policy trying to beat its own high score has similarities to the concept of self-play in two-player settings. While this is sometimes also called self-play, Mandhane et al. [66] introduce the term *self-competition* for this type of approach. In the remainder of this review, we will adopt this term and reserve *self-play* for multi-player settings to avoid confusion. While simple versions of self-competition can be implemented trivially, Laterre et al. [60] introduced a more substantiated form of self-competition named *ranked reward*, followed by the approaches of Mandhane et al. [66] and Schmidt et al. [88].

However, many authors claim to implement self-play without obviously applying any of the concepts described above (see e.g. [80, 111]). While it is hard to be certain about what is meant in such instances, we suspect that two further concepts are sometimes termed self-play in the literature. The first is the practice of keeping track of the best policy by evaluating the current policy against the previous best one. If the current policy can outperform the previous best one on some defined set of problems, it replaces the currently saved best policy. This simply serves the purpose of having access to the best policy after training completes since training does not necessarily improve the policy monotonically. In such cases, the outcomes of evaluation are not used as rewards to train the policy. Consequently, no learning follows from the policy playing against another version of itself, i.e. it is not a mechanism by which the current policy is improved, but merely evaluated.

The last concept, which we suspect is sometimes described as self-play is policy improvement by MCTS as introduced above (see e.g. [12]).

To be clear, we will use the term self-play only to describe multi-player cases where the state generating property as well as the reward generating property hold, and the term self-competition only for single-player cases where the reward generating property holds.

While we would ideally like to report the usage of self-play and self-competition for all publications included in our review, we refrain from doing so when the terms are used ambiguously and instead only report a selection of notable examples where their meaning has been clearly established.

Self-play Actual self-play appears to be fairly rare in the non-game-playing literature. We can only attribute the use of self-play to a single work [27], in which a problem is modelled as a two-player game and a policy learned by self-play. In some other cases, problems are modelled as two-player games as well, but the resulting games are asymmetric, i.e. the players have different action spaces and hence require different policies [32, 138–141]. In such cases, two different neural networks each learn a policy.

Self-competition In terms of self-competition, we observe instances of ranked reward [109, 123, 127] as well as naive approaches (see Table 16). In a naive approach, the performance of the current policy on the current problem is simply evaluated as some score and compared against the score of the best policy observed up to this point on the same problem. If the current policy's score is better, the game is won ($r = 1$), if it is worse, the game is lost ($r = -1$), and if it is equivalent, the outcome is a draw ($r = 0$) [46, 52]. A variation of this is to not use the best policy, but to evaluate against the average score of a group of saved policies [122].

Table 16 Self-competition

Type	Publications
Ranked reward	[109, 123, 127]
Naive (best)	[46, 52]
Naive (average)	[122]

In one case, a naive approach, as described above, is applied, but instead of a past version of the policy, a second, completely independent policy is learned and the two policies continually compete against each other [3].

While self-competition can be used to generate rewards based on the relative performance of the policy, this is not strictly necessary, as the absolute performance can be used to compute rewards just as well. One benefit of self-competition may simply be having a reward in a clearly defined range of $[-1, 1]$ or similar, as optimal choices of MCTS hyperparameters depend on this range [8]. However, there appear to be benefits beyond this, as the ranked reward approach has been shown to outperform agents trained using a standard reward in the range $[0, 1]$ [60]. Whether this is the case for the naive self-competition approaches as well is unclear.

5.3 Guided selection

The previous sections argue that MCTS functions as a policy improvement operator. We now explore the mechanisms of this policy improvement, i.e. the inner workings of neural MCTS. A search iteration in MCTS begins with the selection phase, in which actions are iteratively chosen starting from the root state until a leaf node is encountered. As described in Section 2.3, the choice of action is determined by a tree policy, which generally takes the form

$$a = \operatorname*{argmax}_a Q(s, a) + U(s, a) \tag{4}$$

where $Q(s, a)$ encourages exploitation of known high-value actions, while $U(s, a)$ encourages exploration of the search tree. Variations exist both in the exact formulation of (4) and in how individual terms of the equation are determined, i.e. by learned policies and value functions or by conventional means. We investigate each aspect individually in the following.

Tree policy formulations The tree policies encountered during the review are usually based on some version or extension of the UCT rule, but some variation in the exact formulation of the rule, especially in the exploration part, can be observed.

We provide an overview of variations of $U(s, a)$ identified during our review in Table 17. While compiling the table, we modified the exact formulations reported in individual publications to arrive at a consistent notation. To this end, we assumed that all reported logarithms are natural logarithms and that $N(s) = \sum_b N(s, b)$, i.e. $N(s)$ refers to the visit count of all the children in state s, while $N(s, a)$ refers to the visit count of action a in state s. The exploration constant,

Table 17 Variations of tree policies based on UCT

Frequency	Name	Variant	Formula
27	UCT	0	$c\sqrt{\frac{2\, lnN(s)}{N(s,a)}}$
2		1	$c\sqrt{\frac{lnN(s)}{1+N(s,a)}}$
1		2	$c\,\frac{\sqrt{N(s)}}{1+N(s,a)}$
39	$PUCT$	0	$c\,P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)}$
5		1	$c\,P(s,a)\sqrt{\frac{2\, lnN(s)}{N(s,a)}}$
3		2	$c\,P(s,a)\frac{\sqrt{N(s)}}{N(s,a)}$
3		3	$c\,P(s,a)^{\mu}\sqrt{\frac{N(s)}{1+N(s,a)}}$
2		4	$c\,\frac{P(s,a)}{1+N(s,a)}$
2		5	$c\,P(s,a)\frac{\sqrt{N(s)+1}}{N(s,a)+1}$
2		6	$c\,P(s,a)\frac{N(s)}{1+N(s,a)}$
1		7	$c\,P(s,a)\frac{\sqrt{N(s)+\epsilon}}{N(s,a)+1}$
1		8	$c\,P(s,a)\sqrt{\frac{lnN(s)+1}{1+N(s,a)}}$
1		9	$c\,P(s,a)\sqrt{\frac{N(s)}{1+N(s,a)}}$
1		10	$c\,P(s,a)\sqrt{\frac{lnN(s)}{1+N(s,a)}}$
2	$MuZero$	0	$P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)}\left[c_1 + ln\ \frac{N(s)+c_2+1}{c_2}\right]$
1		1	$P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)}\left[c_1 + ln\ \frac{\sqrt{N(s)+c_2+1}}{c_2}\right]$
1	UCT_D	0	$c\,A(s,a)\sqrt{\frac{2\ ln\ N(s)}{N(s,a)}}$
1	$PUCT_B$	0	$c_1\ P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)} + c_2 B(s,a)$

Different groups of selection formulae are each indiciated with a name and each member of a group additionally has a variant number. The first member of a group is always the most frequently observed one, not necessarily the original formulation of the group

sometimes given as c_{uct}, c_{puct} or similar, is simply referred to as c in this review. $P(s, a)$ represents some prior probability of choosing action a in state s, whether it be given by a learned policy or obtained by other means.

Among the surveyed publications, a large proportion still use (some variant of) the UCT formula (see Table 17), but PUCT as it is used in AlphaZero [100] (PUCT variant 0 in Table 17) is the most frequently used selection mechanism. There are a number of less frequently used PUCT variations mostly concerning the presence of logarithms, constant factors in the numerator and denominator, and scope of the square root. These differences impact the overall magnitude of the exploration term as well as its decay as individual actions are visited more often (see Fig. 5). It is difficult to judge the impact of different formulations on the search, since authors usually do not directly compare them. In a rare exception, Xu and Lieberherr [138] try both the AlphaZero PUCT variant as well as PUCT variant 9 in Table 17 and report that the AlphaZero variant performs much better, although they do not quantify this difference.

One notable PUCT variant, variant 3, introduces a new constant μ which determines the impact of the prior probabilities as $P(s, a)^{\mu}$. This variant seems to have been independently suggested in [95, 104, 123].

Aside from UCT and PUCT variants, the MuZero [89] selection formula or a variant of it is used by three authors. We further find two unique modifications of typical selection formulae that we cannot assign to any of the other groups: UCT_D and $PUCT_B$. The former will be discussed at a later point. $PUCT_B$ aims to exploit the nature of deterministic single-player settings, in which future trajectories are not influenced by the choices of another player. In such cases, rather than simply looking at average state values, it may be advantageous to keep track of the best encountered values

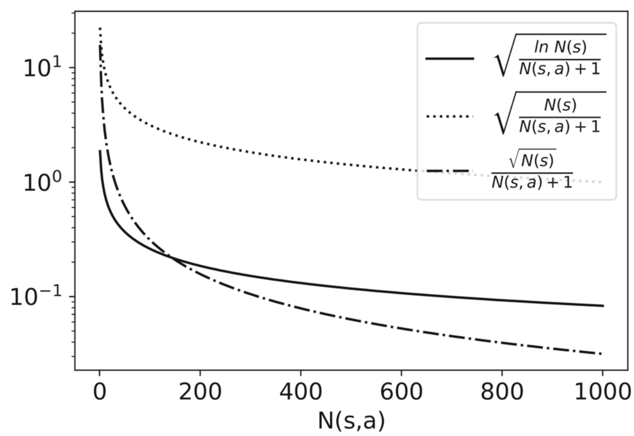

Fig. 5 Different UCT-style fractions with a fixed $N(s) = 1000$. Note that the vertical axis is logarithmic and shows the value the expressions in the legend produce for different $N(s, a)$

during the search. Making decisions based on average values is problematic because most of the actions in a given state may be bad choices, while one specific single action may be a good choice. On average, the value of the node will then be low, even though a promising child exists. In a deterministic setting, the best path can be executed reliably and, accordingly, it makes sense to choose nodes based on their expected best value rather than the average one. Deng et al. [23] design a selection formula that makes use of this fact, which we refer to as $PUCT_B$ here. In $PUCT_B$, the best value of an action is simply scaled by a constant and then added to PUCT variant 0.

Neural guidance in the tree policy Neural guidance may be used in both the exploitation and the exploration part of (4) (see Fig. 6). When used in the exploitation part, neural guidance is typically used to estimate $Q(s, a)$. This does not change how the selection mechanism works, only how the corresponding value is determined. Since value estimation is a function of the evaluation step, this kind of neural guidance

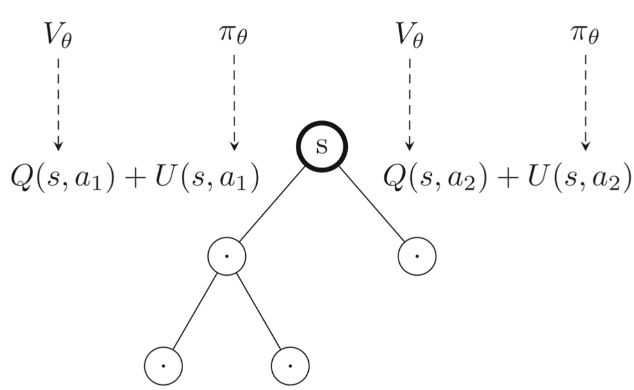

Fig. 6 Neural Selection. Each of the children of the current state s are considered and the one maximizing $Q(s, a) + U(s, a)$ is chosen. Both $Q(s, a)$ and $U(s, a)$ may be influenced by neural guidance in some way

Fig. 7 Proportion of choices in the selection phase among the surveyed articles. *Standard* refers to some selection strategy that does not involve the use of learned functions

will be explored in Section 5.5 and not discussed further at this point.

In the exploration part of (4), neural guidance is typically used to determine the prior probabilities $P(s, a)$ in PUCT-style formulae. As shown in Fig. 7, about 62% of all reviewed articles report guiding the tree search in this way, with less than 30% reporting selection phases without neural guidance. The remaining articles do not report how the selection phase is performed at all. While the latter can probably be interpreted as selection without neural guidance, we try to refrain from interpretations as much as possible and hence give separate categories for standard selection and unreported selection.

Most approaches for neurally guided selection phases take the form described above, with the exception of a few special cases. Zombori et al. [157] argue that a learned policy network tends to make predictions with high confidence even if they are of low quality, which leads to a strong unfounded bias in the search. It may be more desirable to have a policy which makes less confident predictions if the prediction quality is not sufficiently high. To achieve this, they use maximum entropy reinforcement learning and use the resulting policy to compute prior probabilities for the selection phase.

In one exception, neural guidance occurs in a form other than providing prior probabilities, as can be seen in the UCT_D formula in Table 17. It is named after its use of a dueling network, which produces action advantage estimates $A(s, a)$ in addition to state values. In UCT_D, the action advantages are used in place of the prior probabilities $P(s, a)$. Vaguely related to the reasoning of Zombori et al. the authors argue that a policy network trained with policy gradient methods tends to concentrate on the best action for a given state, while not assigning probabilities proportional to the expected usefulness of the other actions [125]. In other words, an overly low entropy policy vector may bias the search to an undesirable degree. In contrast, the action advantages do not overly focus on the best action.

5.4 Guided expansion

Once a leaf node s_L is encountered during the selection phase in MCTS, the expansion step is performed to create child nodes of s_L. Guidance by neural networks can be employed in this step as well to bias and hence speed up the search. To avoid confusion, we will first give some details on possible alternate ways to implement the expansion step and only then return to the topic of neural guidance.

As discussed in Section 2.3, nodes are typically expanded either one at a time whenever an expandable node s_E is encountered, or all children of a node are expanded simultaneously if a leaf node s_L is encountered.

Clearly, implementing an MCTS approach requires deciding how many children are expanded at a given time. There is, however, an additional, related decision to be made: How many and which children are *considered* for expansion? In the naive case, the search is free to choose any action from the set of all possible actions $A(s_L)$ in state s_L. However, it is also possible to be more selective in the expansion step. To limit the growth of the tree, only a limited number of children may be considered for expansion either randomly or according to some rule or heuristic. In other words, the search may be restricted to only choose actions from a set $\tilde{A}(s_L) \subset A(s_L)$. This is especially relevant for continuous cases, where the number of potential children is infinite and necessarily has to be limited in some way. Once such a set $\tilde{A}(s_L)$ has been defined, the corresponding child nodes may be expanded all at once when the leaf is encountered or one by one, whenever the expendable node is encountered during the tree search.

Neural guidance during the expansion step is possible in both paradigms, i.e. when expanding on encountering a true leaf node and when expanding on encountering an expandable node. In the former case, neural guidance means using a learned policy to determine $\tilde{A}(s_L)$, while in the latter case, neural guidance means choosing an action in $A(s_E)$ to create a new child node. Theoretically, it is possible to combine both of these approaches by first determining and saving $\tilde{A}(s_L)$ when a leaf is encountered for the first time, but not expanding all corresponding nodes at this point. The children can then be expanded one by one whenever the node is encountered again by choosing some action $a \in \tilde{A}(s_E)$. However, we do not observe this combined approach in the collected literature.

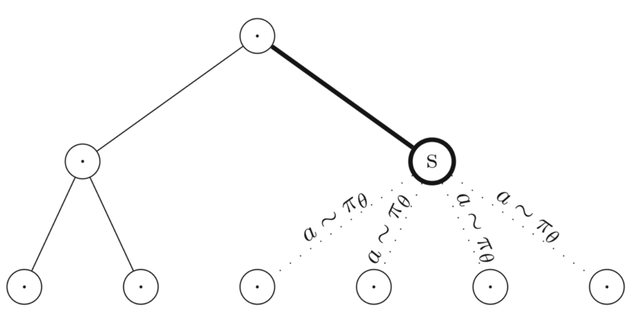

Fig. 8 Neural Expansion. When a leaf node is encountered, possible actions in the leaf node's state are sampled by some mechanism involving the learned policy. For every sampled action, a new child is created

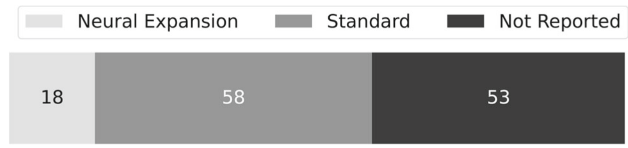

Fig. 9 Proportion of choices in the expansion phase among the surveyed articles. *Standard* refers to some form of expansion that does not involve the use of neural networks

We do observe some form of neurally guided expansion in a sizeable portion of publications (see Figs. 8 and 9) and categorize them in Table 18. In some additional examples, neurally guided expansion is used, but the specifics are not reported [25, 36, 90].

$\tilde{A}(s_L)$ can be determined by randomly sampling actions from a learned policy, but it can also be determined by enumerating all actions in the policy distribution and choosing the top k ones [48, 91, 112]. In the approach of Thakkar et al. [112], the top k actions with a cumulative policy probability of 0.995 or at most 50 actions are selected. In both types of neurally guided expansion, instead of a learned policy, a learned value function can of course be converted to a policy with a softmax operator, as is done in [119].

While the exact impact of neurally guided expansion will vary from application to application, its general potential is demonstrated in [82] who report that their computational time is 20 times reduced with neural expansion while achieving higher quality solutions.

5.5 Guided evaluation

The evaluation step in MCTS serves to estimate the (state-)value of a leaf node encountered during the tree search. While it is often also called the roll-out step or the simulation step, its purpose is the value estimation of a leaf. Roll-outs or simulations are simply approaches to produce a value estimate. Here, we use the term evaluation, because not all evaluation approaches in the neural MCTS literature are based on roll-outs.

There are two obvious ways to use learned policies and value functions during the evaluation phase: a roll-out using the learned policy and a direct prediction by a learned value function. In the former, actions are iteratively sampled from

Table 18 Types of neurally guided expansion

Type	Publications
Choosing $a \in A(s_E)$	[5, 24, 45, 64, 119, 143]
Determining $\tilde{A}(s_L)$	[26, 48, 61, 80, 82, 91, 112, 145, 148]

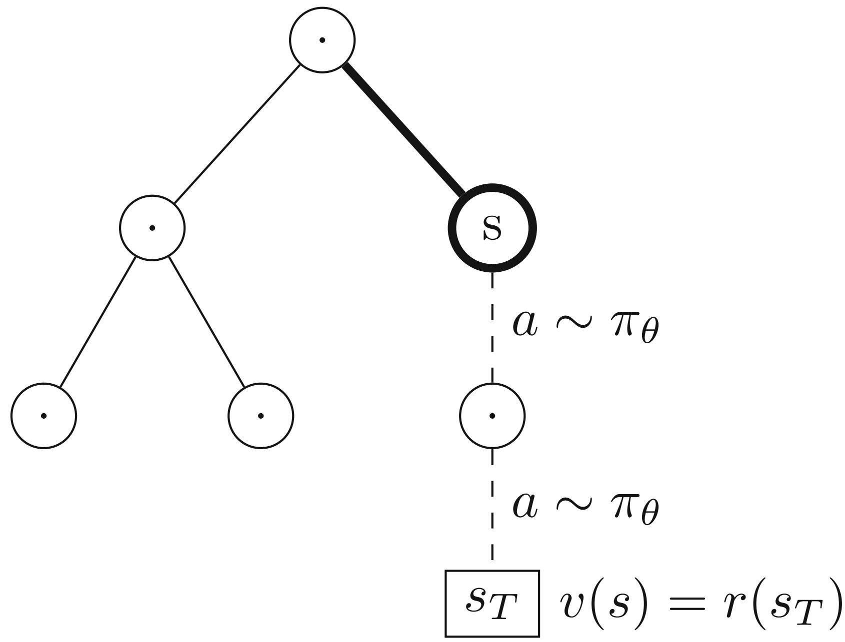

Fig. 10 Evaluation by learned policy roll-out: After arriving at leaf node with state s according to the tree policy, the value of s needs to be determined. Here, the learned policy π_θ is used to generate a roll-out by iteratively sampling actions until a terminal state s_T is reached. The reward of this terminal state serves as an estimate for the value of s

the learned policy starting from the encountered leaf node until a terminal node is reached (see Fig. 10), while in the latter, the value of the leaf node is simply predicted by the learned value function without any roll-out (see Fig. 11).

Most authors use either learned policy or value functions as described above, with 54 occurrences of learned value functions and 26 occurrences of learned policy functions (see Fig. 12) The remaining publications either do not use neural guidance for the evaluation phase or their approach is unclear.

Among the neural evaluation approaches, some authors employ different evaluation approaches depending on how far the training has progressed. Song et al. [104] combine both approaches by performing roll-outs according to a learned policy network in the early phases of training and use a learned value network for estimation in later stages. Zhang et al. [150] use an initial phase of random roll-outs to pretrain a policy network and employ the learned policy network for roll-outs in later stages.

In some cases, roll-outs are not performed by naively using a learned policy, but more complex roll-out procedures are still guided by learned functions. He et al. [40] use a value network to guide a problem-specific roll-out procedure, while Xing et al. [136] use a learned policy function to guide a beam search. Kumar et al. [58] combine a value estimate as predicted by a neural network with a domain-specific roll-out policy, motivated by the fact that their reward function is more fine-grained than those typically observed in board games. Finally, Lu et al. [64] use a value network in a symbolic regression task to estimate whether a leaf node merits further refinement by an optimization method, but their approach is highly problem-specific.

Deng et al. [23] perform different evaluation approaches depending on the depth of the node to be evaluated. If the node is closer to the root of the tree, they use a neural network to estimate the value of the state, while they perform a random

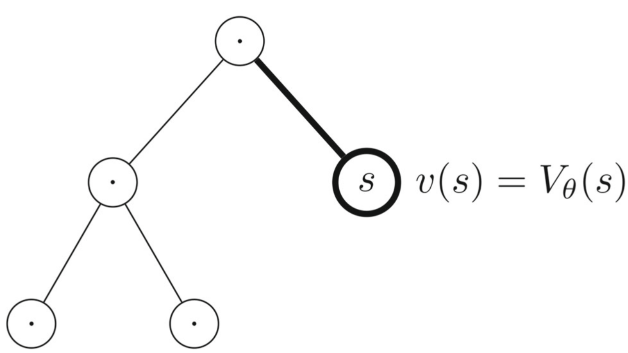

Fig. 11 Evaluation by learned value function: After arriving at leaf node with state s by following the tree policy, the value of s needs to be determined. Here, a learned value function V_θ is used to estimate the value of s directly, without any need for a roll-out

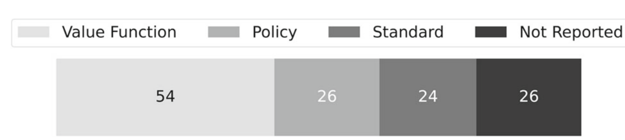

Fig. 12 Proportion of choices in the evaluation phase among the surveyed articles. *Value function* refers to an evaluation by a learned value function, *policy* to a roll-out using the learned policy and *standard* to a random roll-out

roll-out for nodes at deeper levels of the tree. Their experiments show that this hybrid approach can balance solution quality and computation time. Since neural inferences are associated with non-negligible computational cost, replacing them with random roll-outs can decrease the search time especially at deeper levels, where roll-outs will have short lengths.

In the application described by He et al. [41], episodes can result in either a successful or a failed solution. Successful solutions can still differ in quality and are rewarded accordingly. They perform a roll-out by a learned policy, but if this results in a failed terminal state, they backtrack until they find a successful solution. They argue that this leads to better search efficiency because preceding trajectories are not repeated unnecessarily.

Design choices in other MCTS phases can also influence what is required from the evaluation phase. Kovari et al. [56] replace the exploitation term, i.e. the estimated value, of a UCT-style formula with the probability of taking an action as predicted by the policy network. Instead of a roll-out or direct value prediction, they hence simply predict this probability in the evaluation step.

5.6 Guidance in multiple phases

As discussed above, neural guidance can be employed in the selection, expansion, and evaluation of phases of MCTS. Of course, it is not necessary to limit this guidance to one phase at a time and different types of neural guidance can be combined in one approach.

To gain an overview of how different types of neural guidance are typically combined, we visualize their use in Fig. 13.

The most common approach is to guide the selection step, perform standard MCTS expansion, and then use a learned value function for evaluation. Many other combinations exist, but none of them are used as often. When standard selection is used, the relative incidence of neural expansion is higher than when using neural selection. This could, however, be explained by the fact that the respective authors simply wanted to highlight the effect of neural expansion since it is often the main focus of their respective publications.

Infrequently occurring combinations may indicate a need for further research.

5.7 Use of dynamics models

Neural MCTS is a model-based reinforcement learning approach, i.e., it requires access to a dynamics model of the environment to perform planning. Typically this dynamics model is given [99, 101] and can be readily used in the tree search. In contrast to a regular reinforcement learning environment as defined in, e.g. the OpenAI Gym standard [7], a dynamics model allows for the computation of the next state and reward given an arbitrary initial state and action to be executed. An environment, on the other hand, is in a specific state at any given time which can be influenced by actions, but does not allow for dynamics computations on arbitrary states. In other words, an environment is stateful, while a dynamics model is not.

The difficulty of developing such a dynamics model will differ from application to application. In any case, its development will require some additional effort. To circumvent this, it is also possible to *learn* the dynamics model and then use this learned model for planning in MCTS [89, 115]. While this adds additional complexity to the training process, it can be helpful in scenarios where an exact and efficient model of the environment cannot be easily obtained.

During our review we found that the vast majority of approaches utilize an existing dynamics model, but learned models also find some application in practice.

For instance, Chen et al. [14] investigate an autonomous driving task where a model is needed to predict the vehicle state. In this case, the vehicle state is an image, meaning that the model needs to produce an output image given an input image (corresponding to the initial state) and an action. While such a model is not trivial to implement manually, Chen et al. [14] are able to train a convolutional neural network to serve this purpose.

Similarly, Challita et al. [10] apply neural MCTS to enable dynamic spectrum sharing between LTE and NR systems and report that this requires a model of individual schedulers for LTE and NR, which is not trivial to design. Instead, they learn the model in an approach similar to the one proposed in MuZero [89]. That is, the dynamics are not computed on the raw observations, but on hidden representations, which are computed from the observations by a learned representation function. This approach is also taken by others [32, 96], but dynamics models which work directly on the observations can also be observed [21, 125].

In many cases, the dynamics model is not learned during neural MCTS training, but trained separately in advance and then simply used for inference during the tree search [9, 28, 39, 122]

As described above, one motivation to learn a dynamics model may be the difficulty of creating one manually. Another motivation is the speed with which a learned model can be evaluated [109].

In some cases, the state transitions can be modelled fairly easily, while the computation of the reward is time-consuming. Some authors do not train a full dynamics model, but a scoring model, which can be used to assess the quality of a given solution quickly. In contrast to a learned value function, which can be used to evaluate newly expanded nodes at arbitrary depths, a scoring model only assigns a score to full

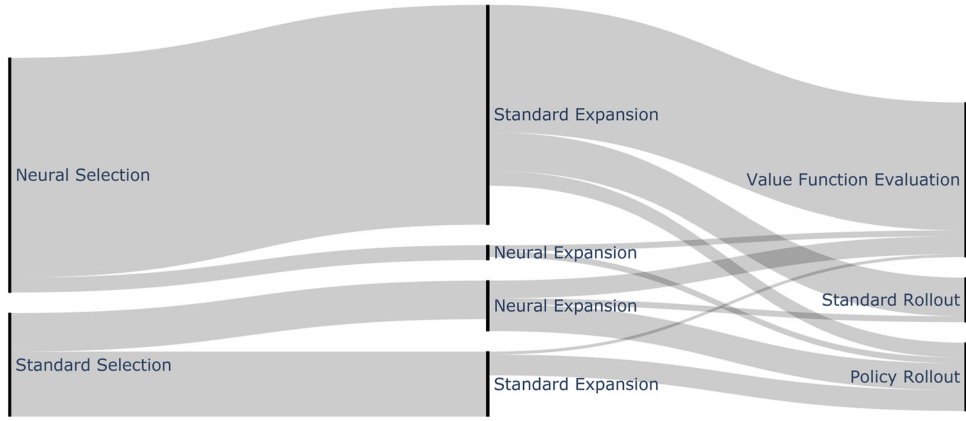

Fig. 13 Parallel sets diagram of surveyed neural guidance configurations. Each vertical bar signifies an option in one of the MCTS phases: selection (left), expansion (middle), evaluation (right). In the expansion step, each option (standard and neural expansion) is displayed twice to allow for easier tracing of the visualized configuration

solutions, i.e. terminal nodes. The resulting scores can then be used as training targets for the value function [42, 106, 151].

5.8 MCTS modifications

We now turn our attention to selected modifications of typical (neural) MCTS procedures as encountered during the review.

Average and best values As briefly mentioned in Section 5.3, deterministic single-player settings pose different requirements than combinatorial games. Action selection based on the average value of a node will lead to sub-optimal results, because a strong child node can be surrounded by weak siblings. While the $PUCT_B$ mechanism described in Section 5.3 is one option to address this, other authors have identified this issue as well and proposed their own solutions.

Deng et al. [23] report that their final search results are usually worse than the best solutions found during roll-outs in empirical experiments. They point out that the final action selection after tree search is performed based on the node visit counts $N(s)$. To rectify the problem, they introduce an oversampling mechanism for good solutions. Whenever a solution is found which outperforms all previously found solutions during a roll-out, this solution will be given preference in subsequent selection phases for a certain amount of time, and will hence be visited more often.

A simpler approach is taken by Peng et al. [77] and Xing et al. [137] to address the same problem. Here, the exploitation part of (4), i.e. the average value of the node, is simply replaced with the best observed value for the node. Fawzi et al. [26] follow a similar strategy.

Zhang et al. [154] simply keep track of the maximum reward encountered in the search and the action sequence that lead to it, which is then returned after the search.

Value normalization While combinatorial games lend themselves well to reward function formulations in the range $[-1, 1]$, in other applications, rewards are often less regular and sometimes completely unbounded. As mentioned earlier, the exploration constant c needs to be tuned for different reward ranges [8]. Further, even with a perfectly tuned c, rewards outside of ranges like $[-1, 1]$ or $[0, 1]$ are typically not conducive to algorithm convergence [97]. A number of authors therefore suggest normalizing Q-values according to the minimum and maximum values observed in the tree search until the current point [78, 97, 137].

The cost of neural inference The main idea behind neural MCTS approaches is to increase the efficiency of the tree search with neural inferences. While neural inferences are not too computationally expensive individually, when performed in large numbers, the required computational time can add up to significant amounts.

Deng et al. [23] vary the amount of neural inferences by switching neural guidance off during the search some proportion of the time. They find that neural guidance generally helps the search, but that further neural guidance after a certain point only increases computational cost without providing additional benefits.

Designing mechanisms to limit the application of neural guidance to where it provides maximum benefits in a targeted way may be an interesting line of research.

5.9 MCTS hyper-parameters

One last aspect in the design of neural MCTS approaches is the choice of appropriate hyper-parameters. While choosing hyper-parameters is highly problem-specific, it can nevertheless be useful to look at average hyper-parameter values to serve as a starting point and determine reasonable bounds for a problem-specific hyper-parameter optimization. To facilitate this, we summarize the values for the MCTS-specific hyper-parameters found during our review in Fig. 14. A large amount of variation in values exists for both the exploration constant c used in UCT-style selection formulae and the MCTS-budget n_{MCTS}, i.e. the number of simulations or play-outs performed during MCTS for a given time-step.

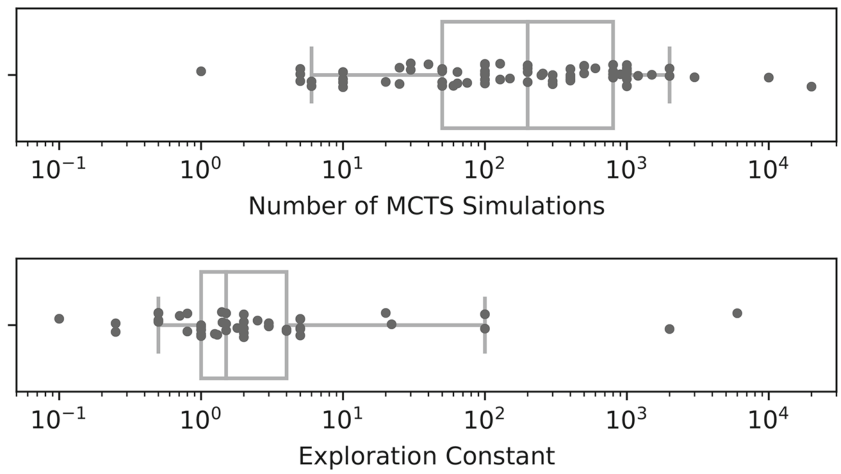

Fig. 14 Distribution of reported MCTS hyper-parameters: Number of MCTS simulations (top) and exploration constant c (bottom). Note that a logarithmic axis is used in both cases. Variation on the vertical axis is not meaningful, but contains random jitter for better visibility of individual data points

It is known that the optimal choice of c depends on the scale of the encountered rewards [8]. While reward scales vary wildly and are not always reported among the literature we survey, the most commonly reported reward scale is in the interval $[-1, 1]$, for which six different values of c ranging from 0.5 to 5.0 are chosen. In other words, the scale of the rewards does not appear to be the only criterion on which authors choose hyper-parameter values.

Instead of having a fixed value, in some cases, the exploration constant c and the MCTS budget n_{MCTS} are changed dynamically depending on circumstances.

Sometimes the exploration constant c is decreased as the training progresses [16, 157], presumably because later training iterations profit more from perfecting the current policy instead of performing further exploration. Wang et al. [126] tune c dynamically based on the currently observed maximum Q-value to balance exploration and exploitation as Q-value estimates evolve and report a significant improvement in the observed results.

Some problem settings feature varying instance sizes, where a larger instance size is generally associated with higher difficulty. Zhong et al. [155] increase c as the problem size grows, presumably because more exploration is required to adequately cover the larger state space. For similar reasons, they and others [123, 136] further choose larger n_{MCTS} for larger instance sizes. Wang et al. [121] even explicitly parameterize n_{MCTS} by the problem size.

In many problems, the size of the remaining search space decreases with increasing depth of the tree. Hu et al. [45] argue that the search budget should depend on the depth of the node the search starts from and present a mechanism that decays n_{MCTS} with increasing tree depth.

Fawzi et al. [26] report an increase in n_{MCTS} after a certain amount of training steps, presumably because later training iterations can profit more from a higher search budget and a larger proportion of the overall training time budget should hence be allocated to those later iterations.

Some authors reduce the number of MCTS simulations at test time. For instance, Chen et al. [14] reduce n_{MCTS} by a factor of ten at test time compared to the training phase. They further limit the search depth in both phases, but do so to a larger degree during test time.

In some cases, MCTS is only used to train a policy network, which is then applied without further tree search at test time [9, 10, 18, 20, 155]. This can be due to the specific requirements of the application, i.e., some applications require fast inferences at test time that render the application of tree search infeasible, but can still profit from MCTS at training time [10]. In some applications, applying MCTS at test time after having used it for training simply does not improve performance to a significant degree [20].

6 Discussion & conclusion

While focusing on usages of neural MCTS outside of games, we investigated the diversity in applications, their characteristics, and the design of employed neural MCTS approaches by performing a systematic literature review.

With regard to **research question 1** posed in the introduction, we find that neural MCTS is applied in a wide variety of domains to solve different problems. While most problems exhibit similar characteristics such as discrete time and actions, finite horizon, and deterministic transitions, many authors also demonstrate that neural MCTS can be applied to problems with differing properties.

The applications encountered during the review usually have slightly different requirements and properties than combinatorial games. This does affect the way solutions are designed, an aspect investigated as part of **research ques-**

tion 2. The concept of self-play, for instance, is generally not applicable to single-player problems. In some cases, single-player problems can be modelled as multi-player problems, but this is the exception rather than the rule. It is possible to employ a mechanism called self-competition, which replicates the way self-play generates rewards in a single-player setting.

Many authors further point out that selecting actions based on *average* node values is not ideal in single-player deterministic environments. Different mechanisms, all based around tracking maximal node value, can be employed to adjust the typical MCTS mechanisms in this regard.

Compared to the neural MCTS ecosystem for games, as well as the traditional RL ecosystem, the neural MCTS landscape is almost completely devoid of standardized implementations and components. Instead, almost all implementations are entirely custom-made. In a few exceptions, domain-specific implementations are reused by others, but they can only be applied to a very narrow set of problems (see e.g. [145]). While this is understandable for more fundamental research, where implementations are inherently in flux, we believe that standardized components could significantly speed up progress on the applied research side.

One reason for the lack of standardized components may be that the design of neural MCTS methods varies substantially, beginning with their training approaches, to their use of self-play related mechanisms, forms of neural guidance, and other modifications to traditional MCTS setups. A widely applicable neural MCTS framework would have to be highly configurable to accommodate different disciplines and problem settings. While this is a difficult task, the traditional RL ecosystem demonstrates that standards [7] and publicly available implementations of algorithms [79] accelerate progress, and that flexible frameworks suitable for research [6] can be designed.

In response to **research question 3**, it can be concluded that the forms of neural guidance used in the game literature are often used in other applications as well. The most common type of guidance consists of neural selection and evaluation by a learned value function, just as in AlphaZero [100]. Other types, and combinations of neural guidance can be found in the literature as well. Given the amount of variation in different neural MCTS systems, a central question for practitioners is in how to set up their own systems depending on the characteristics of their applications. Ideally, we would be able to map observed problem characteristics to observed neural MCTS configurations to provide a guideline for others to use. While some problem characteristics, e.g. the discrete or continuous nature of the action space, can be determined fairly reliably when reviewing existing publications, others are not so easy to ascertain. The breadth and depth of the (full) tree, for instance, are rarely reported explicitly. In some cases, it may be possible to infer them, but in a review with a multitude of different disciplines, trying to do so reliably is difficult.

Some insights can nevertheless be derived from the collected literature. In games and beyond, it is clear that neural guidance can help to increase the efficiency of the tree search, but can also incur computational cost without much additional benefit in some situations. When and how to employ neural guidance should hence be carefully weighed. During the evaluation phase, for instance, the right choice of mechanism depends on the depth of the overall tree as well as the depth of the node to be evaluated. If the length of a roll-out will be short, it may be preferable over an estimate by a learned value function with associated inference cost. At what exact depth one may be preferable over the other will depend on the size of the employed neural network, which will influence the inference cost, as well as the quality of its predictions. Competing with a high quality estimate of a learned value function may require multiple roll-outs, since individual roll-outs are high variance estimates of a node's value. A good initial estimate can help focus the search on promising regions of the solution space, while a bad estimate can lead the search astray.

In applications with large tree breadth, neural guidance may be especially helpful in the expansion phase, where it can be used to prevent certain paths in the search tree from consideration altogether. Of course, this comes at the risk of cutting off high-quality solutions. Here as well, the quality of neural network predictions determines whether such an approach is sensible. Especially in applications with very large tree breadth, or even continuous domains, however, a search may not even be feasible without limiting the solution space to some degree.

Clearly, many questions remain unanswered. Additionally, a purely backward looking review tends to summarize what has been done in the past, rather than what should have been done. What is presented in this review is therefore primarily a map of existing approaches and less so a collection of prescriptive knowledge. The results gathered in this review can, however, serve as a foundation for further experimental studies. It is clear that different applications can benefit from different variants of neural MCTS and that no single algorithmic formulation will be the best choice for all possible problems. Practitioners will hence continue to be faced with the task of designing a suitable algorithm for their specific problem setting. Our review can serve as a starting place for this, as it provides a summary of the large set of known possible design choices. Explicitly performing experiments for multiple applications with different properties, in which the factors identified in this review are systematically controlled, can serve to create a more robust understanding of the design of neural MCTS approaches. From such an understanding, prescriptive rules (of thumb) can be derived to aid practitioners in making appropriate design choices for applications with given properties.

Appendix A: Implementations

Table 19 Publicly accessible implementations of neural MCTS approaches

Source	Link to Implementation
[5]	https://github.com/baifanxxx/NPMO-Rearrangement
[25]	https://github.com/sekijima-lab/mermaid
[32]	https://github.com/werner-duvaud/muzero-general (muzero general)
[35]	https://github.com/HOL-Theorem-Prover/HOL
[36]	https://github.com/MolecularAI/aizynthfinder
[37]	https://mgit.cs.uni-saarland.de/timopgros/carmanufacturin
[82]	https://github.com/bpriviere/decision_making
[94]	https://github.com/lynshao/AlphaSeq
[106]	see [145]
[109]	https://github.com/NREL/rlmolecule
[113]	https://userinterfaces.aalto.fi/adaptive/
[119]	https://github.com/HanqingWangAI/SceneMover
[131]	https://github.com/PhilippeW83440/MCTS-NNET
[135]	see [145]
[136]	https://github.com/CMACH508/2020-GNN-MCTS-TSP
[145]	https://github.com/tsudalab/ChemTS
[148]	https://github.com/zbc0315/synprepy
[156]	https://github.com/zsoltzombori/plcopprolog
[157]	https://github.com/zsoltzombori/plcop
[158]	https://github.com/zoulixin93/FMCTS

Acknowledgements Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2023 Internet of Production - 390621612.

Author Contributions Conceptualization: Marco Kemmerling; Methodology: Marco Kemmerling; Literature search: Marco Kemmerling; Data analysis: Marco Kemmerling; Writing - original draft preparation: Marco Kemmerling; Writing - review and editing: Marco Kemmerling, Daniel Lütticke; Funding acquisition: Robert H. Schmitt; Supervision: Daniel Lütticke, Robert H. Schmitt.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability All data generated or analysed during this study are included in this published article.

Declarations

Competing interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Al-Saffar M, Musilek P (2020) Reinforcement learning-based distributed BESS management for mitigating overvoltage issues in systems with high PV penetration. IEEE Trans Smart Grid 11(4):2980–2994
2. Anthony T, Tian Z, Barber D (2017) Thinking fast and slow with deep learning and tree search. Adv Neural Inf Process Syst 30
3. Audrey G, Francesco M (2019) Deep neural network and Monte Carlo tree search applied to fluid-structure topology optimization. Sci Rep 9(1):15916
4. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn 47(2):235–256
5. Bai F, Meng F, Liu J, Wang J, Meng MQ (2022) Hierarchical policy with deep-reinforcement learning for nonprehensile multiobject rearrangement. Biomim Intell Robot 2(3):100047
6. Bitter C, Thun T, Meisen T (2022) Karolos: an open-source reinforcement learning framework for robot-task environments. arXiv:2212.00906
7. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) OpenAI Gym. arXiv:1606.01540
8. Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games 4(1):1–43
9. Carfora V, Di Massimo F, Rastelli R, Catellani P, Piastra M (2020) Dialogue management in conversational agents through psychology of persuasion and machine learning. Multimed Tools Appl 79(47):35949–35971
10. Challita U, Sandberg D (2021) Deep reinforcement learning for dynamic spectrum sharing of LTE and NR. In ICC 2021 - IEEE international conference on communications, pp. 1–6
11. Chen J, Chen S, Luo S, Wang Q, Cao B, Li X (2020) An intelligent task offloading algorithm (iTOA) for UAV edge computing network. Digit Commun Netw 6(4):433–443
12. Chen J, Chen S, Wang Q, Cao B, Feng G, Hu J (2019) iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks. IEEE Internet Things J 6(4):7011–7024
13. Chen J, Luo S, Zhang L, Zhang C, Cao B (2021) iPAS: A deep Monte Carlo Tree Search-based intelligent pilot-power allocation scheme for massive MIMO system. Digit Commun Netw 7(3):362–372
14. Chen J, Zhang C, Luo J, Xie J, Wan Y (2020) Driving maneuvers prediction based autonomous driving control by deep Monte Carlo tree search. IEEE Trans Veh Technol 69(7):7146–7158
15. Chen P-Y, Ke B-T, Lee T-C, Tsai I-C, Kung T-W, Lin L-Y, Liu E-C, Chang Y-C, Li Y-L, Chao MC-T (2022) A reinforcement learning agent for obstacle-avoiding rectilinear steiner tree construction. In Proceedings of the 2022 international symposium on physical design, pp. 107–115

16. Chen Y-Q, Chen Y, Lee C-K, Zhang S, Hsieh C-Y (2022) Optimizing quantum annealing schedules with Monte Carlo tree search enhanced with neural networks. Nat Mach Intell 4(3):269–278
17. Chen Z, Huang J, Ahn H, Ning X (2021) Costly features classification using Monte Carlo tree search. In 2021 International joint conference on neural networks (IJCNN), pp. 1–8
18. Chen Z, Zhong S, Chen J, Zhao Y (2021) DeepPursuit: uniting classical wisdom and deep RL for sparse recovery. In 2021 55th Asilomar conference on signals, systems, and computers, pp. 1361–1366
19. Cheng Y, Wu Z, Liu K, Wu Q (1826) Wang Y (2019) Smart dag tasks scheduling between trusted and untrusted entities using the mcts method. Sustainability 11:7
20. Dai Y, Wang P, Zhang L (2021) Reinforcement syntactic dependency tree reasoning for target-oriented opinion word extraction. In: International conference on artificial neural networks, Springer, pp. 531–543
21. Dai Z, Liu CH, Ye Y, Han R, Yuan Y, Wang G, Tang J (2022) AoI-minimal UAV crowdsensing by model-based graph convolutional reinforcement learning. In IEEE INFOCOM 2022 - IEEE conference on computer communications, pp. 1029–1038
22. Dalgaard M, Motzoi F, Sørensen JJ, Sherson J (2020) Global optimization of quantum dynamics with AlphaZero deep exploration. NPJ Quantum Inf 6(1):1–9
23. Deng H, Yuan X, Tian Y, Hu J (2022) Neural-augmented two-stage Monte Carlo tree search with over-sampling for protein folding in HP Model. IEEJ Trans Electr Electron Eng 17(5):685-694
24. Dieb S, Song Z, Yin W-J, Ishii M (2020) Optimization of depth-graded multilayer structure for x-ray optics using machine learning. J Appl Phys 128(7):074901
25. Erikawa D, Yasuo N, Sekijima M (2021) MERMAID: an open source automated hit-to-lead method based on deep reinforcement learning. J Cheminformatics 13(1):1–10
26. Fawzi A, Balog M, Huang A, Hubert T, Romera-Paredes B, Barekatain M, Novikov A, R Ruiz FJ, Schrittwieser J, Swirszcz G, Silver D, Hassabis D, Kohli P (2022) Discovering faster matrix multiplication algorithms with reinforcement learning. Nature 610(7):47–53
27. Feng Y, Li B, Zheng Q, Wang D, Xu X, Zhang R (2021) Electromagnetic situation analysis and judgment based on deep learning. IET Commun 15(11):1455–1466
28. Fong J, Campolo D, Acar C, Tee KP (2021) Model-based reinforcement learning with LSTM networks for non-prehensile manipulation planning. In: 2021 21st international conference on control, automation and systems (ICCAS), pp. 1152–1158
29. Fricke C, Wolff D, Kemmerling M, Elgeti S (2023) Investigation of reinforcement learning for shape optimization of 2D profile extrusion die geometries. Advances in computational science and engineering. Publisher: Advances in Computational Science and Engineering
30. Fu Z, Fan Q, Zhang X, Li X, Wang S, Wang Y (2021) Policy network assisted Monte Carlo Tree search for intelligent service function chain deployment. In 2021 IEEE 20th international conference on trust, security and privacy in computing and communications (TrustCom), pp. 1161–1168
31. Gaafar M, Shaghaghi M, Adve RS, Ding Z (2019) Reinforcement learning for cognitive radar task scheduling. In: 2019 53rd Asilomar conference on signals, systems, and computers, pp. 1653–1657
32. Gabirondo-López J, Egaña J, Miguel-Alonso J, Orduna Urrutia R (2021) Towards autonomous defense of SDN networks using MuZero based intelligent agents. IEEE Access 9:107184–107199
33. Ganapathi Subramanian S, Crowley M (2018) Combining MCTS and A3C for prediction of spatially spreading processes in forest wildfire settings. In: Canadian conference on artificial intelligence, Springer, pp. 285–291
34. Gannouni A, Samsonov V, Behery M, Meisen T, Lakemeyer G (2020) Neural combinatorial optimization for production scheduling with sequence-dependent setup waste. In 2020 IEEE international conference on systems, man, and cybernetics (SMC), pp. 2640–2647
35. Gauthier T (2020) Deep reinforcement learning for synthesizing functions in higher-order logic. EPiC Series in Computing 73:230–248
36. Genheden S, Thakkar A, Chadimová V, Reymond J-L, Engkvist O, Bjerrum E (2020) AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. J Cheminformatics 12(1):1–9
37. Gros TP, Groß J, Wolf V (2020) Real-time decision making for a car manufacturing process using deep reinforcement learning. In: 2020 winter simulation conference (WSC), pp. 3032–3044
38. Göppert A, Mohring L, Schmitt RH (2021) Predicting performance indicators with ANNs for AI-based online scheduling in dynamically interconnected assembly systems. Prod Eng 15(5):619–633
39. Ha T, Cho K, Cha G, Lee K, Oh S (2020) Vehicle control with prediction model based Monte-Carlo tree search. In 2020 17th international conference on ubiquitous robots (UR), pp. 303–308
40. He L, Shao B, Xiao Y, Li Y, Liu T-Y, Chen E, Xia H (2018) Neurally-guided semantic navigation in knowledge graph. IEEE Trans Big Data 8(3):607–615
41. He Y, Li H, Jin T, Bao FS (2022) Circuit routing using Monte Carlo Tree Search and deep reinforcement learning. In: 2022 International symposium on vlsi design, automation and test (VLSI-DAT), pp. 1–5
42. He Y, Tao S, Xu J, Guo J, Lan Y, Cheng X (2018) Text matching with Monte Carlo tree search. In China conference on information retrieval, Springer, pp 41–52
43. Hernandez D, Denamganaï K, Gao Y, York P, Devlin S, Samothrakis S, Walker JA (2019) A generalized framework for self-play training. In: 2019 IEEE conference on games (CoG), IEEE, pp 1–8
44. Hoel C-J, Driggs-Campbell K, Wolff K, Laine L, Kochenderfer MJ (2020) Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. IEEE Trans Intell Veh 5(2):294–305
45. Hu Z, Tu J, Li B (2019) Spear: optimized dependency-aware task scheduling with deep reinforcement learning. In: 2019 IEEE 39th international conference on distributed computing systems (ICDCS), pp 2037–2046
46. Huang J, Patwary M, Diamos G (2019) Coloring big graphs with alphagozero. arXiv:1902.10162
47. Huang Y, Yu Y (2018) Distilling deep neural networks with reinforcement learning. In 2018 IEEE international conference on information and automation (ICIA), pp 133–138
48. Ishida S, Terayama K, Kojima R, Takasu K, Okuno Y (2022) AI-driven synthetic route design incorporated with retrosynthesis knowledge. J Chem Inf Model 62(6):1357–1367
49. Jang Y, Lee J, Kim K-E (2020) Bayes-adaptive Monte-Carlo planning and learning for goal-oriented dialogues. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, pp 7994–8001. Issue: 05
50. Jiang Y, Liu M, Li J, Zhang J (2022) Reinforced MCTS for non-intrusive online load identification based on cognitive green computing in smart grid. Math Biosci Eng 19(11):11595–11627
51. Kemmerling M, Samsonov V, Lütticke, D., Schuh G, Gützlaff A, Schmidhuber M, Janke T, Meisen T (2021) Towards production-ready reinforcement learning scheduling agents: A hybrid two-step training approach based on discrete-event simulations. Simulation in Produktion und Logistik, 325–336

52. Kim M, Park J-K, Moon S-M (2022) Solving PBQP-based register allocation using deep reinforcement learning. In 2022 IEEE/ACM international symposium on code generation and optimization (CGO), pp 1–12
53. Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In European conference on machine learning, Springer, pp 282–293
54. Kocsis L, Szepesvári C, Willemson J (2006) Improved Monte-Carlo search. Univ. Tartu, Estonia, Tech. Rep 1
55. Kovari B, Becsi T, Szabo A., Aradi, S (2020) Policy gradient based control of a pneumatic actuator enhanced with Monte Carlo Tree search. In: 2020 6th international conference on mechatronics and robotics engineering (ICMRE), pp 177–182
56. Kovári B, Hegedüs F, Bécsi T (2020) Design of a reinforcement learning-based lane keeping planning agent for automated vehicles. Appl Sci 10(20):7171
57. Kuai Z, Wang T, Wang S (2022) Fair virtual network function mapping and scheduling using proximal policy optimization. IEEE Trans Commun 70(11):7434–7445
58. Kumar A, Dimitrakopoulos R (2021) Production scheduling in industrial mining complexes with incoming new information using tree search and deep reinforcement learning. Appl Soft Comput 110:107644
59. Lao Y, Xu J, Gao S, Guo J, Wen J-R (2019) Name entity recognition with policy-value networks. In: Proceedings of the 42nd International ACM SIGIR conference on research and development in information retrieval, pp 1245–1248
60. Laterre A, Fu Y, Jabri M, Cohen A, Kas D, Hajjar K, Dahl T, Kerkeni A, Beguir K (2018) Ranked reward: enabling self-play reinforcement learning for combinatorial optimization. Advances in neural information processing systems 31 (NeurIPS 2018)
61. Lei L, Luo R, Zheng R, Wang J, Zhang J, Qiu C, Ma L, Jin L, Zhang P, Chen J (2021) KB-Tree: learnable and continuous Monte-Carlo tree search for autonomous driving planning. In: 2021 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 4493–4500
62. Li H, Hu Y, Cao Y, Zhou G, Luo P (2021) Rich-text document styling restoration via reinforcement learning. Front Comput Sci 15(4):1–11
63. Li Y, Pei J, Lai L (2021) Structure-based de novo drug design using 3D deep generative models. Chem Sci 12(41):13664–13675
64. Lu Q, Tao F, Zhou S, Wang Z (2021) Incorporating Actor-Critic in Monte Carlo tree search for symbolic regression. Neural Comput Appl 33(14):8495–8511
65. Ma B, Terayama K, Matsumoto S, Isaka Y, Sasakura Y, Iwata H, Araki M, Okuno Y (2021) Structure-based de novo molecular generator combined with artificial intelligence and docking simulations. J Chem Inf Model 61(7):3304–3313
66. Mandhane A, Zhernov A, Rauh M, Gu C, Wang M, Xue F, Shang W, Pang D, Claus R, Chiang C-H, et al (2022) Muzero with self-competition for rate control in vp9 video compression. arXiv:2202.06626
67. Mao K, Xiao Y (2021) Learning the fastest RNA folding path based on reinforcement learning and Monte Carlo tree search. Molecules 26(15):4420
68. Mańdziuk J (2018) MCTS/UCT in solving real-life problems. In: Advances in data analysis with computational intelligence methods. Springer, pp 277–292
69. Meng X, Inaltekin H, Krongold B (2019) Deep reinforcement learning-based topology optimization for self-organized wireless sensor networks. In: 2019 IEEE global communications conference (GLOBECOM), pp 1–6
70. Mo S, Pei X, Wu C (2022) Safe reinforcement learning for autonomous vehicle using Monte Carlo tree search. IEEE Trans Intell Transp Syst 23(7):6766–6773
71. Mo T-W, Chang RY, Kan T-Y (2022) DeepMCTS: deep reinforcement learning assisted Monte Carlo tree search for MIMO detection. In: 2022 IEEE 95th vehicular technology conference: (VTC2022-Spring), pp 1–6
72. Moerland TM, Broekens J, Plaat A, Jonker CM (2018) A0c: Alpha zero in continuous action space. arXiv:1805.09613
73. Moerland TM, Broekens J, Plaat A, Jonker CM (2022) A unifying framework for reinforcement learning and planning. Front Artif Intell 5
74. Motomichi T, Yutaka I, Michio K. De (2021) novo generation of optically active small organic molecules using Monte Carlo tree search combined with recurrent neural network. J Comput Chem 42(3):136–143
75. Oren J, Ross C, Lefarov M, Richter F, Taitler A, Feldman Z, Di Castro D, Daniel C (2021) SOLO: search online, learn offline for combinatorial optimization problems. In: Proceedings of the international symposium on combinatorial search, vol 12, pp 97–105. Issue: 1
76. Paxton C, Raman V, Hager GD, Kobilarov M (2017) Combining neural networks and tree search for task and motion planning in challenging environments. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp. 6059–6066
77. Peng H, Wu C, Zhan Y, Xia Y (2022) Lore: a learning-based approach for workflow scheduling in clouds. In: Proceedings of the conference on research in adaptive and convergent systems, pp 47–52
78. Qian H, Lin C, Zhao D, Tu S, Xu L (2022) AlphaDrug: protein target specific de novo molecular generation. PNAS Nexus 1(4):pgac227
79. Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N (2021) Stable-Baselines3: reliable reinforcement learning implementations. J Mach Learn Res 22(268):1–8
80. Raina A, Cagan J, McComb C (2022) Learning to design without prior data: Discovering generalizable design strategies using deep learning and tree search. J Mech Des 145(3):031402
81. Rinciog A, Mieth C, Scheikl PM, Meyer A (2020) Sheet-metal production scheduling using AlphaGo Zero. In: Proceedings of the conference on production systems and logistics: CPSL 2020
82. Riviere B, Hönig W, Anderson M, Chung S-J (2021) Neural tree expansion for multi-robot planning in non-cooperative environments. IEEE Robot Autom Lett 6(4):6868–6875
83. Rosin CD (2011) Multi-armed bandits with episode context. Ann Math Artif Intell 61(3):203–230
84. Sadeghnejad-Barkousaraie A, Bohara G, Jiang S, Nguyen D (2021) A reinforcement learning application of a guided Monte Carlo Tree Search algorithm for beam orientation selection in radiation therapy. Mach Learn: Sci Technol 2(3):035013
85. Sadeghnejad Barkousaraie A, Ogunmolu O, Jiang S, Nguyen D (2019) Using supervised learning and guided Monte Carlo tree search for beam orientation optimization in radiation therapy. In: Workshop on artificial intelligence in radiation therapy, Springer, pp 1–9
86. Samsonov V, Kemmerling M, Paegert M, Lütticke D, Sauermann F, Gützlaff A, Schuh G, Meisen T (2021) Manufacturing control in job shop environments with reinforcement learning. In: Proceedings of the 13th international conference on agents and artificial intelligence, pp 589–597
87. Sandberg D, Kvernvik T, Calabrese FD (2022) Learning robust scheduling with search and attention. In: ICC 2022-IEEE international conference on communications, IEEE, pp 1549–1555
88. Schmidt D, Moran N, Rosenfeld JS, Rosenthal J, Yedidia J (2019) Self-play learning without a reward metric. arXiv:1912.07557
89. Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T et al

(2020) Mastering ATARI, GO, Chess and Shogi by planning with a learned model. Nature 588(7839):604–609
90. Segler MH, Preuss M, Waller MP (2018) Planning chemical syntheses with deep neural networks and symbolic AI. Nature 555(7698):604–610
91. Segler MHS, Preuss M, Waller MP (2017) Towards "AlphaChem": chemical synthesis planning with tree search and deep neural network policies. In: 5th International conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings
92. Shafieirad H, Adve RS (2022) On meeting a maximum delay constraint using reinforcement learning. IEEE Access 10:97897–97911
93. Shaghaghi M, Adve RS, Ding Z (2019) Resource management for multifunction multichannel cognitive radars. In: 2019 53rd Asilomar conference on signals, systems, and computers, pp 1550–1554
94. Shao Y, Liew SC, Wang T (2020) AlphaSeq: sequence discovery with deep reinforcement learning. IEEE Trans Neural Netw Learn Syst 31(9):3319–3333
95. Shen Y, Chen J, Huang P-S, Guo Y, Gao J (2018) M-walk: Learning to walk over graphs using Monte Carlo tree search. Adv Neural Inf Process Syst 31
96. Shuai H, He H (2021) Online scheduling of a residential microgrid via Monte-Carlo tree search and a learned model. In: 2021 IEEE power & energy society general meeting (PESGM), pp 01
97. Shuai H, Li F, She B, Wang X, Zhao J (2023) Post-storm repair crew dispatch for distribution grid restoration using stochastic Monte Carlo tree search and deep neural networks. Int J Electr Power Energy Syst 144(4):108477
98. Silva K, Abeyasekare W, Dasanayake D, Nandisena T, Kasthurirathna D, Kugathasan A (2021) Dynamic user interface personalization based on deep reinforcement learning. In: 2021 3rd international conference on advancements in computing (ICAC), IEEE, pp 25–30
99. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529(7587):484–489
100. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, Lillicrap T, Simonyan K, Hassabis D (2018) A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362(6419):1140–1144
101. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A et al (2017) Mastering the game of go without human knowledge. Nature 550(7676):354–359
102. Sinha A, Azad U, Singh H (2022) Qubit routing using graph neural network aided Monte Carlo tree search. In: Proceedings of the AAAI conference on artificial intelligence, vol 36, pp 9935–9943. Issue: 9
103. Skrynnik A, Yakovleva A, Davydov V, Yakovlev K, Panov AI (2021) Hybrid policy learning for multi-agent pathfinding. IEEE Access 9:126034–126047
104. Song S, Chen H, Sun H, Liu M (2020) Data efficient reinforcement learning for integrated lateral planning and control in automated parking system. Sensors 20:24
105. Sridharan B, Mehta S, Pathak Y, Priyakumar UD (2022) Deep reinforcement learning for molecular inverse problem of nuclear magnetic resonance spectra to molecular structure. J Phys Chem Lett 13:4924–4933
106. Srinivasan S, Batra R, Chan H, Kamath G, Cherukara MJ, Sankaranarayanan SK (2021) Artificial intelligence-guided De novo molecular design targeting COVID-19. ACS Omega 6(19):12557–12566
107. Sun R, Liu Y (2021) Hybrid reinforcement learning for power transmission network self-healing considering wind power. IEEE Trans Neural Netw Learn Syst 1–11
108. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press
109. Ss SV, Law JN, Tripp CE, Duplyakin D, Skordilis E, Biagioni D, Paton RS, St. John PC (2022) Multi-objective goal-directed optimization of de novo stable organic radicals for aqueous redox flow batteries. Nat Mach Intell 4(8):720–730
110. Tang C, Fu S, Liu F (2021) Design and implementation of system for generating MOFs for hydrogen storage in hydrogen-fueled vehicles. In: 2021 IEEE international conference on artificial intelligence and industrial design (AIID), IEEE, pp 549–553
111. Thacker HK, Kumar A, Barari A, Damini D, Gupta A, Jagannathachar KK, Yoon D (2021) AlphaRA: an alphazero based approach to redundancy analysis. In: 2021 20th IEEE international conference on machine learning and applications (ICMLA), pp 477–483
112. Thakkar A, Kogej T, Reymond J-L, Engkvist O, Bjerrum EJ (2020) Datasets and their influence on the development of computer assisted synthesis planning tools in the pharmaceutical domain. Chem Sci 11(1):154–168
113. Todi K, Bailly G, Leiva L, Oulasvirta A (2021) Adapting user interfaces with model-based reinforcement learning. In: Proceedings of the 2021 CHI conference on human factors in computing systems, pp 1–13
114. Tomin N (2020) The concept of constructing an artificial dispatcher intelligent system based on deep reinforcement learning for the automatic control system of electric networks. J Comput Syst Sci Int 59(6):939–956
115. Van Eyck J, Ramon J, Guiza F, Meyfroidt G, Bruynooghe M, Van den Berghe G (2013) Guided Monte Carlo tree search for planning in learned environments. In: Asian conference on machine learning, PMLR, pp 33–47
116. Vodopivec T, Samothrakis S, Ster B (2017) On Monte Carlo tree search and reinforcement learning. J Artif Intell Res 60:881–936
117. Vom Brocke J, Simons A, Riemer K, Niehaves B, Plattfaut R, Cleven A (2015) Standing on the shoulders of giants: Challenges and recommendations of literature search in information systems research. Commun Assoc Inf Syst 37(1):9
118. Wang C, Chen X, Luo Y, Zhang G (2022) Solving virtual network mapping fast by combining neural network and MCTS. In: 2021 Ninth international conference on advanced cloud and big data (CBD), pp 19–24
119. Wang H, Liang W, Yu L-F (2020) Scene mover: automatic move planning for scene arrangement by deep reinforcement learning. ACM Trans Graph 39(6):1–15
120. Wang H, Yang R, Yin C, Zou X, Wang X (2021) Research on the difficulty of mobile node deployment's self-play in wireless Ad Hoc networks based on deep reinforcement learning. Wirel Commun Mob Comput 2021
121. Wang JH, Cheng Luo P, Xiong HQ, Zhang BW, Peng JY (2020) Parallel machine workshop scheduling using the integration of proximal policy optimization training and Monte Carlo tree search. In: 2020 Chinese automation congress (CAC), pp 3277–3282
122. Wang K, Sun W (2019) Meta-modeling game for deriving theory-consistent, microstructure-based traction-separation laws via deep reinforcement learning. Comput Methods Appl Mech Eng 346:216–241
123. Wang Q, Hao Y, Cao J (2021) Learning to traverse over graphs with a Monte Carlo tree search-based self-play framework. Eng Appl Artif Intell 105:104422
124. Wang R, Zhou M, Li Y, Zhang Q, Dong H (2019) A timetable rescheduling approach for railway based on Monte Carlo tree

search. In: 2019 IEEE intelligent transportation systems conference (ITSC), pp 3738–3743
125. Wang S, Zhou K, Lai K, Shen J (2020) Task-completion dialogue policy learning via Monte Carlo tree search with dueling network. In Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP), pp 3461–3471
126. Wang X, Qian Y, Gao H, Coley CW, Mo Y, Barzilay R, Jensen KF (2020) Towards efficient discovery of green synthetic pathways with Monte Carlo tree search and reinforcement learning. Chem Sci 11(40):10959–10972
127. Wang X, Thomas JD, Piechocki RJ, Kapoor S, Santos-Rodríguez R, Parekh A (2022) Self-play learning strategies for resource assignment in Open-RAN networks. Comput Netw 206:108682
128. Wang Y, Sun M, Wang H, Sun Y (2022) Research on knowledge graph completion model combining temporal convolutional network and Monte Carlo tree search. Math Prob Eng 2022
129. Wang Z, Li C (2022) Channel pruning via lookahead search guided reinforcement learning. In 2022 IEEE/CVF winter conference on applications of computer vision (WACV), pp 3513–3524
130. Webster J, Watson RT (2002) Analyzing the past to prepare for the future: writing a literature review. MIS Q xiii-xxiii
131. Weingertner P, Ho M, Timofeev A, Aubert S, Pita-Gil G (2020) Monte Carlo Tree search with reinforcement learning for motion planning. In: 2020 IEEE 23rd international conference on intelligent transportation systems (ITSC), pp 1–7
132. Weininger D (1988) SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. J Chem Inf Comput Sci 28(1):31–36
133. Wu Q, Feng Q, Ren Y, Xia Q, Wang Z, Cai B (2021) An intelligent preventive maintenance method based on reinforcement learning for battery energy storage systems. IEEE Trans Industr Inform 17(12):8254–8264
134. Wu Y, Yao L (2021) Research on the problem of 3D bin packing under incomplete information based on deep reinforcement learning. In: 2021 International conference on e-commerce and e-management (ICECEM), pp 38–42
135. Xiangyu Z, Kexin Z, Yongjin L (2020) Machine learning enabled tailor-made design of application-specific metal-organic frameworks. ACS Appl Mater Interfaces 12(1):734–743
136. Xing Z, Tu S (2020) A graph neural network assisted Monte Carlo tree search approach to traveling salesman problem. IEEE Access 8:108418–108428
137. Xing Z, Tu S, Xu L (2020) Solve traveling salesman problem by Monte Carlo tree search and deep neural network. arXiv:2005.06879
138. Xu R, Lieberherr K (2019) Learning self-game-play agents for combinatorial optimization problems. In: Proceedings of the 18th international conference on autonomous agents and multiagent systems, pp 2276–2278
139. Xu R, Lieberherr K (2020) Learning self-play agents for combinatorial optimization problems. Knowl Eng Rev 35
140. Xu R, Lieberherr K (2022) On-the-fly model checking with neural MCTS. In: NASA formal methods symposium, Springer, pp 557–575
141. Xu R, Lieberherr K (2022) Towards tackling QSAT problems with deep learning and Monte Carlo tree search. In: Science and information conference, Springer, pp 45–58
142. Yan M, Feng G, Zhou J, Qin S (2018) Smart multi-RAT access based on multiagent reinforcement learning. IEEE Trans Veh Technol 67(5):4539–4551
143. Yang J, Hou X, Hu YH, Liu Y, Pan Q (2020) A reinforcement learning scheme for active multi-debris removal mission planning with modified upper confidence bound tree search. IEEE Access 8:108461–108473
144. Yang R, Zou X, Nie Z, Yin C (2018) Research on deployment of communication node vehicles based on deep reinforcement learning. In: 2018 5th IEEE international conference on cloud computing and intelligence systems (CCIS), pp 484–487
145. Yang X, Zhang J, Yoshizoe K, Terayama K, Tsuda K (2017) ChemTS: an efficient python library for de novo molecular generation. Sci Technol Adv Mater 18(1):972–976
146. Yin C, Yang R, Zhu W, Zou X, Zhang J (2021) Optimal planning of emergency communication network using deep reinforcement learning. IEICE Trans Commun 104(1):20–26
147. Yu T, Huang J, Chang Q (2020) Mastering the working sequence in human-robot collaborative assembly based on reinforcement learning. IEEE Access 8:163868–163877
148. Zhang B, Zhang X, Du W, Song Z, Zhang G, Zhang G, Wang Y, Chen X, Jiang J, Luo Y (2022) Chemistry-informed molecular graph as reaction descriptor for machine-learned retrosynthesis planning. Proc Natl Acad Sci 119(41):e2212711119
149. Zhang C, Song W, Cao Z, Zhang J, Tan PS, Chi X (2020) Learning to dispatch for job shop scheduling via deep reinforcement learning. Adv Neural Inf Process Syst 33:1621–1632
150. Zhang J, Chen H, Song S, Hu F (2020) Reinforcement learning-based motion planning for automatic parking system. IEEE Access 8:154485–154501
151. Zhang J, Zhou K, Schelter S (2020) Alphajoin: join order selection á la AlphaGo. In: CEUR workshop proceedings, vol 2652
152. Zhang K, Wu J, Yoo H, Lee Y (2021) Machine learning-based approach for tailor-made design of ionic liquids: Application to CO2 capture. Sep Purif Technol 275:119117
153. Zhang M, Huang Q, Wang S, Wang Z (2018) Construction of LDPC codes based on deep reinforcement learning. In: 2018 10th international conference on wireless communications and signal processing (WCSP), pp 1–4
154. Zhang Y, Wang W, Zhang P, Huang P (2022) Reinforcement-learning-based task planning for self-reconfiguration of cellular satellites. IEEE Aerosp Electron Syst Mag 37(6):38–47
155. Zhong S, Zhao Y, Chen J (2019) Learning to recover sparse signals. In: 2019 57th Annual allerton conference on communication, control, and computing (Allerton), pp 995–1000
156. Zombori Z, Urban J, Brown CE (2020) Prolog technology reinforcement learning prover. In: International joint conference on automated reasoning, Springer, pp 489–507
157. Zombori Z, Urban J, Olšák M (2021) The role of entropy in guiding a connection prover. In: International conference on automated reasoning with analytic tableaux and related methods, Springer, pp 218–235
158. Zou L, Xia L, Ding Z, Yin D, Song J, Liu W (2019) Reinforcement learning to diversify top-n recommendation. In: International conference on database systems for advanced applications, Springer, pp 104–120
159. Zou X, Yang R, Yin C (2019) Research on node layout model optimization of MANET based on AlphaZero technology under incomplete visual terrain. In: Proceedings of the 2019 international conference on artificial intelligence and computer science, pp 562–565
160. Zou X, Yang R, Yin C, Nie Z, Wang H (2020) Deploying tactical communication node vehicles with AlphaZero algorithm. IET Commun 14(9):1392–1396
161. Świechowski M, Godlewski K, Sawicki B, Mańdziuk J (2022) Monte, Carlo tree search: A review of recent modifications and applications. Artif Intell Rev 1–66