Monte Carlo Tree Search: a review of recent modifications and applications

Maciej Świechowski[1] · Konrad Godlewski[2] · Bartosz Sawicki[2] · Jacek Mańdziuk[2,3]

Abstract

Monte Carlo Tree Search (MCTS) is a powerful approach to designing game-playing bots or solving sequential decision problems. The method relies on intelligent tree search that balances exploration and exploitation. MCTS performs random sampling in the form of simulations and stores statistics of actions to make more educated choices in each subsequent iteration. The method has become a state-of-the-art technique for combinatorial games. However, in more complex games (e.g. those with a high branching factor or real-time ones) as well as in various practical domains (e.g. transportation, scheduling or security) an efficient MCTS application often requires its problem-dependent modification or integration with other techniques. Such domain-specific modifications and hybrid approaches are the main focus of this survey. The last major MCTS survey was published in 2012. Contributions that appeared since its release are of particular interest for this review.

✉ Maciej Świechowski
maciej.swiechowski@qed.pl

Konrad Godlewski
jowisz11@gmail.com

Bartosz Sawicki
bartosz.sawicki@pw.edu.pl

Jacek Mańdziuk
jacek.mandziuk@pw.edu.pl

1   QED Software, Warsaw, Poland

2   Warsaw University of Technology, Warsaw, Poland

3   AGH University of Science and Technology, Kraków, Poland

1 Introduction

Monte Carlo Tree Search (MCTS) is a decision-making algorithm that consists in searching combinatorial spaces represented by trees. In such trees, nodes denote states, also referred to as configurations of the problem, whereas edges denote transitions (actions) from one state to another.

MCTS has been originally proposed in the work by Kocsis and Szepesvári (2006) and by Coulom (2006), as an algorithm for making computer players in Go. It was quickly called a major breakthrough (Gelly et al. 2012) as it allowed for a leap from 14 *kyu*, which is an average amateur level, to 5 *dan*, which is considered an advanced level but not professional yet. Before MCTS, bots for combinatorial games had been using various modifications of the minimax alpha–beta pruning algorithm (Junghanns 1998) such as MTD(f) (Plaat 2014) and hand-crafted heuristics. In contrast to them, the MCTS algorithm is at its core *aheuristic*, which means that no additional knowledge is required other than just rules of a game (or a problem, generally speaking). However, it is possible to take advantage of heuristics and include them in the MCTS approach to make it more efficient and improve its convergence.

Moreover, in practical applications, the given problem often tends to be difficult for the base variant of the algorithm. The utilitarian definition of "too difficult" is that MCTS achieves poor outcomes in the given setting under practical computational constraints. Sometimes, increasing the effective computational budget would help, although in practical applications it may not be possible (e.g., because of strict response times, hardware costs, parallelization scaling). However, often, a linear increase is not enough to tackle difficult problems represented by trees and solving them would require effectively unlimited memory and computational power. There can be various reasons for a given problem being hard for MCTS. To name a few—combinatorial complexity, sparse rewards or other kinds of inherent difficulty. Whenever the *vanilla* MCTS algorithm, i.e., implemented in its base unmodified form, fails to deliver the expected performance, it needs to be equipped with some kind of enhancements. In this survey, we will **focus only on such papers** that introduce at least one modification to the vanilla version of the method. Although works that describe the use of standard MCTS in new domains have been published, they are not in the scope of this survey.

More recently, MCTS combined with deep reinforcement learning (RL) has become the backbone of AlphaGo developed by Google DeepMind and described in the article from Silver et al. (2016). It has been widely regarded as not only another major breakthrough in Go, but in artificial intelligence (AI) in general. It is safe to say that MCTS has been tried in most of combinatorial games and even some real-time video games (Farooq et al. 2016; Kim and Kim 2017). Examples include Poker (Van den Broeck et al. 2009), Chess (Silver et al. 2018), Settlers of Catan (Szita et al. 2009), Othello (Robles et al. 2011), Hex (Arneson et al. 2010), Lines of Action (Winands et al. 2010), Arimaa (Syed and Syed 2003), Havannah (Teytaud and Teytaud 2010) and *General Game Playing* (GGP) (Genesereth and Thielscher 2014; Perez et al. 2014). The latter is particularly interesting because in GGP, computer bots are pitted against each other to play a variety of games without any domain knowledge. This is where MCTS is a commonly applied approach.

Driven by successes in games, MCTS has been increasingly often applied in domains outside the game AI such as planning, scheduling, control and combinatorial optimization. We include examples from these categories in this article as well. Formally, MCTS is directly applicable to problems which can be modelled by a Markov decision process

(MDP) (Lizotte and Laber 2016), which is a type of discrete-time stochastic control process. Certain modifications of MCTS make it possible to apply it to partially observable MDPs (POMDPs).

1.1 Methodology

We review only such applications of the MCTS algorithm in which the standard version of the algorithm has been deemed ineffective or unsatisfactory, because of the given problem difficulty or unsuitability for the standard version as it is. This is a common scenario, for instance, in real world problems such as vehicle routing problem (VRP) or real-time video games, which tend to be more complex than combinatorial games due to an enormous branching factor, large state spaces, a continuous environment (time and space) and fast-paced action that requires making constant decisions.

It is useful to distinguish between two classes of environments—single-agent and multi-agent ones. Let us comment on this using the game domain example. The differences between single-player and multi-player games make various AI techniques suitable for a particular environment. For instance, deterministic single-player games are like puzzles to solve. The success in solving them depends only on the agent and not on external factors such as chance or opponents. Multi-player games require some form of opponent's modelling or cooperation (if they are cooperative). The implementation of MCTS can be optimized appropriately for the given case.

We focus on multi-agent problems, e.g., multi-player games, with a few exceptions. The exception is that we include such approaches of using the MCTS algorithm that, in our opinion, could be easily applied if the given problem was a multi-agent one. This is due to our subjective evaluation. The idea behind this was to provide readers with relatively well-generalizable modifications and not necessarily tailored for a given single-player game.

Moreover, certain problems may be modelled as single-player or multi-player games. For example, VRP can be modelled as a multi-player cooperative game, in which each vehicle has its own game tree and the synchronization between them is treated as in a multi-agent system. However, it can also be modelled as a single-player game, in which there is one coordinator player.

This survey covers contributions made after 2012. The reasoning behind 2012 was to start from the year in which the previous comprehensive MCTS survey was published by Browne et al. (2012). We recommend it to readers interested in earlier developments related to MCTS and as an additional introduction to the method. Section 2 contains an introduction to the MCTS algorithm as well.

To start with, we focused on *IEEE Transactions on Games* (ToG), which was previously known as *IEEE Transactions on Computational Intelligence and AI in Games* (TCIAG, until 2018). This is the leading journal concerning scientific, technical, and engineering aspects of games and the main source of high-quality contributions related to MCTS. We have also included the major conference on games—*IEEE Conference on Games* (CoG). It also changed the name and we included proceedings published under the previous name of *IEEE Computational Intelligence and Games* (CIG). Given those two sources, we performed an exhaustive search within each issue of ToG/TCIAG and each proceedings of CoG/CIG and analyzed all articles whether they fit the scope of this survey.

The remaining articles were found through scientific bibliography providers (online libraries): *dblp*, *IEEE Xplore Digital Library* and *Google Scholar*. Using them, we performed search over **titles** and **abstracts** by keywords. The following keywords were

chosen: "Monte Carlo", "Monte Carlo", "Tree Search", "MCTS", "Upper Confidence Bounds", "UCT", "UCB", "Multi-Armed Bandit", "Bandit-Based", "General Game Playing" and "General Video Game Playing". The results were curated by us with the bar set low (i.e., an acceptance rate was high). Using common sense, we excluded articles which had not been peer reviewed or were available only from unverified sources. The latter applies to *Google Scholar* which can index such sources too.

1.2 Structure of the survey

The following survey structure is mainly organized by types of extensions of MCTS algorithm. However, those modifications are usually motivated by challenging applications of the algorithm. These two classifications cross each other multiple times, which makes unambiguous navigation through the survey content troublesome. This section is to present the main structure of the text. The complete overview of references is presented in two Tables, grouped by application and grouped by method. It could be found in the last section Conclusions on pages 53, 54, 55.

The main content of the survey is composed of seven sections.

Section 2—***Classic MCTS*** introduces the MCTS algorithm in its classic (vanilla) variant. The basics are recalled for readers with less experience with the method.

Section 3—***Games with Perfect Information*** is devoted to the modifications of MCTS in games with perfect information, which are simpler in the combinatorial aspect. The topic is divided into three subsections:

Action Reduction (Sect. 3.1)—extensions based on elimination of possible actions in decision states. Usually, obviously wrong actions are skipped. This could be achieved by manipulation of exploration term in the tree policy function.
UCT Alternatives (Sect. 3.2)—Upper Confidence Bounds for Trees (UCT) is the core equation in the MCTS method. Several modification of UCT are discussed.
Early Termination (Sect. 3.3)—techniques for improving efficiency by setting cut-off depth for random playouts.
Opponent Modelling (Sect. 3.4)—chosen aspects of modelling opponent's moves in perfect information games

Section 4—***Games with Imperfect Information*** is dedicated to imperfect information games also referred to as games with hidden information. We distinguish six different types of MCTS extensions related to this game genre.

Determinization (Sect. 4.1)—removing randomness from the game by setting a specific value for each unknown feature.
Information Sets (Sect. 4.2)—a method to deal with imperfect information games, where states, which are indistinguishable from player's perspective are grouped in information sets.
Heavy Playouts (Sect. 4.3)—extension of simple, random playouts by adding domain-specific knowledge. Methods from this group proved to be effective; however, one should be aware that utilization of domain knowledge is always at the expense of increased algorithm's complexity.
Policy Update (Sect. 4.4)—a general group of methods designed to manipulate policy during the tree building.

Master Combination (Sect. 4.5)—algorithms taking part in competitions are usually a combination of different MCTS extensions.

Opponent Modelling (Sect. 4.6)—modelling the game progress requires modelling the opponent's actions. This is actually true for all multi-player games, but is particularly challenging in imperfect information games.

Section 5—*Combining MCTS with Machine Learning* summarizes the approaches that combine MCTS with machine learning (ML).

MCTS + Neural Networks (Sect. 5.3)—AlphaGo approach and other inspired methods. MCTS + Temporal Difference Learning (Sect. 5.4)—MCTS combined with temporal difference (TD) methods.

Advantages and Challenges of Using ML with MCTS (Sect. 5.5)—summary of the section.

Section 6—*MCTS with Evolutionary Methods* summarizes the approaches that combine MCTS with methods inspired by biological evolution.

Evolving Heuristic Functions (Sect. 6.1)—evaluation functions for MCTS are evolved. Evolving Policies (Sect. 6.2)—the aspects related to performing simulations (roll-outs) undergo the evolution process.

Rolling Horizon Evolutionary Algorithm (RHEA, Sect. 6.3)—a competitor algorithm to MCTS with elements inspired by evolutionary algorithms (EA).

Evolutionary MCTS (Sect. 6.4)—combination of MCTS and EA in a single approach.

Section 7—*MCTS Applications Beyond Games* is focused on applications to "real-world" problems from outside the games domain.

Planning (Sect. 7.1)—optimal planning in logistics and robotics. Security (Sect. 7.2)—finding optimal patrolling schedules for security forces in attacker–defender scenarios.

Chemical synthesis (Sect. 7.3)—planning in organic chemistry domain. Scheduling (Sect. 7.4)—real business challenges, such as scheduling combined with risk management.

Vehicle Routing (Sect. 7.5)—solutions to various formulations of the VRP.

Section 8—*Parallelization* discusses parallel MCTS implementations. Both basic approaches and more advanced methods are discussed.

Section 9—*Conclusions* presents an overview of the discussed methods and applications in the form of two summary tables, as well as conclusions and open problems.

Note—The majority of the literature works analysed in this survey apply some kind of parameter optimization or fine-tuning. In particular, the focus is on the *exploration constant* C, the *maximum search depth*, the *efficient use of an computational budget*, or *parameters related to choosing and executing policies (tree policy and default policy)*. Typically such an optimisation is performed empirically.

Due to a variety of possible parameter-tuning approaches and their direct connections to particular implementations of the MCTS mechanism, we decided to address the topic of parameter optimization alongside introduction and discussion of particular MCTS variants/related papers—instead of collecting them in a separate dedicated section. We believe that

such a presentation is more coherent and allows the reader to take a more comprehensive look at a given method.

1.3 Definitions of the considered game genres

In this section we present selected definitions of the most popular genres of games considered in this paper. An application-oriented taxonomy is presented in Table 1 in Sect. 9.

Perfect Information Games. Game is classified as perfect information when every player at any time has a complete knowledge about the history and current state of the game. Chess and Go are prominent representatives of this group.

Imperfect Information Games. In games with imperfect information a game state is not fully available to the players, which means that decisions are taken without the full knowledge of the opponent's position. There can be hidden elements of the game, for example cards in hand or face down cards. Game mechanics can include also randomness e.g. drawing a starting hand of cards at the beginning of gameplay. Most modern games are classified as imperfect information.

General Game Playing (GGP). GGP (Genesereth and Thielscher 2014) is a research framework that concerns creating computer agents (AI players) capable of playing a variety of games without any prior knowledge of them. Despite a proposal by Thielscher (2010) specifying how to include games of chance and incomplete information, the official GGP Competition that is organized each year in this area has only involved finite combinatorial games with perfect information. Apart from these three conditions, the games can be of any kind, e.g. they do not need to be only board ones. Also, they can involve any finite number of players including one, i.e., single-player games. The agents have to figure out how to play from scratch, being given only rules of the game to play. This is possible due to the fact that rules are written in a formal logic-based language, called game description language (GDL) discussed in details by Świechowski and Mańdziuk (2016b), which is a concise way to represent a state machine of the initial state, terminal states and all possible transitions. Therefore, agents are able to simulate games in order to do reasoning. Since 2007, all GGP winners, such as CadiaPlayer developed by Finnsson and Björnsson (2011) have been simulation-based. An introduction to GGP and review of the related literature can be found in the survey paper by Świechowski et al. (2015b).

General Video Game Playing (GVGP). GVGP is also referred to as General Video Game AI (GVGAI) (Perez-Liebana et al. 2016). The GVGP Competition, first hosted in 2014 at IEEE Conference on Computational Intelligence and Games, has been inspired by a more mature GGP Competition. However, it is not a successor to GGP but rather a different framework where the time to decide on an action is significantly shorter. The goal remains the same—to create agents capable of playing various games based only on rules given to them as a run-time parameter. The games of choice, in GVGP, are simple 2D realtime video games. The examples are Sokoban, Asteroids, Crossfire or Frogger. Similar to GGP, this framework has its own language for game definitions that features concepts such as sprites, levels and it is generally more suitable for said games. The agents have only 40 ms to make a move in the game. Despite such a vast difference in response times, both GGP and GVGP competitions are dominated by simulation-based approaches including ones based on MCTS. The competition is open for entries to various specific tracks such as single-player planning, single-player learning, two-player planning and level generation.

Arcade Video Games. Arcade Video Games is a group of simple computer games that gained popularity in the early 1980s. When the MCTS algorithm had been developed,

many researchers validated its applicability for games such as Pac-Man, Mario or Space Invaders. In many cases interesting modifications to standard MCTS were formulated.

Real Time Strategy Games. In Real Time Strategy games players combat each other through building units and simulating battles on a map. In general, there are two types of units which the players can deploy: economy-based and military. The former gather resources from certain places on the map; they can be either renewable or depletable depending on the game. These resources are spent on recruiting military units, which fight in battles controlled by the players. This combination of financial and tactical management results in RTS games posing a significant challenge to tree search algorithms. The three main issues associated with RTS games are:

– huge decision space—Starcraft,[1] the precursor of the genre, with the branching factor about 10^{50} or higher Ontañón et al. (2013), whereas Chess about 35 and Go roughly 180. Moreover, the player typically has to manage tens of units simultaneously,
– real-time—actions can be performed simultaneously, at any time, as opposed to turn-based games where players must take actions one at a time, in turns. Typically, actions in strategy games are divided into macro-management (e.g., strategic decisions) and micro-management (e.g., units' movement). The latter are taken with a higher frequency,
– partial observability—the map is covered with *fog of war*, meaning that some parts are hidden from the players. Handling partial observability in RTS games is presented in Uriarte and Ontañón (2017). Given the current observation and past knowledge the method estimates the most probable game state. This state is called *believe state* and is then sampled while performing MCTS.

2 Classic MCTS

2.1 Basic theory

MCTS is an iterative algorithm that searches the state space and builds statistical evidence about the decisions available in particular states. As mentioned in the "Introduction", formally, MCTS is applicable to MDPs. MDP is a process modelled as a tuple (S, A_S, P_a, R_a), where:

– S—is a set of states that are possible in an environment (state space). A specific state $s_0 \in S$ is distinguished as the initial state.
– A_s—denotes a set of actions available to perform in state s. The subscript S can be omitted if all actions are always available in the given environment.
– $P_a(s, s')$—is the transition function modelled by a probability that action a performed in state s will lead to state s'. In deterministic games, the probability is equal to 1 if the action in state s leads to s', whereas 0 if it does not.
– $R_a(s)$—is the immediate reward (payoff) for reaching state s by action a. In Markov games, where states incorporate all the necessary information (that summarizes the history), the action component can be omitted.

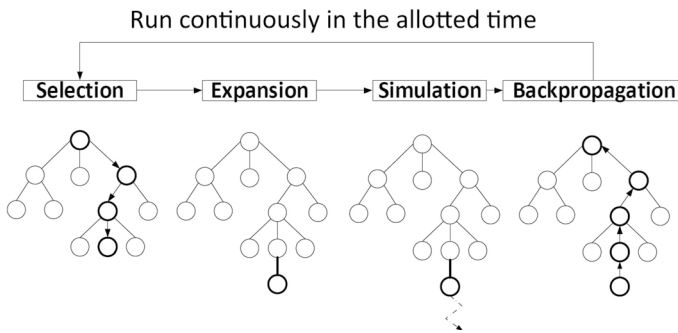[1] Blizzard Entertainment, Inc.

Fig. 1 Monte Carlo Tree Search phases

This formalism enables to model a simulated environment with sequential decisions to make in order to receive a reward after certain sequence of actions. The next state of the environment can be derived only using the current state and the performed action. We will use the game-related term *action* and a broader one—*decision*—interchangeably. The same applies for the term *game* and *problem*.

In non-trivial problems, the state space (e.g. a game tree) cannot be fully searched. In practical applications, MCTS is allotted some computational budget which can be either specified by the number of iterations or the time available for making a decision. MCTS is an *anytime* algorithm. This property means that it can be stopped at anytime and provide the currently best action (decision) using Eq. 1:

$$a^* = \arg \max_{a \in A(s)} Q(s, a), \tag{1}$$

where $A(s)$ is a set of actions available in state s, in which decision is to be made and $Q(s, a)$ denotes the empirical average result of playing action a in state s. Naturally, the more the iterations, the more confident the statistics are and the more likely it is that the recommended best action is indeed the optimal one.

Each iteration consists of four phases as depicted in Fig. 1:

(1) **Selection**—the algorithm searches the portion of the tree that has already been represented in the memory. Selection always starts from the root node and, at each level, selects the next node according to the *selection policy* also referred to as *tree policy*. This phase terminates when a leaf node is visited and the next node is either not represented in the memory yet or a terminal state of the game (problem) has been reached.

(2) **Expansion**—unless *selection* reached a terminal state, *expansion* adds at least one new child node to the tree represented in memory. The new node corresponds to a state that is reached by performing the last action in the *selection* phase. When *expansion* reaches the terminal state, which is extremely rare, then the current iteration skips directly to *backpropagation*.

(3) **Simulation**—performs a complete random simulation of the game/problem, i.e. reaching a terminal state and fetches the payoffs. This is the "Monte Carlo" part of the algorithm.

(4) **Backpropagation**—propagates the payoffs (scores), for all modelled agents in the game, back to all nodes along the path from the last visited node in the tree (the leaf one) to the root. The statistics are updated.

Fig. 2 The root denotes the starting state. The average scores [$Q(s, a)$] of actions leading to next states are shown inside the circles. The bars denote how many simulations started from a particular action. This is an abstract example—not taken from any particular game
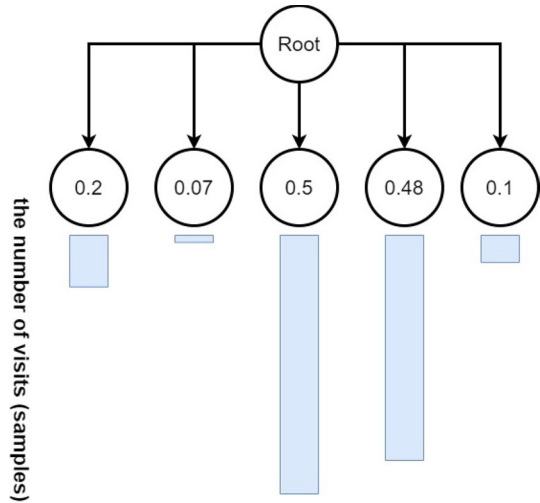
2.2 Tree policy

The aim of the selection policy is to maintain a proper balance between the exploration (of not well-tested actions) and exploitation (of the best actions identified so far). The most common algorithm, which has become de facto the enabler of the method is called Upper Confidence Bounds applied for Trees (UCT) introduced by Kocsis and Szepesvári (2006). First, it advises to check each action once and then according to the following formula:

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{ln[N(s)]}{N(s, a)}} \right\}, \tag{2}$$

where $A(s)$ is a set of actions available in state s, $Q(s, a)$ denotes the average result of playing action a in state s in the simulation performed so far, $N(s)$ is a number of times state s has been visited in previous iterations and $N(s, a)$—a number of times action a has been sampled in state s. Constant C controls the balance between exploration and exploitation. The usually recommended value of C to test first is $\sqrt{2}$, assuming that Q values are normalized to the [0, 1] interval, but it is a game-dependent parameter, in general.

Due to the UCT formula (c.f. 2) and random sampling, MCTS searches the game tree in an asymmetric fashion in contrast to traditional tree search methods such as minimax. The promising lines of play are searched more thoroughly. Figure 2 illustrates the type of asymmetrical MCTS search.

UCT is such a dominant formula for the exploration vs. exploitation trade-off in MCTS, that the approach is often called MCTS/UCT. However, there have been alternatives proposed. One of them is called *UCB1-Tuned*, which is presented in Eq. 3 from the work by Gelly and Wang (2006). The only new symbol here compared to Eq. 2 is σ_a which denotes the variance of the score of action a.

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln [N(s)]}{N(s, a)} \min \left(\frac{1}{4}, \sigma_a + \frac{2 * \ln [N(s)]}{N(s, a)} \right)} \right\}. \tag{3}$$

The *Exploration–Exploitation with Exponential weights* (EXP3) proposed by Auer et al. (2002) is another established tree policy algorithm. In contrast to UCT, it makes no assumption about the reward distribution. For a fixed horizon, EXP3 provides an expected cumulative regret bound against the best single-arm strategy. It is often used in combination with UCT in games with partial observability and/or simultaneous actions.

Thomson sampling is a different than UCT, heuristic approach to the *exploration vs. exploitation* problem. It preceded the MCTS algorithm (Thompson 1933) but has found its use as a tree policy as shown by Bai et al. (2013). Thomson sampling is a model that incorporates Bayesian concepts:

– parameterized likelihood function,
– prior distribution,
– set of past rewards,
– posterior distribution.

The model is updated as more information comes in. The expected reward of given action is sampled using the posterior distribution.

Typically, in the *expansion* phase, only one node is added to the tree which is a practical trade-off based on empirical experiments in lots of games. Adding more nodes visited in simulations would slow down the algorithm and increase its memory usage. The further down the road in a simulation a state is visited, the less likely it is for the actual game to reach it. However, if the states (and therefore potential nodes) belonged to the optimal path, it could be beneficial to add them. To make more use of simulations, Gelly and Silver (2011) proposes a method called *Rapid Value Action Estimation* (RAVE). In RAVE, an additional score is maintained called Q_{RAVE}, which is updated for each action performed in the simulation phase and not only for those chosen in the selection steps. The idea is to minimize the effect of the cold start, because especially at the beginning when there are not many samples, the MCTS algorithm behaves chaotically. One of the standard procedures is to linearly weight the RAVE evaluation with the regular average score [c.f. $Q(s, a)$ in Eq. 2] and reduce the impact of RAVE when more simulations have been performed. The new formula is shown in Eq. 4:

$$\sqrt{\frac{k}{3 * N(s) + k}} * Q_{RAVE}(s, a) + \left(1 - \sqrt{\frac{k}{3 * N(s) + k}} \right) * Q(s, a), \tag{4}$$

where k is the so-called RAVE equivalence constant.

There are a few more algorithms that modify or build upon the UCT formula such as *Move-Average Sampling Technique* (MAST) or *Predicate-Average Sampling Technique* (PAST). We recommend papers by Finnsson and Björnsson (2010, 2011) for details.

2.3 Transposition tables

The RAVE algorithm is often used with *Transposition Tables* (Kishimoto and Schaeffer 2002). They have been originally proposed as an enhancement to the alpha–beta algorithm,

which reduces size of minmax trees. As shown in paper by Kishimoto and Schaeffer (2002), within the same computational budget, the enhanced algorithm significantly outperforms the basic one without the transposition tables. The term "transpositions" refers to two or more states in the game that can be achieved in different ways. Formally speaking, let S_i, S_j, S_k be some arbitrary states in the game. If there exists a sequence of actions $(a_i, \ldots, a_k)$, that defines the path from state S_i to S_k, and a sequence of actions $(a_j, \ldots, a_k)$ which differs from $(a_i, \ldots, a_k)$ on at least one position, and transforms the state from S_j to S_k, then S_k represents transposed states. For simpler management, the MCTS tree is often modeled in such a way, that each unique sequence of actions leads to a state with a unique node in the tree. This may result in a duplication of nodes, even for indistinguishable states. However, one of the benefits of such a duplication, is the fact that, whenever an actual action in the game is performed, the tree can be safely pruned into the sub-tree defined by the state the action lead to. All nodes that are either above the current one or on an alternative branch cannot be visited anymore, so there is no need to store them anymore. The problem is more complicated when transpositions are taken into account, so there is one-to-one mapping between states and nodes. In such a case, the structure is no longer a tree per se, but a directed acyclic graph (DAG). When an action is played in the game, it is non-trivial to decide which nodes can be deallocated and which cannot because they might be visited again. In general, it would require a prediction model that can decide whether a particular state is possible to be encountered again. Storing all nodes, without deallocations, is detrimental not only for performance but mostly for the memory usage, which can go too high very quickly.

2.4 History heuristic

History Heuristic, pioneered by the work of Schaeffer (1989), is an enhancement to game-tree search that dates back to 1989. It dwells on the assumption that historically good actions are likely to be good again regardless of the state they were performed in. Therefore, additional statistics $Q(a)$—the average payoff for action a and $N(a)$—the number of times action a has been chosen, are stored globally. There are a few ways of how these statistics can be used. The most common approach is to use the $Q(a)$ score to bias simulations. In a simulation, instead of uniform random choice, the historically best action is chosen with ϵ probability, whereas with $1 - \epsilon$ probability, the default random choice is performed. This method is called $\epsilon - greedy$. An alternative approach is to use the roulette sampling based on $Q(a)$ or a particular distribution such as Boltzmann Distribution, which is the choice in the article from Finnsson and Björnsson (2010). A similar enhancement to History Heuristic is called *Last-Good Reply Policy* (LGRP), which is presented in Tak et al. (2012). The idea is to detect countermoves to particular actions and store them globally. For each action, the best reply to it is stored. The authors use the LGRP to rank the unexplored actions in the selection step and the simulation step. In the same article (Tak et al. 2012), the idea is further expanded to the so-called *N-grams Selection Technique* (NST). In NST, the statistics are stored not just for actions but for sequences of actions of length N.

2.5 Section conclusion

There is no "free lunch" (Wolpert and Macready 1997), which in the area of MCTS means that most modifications of the base algorithm introduce a bias. Such a bias may increase the performance in a certain problem but decrease in others. There are, however, certain

universal enhancements that can be applied such as parallelization. Opening books, discussed e.g. by Gaudel et al. (2010), are another example of a universally strong enhancement, but they are not always available. MCTS with opening books can be considered a *meta-approach*, because the first actions are chosen from the opening book and when, at some point in the game, the available actions are not present in the opening book, the remaining ones are chosen according to the MCTS algorithm. MCTS with relaxed time constraints, e.g. run for days, can even be used to generate the opening books for subsequent runs of the MCTS-based agents (Chaslot et al. 2009). Similarly, endgame books can be integrated with MCTS, but this is rarely applied as MCTS is a very strong end-game player and far less efficient at the beginning, where the whole game tree is to be searched.

In this section, we presented the MCTS algorithm as it was originally defined and discussed several standard modifications that were proposed before 2012. In the next sections, we focus exclusively on modifications proposed after 2012.

3 Games with perfect information

Almost any modern game features a huge branching factor, which poses a great challenge for search-based algorithms. In recent years, several MCTS modifications intended to tackle this problem such as action abstractions or rapid value estimations have been developed. Most of the modifications can be divided into the three categories:

– Action Reduction—limiting the number of available actions,
– Early Termination—ending the simulation phase (Fig. 1) prematurely,
– UCT alternatives—other approaches to MCTS selection phase.

In the following subsections, those three types of extension will be discussed with several examples. Another well-known solution, though recently less common, is to simplify the model of the game. One can construct a prediction model capable of anticipating promising moves in a given situation of the game. Graf and Platzner (2014) introduce a prediction model for the game of Go. The model features patterns extracted from *common fate graph*—a graph representation of the board (Graepel et al. 2001). These patterns are updated incrementally during the game both during the tree search and the playouts. This process allows reduction of the branching factor without affecting the prediction performance. Model simplification appears to be indispensable in video games. It allowed the use of the MCTS technique for campaign bots in a very complex game—*Total War: Rome II* as discussed in Preuss and Risi (2020). Swiechowski and Slęzak (2018) propose the so-called granular games as a framework for simplification of complex video games in order to apply MCTS to them.

3.1 Action reduction

In the expansion phase, the MCTS algorithm adds new nodes into the tree for states resulting after performing an action in the game. For many problems, the number of possible actions can be too high. In such a case, the tree is growing sideways, with limited chances for the in-depth inspection of promising branches. The aim of the *action reduction* methods is to limit this effect by eliminating some of the actions.

Fig. 3 Representation of options within a tree
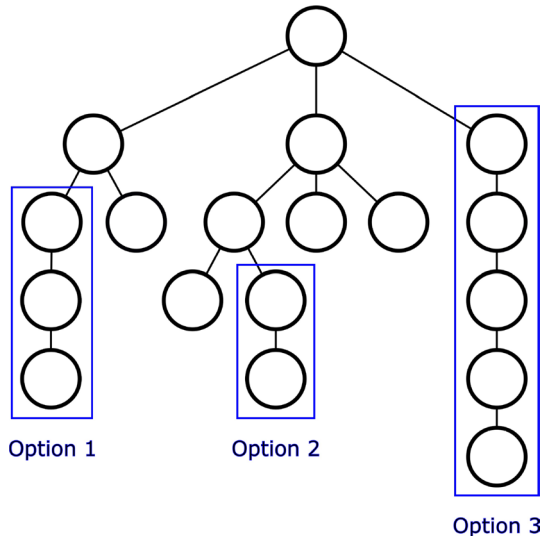
Sephton et al. (2014) apply heuristic move pruning to a strategy card game Lords of War.[2] The heuristic knowledge is split into two categories: measuring the fitness of a state card-based statistics and evaluating card positions based on heat maps. The former proves more efficient, but the heat maps make use of state extrapolation.

Justesen et al. (2014) extend the UCT Considering Durations (UCTCD) algorithm for combats in Starcraft[3] (Churchill and Buro 2013). Unlike UCTCD, where the algorithm searches for unit actions, the proposed method considers sequences of scripts assigned to individual units. The approach is elaborated even further—the experiments prove that the scripts appointed to groups of units are more effective in terms of branching factor reduction.

Similarly to the scripts, Option MCTS (O-MCTS) is presented by De Waard et al. (2016). The option is a predefined method of reaching a particular subgoal, e.g. conquering an area in an RTS game. In order to do this, options that contain several consecutive actions within the tree—Fig. 3 were introduced. O-MCTS selects options instead of actions as in vanilla MCTS. Once the selected option is finished (the subgoal has been reached), the expansion phase (Fig. 1) is conducted. This process allows a more in-depth search in the same amount of time and diminishes the branching factor. The paper includes an empirical optimisation of the basic parameters of the algorithm such as discount factor, maximum action time, maximum search depth and UCT constant C.

Another approach to reducing the branching factor is imposing constraints. Constraints determine situations to be avoided, i.e. actions which result in a defeat, whereas options lead to a specific sub-goal. Subramanian et al. (2016) propose a new technique of applying options and constraints to the search policy called Policy-Guided Sparse Sampling (PGSS). PGSS uses constraints for the possibility of pruning a node and options to bias the search towards the desired trajectories. Endowing MCTS with both techniques provides more accurate value estimation. Different action abstraction

---

[2] Black Box Games Publishing.

[3] Blizzard Entertainment.

Fig. 4 A sketch of tree pruning at depth $d = 2$ for BMCTS: **(a)** a beam of the three most visited nodes ($W = 3$) is selected, and **(b)** the rest is pruned
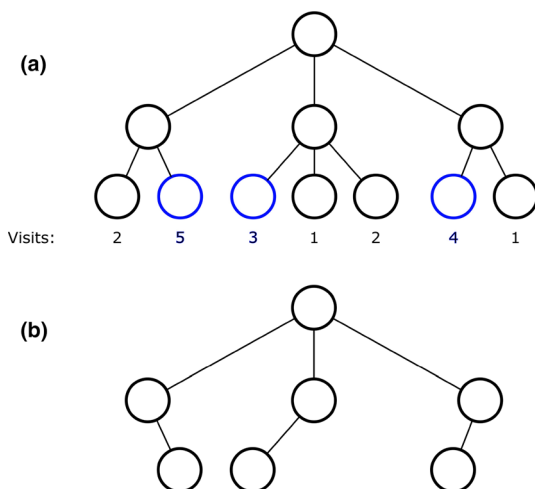
methods can be found in Gabor et al. (2019), Moraes et al. (2018), and Uriarte and Ontanón (2014).

Real Time Strategy games are challenging for MCTS due to a huge branching factor. Ontanón (2016) proposes Informed MCTS (IMCTS)—an algorithm dedicated for games with large decision spaces. IMCTS aggregates information of a game using two novel probability distribution models: Calibrated Naive Bayes Model (CNB) and Action-Type Independence Model (AIM). CNB is a Naive Bayes classifier with a simplified calibration of posterior probability. AIM derives the distribution only for legal actions in the current game state. The models are incorporated into the tree policy by computing the prior probability distributions in the exploration phase. The models are trained on the data from bot duels in an RTS testbed. The experiments show that the models outperform other MCTS approaches in the RTS domain.

One of the basic techniques of heuristic action reduction is Beam Search (Lowerre 1976). It determines the most promising states using an estimated heuristic. These states are then considered in a further expansion phase, whereas the other ones get permanently pruned. Baier and Winands (2012) combine Beam Search with MCTS (BMCTS). BMCTS features two parameters: beam width W and tree depth d. They describe how many of the most promising nodes should be selected at an arbitrary depth—Fig. 4. The nodes outside the beam are pruned. Manual experimentation is used for finding optimal values of parameters: C—exploration coefficient, L—simulation limit, W—beam width. Different variants of MCTS with a variable depth can be found in Pepels and Winands (2012), Soemers et al. (2016), Zhou et al. (2018).

Ba et al. (2019) present Variable Simulation Periods (VSP-MCTS). The method extends the decision variables with the simulation time. In VSP-MCTS actions are selected with respect to the possible simulation time before the next selection phase. The available set of action is initially limited to the most promising ones and gradually progressively expanded (unpruned). To achieve this, the authors combine *progressive widening* and *Hierarchical optimistic optimization applied to tree* (HOOT). This approach supports the balance between the frequent action choice and the time dedicated to simulations.

Progressive Bias (Chaslot et al. 2008b) is another method of heuristic action reduction. Progressive bias extends the UCB function with a heuristic value, which is reduced as the number of node visits increases. Gedda et al. (2018) propose a progressive win bias:

$$P = W \frac{H_i}{T_i(1 - \bar{X}_i) + 1},$$ (5)

where H_i is the heuristic value, $\bar{X}_i$ is the average reward for the node, T_i is the number of node visits and W is a positive constant controlling the impact of the bias. Unlike the basic approach, in this formula the heuristic value depends on the number of losses. Other extensions to UCB can be found in Liu and Tsuruoka (2015), Mandai and Kaneko (2016), Tak et al. (2014) and Yee et al. (2016). Perick et al. (2012) compare different UCB selection policies. Ikeda and Viennot (2013a) involve Bradley–Terry Model of static knowledge (Coulom 2007) to bias UCB. A Bias in MCTS can affect the performance in terms of suboptimal moves (Imagawa and Kaneko 2015).

Demediuk et al. (2017) introduce a set of *Outcome-Sensitive Action Selection* formulas to create an MCTS agent in a fighting game. The equations evaluate actions with a score function, which in this case is the difference between the player's and the enemy agent's health. The action with the score closest to zero is selected.

A large branching factor can be limited by manipulation of the exploration term in the tree policy function (UCT, Eq. 2), which is covered in Sect. 3.2.

3.2 UCT alternatives

In the vanilla MCTS, every state or state–action pair is estimated separately according to the Eq. 2 (on p. 10). Some games like the game of Go feature actions which do not strictly depend on the state in which they were played.. To handle this the *all-moves-as-first* (AMAF) heuristic has been developed (Brügmann 1993). It evaluates each move a by the total reward $\bar{W}_a$ divided by the number of simulations in which a was ever played $\bar{N}_a$.

Rapid Value Estimation, which has already been discussed in Sect. 2.2, is a generalized idea of AMAF to search trees. It shares action values among subtrees. The value estimation of action a is performed with respect to all the simulations involving a at the subtrees. This process provides more data for each action in comparison to vanilla MCTS, although the additional data may be needless in some cases. In RTS games, any change in positions of nearby units can mislead the action value. The MCTS–RAVE method handles this observation through interpolating the unbiased Monte Carlo and AMAF value estimate. Thus, the score of state–action pair is as follows (Gelly et al. 2012):

$$\bar{Z}(s,a) = (1 - \beta(s,a))\frac{W(s,a)}{N(s,a)} + \beta(s,a)\frac{\bar{W}(s,a)}{\bar{N}(s,a)},$$ (6)

where $\bar{N}(s,a)$ is the AMAF visit count, $\bar{W}(s,a)$ is the corresponding total reward and $\beta(s,a)$ is a weighting parameter that decays from 1 to 0 as the visit count $N(s,a)$ increases. In recent years, several RAVE improvements within the GGP domain have been proposed— Sironi and Winands (2016) and Cazenave (2015). Sarratt et al. (2014) and Sironi and Winands (2019) discuss effects of different parameterization policies on convergence.

A case analysis of the classical position in the Go game (Browne 2012) shows that the inclusion of domain knowledge in the form of heavy playouts (see also Sect. 4.3) dramatically improves the performance of MCTS, even in the simplest flat version. Heavy playouts

reduce the risk of convergence to a wrong move. Exploration constant has been found empirically to give the best convergence. Incorporating domain-independent RAVE technique can actually have a negative effect on performance, if the information taken from playouts is unreliable.

The non-uniformity of tree shape is another serious challenge for the MCTS method. The Maximum Frequency method (Imagawa and Kaneko 2015) adjusts a threshold used in recalculation of playout scores into wins or losses. The method is based on maximizing the difference between the expected reward of the optimal move and the remaining ones. The Maximum Frequency method observes the histogram of the results and returns the best threshold based on it. Algorithm parameters are manually adjusted in order to minimize failure rate. The proposed solution has been verified on both random and non-random trees.

Algorithm 1: MCTS with adaptive playouts (Graf and Platzner, 2016)

---

function $MCTS(s,\ b)$
    if $s \notin tree$ **then**
        | $expand(s)$
    if $totalCount(s) > 32$ **then**
        | $b \leftarrow winrate(s)$
    $a \leftarrow select(s)$
    $s' \leftarrow playMove(s,a)$
    if $expanded$ **then**
        | $(r,g) \leftarrow playout(s')$
        | $policyGradientUpdate(g,b)$
    else
        | $r \leftarrow MCTS(s',b)$
    $update(s,a,r)$
    return r

The playout policy can be changed during a tree search. Graf and Platzner (2016, 2015) show that updating policy parameters (see Algorithm 1) increases the strength of the algorithm. An adaptive improvement of the policy is achieved by the gradient RL technique.

Despite its effectiveness in GGP, there are some domains in which the UCT selection strategy does not work nearly as well. Chess-like games featuring many tactical traps pose a challenging problem for MCTS. One of the tactical issues are optimistic actions—initially promising moves (even leading to a win), but easy to refute in practice (Gudmundsson and Björnsson 2013). Until MCTS determines the true refutation, the move path leading to an optimistic action is scored favorably. To diminish this positive feedback UCT (see Eq. 2) has been extended with *sufficiency threshold* α (Gudmundsson and Björnsson 2013). It replaces constant C with the following function:

$$\hat{C} = \begin{cases} C & \text{when all } Q(s,a) \leq \alpha, \\ 0 & \text{when any } Q(s,a) > \alpha. \end{cases} \tag{7}$$

When the state–action estimate $Q(s,a)$ is sufficiently good (the value exceeds α) then the exploration is discounted; otherwise the standard UCT strategy is applied. This approach allows rearrangement of the order of moves so that occasional fluctuations of the estimated value can be taken into account.

Hybrid MCTS-minimax algorithms (Baier and Winands 2014; Baier and Mark 2013) have also been proposed for handling domains with a large number of tactical traps. In this

context, the basic drawback of vanilla MCTS is the averaging of the sampled outcomes. It results in missing crucial moves and underestimation of significant terminal states. To overcome this, the authors employ shallow-depth minimax search at different MCTS phases (Sect. 2). Since the minimax search is performed without any evaluation functions, the method does not require domain knowledge and finds its application in any game. Parameters tuning covers exploration constant, minimax depth and visit counter limit.

3.3 Early termination

MCTS in its classical form evaluates game states through simulating random playouts i.e. complete play-throughs to the terminal states. In dynamic domains, where actions are taken in a short period of time, such outcome sampling appears to be a disadvantage of vanilla MCTS. One of the alternatives is to *early terminate* the playouts—a node is sampled only to an arbitrary depth and then the state is evaluated similarly to the minimax search. One of the key issues is to determine the cut-off depth. Earlier terminations save the simulation time although they result in evaluation uncertainty. On the other hand, later terminations cause the algorithm to behave more like vanilla MCTS.

Lorentz (2016) finds the optimal termination point for diverse evaluation functions in the context of three different board games. The results prove that even a weak function can compare favorably to a long random playout. The evaluation functions can also augment the quality of simulations (Lanctot et al. 2014). The proposed method replaces $Q(s, a)$ in the classic UCB selection—Eq. 2—with the following formula:

$$\hat{Q}(s, a) = (1 - \alpha)\frac{r_{s,a}^{\tau}}{n_{s,a}} + \alpha v_{s,a}^{\tau}, \tag{8}$$

where $r_{s,a}^{\tau}$ is the cumulative reward of the state–action pair with respect to the player τ, $n_{s,a}$ is the visit count, $v_{s,a}^{\tau}$ is the implicit minimax evaluation for that player and *alpha*— weighting parameter. The minimax evaluations are computed implicitly along with standard updates to the tree nodes and maintained separately from the playout outcomes. Hierarchical elimination tournament with 200 games has been used to tune the parameters of the algorithm.

The generation of evaluation functions is a challenging research especially in GGP (see Sect. 1.3). Waledzik and Mańdziuk (2011, 2014) construct a state evaluation function in GGP automatically based solely on the game rules description, and apply this function to guide the UCT search process in their *Guided UCT* method.

Moreover, evaluation functions can be approximated with neural networks (Wu et al. 2018). Goodman (2019) trains an evaluation network on the data from offline MCTS, which allows omission of the simulation step during the tree search.

3.4 Opponent modelling

Opponent modelling is a challenge that appears in multi-player games. The more information we possess or can infer about the opponent, the better simulation model of their actions we can build. Opponent modelling is a complex topic that is related to games, game theory and psychology. The model of the opponent can be independent of the algorithm an AI agent uses. In this section, we will only scratch a surface and show a

few examples of how opponent modelling is incorporated into the MCTS algorithm in the context of games with perfect information.

The paper by Baier and Kaisers (2020) emphasizes the idea of "focusing on yourself" during MCTS simulations. The authors argue that in multi-player games the agent's own actions have often a significantly greater impact on the final score than appropriate opponent prediction. Naturally, there are exceptions to this, such as complex tactical situations. The proposed approach considers abstractions over opponent moves (OMA) that only consider the history (past moves) of the main agent's moves. The abstracted moves share the same estimates that are combined with the UCT formula in various ways. The authors report a better win-rate than vanilla MCTS in six perfect-information, turn-taking, board games with 3+ players.

Fighting games pose a special challenge, since they require a short response time. However, players often repeat the same patterns, e.g., sequences of punches and kicks. An algorithm designed for such a game is described by Kim and Kim (2017). An action table is used to describe the opponent's playing patterns. A default action table has three columns depending on the distance to the opponent. Each column corresponds to a particular distance range. After each round the action table is updated based on the observed action frequency.

Opponent modelling is also present in games such as Pacman. In a paper by Nguyen and Thawonmas (2012), the main enhancement was related to the prediction of the opponent's moves, which reduced the number of states analyzed. In the backpropagation phase, the node reward scheme combines the final playout score, but also the simulation time. The playouts are not completely random, the space of the analyzed moves is limited by heuristic rules.

4 Games with imperfect information

The success of MCTS applications to games with perfect information, such as Chess and Go, motivated the researchers to apply it to other games such as card games, real-time strategy (RTS) and other video games. The main feature, which challenges the effective applicability of tree search algorithms is imperfect information. For instance, in RTS games the "fog of war" often appears, covering the map including the enemy's units.

Imperfect information can increase the branching factor significantly, as every hidden element results in a probability distribution of possible game states. Those states should be included within the tree in the form of additional nodes.

In order to handle imperfect information there have been developed several improvements of the methods such as Perfect Information MCTS (PIMC) (see Sect. 4.1) and Information Set MCTS (ISMCTS) (see Sect. 4.2).

Rules of the game and human expert knowledge can be used to increase the reliability of playouts (see Sect. 4.3 for heavy playouts), or to optimize the tree development process (see Sect. 4.4 for policy update). The highest quality results can be achieved by combining several methods. Papers of this type usually concern algorithms created for competitions and is described in Sect. 4.5.

The last issue related to imperfect information discussed in this section is especially relevant for games and concerns modelling the opponent (see Sect. 4.6).
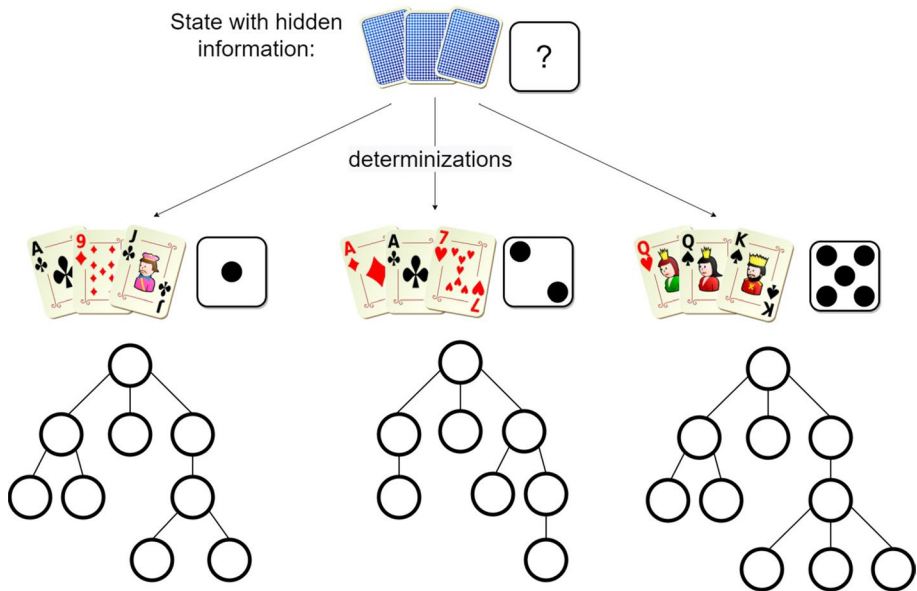
Fig. 5 Determinization method. The dice and the cards in the opponent's hand are sampled in order to form possible subsets. Then each subset rolls out a new subtree

4.1 Determinization

One method of solving the problem of randomness in games is determinization, which involves sampling from the space of possible states and forming games of perfect information for each such sampling, correspondingly. The sampling requires setting a specific instance (value) for each unknown feature, thus determining it, so it is no longer unknown. For instance, determinization may involve guessing the cards an opponent might have in their hand in a card game (see Fig. 5).

Perfect Information MCTS (PIMC) assumes that all hidden information is determined and then the game is treated as a perfect information one with respect to a certain assumed state of the world. PIMC faces three main challenges (Cowling et al. 2012a):

- strategy fusion—since a given imperfect information state can contain tree nodes with different strategies, a (deterministic) search process may make different decisions in that state depending on particular determinization,
- non-locality—optimal payoffs are not recursively defined over subgames,
- sharing computational budget across multiple trees.

Cowling et al. (2012b) apply this method to deal with imperfect information in the game "Magic: The Gathering" (MtG). MtG is a two-person, collectible card game, in which players draw cards for their starting hand. The cards are not visible to the opponent until they are placed on the table, therefore MtG features partially observable states and hidden information. To approach this game with MCTS, Cowling et al. (2012b) propose multiple perfect information trees. This method considerably increases the branching factor, so a few enhancements are employed such as move pruning with domain

Fig. 6 Two information sets:red (with 1 inside the circles) and blue (with 2 inside the circles). Each set contains nodes indistinguishable from a player's perspective. (Color figure online)
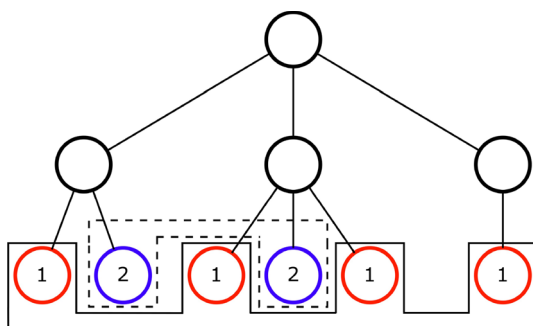
knowledge and binary representation of the trees. The improvements reduce CPU time per move, rendering the computational budget more manageable.

4.2 Information sets

The concept of Information Sets has been introduced by Frank and Basin (1998). It is illustrated in Fig. 6. The game starts with a chance node, which has three possible moves (circle) for the first player. The moves of the second player are split into two information sets: red (1) and blue (2). Nodes in each information set are treated as equivalent since the player is only aware of the previous moves, not of the moves below the chance node.

Cowling et al. (2012a) propose Information Set MCTS (ISMCTS) to handle the determinization problems: strategy fusion and suboptimal computational budget allocation. The information set is an abstract group of states, which are indistinguishable from a player's point of view (Wang et al. 2015). Instead of a single state, each node of ISMCTS comprises an information set that allows the player to act upon their own observations. Information sets bring all the statistics about moves in one tree, providing the use of the computational budget in the optimal way. Moreover, applying information sets reduces the effects of strategy fusion, as the search is able to exploit moves that are beneficial in many states of the same information set. For an extension of ISMCTS that incorporates advanced opponent modelling, please refer to Sect. 4.6.

In games where non-locality is a key factor, Lisỳ et al. (2015) introduce Online Outcome Sampling (OOS). The algorithm extends Monte Carlo Counterfactual Regret Minimization with two features: building its tree incrementally and targeting samples to more situation relevant parts of the game tree. Performance evaluated on both OOS and ISMCTS shows that only OOS consistently converges close to Nash Equilibrium, and it does so by adapting an offline equilibrium computation to a limited time. This is achieved using the information set about the current position in the game. To tune efficiency of the algorithm sampling technique is used for strategy probabilities parameters.

Furtak and Buro (2013) introduce Recursive Imperfect Information Monte Carlo (IIMCTS) which is used for playouts with a fixed maximum recursive depth. One of the differences to Cowling et al. (2012a) is that ISMCTS implicitly leaks game state information by restricting the playouts to be consistent with the player's hidden/private information (e.g. cards in the player's hand in Poker). IIMCTS prevents the leakage by not adapting the players' playout policies across playouts.

Moreover, the move computation in IIMCTS allows use of the inference mechanism at any node during the lookup, while Information Set MCTS considers nodes only at the root or sets uniform distributions of the hidden state variables.

Another collectible card game with imperfect information in which MCTS was used is Pokemon (Ihara et al. 2018). In this research, authors compare the effectiveness of determinization versus information sets within MCTS. The results indicate a significant advantage of information sets as ISMCTS reduces the effects of strategy fusion and allocates computing resources more effectively. A similar approach has been presented for Scopone game (Di Palma and Lanzi 2018).

Sephton et al. (2015) propose the RobustRoulette extension for the selection mechanism in ISMCTS, which averages across many possible states and has a positive effect on the playing strength.

Uriarte and Ontañón (2017) use a similar method to generate a single believe state—an estimation of the most probable game state. The estimation derives from the current game state, which is not fully observable to the player. This situation is often encountered in Real Time Strategy games (see Sect. 1.3), where enemy units' positions are unknown (covered with "fog of war").

The concept of Information Sets, which is inherent in MCTS application to solving imperfect information problems, has been recently utilized also in domains other than games, e.g. in Security (Karwowski and Mańdziuk 2015, 2019a, b, 2020). A more detailed discussion is presented in Sect. 7.2.

4.3 Heavy playouts

Heavy playouts is a term defined in the classical paper by Drake and Uurtamo (2007). Authors define two ways of adding domain-specific knowledge into vanilla MCTS: move ordering and heavy playouts. The first one concerns the issue of creating new nodes during the tree's expansion phase, whereas the second one uses a heuristic to bias the "random" moves during the rollout.

The implementation of heavy playouts results in more realistic games in the simulation phase. However, it should be remembered that playouts need some degree of randomness to efficiently explore the solution space. Experiments in a card game *Magic: the Gathering* show the importance of the rollout strategy (Cowling et al. 2012b). A deterministic expert-rule-based rollout strategy provides an inferior performance to utilizing the reduced-rule-based player that incorporates some randomness within its decisions. This is true even though the expert-rule-based algorithm used as a standalone player is more efficient than the reduced-rule-based algorithm. On the other hand, relying on completely random playouts gives poor performance.

GGP defines a special challenge for the AI agents since they are expected to be able to play games with very different rules (see Sect. 1.3). On the one hand, a fully random playout strategy is universal although, it avoids the use of domain knowledge gained from the past simulations. On the other hand, too much heuristics can lead to omitting a potentially important group of moves. Therefore, one has to find a balance between these two extremes. Świechowski and Mańdziuk (2014) present a method based on switching between six different strategies (from simple random or History Heuristic to more advanced ones). Their solution has been implemented in the MINI-Player agent, which reached the final round in the 2012 GGP Competition. The method was further extended towards handling single-player games more efficiently (Świechowski et al. 2016).

Genetic programming (GP) has been successfully used to enhance the default MCTS policy in Pac-Man (Alhejali and Lucas 2013). The GP system operated using seven categories of elements such as IF ELSE statements, numerical operators and comparison operators. Based on these components, a tree policy function was evolved. After sweeping for optimal value of nine different GP and MCTS algorithm parameters, it allowed achievement of a 18% greater average score than the random default policy. Moreover, the constructed function did not introduce a significant computational overhead.

Experiment with different 'weights' of heavy playouts are described by Godlewski and Sawicki (2021). Increasing the amount of expert knowledge added to the playout function generally improves the performance of the AI agent. However, for multistage card games like "The Lord of the Rings", each stage might have a different value of optimal playout weight.

4.4 Policy update

The previous subsection concerned the introduction of knowledge through heavy playouts. In this section, the methods of modification of the tree building policy is presented.

The classical RAVE approach (see Sect. 2.2) has been extended by Kao et al. (2013). They introduce the RIDE method (Rapid Incentive Difference Evaluation) where the default MCTS policy is updated by using differences (9) between action values for the same state s.

$$D^{RIDE}(a, b) = E\{Q(s, a) - Q(s, b)|s \in S\}. \tag{9}$$

The RAVE and RIDE approaches differ in their sampling strategy. RAVE applies an independent sampling strategy, whereas RIDE applies a pairwise sampling strategy.

The playout policy can be updated by moves, but also based on features of the moves. In the solution presented by Cazenave (2016), the tree is developed exactly as UCT does. However, the playouts have weights for each possible move and choose randomly, proportionally to the exponential of the weight.

Heuristics can be added at the tree development phase or at the playouts phase. Naturally, it is also possible to combine these two approaches (Trutman and Schiffel 2015). The authors propose three schemes (playout heuristics, tree heuristics, combined heuristics) in the GGP framework. They have been compared with MAST (Finnsson and Björnsson 2008), which is used to bias random playouts, and the RAVE technique for trees. None of the proposed heuristics is universally better than the others and their performance depends on a particular type of game.

The default policy update using heuristics is one of the key enhancements in MCTS experiments for Hearthstone. Since the release in 2014, every year the game has attracted thousands of players, quickly becoming the most popular card game, exceeding 100 million of users worldwide in 2018.[4]

The game state includes imperfect information—each player's hand is hidden for the opponent. Moreover, the game-play imposes random events and complex decisions. Because of this, state evaluation for this game has always been under research. Santos et al. (2017) propose heuristic functions for evaluating subsequent states based on hand-picked features. In addition, they enhance the state search with a database of cards, which contains

---

[4] https://videogamesstats.com/hearthstone-facts-statistics/.

Fig. 7 Comparison of MCTS enhancements for Mario arcade game (Vanilla MCTS—yellow line, Macro Actions—red line, Best combination—blue line). Figure reproduced from Jacobsen et al. (2014). (Color figure online)

cards already played by the opponent. The database is used for prediction of cards in the next turn.

State evaluation with a value network (Świechowski et al. 2018) is another approach. The authors employ the network to predict the winner in a given state. To transform the state vector as the input of the network, they use skip-gram model instead of hand-picking. A value network from Zhang and Buro (2017) maps a game state to a vector of probabilities for a card to be played in the next turn. Training data for both networks is generated by bot players.

4.5 Master combination

A special category of research papers are those describing methods developed for game AI competitions (Świechowski 2020). Each year, there are many events in which AI players compete against each other. One of the largest ones among academic community members is the IEEE Conference of Games (CoG) (formerly Computational Intelligence and Games—CIG) with over ten categories for popular games such as Angry Birds, Hearthstone and StarCraft; and more general disciplines (General Video Game Playing—GVGP, MicroRTS, Strategy Card Game) e.g. IEEE-CoG (2020). Open competitions stimulate development of new methods, but particularly good results are achieved by combining multiple methods masterfully. Strong problem-specific enhancements can address different phases of the MCTS algorithm.

One example of such a master combination is the algorithm developed to win the Pac-Man versus Ghost Team competition (Nguyen and Thawonmas 2012). The team of ghosts was controlled by MCTS to catch Pac-Man. It is intrinsically interesting that the winner on the Pac-Man side was also an agent using the MCTS algorithm. The set of several improvements was described by Pepels and Winands (2012) and Pepels et al. (2014). Most of them refer to heuristic changes in the policy:

– a tree with variable length edges (taking the distance between decision points into account),
– saving the history of good decisions to improve the playout policy,
– changes of policy (different modes to eliminate some obviously wrong decisions).

Another popular challenge is the Mario arcade game (see Fig. 7). Jacobsen et al. (2014) describe a combination of several improvements to vanilla MCTS that shows excellent results. However, most of them could be classified as an incorporation of domain knowledge into the general algorithm: MixMax (high rewards for good actions), Macro Actions (avoid monsters in a sequence of moves), Partial Expansion (eliminate obvious choices), Hole Detection (extra heuristic to jump over a deadly trap). The paper contains explanation of optimization process for playout depth and exploration constant.

GVGP is a class of AI competitions where agents do not know in advance what game will be played (see Sect. 1.3). The vanilla MCTS is an algorithm which does not rely on domain-specific heuristics, so it is naturally a good candidate for such problems. In the first GVG-AI competition at CIG 2014 (Preuss and Gunter 2015), the vanilla MCTS agent surprisingly came in 3rd place, achieving a win-rate of about 32%. These impressive results attracted attention to the MCTS method, however, Nelson (2016) found out that simply increasing the playout budget is not enough to significantly improve the win-rate.

Soemers et al. (2016) describe several enhancements of the method, such as N-gram Selection Technique (Tak et al. 2012)—a playout policy update introducing a bias in the steps towards playing sequences of actions that performed well in earlier simulations. The idea of *Loss Avoidance* is particularly interesting, where an agent tries to reduce losses by immediately searching for a better alternative whenever a loss is encountered. A combination of enhancements allows an increase in the average win-rate in over 60 different video games from 31.0 to 48.4% in comparison with a vanilla MCTS implementation.

Frydenberg et al. (2015) show that classical MCTS is too cautious for some games in GVGP. Better results are observed after increasing risk-seeking behavior by modification of the exploitation part of UCT. Undesirable oscillations of the controlled character can be eliminated by adding a reversal penalty. Macro-actions and limiting tree expansion are other important improvements.

Adding a heuristic into the algorithm is a common procedure. Guerrero-Romero et al. (2017) formulate a conclusion that the best strategy is to change the type of utilized heuristic during the game. They investigate four heuristics: winning maximization, exploration maximization, knowledge discovery and knowledge estimation. It is beneficial to start with enhancing knowledge discovery or knowledge estimation and only then add more winning or exploration heuristics. These rules could be described by a high level meta-heuristic extension.

An interesting combination of MCTS enhancements is shown by Goodman (2019) describing their efforts to win the Hanabi competition at CIG in 2018. Hanabi is a cooperative game, so communication is an essential factor. The convention used for communication is a particularly interesting sub-problem for AI. When all players are controlled by the same AI agent, hidden information can be naturally shared. In the case of the cooperation between different agents, the communication convention has to be established. For this task, a novel re-determinizing ISMCTS algorithm has been proposed, supported by adding rules to restrict the action space. The rules of competition give an agent 40 ms time for a decision but these were easily achieved using a supervised neural network with one hidden layer.

Nowadays, AI agents usually outperform human players, which could be perceived as a problem when human enjoyment of the game is necessary. This topic has been studied by Khalifa et al. (2016) in terms of GVGP. Several modifications of the UCT formula are introduced, such as an action selection bias towards repeating the current action, making pauses, and limiting rapid switching between actions. In this way, the algorithm

behaves in a manner similar to humans, which increases the subjective feeling of satisfaction with the game in human players.

This section can be concluded by stating that the best results are achieved by combining different types of extensions of the MCTS algorithm. However, there is no one-size-fits-all solution. So this should encourage scientists to experiment with various methods.

4.6 Opponent modelling

Problem of opponent modelling is also relevant for games with imperfect information. This section presents a few examples, which incorporate analysis of the opponent into the MCTS algorithm.

Goodman and Lucas (2020) analyse how accurate the opponent model must be in a simple RTS game called *Ground War*. Two types of opponents are considered—one that uses the same policy as the MCTS-based agent and a parameterized heuristic model. The authors not only have shown that having an accurate opponent model is beneficial to the performance of the MCTS-based agents but also compared its performance against another agent based on RHEA. The findings show that the latter is more sensitive to the quality (accuracy) of the opponent model.

A game in which opponent modelling is crucial in order to succeed is Poker. Cowling et al. (2015) extend ISMCTS with the ability to perform inference or bluffing. Inference is a method of deriving the probability distribution of opponent moves based on the observed data (actions already done). However, the opponent can assume that his moves are inferred, and then bluffing comes into play—they can take misleading actions deliberately. Bluffing as an ISMCTS extension is realized by allowing sampling of states that are possible only from the opponent's perspective, the search agent's point of view is neglected. This technique is called self-determinization. The frequency with which the determinization samples each part of the tree is also recorded, thereby allowing performance of inference.

Another example of opponent modelling is Dynamic Difficulty Adjustment (DDA; Hunicke 2005). DDA is a family of methods focused on creating a challenging environment for the player. In order to keep the player engaged during the whole game, DDA changes the difficulty of encountered opponents according to the player's actual skill level.

A topic which can be related to opponent modelling is making an MCTS-driven player to model a certain behavior. Here, instead of predicting what other players may do in the game, the goal is to make the agent mimic a certain group of players. Baier et al. (2018) consider the problem of creating a computer agent that imitates a human play-style in a mobile card game *Spades*. The human-like behavior leads to a more immersive experience for human players, increases retention as well as enables prediction what human players will do, e.g. for testing and QA purposes. The MCTS-based bots are often very strong but rarely feel human-like in their style of playing. The authors train neural networks using approximately 1.2 million games played by humans. The output of the networks is the probability, for each action, that such an action would be chosen by a human player in a given state. Various network architectures were tested, but all of them were shallow, i.e., not falling into the category of deep learning. Next, various *biasing techniques*, i.e., ways of including the neural networks in the MCTS algorithm were tested. The notable one was based on *knowledge bias*, in which the UCT formula (c.f. 2) was modified as in Eq. 10:

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln [N(s)]}{N(s,a)}} + C_{BT}\sqrt{\frac{K}{N(s) + K}} P(m_i) \right\}, \qquad (10)$$

where $P(m_i)$ is the output from the neural network trained on human data; C_{BT} is a weight of how the bias blends with the UCT score and K is a parameter controlling the rate at which the bias decreases. Values of C_{BT} and K have been experimentally optimized.

5 Combining MCTS with machine learning

5.1 General remarks

ML has been on the rise in the computer science research. It has lead to many breakthrough innovations and its popularity and applicability in practical problems is still growing rapidly. The field of ML consists of techniques such as neural networks, decision trees, random forests, boosting algorithms, Bayesian approaches, support vector machines and many others. Although, MCTS is not typically associated with ML, but rather with the classical AI and search problems, it can be considered a ML approach, and a type of reinforcement learning, specifically. However, MCTS is often combined with other ML models to great success. ML, as a general technique, can be used in various ways and contexts. A good example is creating an evaluation function through ML. We have already referred to such an approach in Sect. 3.3 and works such as authored by Wu et al. (2018) and Goodman (2019). In this section, this topic will be investigated in more details. In Sect. 4, we have also already shown that convolutional neural networks (CNNs) can be useful tools for map analysis in RTS games (Barriga et al. 2017).

5.2 The AlphaGo approach: policy and value

The ancient game of *Go* has been considered the next *Grand Challenge* of AI in games since *Chess* and the famous match between Garry Kasparov and Deep Blue, which was analysed by Campbell et al. (2002). Following Gelly et al. (2012), *Go* is considerably more difficult for computer programs to play than Chess. Until 2006, computer programs were not stronger than average amateur human player. In 2006–2007, MCTS was introduced and became a quantum leap in terms of bots competence levels. MCTS has become the first major breakthrough in computer *Go*, whereas MCTS combined with ML techniques have became the second major breakthrough. In 2015, *AlphaGo* program created by Silver et al. (2016) under the auspices of Google, has become first to beat a professional human player, Fan Hui, on a regular board without a handicap. Subsequently, in 2016, *AlphaGo* won 4–1 in a match versus Lee Sedol (Silver et al. 2017), one of the strongest *Go* players of all time. By many scientists, e.g. Wang et al. (2016), this result was considered one of the most important achievements of AI in the recent years. *AlphaGo* inspired many approaches that combine MCTS and ML models. Because many of them share the same basic structure, we will now outline it. The structure is based on two ML models that represent the so-called **value** and **policy** functions, respectively. Such models are often referred to as "heads" if they are constructed using neural networks, which is the most common choice to modelling them.
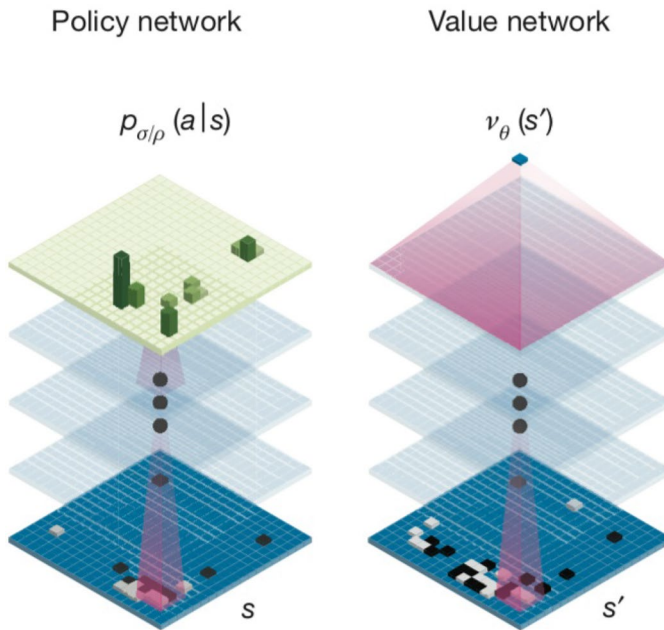
Fig. 8 Policy and value deep convolutional networks used in AlphaGo. The figure is reproduced from Silver et al. (2016)

- **Value function**—a function $v^*(s)$ that approximates the outcome of the game in a given state s. A perfect (non-approximate) value function is often used when solving a game, e.g. *Checkers* by Schaeffer et al. (2007). This is a similar concept to the state evaluation function. The use of them can be technically the same, however, the term "state evaluation function" is usually used when the function is static, whereas value function is self-improving through ML.
- **Policy function**—the policy, denoted by $p(a|s)$ informs which action—a—should be chosen given state s. A policy can be deterministic or stochastic. It is often modelled as a probability distribution of actions in a given state.

The AlphaGo approach employs deep convolutional networks for modelling both value and policy functions as depicted in Fig. 8. In contrast to a later iteration of the program called *AlphaZero*, AlphaGo's policy function is kick-started by supervised learning (SL) over a corpus of moves from expert human players. Readers interested in the details of the ML pipelines pursued in various versions of *AlphaGo* and *AlphaZero* are advised to check the papers from Silver et al. (2018, 2016, 2017). The initial policy is called the SL policy and contains 13 layers (Silver et al. 2016). Such a policy is optimized by a RL policy through self-play and the policy gradient optimization method. This adjusts the SL policy to correctly approximate the final outcome rather than maximize the predictive accuracy of expert moves. Next, the value network is trained to predict the winner in games played by the RL policy.

The authors propose to use a slightly modified formula for selection in MCTS compared to 2:

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + \frac{P(s, a)}{1 + N(s, a))} \right\}, \tag{11}$$

where $P(s, a)$ is the prior probability and the remaining symbols are defined in the same way as in Eq. 2.

When a leaf node is encountered, its evaluation becomes:

$$V(s_L) = (1 - \lambda)v_\Theta(s_L) + \lambda z_L, \tag{12}$$

where $v_\Theta(s_L)$ is the value coming from the value network, z_L is the result of a rollout performed by the policy network and λ is the mixing parameter.

5.3 MCTS + neural networks

In this section, we outline approaches that use both the value and the policy functions, so they can be considered as inspired by the AlphaGo approach. Chang et al. (2018) propose one such work called the "Big Win Strategy". It is essentially the AlphaGo approach applied for 6×6 Othello. The authors propose small incremental improvements in the form of an additional network that estimates how many points a player will win/lose in a game as well as using depth rewards during the MCTS search.

5.3.1 AlphaGo inspired approaches

The AlphaGo style training was also applied to the game of *Hex*. Gao et al. (2017) proposed a player named *MoHex-CNN* that managed to outperform the previous state-of-the-art agent named *MoHex 2.0* (Huang et al. 2013). *MoHex-CNN* introduced a move-prediction component based on a deep convolutional network that serves as a policy network and is used together with MCTS. Next, Gao et al. (2018) proposed a novel three-head neural network architecture to further significantly improve *MoHex-CNN* and named the new player *MoHex-3HNN*. The network outputs include policy, state- and action-values.

Takada et al. (2019) present another approach, called *DeepEzo*, that has been reported to achieve a winning rate of 79.3% against *MoHex 2.0*, then world champion. The authors train value and policy functions using RL and self-play, as in the case of AlphaGo. The distinctive feature, however, is that the policy function is trained directly from game results of agents that do not use the policy function. Therefore, the policy function training is not a continuous process of self-improvement. Although *DeepEzo* won against *MoHex 2.0*, it lost against *MoHex-3HNN* in a direct competition during the 2018 Computer Olympiad (Gao et al. 2019).

Yang et al. (2020) build upon the AlphaGo approach to work without the prior knowledge of *komi*, which is the handicap that can be dynamically applied before a match. AlphaGo as well as most of the programs are trained using a komi fixed beforehand, i.e. equal to 7.5. In the described work, the authors designed the networks that can gradually learn how much komi there is. In addition, they modified the UCT formula to work with the game score directly and not with the win-rate, which is better for unknown komi. Wu et al. (2018) propose another approach to tackle the dynamic komi settings. Here, the authors proposed a novel architecture for the value function called a multi-labelled value network. Not only it supports dynamic komi but also lowers the mean squared error (MSE).

Tang et al. (2016) state in the first sentence of the abstract that their method for creating a *Gomoku* agent was inspired by AlphaGo. The main differences are that (1) here only one

neural network is trained that predicts the probability of players winning the game given the board state, (2) the network is only three-layer deep, and (3) the network is trained using *Adaptive Dynamic Programming* (ADP).

Barriga et al. (2017) in their work aim at creating AI agents for RTS games in the μ RTS framework. You can find more details about the approach in the RTS games Sect. 1.3 of this survey. The use of ML is very similar to the AlphaGo's approach as both value and policy functions are modelled as CNNs. However, the networks are only used for the strategic search, which outputs a strategic action. Next, a purely adversarial search replaces the strategic action by lower level tactical actions. This can be considered a two-level hierarchical approach.

Value network	Policy network
Input: 128×128, 25 planes	Input: 128×128, 26 planes
8 Conv. layers	8 Conv. layers
Global averaging over 16×16 planes	
2-Way softmax	4-Way softmax

Hearthstone, which has already been introduced in Sect. 4.4, is a two player, competitive card game in which the information about game state is imperfect. That partial observability in practice means two situations—when player draws a card and when takes combat decisions without knowing the opponent's cards in hand and the so-called secrets on the board. To handle the first case, Zhang and Buro (2017) introduce chance event bucketing. Chance event is a game mechanics with a random outcome e.g. a player rolls a dice or draws a card. Those tens or hundreds different outcomes produced by a single chance event can be divided into an arbitrary number of buckets. Then at the simulation phase each bucket containing similar outcomes is sampled, that process allows to reduce branching factor in the tree search. Other sampling methods based on chance nodes are described by Choe and Kim (2019) and Santos et al. (2017).

Świechowski et al. (2018) use the MCTS algorithm together with SL for the game of Hearthstone. In this work, the authors describe various ways in which ML-based heuristics can be combined with MCTS. Their variant of choice uses a value network for the early cutoff which happens at the end of the opponent's turn, therefore the simulation phase of the MCTS algorithm searches the tree only two turns ahead. However, players can make multiple moves within their turns. The policy network is used to bias the simulations. The authors use pseudo-roulette selection in two variants. In the first one, the probability of choosing an action in a simulation was computed using Boltzmann distribution over heuristic evaluations from the value network. In the second variant, the move with the highest evaluation was chosen with the probability of ϵ, whereas a random one (the default policy) was chosen otherwise, i.e., with the probability of $1 - \epsilon$.

Dots-and-Boxes is a paper-and-pencil game that involves capturing territory. For actions, players draw horizontal or vertical lines that connect two adjacent dots. Zhuang et al. (2015) present an approach to creating a strong agent for this game. The program is based on the MCTS algorithm and a sophisticated board representation. The MCTS is combined with a traditional, i.e., shallow, neural network that predicts the chance of winning of the players. Although this is effectively a value network, it is used to guide the simulation phase of MCTS. The network is trained using stochastic gradient descent.

Yang and Ontañón (2018) propose a neural network approach to RTS games in the μRTS framework. It relies on learning map-independent features that are either global (e.g.
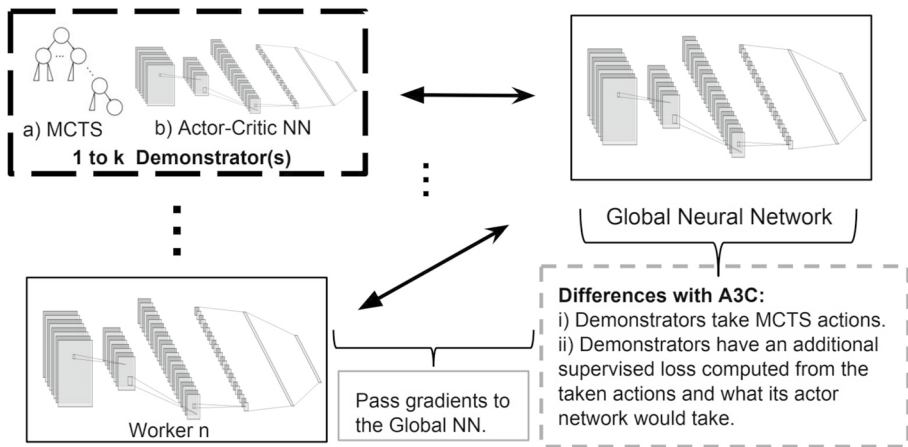
Fig. 9 Overview of the Planner Imitation based A3C framework. There are multiple demonstrators making MCTS actions while keeping track of what action the actor network would take. The figure is reproduced from Kartal et al. (2019b)

resources, game time left) or belong to one of 15 feature planes. Each map is represented by a tensor of size $15 \times width \times height$. Such tensor are passed to CNNs that are trained to estimate winning probability in the current state. The generalization ability of the trained network then allows extension to larger maps than the ones used for training.

Hu et al. (2019) present work that is another of the many examples of approaches that have been inspired by AlphaGo. Here, a system called Spear has been proposed for dependency-aware task scheduling that considers both task dependencies and task resource demands for various resources. It is based on MCTS in which the simulation phase is replaced by a deep RL model that acts in the environment and chooses actions according to its policy rather than randomly.

5.3.2 MCTS as a trainer for ML models

Soemers et al. (2019) show that is possible to learn a policy in a MPD using the policy gradient method and value estimates directly from the MCTS algorithm. Kartal et al. (2019a) propose a method to combine deep RL and MCTS, where the latter acts as a demonstrator for the RL component. The authors motivate this approach as "safer reinforcement learning" in "hard-exploration" domains, where negative rewards may have profound consequences. The domain of choice is a classic video game called *Pommerman*. In each state, the agent may move in one of four directions on the map, stay in place or plant a bomb. The MCTS is run with relatively shallow depth for time-constraints. However it is sufficient to learn, for instance, to avoid being killed by agent's own bomb. Please note that the bombs placed by the agent explode with a delay and this interaction is hard to learn from scratch by pure RL methods. The approach, depicted in Fig. 9, is based on the actor-critic idea.

A relatively similar idea of using MCTS as a trainer for reinforcement learner was proposed by Guo et al. (2014) for simple Atari games. In contrast to the previous approach, the authors use deep CNNs as a model for policy. The input to the CNNs are raw pixel images of what is displayed on the screen. This is essentially the same representation human players observe. The original size of 160×120 pixels is downgraded to 84×84

in a preprocessing step. The last four frames, e.g., a tensor of size $84 \times 84 \times 4$ is passed to the network. The output is a fully connected linear layer with a single output for each valid action in the game. The authors compare three different variants of how MCTS can train a CNN: *UCTtoRegression*, *UCTtoClassification* and *UCTtoClassification-Interleaved*.

Pinto and Coutinho (2018) present another approach in which MCTS and RL are combined. Here, MCTS operates in the "Options Framework". Options are defined as a tuple (I, μ, β):

(1) *I*—is the initiation set that contains states from which options are valid;
(2) *μ*—is a policy that tells what actions to make as part of the current option;
(3) *β*—is the termination condition for the option.

The MCTS algorithm searches the combinatorial space of possible options and prepares them for RL. Using options allowed the authors to reformulate the MPD problem into a semi-Markov decision problem (SMDP). In the latter, the Q-learning rule can be defined over options as shown in Eq. 13.

$$Q(s,o) = (1 - \alpha)Q(s,o) + \alpha\left[r + \gamma \max_{o' \in O} Q(s',o')\right], \tag{13}$$

where *O* denotes the possible options.

5.4 MCTS with temporal difference learning

Temporal Difference (*TD*) learning was one of the first notable applications of the RL paradigm in the domain of games Tesauro (1994). Here, the game is modeled as time sequences of pairs (x_t, y_t), where *t* is the time step, x_t denotes a state at this step and y_t is the predicted reward associated with it. The main idea that underpins *TD* learning is that the reward at the given time step can be approximated by a discounted sum of rewards at future time steps:

$$Y_t = y_{t+1} + \gamma y_{t+2} + \gamma^2 y_{t+3} + \cdots \tag{14}$$

The predictions are iteratively updated to be closer to the prediction of future states, which are more accurate. Please note that the last prediction, which is the final game outcome, is the most accurate. The goal is to learn the prediction for the current state or states available from the current states using the training signal from future predictions. The learning process typically runs for many game repeats. This idea is called bootstrapping.

TD methods can be applied to any pairs of values, i.e., input and output signals. Silver et al. (2012) use it to learn the weights of an evaluation function components for Go rather than for complete states.

The value function update in *TD* learning is most commonly defined for any state in the game as:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)), \tag{15}$$

where *α* is the learning factor; *γ* is the discount factor; R_{t+1} is the reward observed in step $(t + 1)$ and $V(s_t)$ is the value of state encountered in step *t* of the game.

Vodopivec and Šter (2014) incorporate the ideas of *TD* learning to replace the UCT formula (c.f. Eq. 2) and calculate state estimates in nodes in a different fashion. Using

TD learning inside MCTS affects only the backpropagation phase, in which values are updated, and the selection phase—because the tree policy no longer uses the UCT score. The remaining parts of the MCTS algorithm remain unchanged. The most general formula, which has specific variants in the paper, is based on weighted average of the classic *UCT* estimate and the *TD* estimate:

$$Q_{TD-UCT} = \omega V + (1 - \omega) * Q + C\sqrt{\frac{\ln [N(s)]}{N(s,a)}}, \tag{16}$$

where ω is the weight, V is the estimate sampled from the TD value function and the remaining symbols are as defined in Eq. 2.

Equation 16 is used instead of the UCT formula as the tree policy. The *TD* value function update $((V))$ is performed in the back-propagation phase. For this task, the authors specifically chose the $TD(\lambda)$ algorithm, in which a state value $V(s_t)$ is updated based on the immediate reward—R_{t+1}—and the value in the following state—$V(s_{t+1})$:

$$V(s_t) \leftarrow V(s_t) + \alpha e(s_t)\delta_t, \tag{17}$$

where

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{18}$$

δ_t is the temporal difference error (TD error) and $e(s_t)$ denotes the *eligibility trace*. The remaining symbols have already been defined before.

Three variants of the $TD(\lambda)$ integration have been proposed:

– *TD-UCT Single Backup*—here, the authors introduce a simplification that all future values of states have already converged to the final value of the playout. The TD error depends on the distance to the terminal state and the value estimated in a leaf node: $\delta_1 = \gamma^P R_i - V(leaf)$, where P is the number of steps in a playout.
– *TD-UCT Weighted Rewards*—this variant is even more simplified version of *TD-UCT Single Backup* obtained by fixing most of the parameters. In particular, they set $\gamma = 1$ (in Eq. 18) and $\omega = 1$ (in Eq. 16) which makes V fully replace the average score estimate Q. However, the exploration part of Eq. 2 remains unchanged, so this variant combines the idea of TD learning with the UCT exploration part.
– *TD-UCT Merged Bootstrapping*—here, there is no longer the assumption that all future state values converge to the final reward of the playout. Instead, this variant includes the regular TD bootstrapping, i.e. the state value is updated based on the value of the next state.

In addition, the combinations with the AMAF heuristic and RAVE (c.f. Eq. 4) are considered in the experiments. The various combinations were tested on four games: Hex, Connect-4, Gomoku and Tic-Tac-Toe. One variant, namely *TD-UCT Merged Bootstrapping* with *AMAF*, achieved the best performance in all four games compared to the other variants as well as compared to the plain UCT.

The TD method can also be used to adapt the policy used in the simulation phase of the MCTS algorithm (Ilhan and Etaner-Uyar 2017). This is especially useful for games (or other domains) in which uniform random simulation performs poorly. This idea is explored in General Video Game AI games: *Frogs*, *Missile Command*, *Portals*, *Sokoban* and *Zelda*. The TD method of choice is Sarsa(λ). The authors use a linear value function

approximation (as suggested in true online Sarsa(λ)) along with a state representation method. The representation is a vector of numerical features that describe the game state and its weights. Here, the goal is to learn the weights of the state vector and then use a custom rollout procedure that incorporates state evaluation. The outcome of a rollout is calculated as in Eq. 14, so the outcome at step t is a discounted sum of future steps. Each consecutive step is performed by a separate rollout. This is a different approach to a vanilla MCTS, in which there is one rollout that fetches the game score, obtained at the terminal state, and propagates the same score to all nodes visited along the path. Here, states on the visited path receive different scores according to the *TD* learning idea. The rollouts are performed according to the ϵ−Greedy selection policy that maximizes the weighted linear combination of features in a state over possible actions that lead to it. The authors achieve better results than vanilla MCTS in 4 out 5 tested games.

5.5 Advantages and challenges of using ML with MCTS

We have reviewed approaches that combine MCTS with ML techniques in various ways. Let us conclude this section with a short discussion about the advantages and specific challenges that arise when such a joint approach is pursued.

Advantages

– *New quality results.* The main factor that motivates researchers to introduce ML into MCTS is to obtain better results than the current state-of-the-art. In the game domain, the common goal is to create a next-level player that will win against the strongest programs or human champions. The examples of *AlphaGo* (Silver et al. 2016), *MoHex-3HNN* (Gao et al. 2018) and other show that ML can indeed be an enabler for such spectacular results. In Sect. 7, which is devoted to non-game applications, we will also show an approach that combines ML and MCTS and achieves novel results in chemical synthesis (Segler et al. 2018).
– *Faster convergence.* Another motivation, and ultimately an advantage, that ML brings to MCTS is that it enables MCTS to converge faster. We have seen three groups of approaches in this category. First, e.g. by Zhuang et al. (2015), Wu et al. (2018) and Świechowski et al. (2018), achieve it by introducing an ML model for the evaluation function in MCTS. Second, which is often combined with the first as in *AlphaGo*'s inspired approaches, consists in using an ML model as the policy for simulations. Lastly, RL algorithms such as *TD*(λ) can be combined with the UCT formula or used instead in MCTS as shown by Vodopivec and Šter (2014).
– *Approximation capabilities.* Let us discuss the last two advantages on a theoretical level. MCTS is a search-based method. In order to properly evaluate the quality of a state, the state must be visited enough number of times (to gain the confidence about the statistics). The MCTS algorithm has no inherent ability to reason about the states rather than based on statistics stored directly for them. Many ML models, however, such as neural networks or logistic regression are universal approximators. They can provide answers for states that have never been observed during simulations as long as the ML models are well-trained, i.e., achieve good performance on the training data and are neither underfit nor overfit.
– *Generalization.* Continuing on the aspect the approximation—ML models can generalize to some extend the knowledge encoded in MCTS trees. Although the training process usually takes a long time, using the trained models in real-time is by orders

of magnitude faster than using MCTS or any search-based method. Moreover, let us imagine a solved end-game position in Chess, in which there is a strategy for a king that involves a traversal around an enemy piece. If we increased the board size from 8×8 to 9×9, then MCTS not only would have to compute everything from scratch but also the complexity of the problem would increase (bigger board). Thanks to generalization and abstraction capabilities, ML models can bring new quality here.

Challenges

– *Data volume and quality.* The biggest challenge of introducing ML into MCTS is data availability. ML models, especially those based on deep learning such as *AlphaZero* (Silver et al. 2017) require a lot of data. Moreover, there is a saying "garbage in, garbage out" so the data need to be of high quality. In game AI, it means that it must be generated by strong AI players. However, if ML is to help MCTS in the development of strong players, then we have a situation that strong players are not yet available. This is a major obstacle for using SL algorithms. To tackle this problem, self-play and RL (Kartal et al. 2019a) can be applied.
– *Training accurate models.* This is a general challenge when creating ML models. In game AI, overfitting may appear, for example, when heuristic-based and non-adaptive AI players are used in training games to generate data. It may lead to a scenario, in which a lot of similar states are present in the training data, whereas a lot of areas are under-represented. A proper variety of positions must be ensured in the training process as well as proper training methodology. Often, experts at game AI, especially in video game development industry, are not experts in ML.
– *Technology problems.* The most common programming languages used for games are *C++* and *C#*. The most common programming language used for ML is *Python*, followed by *R*. Therefore, using ML techniques in games usually requires either custom implementations of the ML models or having complex projects that combine multiple programming languages.

6 MCTS with evolutionary methods

6.1 Evolving heuristic functions

Benbassat and Sipper (2013) describe an approach, in which each individual in the population encodes a board-evaluation function that returns a score of how good the state is from the perspective of the active player. The authors use strongly typed genetic programming framework with boolean and floating-point types. The genetic program, i.e., the evaluation function, is composed of terminal nodes and operations such as logic functions, basic arithmetic functions and one conditional statement. The terminal nodes consist of several domain-independent nodes such as *random constant*, *true*, *false*, *one* or *zero* as well as domain-specific nodes such as *EnemyManCount*, *Mobility* or *IsEmptySquare(X,Y)*. The paper considers two domains of choice: *Reversi* and *Dodgem* games. Figure 10 illustrates exemplar function for Reversi.

The authors incorporate the evolved heuristics in the simulation phase of MCTS. Instead of simulating a random move, the agent randomly chooses a parameter called *playoutBranchingFactor* in each step. The upper bound of this value is equal to the number

Fig. 10 An example of a genetic programming tree used to evolve an evaluation function. The figure was inspired by a depiction of the evaluation function for Reversi in work by Benbassat and Sipper (2013)
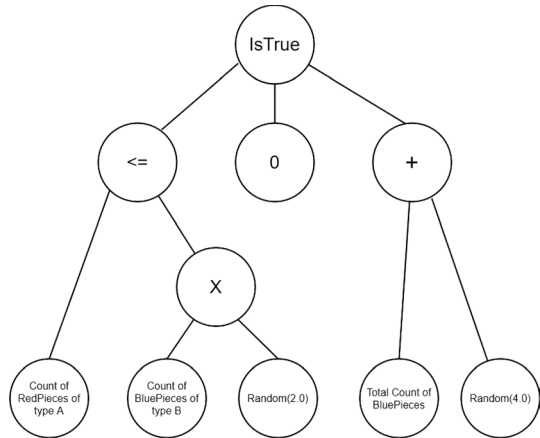
of currently available actions. Then, a set of *playoutBranchingFactor* random moves is selected and the one with the highest heuristic evaluation is chosen to be played in the simulation. This way, there is still stochastic element in simulations, which is beneficial for the exploration of the state-space.

The authors conclude that evolving heuristic evaluation function improves the performance of the MCTS players in both games. However, it is time consuming, therefore it is most suitable in scenarios, when it can be done offline, i.e., before the actual game, which is played by MCTS with an already prepared evaluation function.

Alhejali and Lucas (2013) propose a similar idea to evolving heuristics by means of GP for *Ms Pac-Man* agent. In this research, the new default policy to carry out simulations is evolved directly.

6.2 Evolving policies

The approaches presented in this section consist in optimizing the policies of MCTS—tree policy or default simulation policy—using evolutionary computation. This is a different case to the previous section, whereby a new evaluation function is constructed.

One of the first papers on this topic was published by Lucas et al. (2014). The authors introduce a weight vector w that is used to influence both tree policy $T(w)$ and default policy $D(w)$. The weight vectors are stored individuals optimized by a $(1 + 1)$ Evolution Strategy (ES). For default policy, a mapping from the state space to a feature space with N features is introduced. Those features are assigned weights that are used to bias actions during a simulation towards states with a greater aggregated sum of weights. To maintain exploration, softmax function is used instead of a greedy selection.

Perez et al. (2014) introduce an approach called *Knowledge-Based Fast Evolutionary MCTS (KB Fast-Evo MCTS)* which extends the work by Lucas et al. (2014). The environment of choice is GVGP, which as a multi-domain challenge, encourages adaptive algorithms that are not tailored for a particular game. The authors dynamically extract features from a game state, in contrast to a fixed number of features employed in the previous work. The features not only vary between games but also between steps of the same game. Each feature i has a weight w_{ij} assigned for action a_i. The weights are evolved dynamically and they are used to bias the MCTS simulations in a similar way as proposed by Lucas et al.
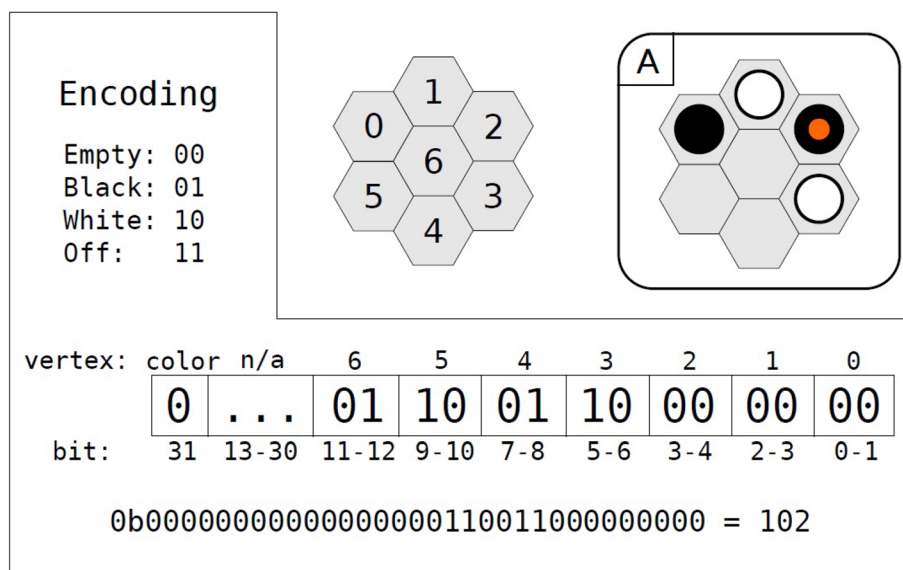
Fig. 11 Example of a board pattern encoded in an individual of a population optimized by EA (Pettit and Helmbold 2012)

(2014). Here, also a soft-max function is used to calculate the probability of choosing an action.

KB Fast-Evo MCTS also introduces a dynamic construction of a knowledge base. The knowledge base consists of knowledge items, for which statistics such as occurrences and average scores changes are maintained. The authors introduce the notions of *curiosity* and *experience* and the formula to calculate them for a knowledge base. Next, both the values, i.e. *curiosity* and *experience*, are combined to calculate the *knowledge change* for a knowledge base. Finally, the authors propose a formula to calculate the predicted end-game reward that can be used for the MCTS roll-out. The reward is either equal to the change of the game score or equals to the weighted sum of the *knowledge change* and the *distance change* if no game score change is observed. The *distance change* is a heuristic value introduced for GVGP. This approach was tested using 10 games from the GVGP corpus. The proposed *KB Fast-Evo MCTS* outperformed the vanilla MCTS in all of the games, whereas the previous *Fast-Evo MCTS* in 7 out of 10 games.

Pettit and Helmbold (2012) introduced a system called *Hivemind* that learns how to play abstract strategy games on regular boards. The game of choice here is Hex. The system uses MCTS/UCT and Evolution Strategies to optimize the default policy for simulations. The subject of evolution are individuals that encode:

– Local patterns on a board. For example, the status of a given hex cell with its six neighbors as shown in Fig. 11.
– Move selection strategies (default, uniform local, uniform local with "tenuki").

Thanks to using ES, the authors were able to evolve meaningful patterns that used by move selection strategies outperform baseline default policies, including uniform random selection. The obtained results are stable, i.e., the evolutionary process finds similar

efficient policies each time. The authors also show that the learnt policies generalizes to different board sizes.

The tree policy, i.e., in the selection part of MCTS, can also be optimized alone. Bravi et al. (2016) investigated whether it is possible to evolve a heuristic that will perform better than UCB in the field of GVGP. They experimented with five different games from this framework: *Boulderdash*, *Zelda*, *Missile Command*, *Solar Fox* and *Butterflies*. In this work, GP is employed with a syntax tree that serves as an encoding of a chromosome. The syntax tree denotes a formula to use instead of UCB.

The possible components are:.

– 16 Predefined constants from the range $[-30, 30]$,
– Unary functions—square root, absolute value, multiplicative inverse,
– Binary functions—addition, subtraction, multiplication, division and power,
– Variables such as child visits, child depth, parent visits, child max value.

In a similar paper, Bravi et al. (2017), the authors added certain *Game Variables* as possible components that describe features of the state of the game. The evolved heuristic functions perform on a similar level, i.e., not significantly better or worse, to UCB in four of five tested games. In the last game, they yield poor results. However, these heuristics give a lot of insights about particular games, e.g., whether exploration is an important factor. The authors conclude that this approach can be promising when combined with additional variables and mechanisms.

6.3 Rolling horizon evolutionary algorithm

In this subsection, we will provide a short background of RHEA. In general, RHEA is a competitor technique for creating decision-making AI agents in games. As the name implies, it is a representative of a broad class of evolutionary algorithms (EA). However, it should not be mistaken with Evolutionary-MCTS, which has been inspired by RHEA. Although MCTS and RHEA are different techniques, they are often compared in terms of performance in particular games as well combined within the same agent.

The first application of RHEA was published in Perez et al. (2013) for the task of navigation in single-player real-time games. For a recent summary of this method and its modifications so far, we recommend the article by Gaina et al. (2021).

Instead of doing the traditional selection and simulation steps, the RHEA algorithms evolve sequences of actions starting from the current state up to some horizon of N actions. Each individual in a population encodes one sequence. Only the first action from the best sequence is played in the actual game. To compute the fitness value, the simulator applies encoded actions one by one. If an action is not legal, then it is replaced by a default one ("do nothing"). After applying the last action, the score of the game is computed as shown in Eq. 19:

$$h(s) = \begin{cases} 1 & \text{win} = \text{True} , \\ 0 & \text{win} = \text{False} , \\ score \in [0, 1] & \text{otherwise} . \end{cases} \tag{19}$$

The algorithm incorporates EA operators of crossover and mutation. The selection is typically the tournament selection. Elitism, i.e., directed promotion of the most fit parents to
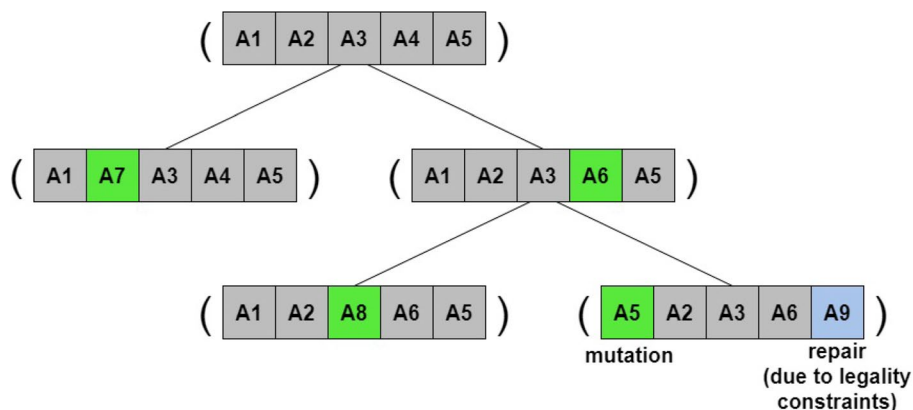
Fig. 12 Evolutionary MCTS. Instead of performing rollouts, sequences of actions are mutated. Repair operators are introduced in case of possible illegal moves generated this way. The figure was inspired by the work of Baier and Cowling (2018)

the generation, can be incorporated. In the mentioned article by Gaina et al. (2021), you can find a description of an RHEA framework tested using a set of 20 different games. As this survey is focused on modifications of the vanilla MCTS algorithm, the work by Gaina et al. (2017b) presents enhancements to the vanilla RHEA approach. The online self-adaptation of MCTS (e.g.) has also its counterparts for RHEA as described by Gaina et al. (2020).

Horn et al. (2016) consider a few hybridization schemes of the MCTS and EA. One is called RHEA with rollouts, in which after the application of a sequence of actions stored in a genome, a number of rollouts (Monte Carlo simulations) is performed. The final fitness is the average of the classic RHEA fitness and the average value from additional rollouts. Another hybridization scheme proposed is an ensemble approach, which first runs RHEA for a fixed time and then MCTS to check for alternative paths.

The analysis by Gaina et al. (2017a) compares the MCTS and RHEA approaches in GVGP. In addition, in this works, the impact of various parameters of RHEA is measured. Within the considered corpus of various games used in GVGP, there are cases, in which one method is stronger than the other. The conclusion is that neither algorithm is inherently stronger and their efficacy depends on a particular game.

6.4 Evolutionary MCTS

The common element of the three approaches described in papers by Baier and Cowling (2018) and Horn et al. (2016) is that EA is responsible for performing simulations. Baier and Cowling (2018) propose *Evolutionary MCTS* (EMCTS), which builds a different type of tree than the vanilla MCTS. It does not start from scratch, but instead all nodes, even the root, contain sequences of actions of certain length. In the presented study with *Hero Academy* game, the sequences are 5 action long. Each node is a sequence, whereas edges denote mutation operators as shown in Fig. 12. Instead of performing rollouts, the sequences in leaf nodes are evaluated using a heuristic function.

7 MCTS applications beyond games

7.1 Planning

Automated planning is one of the major domains of application of the MCTS algorithm outside games. The planning problem is typically formulated as *MDP*, which was defined in Sect. 2, or *POMDP* (Partially Observable MDP). Similarly to games, in AI planning, there is a simulated model that can be reasoned in. The model consists of an environment with the initial state, the goal states (to achieve) and available actions. The solution is a strategy—either deterministic or stochastic, depending on a particular problem, that transitions the initial state to the goal state, playing by the rules of the environment, in the most efficient way. The most efficient fashion may be, e.g., the shortest transition or having the smallest cost. Particular applications differ between each other with respect to various constraints, extensions and assumptions. In this subsection, we focus on general planning problems. Scheduling, transportation problems and combinatorial optimization have their own dedicated sections.

Vallati et al. (2015) present a summary of the International Planning Competitions up to 2014, in which we can read that winning approaches of the probabilistic track (*International Probabilistic Planning Competition*) in 2009–2014 were using Monte Carlo Tree Search. It was shown that MCTS-based planners were very strong but they lost ground in the largest problems. They often needed to be combined with other models such as Bayesian RL (Sharma et al. 2019). This is along the motivation behind this survey, which focuses on extensions aimed at tackling the most difficult problems from the combinatorial point of view.

Scheduling is a similar problem to planning, wherein resources are assigned to certain tasks that are performed according to a schedule. Ghallab et al. (2004) point out that both automated planning and scheduling are often denoted as AI planning. There have been several standard implementations of MCTS to scheduling such as of Amer et al. (2013) for activity recognition and by Neto et al. (2020) for forest harvest scheduling.

MCTS has been used in other discrete combinatorial problems in a relatively standard variant, e.g. by Kuipers et al. (2013) for optimizing the Horner's method of evaluating polynomials. In the work by Jia et al. (2020), MCTS is used for optimizing low latency communication. The authors present a parallelized variant, in which each parallel node is initialized with a different seed. The master processor merges the MCTS trees constructed with different seeds. Shi et al. (2020) apply MCTS to automated design of generating large scale floor plans with adjacency constraints. The only non-standard modification there is discarding the rollout phase entirely. Instead the tree is expanded by each node encountered by the tree policy. The MCTS algorithm has also been used as part of a more general framework, e.g., to the imbalance subproblem in a complex framework based on cooperative co-evolution to tackle large optimization problems.

7.1.1 Simplifications of a problem/model

To tackle large planning problems, Anand et al. (2015) propose an abstraction of state–action pairs for UCT. Later, Anand et al. (2016) came up with a refined version of this idea called "On-the-go abstractions" . The abstraction is based on the equivalence

function that maps state–action pairs to equivalent classes. In their work, Anand et al. (2015) consider the following problems:

– How to compute abstractions—the algorithm is presented.
– When to compute abstractions—time units are introduced and when the UCT is sampled up to a certain level, abstractions are computed every t units, where t is computed using a dynamic time allocation strategy.
– Efficient implementation of computing abstractions—let n denote the number of State–Action Pairs (SAPs). The authors reduced the complexity of the naive algorithm of computing the abstractions from $O(n^2)$ to $O(rk^2)$, where r is the number of hash buckets and k is the size of a bucket and $n^2 > rk^2$.

The novel method with abstractions, called ASAP-UCT, has been compared with the vanilla UCT planner and other three algorithms in six problems from three domains. The ASAP-UCT was the clear winner with statistically significantly better results in all but one comparison.

Keller and Eyerich (2012) introduce a system called PROST aimed at probabilistic planning problem and acting under uncertainty. As previously mentioned, in this subproblem of planning, MCTS approaches has been the most successful. PROST is equipped with several modifications of vanilla MCTS/UCT approach such as:

– Action pruning—the model includes the concepts of *dominance* and *equivalence* of actions. Dominated actions are excluded, whereas only one action from equivalent sets is considered.
– Search space modification—the tree contains *chance nodes* and *decision nodes* and even after action pruning the search space is strongly connected.
– Q-value Initialization—PROST uses *single-outcome determinization* of the MDP using *iterative deepening search* (IDS) to initialize Q-values and avoid the cold start.
– Search depth limitation—search depth limit is introduced as a parameter λ.
– Reward Locks—states, in which no matter what the agent does, it will receive the same reward. The authors explain this is often the case close to goals or dead ends. The method of detecting reward locks is presented in the paper. The MCTS uses such information in two ways: (1) to terminate a rollout earlier and (2) minimize the error induced by limited depth search by changing the simulation horizon.

7.1.2 Modifications of UCT

Painter et al. (2020) modify the standard UCT formula (c.f. Eq. 2) to include the so-called *Contextual Regret* and adapted to multi-objective settings. The approach has been coined as *Convex Hull Monte Carlo Tree Search*. In a multi-objective setting the reward function is multi-dimensional and produces vectors instead of single values. The approach was validated in experiments using the *Generalized Deep Sea Treasure* (GDST) problem in both online and offline planning scenarios. A similar type of modification of UCT in planning problems, that is based on simple regret optimization, can be found in the article published by Feldman and Domshlak (2014)

Hennes and Izzo (2015) consider the problem of interplanetary trajectory planning and introduces a handful of modifications to MCTS that are unique to this problem. One of the modifications, however, is the fifth step of the MCTS routine called by the authors
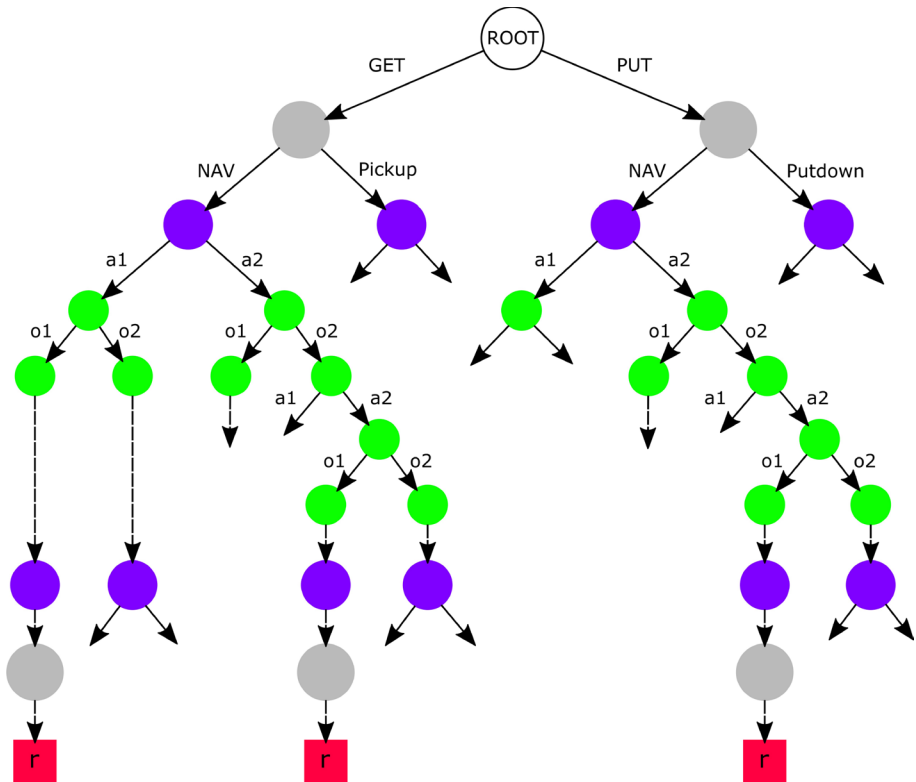
Fig. 13 Hierarchical Monte Carlo Planning. The figure is inspired by Vien and Toussaint (2015)

contraction. In the considered problem, the search-tree is often broad, i.e. the branching factor is as high as 500, but shallow, i.e., the depth is limited. In the contraption phase, the algorithm checks whether there is a sub-tree with terminal states. If each nodes of such a subtree have already been visited it can be considered solved.

7.1.3 Hierarchical and distributed MCTS

Vien and Toussaint (2015) propose a hierarchical Monte Carlo planning method to tackle the problem of large search spaces and combinatorial explosion in planning. The hierarchy is introduced at the action level. The top-level tasks are subsequently decomposed into sub-tasks that cannot be decomposed any further. The authors discuss the concept of a hierarchical policy in POMDPs. This research contains a successful application of macro actions in a non-game domain. The idea is illustrated in Fig. 13, in which there are four actions: $\{a_1, a_2, Pickup, Putdown\}$ and three macro actions $\{GET, PUT, NAV\}$.

The vertical dotted lines denote rollouts, whereas solid ones are tree policy executions. The green nodes (i.e., the smallest ones with o1, o2, a1, a2 annotations on the edges to them) denote subtree with only primitive actions. Nodes with other colors denote subtrees with particular macro actions. Each macro action defines a particular subgoal for the algorithm and has a nested routine (subroutine) for action selection. The authors show how to traverse such a MCTS tree with nodes of various hierarchy and
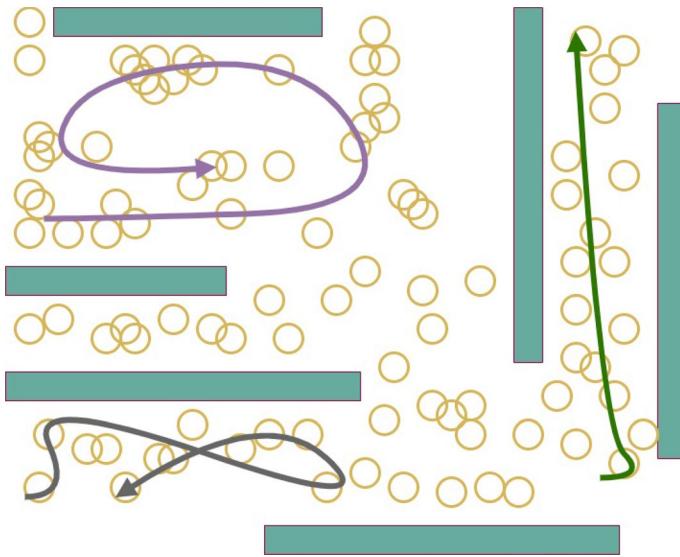
Fig. 14 Decentralized planning. The figure is drawn inspired by the work of Best et al. (2019)

how to expand it. The paper contains not only empirical evaluation using the Taxi problem but also theoretical results such as estimation of the bias induced by the Hierarchical UCT (H-UCT). They conclude that the proposed approach aims to mitigate two challenging problems by means of the "curse of dimensionality" and the "curse of history". Hierarchical MCTS planning is efficient whenever a hierarchy can be found in the problem. It has been featured in recent works as well (Patra et al. 2020).

7.1.4 Planning in robotics

MCTS has also been used for planning in robotics. Best et al. (2019) propose a decentralized variant of MCTS for multi-robot scenario, in which each robot maintains its individual search tree. The trees in compressed forms are periodically sent to other robots, which results in an update of joint distribution over the policy space. The approach is suitable for online replanning in real-time environment. The algorithm was compared to standard MCTS with a single tree with actions of all robots included. Therefore, the action space of the tree was significantly larger than any of the decentralized trees of the Dec-MCTS approach. Apart from theoretical analysis, the experiments were carried out in eight simulated environments based on the team orienteering problem as shown in Fig. 14. The Dec-MCTS achieved a median 7% better score than the standard algorithm.

Wu et al. (2015) introduce the *k-Rejection Multi-Agent Markov Decision Process* *k*-RMMDP model in the context of real-world disaster emergency responders. In this model, plans can be rejected under certain conditions e.g. if the human responders decide that they are too difficult. Because their approach is based on the two-pass UCT planning as shown in Algorithm 2, it can be considered hierarchical as well.

Algorithm 2: Two-Pass UCT Planning (Wu et al., 2015)

Input: The k-RMMDP Model: M, The Initial State: $(s^0, 0)$
// The first pass:
// Compute the policy for $\forall s, (s, 0)$
Solve the underlying MMDP of M and build a search tree:
  Run UCT on the underlying MMDP: $\langle S, \overrightarrow{A}, T, R \rangle$
  Update all the Q-values: $Q(s, \overrightarrow{a}) \leftarrow Q(s, \overrightarrow{a}) \cdot \Delta_0(s, \overrightarrow{a})$

// The second pass:
// Compute policy for $\forall s, j \neq 0, (s, j)$
foreach *state node $(s, 0)$ of the tree* **do**
  Create a sub MMDP:
    Initial state: $(s, 0)$
    Normal states: $\forall j, (s, j)$
    Terminal states: $\forall s', (s', 0)$
    Actions: $\overrightarrow{A}$, Transition: **T**, Reward: **R**
  Run UCT on the sub MMDP:
    Build a subtree with states $\forall j, (s, j)$
    Propagate values of $\forall s', (s', 0)$ to the subtree
    Propagate values of the subtree to node $(s, 0)$
  Update all the Q-values of $(s, 0)$'s ancestors in the tree
// The policy: $\pi(s^0, j) = argmax_{\overrightarrow{a}} Q((s^0, j), \overrightarrow{a})$
return *the policy computed with Q-values in the search tree*

Monte Carlo Discrepancy Search (Clary et al. 2018) is another MCTS variant with applications in robotics. The idea is to find and store the current best path out to a receding horizon. Such a path is invalidated either by a change in the horizon or by stochastic factors. The simulations are performed around the best plan from the previous iteration. Then a so-called discrepancy generator selects a new node to branch from in the tree. The method was applied to planning the movement of a self-stable blind-walking robot in a 3D environment, which involved navigating around the obstacles.

7.2 Security

One of very recent areas of MCTS application are Security Games (SG). In short, in SG methods from game theory (GT) are used to find optimal patrolling schedules for security forces (e.g. police, secret service, security guards, etc.) when protecting a set of potential targets against the attackers (criminals, terrorists, smugglers, etc.). Recently, SGs gained momentum due to rising terrorist threats in many places in the world. While the most noticeable applications refer to homeland security, e.g. patrolling LAX airport (Jain et al. 2010) or patrolling US coast (An et al. 2013), SGs can also be applied to other scenarios, for instance poaching preventing (Fang et al. 2015; Gholami et al. 2018; Bondi et al. 2020) or cybersecurity (Wang et al. 2020).

In SG there are two players, the *leader* (a.k.a. the *defender*) who represents the security forces, and the *follower* (a.k.a. the *attacker*) who represents their opponents. The vast majority of SGs follow the so-called Stackelberg Game (StG), which is an asymmetric game model rooted in GT. In StG each of the two players commits to a certain *mixed* strategy, i.e. a probability distribution of *pure* strategies. In a typical patrolling example, a pure strategy would be a deterministic patrolling schedule which happens with certain

probability. A set of such deterministic schedules (pure strategies) whose probabilities sum up to 1 constitutes mixed leader's strategy.

The aforementioned information asymmetry of StG consists in the underlying assumption that first the leader commits to a certain mixed strategy, and only then the follower (being aware of that strategy) commits to their strategy. This assumption reflects common real-life situations in which the attacker (e.g. a terrorist or a poacher) is able to observe the defender's behavior (police or rangers) so as to infer their patrolling schedules, before committing to an attack. SGs relying on StG model are referred to as Stackelberg SGs (SSG). SSG are generally imperfect-information games as certain aspects of the current game state are not observable by the players. Solving SSG consists in finding the Stackelberg Equilibrium (SE), which is typically approached by solving the Mixed Integer-Linear Program (MILP) corresponding to a given SSG instance (Paruchuri et al. 2008).

Calculation of SE is computationally expensive in terms of both CPU time and memory usage, and therefore, only relatively simple SSG models can be solved (exactly) in practice. For this reason, in many papers various simplifications are introduced to MILP SG formulations, so as to scale better for larger games, at the expense of obtaining approximate solutions. However, these approximate approaches still suffer from certain limitations of the baseline MILP formulation, e.g. the requirement of matrix-based game representation. Consequently, the vast majority of deployed solutions consider only single-act (one-step) games.

The lack of efficient methods for solving multi-act games motivated the research on entirely different approach (Karwowski and Mańdziuk 2015) which, instead of solving the MILP, relies on massive Monte Carlo simulations in the process of finding efficient SE approximation.

In the next couple of years the initial approach was further developed into a formal solution method, called Mixed-UCT (Karwowski and Mańdziuk 2016)

which is generally suitable for a vast range of multi-act games. Mixed-UCT consists in iterative computation of the leader's strategy, based on the results of MC simulations of the game, played against a gradually improving follower. The currently considered follower is defined as a combination of a certain number of past followers from previous Mixed-UCT iterations. The method scales very well in both time and memory and provides efficient SE approximations (approximate mixed leader's strategies) for various types of SGs (Karwowski and Mańdziuk 2019a). A general concept of Mixed-UCT is depicted in Fig. 3.

Recently, the same authors have proposed another UCT-based method for SE approximation in general-sum multi-step SGs. The method, called Double Oracle UCT (O2UCT), iteratively interleaves the following two phases: (1) guided MCTS sampling of the attacker's strategy space; and (2) constructing the defender's mixed-strategy, for which the sampled attacker's strategy (1) is an optimal response (Karwowski and Mańdziuk 2019b, 2020).

One of the underlying assumptions of Stackelberg Games (SGs) is perfect rationality of the players. This assumption may not always hold in real-life scenarios in which the (human) attacker's decisions may be affected by the presence of stress or cognitive biases. There are several psychological models of bounded rationality of human decision-makers, among which the Anchoring Theory (AT; Tversky and Kahneman 1974) is one of the most popular. In short, AT claims that humans have a tendency to flatten probabilities of available options, i.e. they perceive a distribution of these probabilities as being closer to the uniform distribution than it really is. An extension of O2UCT method that incorporates the AT bias into the attacker's decision process (AT-O2UCT) has been recently considered in Karwowski et al. (2020).

Algorithm 3: Mixed-UCT for security games (Karwowski and Mańdziuk, 2019a)

Input: problem specification
Output: defender's strategy - the best of strategies developed in all iterations
$tree \leftarrow \emptyset$
$evaders \leftarrow [InitES()]$ // Vector of attacker's strategies in subsequent
    iterations initialized with the optimal strategy against uniform
    defender
while *not EndConditions()* **do**
    $moves \leftarrow []$
    for *depth* $\leftarrow$ *1...T* **do**
        $tree \leftarrow I2Uct(tree, EvaderStrategy(evaders), depth, moves)$
        $tree \leftarrow Append(moves, BestMove(tree, depth))$
    $defender \leftarrow StrategyFromTree(tree)$
    $evaders \leftarrow Append(evaders, OptEvader(defender))$
return *the best strategy among all calculated in the loop*

7.3 Chemical synthesis

Segler et al. ([2018](#)) in an article published in *Nature* show an outstanding approach to planning chemical syntheses that combines MCTS with deep neural networks. These networks have been trained on most reactions known from the history of organic chemistry research. The particular type of synthesis considered in this paper is computer-aided retrosynthesis, in which molecules are transformed into increasingly simpler precursors. In this approach, named **3N-MCTS**, three neural networks guide the MCTS algorithm:

– Rollout policy network—trained on 17,134 rules describing reactions, which were selected from the set of all published rules with a requirement for each rule to appear at least 50 times before 2015. The actions in the simulation phase of MCTS are sampled from this network.
– In-score filter network—this network provides retroanalysis in the expansion phase and predicts whether a particular transformation resulting by applying an action from the rollout policy is expected to actually work. If not—the action is discarded and a different one has to be chosen. Awais et al. ([2018](#)) based their work on a similar idea for synthesis of approximate circuits that discards circuit designs which are not promising in terms of quality. This way, the combinatorial complexity could be drastically reduced.
– Expansion policy network—this network selects the most promising node to add in the MCTS tree. This is a significant change compared to the standard approach, in which the first non-tested action defines the node to expand.

Combining deep neural networks with MCTS has been proven a viable approach to various chemical or physical problems including *FluidStructure Topology Optimization* by Gaymann and Montomoli ([2019](#)). The problem is formulated as a game on a grid, similar to Go.

7.4 Scheduling

Combining MCTS with additional heuristics is a pretty common approach to large computational problems. Asta et al. (2016) use MCTS to generate initial solutions to a Multi-mode Resource-constrained Multi-project Scheduling Problem, which are then modified through a local search guided by carefully crafted hyper-heuristics.

In order to create a test-bed for Computational Intelligence (CI) methods dealing with complex, non-deterministic and dynamic environments Waledzik et al. (2014) introduces a new class of problems, based on the real-world task of project scheduling and executing with risk management. They propose Risk-Aware Project Scheduling Problem (RAPSP) as a significant modification of the Resource-Constrained Project Scheduling Problem (RCPSP).

In Waledzik et al. (2015), they describe a method called *Proactive UCT* (ProUCT) that combines running heuristic solvers with the UCT algorithm. The heuristic solvers create baseline schedules, whereas UCT is responsible for testing various alternative branches caused by dynamic events. The method is further extended and summarized in Waledzik and Mandziuk (2018).

Wijaya et al. (2013) present an approach to energy consumption scheduling in a smart grid system that is based on MCTS. The application of the algorithm is relatively standard, however, the solutions can vary in terms of the so-called *laziness*. The lazier the solution, the less frequent consumption schedule changes. Since the lazier solutions are preferable, the UCT formula includes an additional penalty that is inversely proportional to the laziness value of the solution in a node.

An interesting scheduling approach inspired by AlphaGo Silver et al. (2016) has been proposed in Hu et al. (2019). Spear system, briefly summarized in Sect. 5.3.1, was reported to outperform modern heuristics and reduce the maskespan of production workloads by 20%.

7.5 Vehicle routing

Transport problems such as Capacitated VRP (CVRP) are special class of problems that often involve both planning and scheduling. We recommend the surveys by Caceres-Cruz et al. (2014) or Mańdziuk (2019) for the introduction to various problem specifications, the most commonly applied techniques other than MCTS and the latest advances. There are many variants of CVRPs. The common elements include planning routes for vehicles, which resembles solving multiple *travelling salesman problems* (TSP), as well as allocating goods to the vehicles that need to be delivered to customers, which resembles the *knapsack* problem. Therefore, even the basic version of CVRP is a cross of two NP-hard problems. The particular CVRP variants may operate with discrete or continuous time, time windows for loading and unloading, traffic, heterogeneous goods, specific client demands, the number of depots the vehicle start in, pickup and delivery, split delivery and many more. The goal, in the order of importance, is to: (1) deliver goods to the clients that meet their demand, (2) minimize the total length of the routes, (3) sometimes minimize the number of vehicles used. At the same time, all the constraints such as not exceeding the capacity of any vehicle must be preserved.

There have been several papers, in which an MCTS-based approach has been applied to a CVRP problem or a similar one such as Multi-Objective Physical TSP. Since we focus on

MCTS in this survey and the approaches share many similarities regarding how this algorithm is applied, we grouped them into topics:

Problem simplification by having a "good" initial solution—featured in the articles by Powley et al. (2013b), Mańdziuk and Świechowski (2017, 2016), and Juan et al. (2013). Powley et al. (2013b) use a classical TSP solver for the initial solution and MCTS acted as a steering controller over it. Mańdziuk and Świechowski (2017, 2016) as well as Juan et al. (2013) generate the initial solution with the use of a modified Clarke and Wright savings algorithm. In the mentioned works by Mańdziuk and Świechowski (2017, 2016), the MCTS algorithm was used to test various incremental modifications of the initial solutions. Therefore, an action of the MCTS algorithm resembled a mutation operator in the evolutionary algorithm. Juan et al. (2013) employ the MCTS algorithm to generate a distribution of various starting solutions. Each solution is sent to a parallel processing node for further modification.

Problem factorization—featured in papers by Kurzer et al. (2018), as well as Mańdziuk and Świechowski (2017, 2016) and Mańdziuk (2018). Here instead of maintaining one MCTS tree for the complete solution, each vehicle (route) is attributed with a separate tree that MCTS iterates over. However, when the simulation takes into account effects of applying actions in each tree in a synchronized manner. The final decision, i.e., "the action to play" is a vector of actions from each tree. *Macro-actions and action heuristics*—featured in Powley et al. (2013b), Mańdziuk (2018), Mańdziuk and Świechowski (2017, 2016), and Kurzer et al. (2018). In each approach, the authors proposed actions that are not the atomic actions available in the model. This modification was commonly referred to in this survey as it drastically reduces the search space. Kurzer et al. (2018) interleave macro actions with regular actions.

Replacement of the UCT formula—featured in Mańdziuk and Nejman (2015) and Kurzer et al. (2018). In the latter work the UCT formula is replaced by a solution assessment formula designed for the CVRP problem to better distinguish between good and poor routes. Three gradation patterns were tested - hyperbolic, linear and parabolic. Kurzer et al. (2018) replace the UCT formula by the $\epsilon - Greedy$ strategy.

Depth limited search—featured in Powley et al. (2013b). The fitness function was introduced to evaluate non-terminal states.

7.6 Multi-domain MCTS with heuristics

Sabar and Kendall (2015) present a relatively general framework that combines MCTS with heuristics. Herein, MCTS tree is composed of nodes that correspond to low level heuristics applied to given states. Therefore, the whole approach can be considered as a hierarchical hyper- or meta-heuristic. The feasibility of the approach was tested across six different domains. The results were competitive with the state-of-the-art methods.

8 Parallelization

The MCTS algorithm consists of iterations shown in Fig. 1. The iterations as well as the respective phases within an iteration are performed sequentially in the classical variant. However, there have been many approaches to parallelization of the algorithm predominantly aimed at increasing the overall number of iterations per second. The more statistics gathered, the more confident the MCTS/UCT evaluation. In many traditional approaches

to the parallel execution of algorithms such as matrix multiplication ones, the result of sequential and parallel versions must be exactly the same. However, in contrast to them, MCTS implementations drift away from the classic version of the algorithm in order to enable parallelism. Therefore, they can be considered variants or modifications of MCTS. For instance, the action selection formula may no longer be optimal, but multiple selections can be done at the same time and, as a result, the overall performance of the algorithm might be better.

8.1 Base approaches

There have been three major approaches to parallel MCTS proposed—*Leaf Parallelization*, *Root Parallelization* and *Tree Parallelization*.

(1) **Leaf Parallelization** (Cazenave and Jouandeau 2007)—the simplest variant, in which there is one MCTS/UCT tree, which is traversed in the regular way in the selection phase. Once a new node is expanded, there are K independent playouts performed in parallel instead of just one. All of them start with the same newly added node. The results of simulations are aggregated, e.g. added or averaged, and back-propagated in the regular fashion as in the sequential algorithm. In Leaf Parallelization, the tree does not grow quicker compared to the non-parallel variant. However, the evaluation of new nodes is more accurate.

(2) **Root Parallelization** (Chaslot et al. 2008a)—maintains K parallel processes. Each process builds its independent MCTS/UCT as the sequential algorithm would do. At certain synchronization points, e.g. just before making the actual (non-simulated) decision, the statistics from the first level in the tree are aggregated from the parallel processes. Typically, weighted aggregation is performed based on the number of visits. Although, this method does not result in deeper trees, the combined statistics of nodes are more confident. Moreover, Root Parallelization comes with minimal communication overhead.

(3) **Tree Parallelization** (Cazenave and Jouandeau 2008)—like in Leaf Parallelization and in contrast to Root Parallelization, there is one MCTS/UCT tree. The tree is shared, iterated and expanded by K parallel processes. In contrast to Leaf Parallelization, the whole iteration is done by such process and not only the playout. Therefore, synchronization mechanisms such as *locks* and *mutexes* are used. In order to avoid selecting the same node by subsequent processes, the so-called *Virtual Loss* is usually used. It tells to assign the worst possible outcome for the active players as soon as the node is visited and correct this result in the backpropagation phase. Such a mechanism reduces the UCT score and discourages the algorithm to follow the previous choices until the correct result is back-propagated.

The algorithms that dynamically modify policies are particularly difficult to implement in parallel. Graf and Platzner (2015) show that manipulation of a playout policy can reduce the efficiency of parallelization. The authors use adaptive weights (c.f. Fig. 1 in Sect. 3) that are updated after each playout and shared among all threads. Although they lead to a stronger player, they require synchronization using a global lock which is harmful for computational performance. The performance ratio between adaptive and static playouts in case of 16 threads computations has been measured as 68%.

8.2 Using specialized hardware

Barriga et al. (2014) propose a parallel UCT search that runs on modern GPUs. In their approach, the GPU constructs several independent trees that are combined using the idea of Root Parallelism. In addition, each three is assigned a multiple of 32 GPU threads for leaf parallelism, i.e., simulations done in parallel from the same node. The method was tested using the game *Ataxx*. Various setups parameterized by the number of trees and threads have been tested . The best variant achieved a 77.5% win-rate over the sequential counterpart. However, there are many challenges to such an approach. Firstly, the game simulator must be implemented on the GPU, which is often infeasible depending on the complexity of the game. Secondly, the programmer has no control over thread scheduling and CPU/GPU communication, which can make the approach risky to apply in tournament scenarios that require precise time controls.

Mirsoleimani et al. (2015) study the MCTS method running in parallel on Intel Xeon Phi, which is a processing unit designed for parallel computations. Each unit allows shared memory scaling up to 61 cores and 244 threads. In their paper, there are three contributions outlined: (1) the analysis of the performance of three common threading libraries on the new type of hardware (Xeon Phi), (2) a novel scheduling policy and (3) a novel parallel MCTS with grain size control. The authors report 5.6 times speedup of the modified MCTS variant compared to the sequential version on the Xeon E5-2596 processor. The proposed variant is based on Tree Parallelization. The iterations are split into *chunks*, which are tasks to be executed serially.

Finally, Hufschmitt et al. (2015) present a parallelization method aimed for *Multi-Purpose Processor Arrays* (MPPA). Each MPPA chip contains 256 processing cores organized into 16 clusters. The authors investigate parallelization of various elements and their impact on the overall performance of the resulting player. Firstly, distributed propnet evaluation is proposed. The propnet is an inference engine responsible for game-state manipulations, i.e., computing the next state in the simulation. This type of parallelization ended up introducing too much of the synchronization overhead. Secondly, playouts parallelization was evaluated requiring fewer synchronization layers. The authors report "good" scaling in three games (TicTacToe, Breakthrough and EightPuzzle) with the number of clusters ranging from 1 to 16 and the maximum number of 16 threads per cluster. However, relatively high synchronization overhead was still reported and authors conclude that there are many ways, in which the approach could be further improved.

8.3 Lock-free parallelization

A "lock" is a mechanism that allows to give a thread the exclusive access to certain code section. It is to prevent deadlocks, races, crashes and inconsistent memory state. In the MCTS algorithm, there are two critical moments that are usually placed within the lock—expansion of a new node and back-propagation. The shared selection phase can also be a potential race condition scenario depending on the implementation. Mirsoleimani et al. (2018) propose a novel lock-free algorithm that takes advantage of the C++ multi-threading-aware memory model and the *atomic* keyword. The authors shared their work as an open source library that contains various parallelization methods implemented in the lock-free fashion. The efficacy was shown using the game of Hex and both Tree Parallelization and Root Parallelization.

8.4 Root–tree parallelization

The approach proposed by Świechowski and Mańdziuk (2016a) defines a way of combining Root- and Tree Parallelization algorithms that leads to a very high scalability. The setup consists of the so-called worker nodes, hub nodes and the master node, for the purpose of message passing and scalability. Each worker node maintains its own MCTS/UCT tree that is expanded using Leaf Parallelization. On top of them, there is a Root Parallelization method (running on hub and master nodes) that gathers and aggregates statistics from worker nodes in a hierarchical manner. In addition, the authors propose a novel way, called *Limited Root–Tree Parallelization*, of splitting the tree into certain subtrees to be searched by worker nodes. The subtrees are not entirely disjoint as some level of redundancy is desirable for Root Parallelization. This allows to push the scaling boundary further when compared with just the hierarchical combination of Root and Tree Parallelization methods. The resulting hybrid approach is shown to be more effective than either method applied separately in a corpus of nine GGP games.

8.5 Game specific parallelization

Parallelism can become an enabler to solving games, which are not too combinatorially complex. One such example is Hex (Arneson et al. 2010). Liang et al. (2015) propose an approach to solving Hex in a parallel fashion. The work builds upon the Scalable Parallel Depth-First Proof-Number Search (SPDFPN) algorithm, which has the limitation that the maximum number of threads that can be utilized cannot be greater than the number of CPU cores. The proposed Job-Level UCT search no longer has this limitation. The authors introduced various techniques aimed at optimizing the workload sharing and communication between the threads. The resulting solver is able to solve four openings faster than the previous state-of-the-art approach. However, solving the game universally is not yet feasible.

Schaefers and Platzner (2014) introduce a novel approach to parallelization of MCTS in Go. The novelty includes efficient parallel transposition table and dedicated compute nodes that support broadcast operations. The algorithm was termed *UCT-Treesplit* and involves ranking nodes, load balancing and certain amount of node duplication (sharing). It is targeted at HPC clusters connected by *Infiniband*. The MPI library is employed for message passing. The major advantage of using the *UCT-Treesplit* algorithm is its great scalability and rich configuration options that enable to optimize the algorithm for the given distributed environment.

8.6 Information set parallelization

Sephton et al. (2014) report a study aimed at testing the base parallelization methods, i.e. Leaf, Root and Tree, adapted to Information Set MCTS (ISMCTS). In addition, the Tree Parallelization was included in two variants—with and without the so-called *Virtual Loss*. The game of choice for the experiments is a strategic card game *Lords of War*. The authors conclude that the Root Parallelization method is the most efficient approach when combined with ISMCTS.

9 Conclusions

MCTS is a *state-of-the-art* tree-search algorithm used mainly to implement AI behavior in games, although it can be used to support decision-making processes in other domains as well. The baseline algorithm, described in Sect. 2, was formulated in 2006, and since then multitude of enhancements and extensions to its vanilla formulation have been published. Our main focus in this survey is on works that have appeared since 2012, which is the time of the last major MCTS survey authored by Browne et al. (2012). Our literature analysis yielded 240 papers cited and discussed in this review, the vast majority of which fell within the above-mentioned time range. An overview of the considered papers grouped by application domains and by enhancements introduced to baseline MCTS are presented in Tables 1 and 2, respectively.

In their 2012 survey, Browne et al. (2012) concluded that MCTS would be "extensively hybridised with other search and optimisation algorithms". In the consecutive 8 years this prediction was positively validated in many papers and, based on their analysis, the following main trends of such hybridized approaches can be distinguished:

(1) **MCTS + ML**—combining MCTS with ML models, in particular deep neural networks trained with RL (Silver et al. 2016; Anthony et al. 2017), has become a significant research trend which is discussed in Sect. 5. Its popularity is to a large extent due to the unprecedented success of *AlphaGo* (Silver et al. 2016) and the rapid increase of interest in the ML field, in general. Further extensive growth in this direction can be safely envisaged.

(2) **AutoMCTS**—an enhancement of MCTS with self-adaptation mechanisms (Świechowski and Mańdziuk 2014; Gaina et al. 2020), which rely on finding optimal hyperparameters or the most adequate policies for a given problem being solved. As discussed in Sect. 5, MCTS is often combined with optimization techniques (e.g. evolutionary approaches) that are executed *offline* in order to find the set of parameters for the *online* MCTS usage (Horn et al. 2016; Lucas et al. 2014; Pettit and Helmbold 2012; Bravi et al. 2016).

(3) **MCTS + Domain Knowledge**—an inclusion of domain knowledge into MCTS operation, either in the form of the heavy playouts, or action prioritization, or exclusion of certain paths in the search space (action reduction), which are not viable or plausible due to the possessed expert knowledge. Other possibilities include building a light evaluation function for action assessment or early MCTS termination due to certain problem-related knowledge (Lorentz 2016).

Elaborating more on the last point, let us observe that MCTS has become the state-of-the-art technique for making strong computer players in turn-based combinatorial games. It has led to many breakthroughs, e.g. in Go and Chess. One of its main strengths is no need for game-related expert knowledge. In the vanilla form, only the simulation engine, i.e. rules of the game, is needed. Hence, it has been widely applied in multi-game environments such as GGP and General Video Game AI.

At the same time, embedding knowledge into MCTS without introducing too much bias is a potential way of improving its performance.

We expect that injecting knowledge into MCTS, especially in a semi-automatic fashion, is one of the future directions. We consider a direct inclusion of a human knowledge into the MCTS algorithm as a special case of domain knowledge utilization:

Table 1 References grouped by application

Games	References
Go	Silver et al. (2018, 2016, 2017), Browne (2012), Graf and Platzner (2016, 2015), Chen (2012), Ikeda and Viennot (2013b), Baier and Winands (2015), Graf and Platzner (2014), Yang et al. (2020), Wu et al. (2018), Brügmann (1993), Coulom (2007), Drake and Uurtamo (2007)
Hex	Gao et al. (2018, 2017), Takada et al. (2019) Huang et al. (2013), Gao et al. (2019)
Other with perfect information	Chang et al. (2018), Tang et al. (2016), Zhuang et al. (2015), Soemers et al. (2019), Benbassat and Sipper (2013)
Card games	Cowling et al. (2012b), Santos et al. (2017), Świechowski et al. (2018), Zhang and Buro (2017), Sephton et al. (2014, 2015), Baier et al. (2018), Ihara et al. (2018), Di Palma and Lanzi (2018), Choe and Kim (2019), Santos et al. (2017), Godlewski and Sawicki (2021), Goodman (2019)
Arcade Video games	Alhejali and Lucas (2013), Pepels and Winands (2012), Pepels et al. (2014), Nguyen and Thawonmas (2012), Jacobsen et al. (2014), Kartal et al. (2019a), Guo et al. (2014), Pinto and Coutinho (2018), Ilhan and Etaner-Uyar (2017), Perick et al. (2012), Gedda et al. (2018), Zhou et al. (2018)
Real Time Strategy games	Uriarte and Ontañón (2017), Justesen et al. (2014), Moraes et al. (2018), Uriarte and Ontañón (2014), Yang and Ontañón (2018), Ontañón (2016), Uriarte and Ontañón (2016) Preuss and Risi (2020)
General Game Playing	Waledzik and Mańdziuk (2011), Świechowski and Mańdziuk (2014), Cazenave (2016), Sironi et al. (2018), Nelson (2016), Finnsson and Björnsson (2008), Finnsson and Björnsson (2010), Waledzik and Mańdziuk (2014), Trutman and Schiffel (2015), Sironi and Winands (2016), Świechowski et al. (2016), Sironi and Winands (2019)
General Video Game Playing	Park and Kim (2015), Joppen et al. (2017), Frydenberg et al. (2015), Guerrero-Romero et al. (2017), Perez et al. (2014), Horn et al. (2016), Gaina et al. (2017a), De Waard et al. (2016), Soemers et al. (2016)
Game development	Hunicke (2005), Keehl and Smith (2018, 2019), Maia et al. (2017), Baier et al. (2018), Zook et al. (2019), Demediuk et al. (2017), Ishihara et al. (2018), Khalifa et al. (2016)
Non-games	References
Optimisation	Uriarte and Ontañón (2016), Kuipers et al. (2013), Jia et al. (2020), Shi et al. (2020), Segler et al. (2018), Awais et al. (2018), Gaymann and Montomoli (2019), Sabar and Kendall (2015)
Security	Karwowski and Mańdziuk (2015, 2019a), Paruchuri et al. (2008), Karwowski and Mańdziuk (2020)
Chemical Synthesis	Segler et al. (2018), Awais et al. (2018), Gaymann and Montomoli (2019)
Planning and Scheduling	Ghallab et al. (2004), Anand et al. (2015), Waledzik et al. (2014), Anand et al. (2016), Keller and Eyerich (2012), Painter et al. (2020), Feldman and Domshlak (2014), Waledzik et al. (2015), Vien and Toussaint (2015), Patra et al. (2020), Best et al. (2019), Wu et al. (2015), Clary et al. (2018), Hu et al. (2019), Amer et al. (2013), Neto et al. (2020), Asta et al. (2016), Wijaya et al. (2013), Waledzik and Mandziuk (2018)
Vehicle Routing	Caceres-Cruz et al. (2014), Powley et al. (2013b), Mańdziuk (2018), Mańdziuk and Świechowski (2017), Mańdziuk and Nejman (2015), Mańdziuk and Świechowski (2016) , Juan et al. (2013), Kurzer et al. (2018) Mańdziuk (2019)

(4) **Human-in-the-Loop**—a direct implosion of human expert domain knowledge to the MCTS operational pattern by means of a *direct, online* steering of the MCTS by the

Table 2 References grouped by method, part 1/2, continued on Table 3

Methods	Section	References
Classic MCTS	2	Kocsis and Szepesvári (2006), Gelly and Wang (2006), Auer et al. (2002), Thompson (1933), Bai et al. (2013), Gelly and Silver (2011), Finnsson and Björnsson (2010), Finnsson and Björnsson (2011), Kishimoto and Schaeffer (2002), Schaeffer (1989), Tak et al. (2012), Gaudel et al. (2010), Chaslot et al. (2009)
Action Reduction	3.1	Sephton et al. (2014), Justesen et al. (2014), Churchill and Buro (2013), De Waard et al. (2016), Subramanian et al. (2016), Gabor et al. (2019), Moraes et al. (2018), Uriarte and Ontanón (2014), Ontanón (2016), Baier and Winands (2012), Pepels and Winands (2012), Soemers et al. (2016), Zhou et al. (2018), Chaslot et al. (2008b), Gedda et al. (2018), Liu and Tsuruoka (2015), Mandai and Kaneko (2016), Tak et al. (2014), Yee et al. (2016), Perick et al. (2012) Ikeda and Viennot (2013a), Coulom (2007), Imagawa and Kaneko (2015), Demediuk et al. (2017)
UCT Alternatives	3.2	Brügmann (1993), Gelly et al. (2012), Sironi and Winands (2016), Cazenave (2015), Sarratt et al. (2014), Sironi and Winands (2019), Browne (2012), Imagawa and Kaneko (2015), Graf and Platzner (2016, 2015), Gudmundsson and Björnsson (2013), Baier and Winands (2014), Baier and Mark (2013)
Early Termination	3.3	Lorentz (2016), Lanctot et al. (2014), Waledzik and Mańdziuk (2014), Wu et al. (2018), Goodman (2019), Cazenave (2015), Sironi and Winands (2016)
Determinization	4.1	Cowling et al. (2012a), Cowling et al. (2012b)
Information Sets	4.2	Cowling et al. (2012a), Karwowski and Mańdziuk (2015), Wang et al. (2015), Cowling et al. (2015), Lisý et al. (2015), Furtak and Buro (2013), Karwowski and Mańdziuk (2019a), Ihara et al. (2018), Di Palma and Lanzi (2018), Sephton et al. (2015), Karwowski and Mańdziuk (2020), Goodman (2019), Powley et al. (2013a), Uriarte and Ontañón (2017)
Heavy Playouts	4.3	Browne (2012), Świechowski and Mańdziuk (2014), Alhejali and Lucas (2013), Godlewski and Sawicki (2021)
Policy Update	4.4	Kao et al. (2013), Cazenave (2015), Cazenave (2016) (Trutman and Schiffel 2015), Santos et al. (2017), (Świechowski et al. 2018)
Master Combination	4.5	Świechowski (2020), Nguyen and Thawonmas (2012), Pepels and Winands (2012), Pepels et al. (2014), Jacobsen et al. (2014), Nelson (2016), Soemers et al. (2016), Tak et al. (2012), Frydenberg et al. (2015), Guerrero-Romero et al. (2017), Kim and Kim (2017), Khalifa et al. (2016)
Opponent Modelling	4.6	Goodman and Lucas (2020), Cowling et al. (2015), Baier and Winands (2012), Kim and Kim (2017), Hunicke (2005), Nguyen and Thawonmas (2012), Baier et al. (2018)

Table 3 References grouped by method, part 2/2, continuation from Table 2

Methods	Section	References
MCTS + Neural Networks	5.3	Chang et al. (2018), Gao et al. (2018, 2017), Takada et al. (2019), Yang et al. (2020), Wu et al. (2018), Tang et al. (2016), Gao et al. (2019) Zhang and Buro (2017), Świechowski et al. (2018), Zhuang et al. (2015), Yang and Ontañón (2018), Baier et al. (2018), Soemers et al. (2019), Kartal et al. (2019a), Guo et al. (2014), Pinto and Coutinho (2018)
MCTS + Reinforcement Learning	5	Silver et al. (2018, 2016, 2017), Vodopivec and Šter (2014), Ilhan and Etaner-Uyar (2017), Anthony et al. (2017)
MCTS + Evolutionary Algorithms	6	Benbassat and Sipper (2013), Alhejali and Lucas (2013), Lucas et al. (2014), Pettit and Helmbold (2012), Bravi et al. (2016), Bravi et al. (2016), Perez et al. (2013), Gaina et al. (2021), Baier and Cowling (2018), Perez et al. (2014), Horn et al. (2016), Gaina et al. (2017a)
Classic Parallelization	8	Cazenave and Jouandeau (2007), Chaslot et al. (2008a), Cazenave and Jouandeau (2008)
New Parallelization	8	Graf and Platzner (2015), Barriga et al. (2014), Mirsoleimani et al. (2015), Hufschmitt et al. (2015), Mirsoleimani et al. (2018), Świechowski and Mańdziuk (2016a), Arneson et al. (2010), Liang et al. (2015), Schaefers and Platzner (2014), Sephton et al. (2014)

human operator (Świechowski et al. 2015a). A human expert has the power of excluding parts of the search tree, disable/enable certain actions or increase the frequency of visits in certain search space states (tree search nodes). In this context, it may be worth trying to develop some frameworks for semi-automatic translation of expert knowledge into MCTS steering or adjusting so as to make the above Human–Machine model of cooperation as effortless and efficient as possible. We believe that such direct teaming of the human expert (knowledge) with (the search speed of) MCTS has potential to open new research path in various complex domains, extending beyond combinatorial games or deterministic problems.

The fifth MCTS related research prospect that we envisage is

(5) **MCTS in New Domains**—in principle, MCTS can be applied to any sequential decision-making problem that can be represented in the form of a tree whose leaf nodes (solutions) can be assessed using some form of an evaluation/reward function. This includes combinatorial optimization (Sabar and Kendall 2015), planning (Feldman and Domshlak 2014), and scheduling (Neto et al. 2020) problems encountered in a variety of fields, e.g. logistics (Kurzer et al. 2018), chemistry (Segler et al. 2018), security (Karwowski and Mańdziuk 2019a), or in UAV (Unmanned Aerial Vehicle) navigation (Chen et al. 2016). We are quite confident that MCTS will gradually become more and more popular in the areas other than combinatorial games.

Finally, one of the potential research prospect, which we regard more as an opportunity than a definite prediction, is

(6) **MCTS in Real Time Games**—although MCTS has been applied to some real-time video games such as *Pac-Man* (Pepels et al. 2014) or *Starcraft* (Uriarte and Ontañón 2016), it is not as popular a tool of choice in this field as it is in combinatorial games. Rabin (2013) summarizes the techniques that are predominantly used in real-time video games, which include Behaviour Trees, Goal-Oriented Action Planning, Utility AI, Finite State Machines and even simple scripts. The main reason for using them is the limited computational budget. Video games tend to have quite complex state representations and high branching factors. Moreover, the state can change rapidly in real-time. In research environment, it is often possible to allocate all available resources to the AI algorithms. In commercial games, the AI module is allowed to use only a fraction of computational power of a single consumer-level machine.

Besides computational complexity, the other reason for MCTS not to be the first choice in video games is the degree of unpredictability introduced by stochastic factors. The game studios often require all interactions to be thoroughly tested before a game is released. Developing more formal methods of predicting and controlling the way MCTS-driven bots will behave in new scenarios in the game seems to be another promising future direction.

The third impediment of MCTS application to video games is the difficulty of implementing a forward model able to simulate games in a combinatorial-like fashion (Preuss and Risi 2020). This issue has already been addressed by using simplified (proxy) game models and letting the MCTS algorithm operate either on abstraction (simplification) of the actual game or on some subset of the game (using a relaxed model).

In the view of the above, we believe that some methodological and/or computational breakthroughs may be required before MCTS becomes a state-of-the-art (or sufficiently efficient) approach in RTS games domain.

In summary, the main advantages of MCTS which differentiate this method from the majority of other search algorithms can be summarized as follows. First of all, the problem tree is built asymmetrically, with the main focus on the most promising lines of action and exploring options that may most likely become promising. Second of all, the UCT formula is the most popular way of assuring the *exploration vs. exploitation* balance. In practice, the method has proven efficient in various (though not all) search problems. Its additional asset is the theoretical justification of convergence. Furthermore, the method is inherently easy to parallelize (cf. Sect. 8). And finally, MCTS is the so-called *anytime* algorithm, i.e. it can be stopped at any moment and still return a reasonably good solution (the best one found so-far).

On the downside, since MCTS is a tree-search algorithm, it shares some weaknesses with other space-search. For instance, it is prone to search space explosion. MCTS modifications, such as model simplifications or reducing the actions space are one way to tackle this problem. A different idea is to use MCTS as an *offline* teacher that can be used for sufficiently long time and generate high-quality training data. Yet another method that is not search-based but rather learning-based and inherently capable of generalizing knowledge (e.g. an ML model) is trained on these MCTS-generated training samples. Consequently, a computationally heavy process is run just once (*offline*) and then this time-efficient problem representation can be used in subsequent *online* applications. The approach of combining an MCTS trainer with a fast learning-based representation can be hybridised in various ways, specific to a particular problem/domain of interest (Kartal et al. 2019a; Guo et al. 2014; Soemers et al. 2019).

Finally, we would like to stress the fact that MCTS, originally created for *Go*, was for a long time devoted to combinatorial games and only recently started to "expand" its usage beyond this domain. We hope that the researchers working with other genres of games or in other fields will make more frequent attempts at MCTS utilization in their domains, perhaps inspired by the MCTS modifications discussed in this survey.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

Ethical approval Not applicable.

Informed consent Not applicable.

Consent for publication The authors declare that they received permissions from the original authors to reuse their figures in this article. This applies to Figs. 7, 8, 9 and 11, 12, 13, 14. The remaining figures have been entirely created by the authors of this manuscript.

References

Alhejali AM, Lucas SM (2013) Using genetic programming to evolve heuristics for a Monte Carlo Tree Search Ms Pac-Man agent. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8

Amer MR, Todorovic S, Fern A, Zhu SC (2013) Monte Carlo Tree Search for scheduling activity recognition. In: Proceedings of the IEEE international conference on computer vision, pp 1353–1360

An B, Ordóñez F, Tambe M, Shieh E, Yang R, Baldwin C, DiRenzo J, Moretti K, Maule B, Meyer G (2013) A deployed quantal response-based patrol planning system for the U.S. coast guard. Interfaces 43(5):400–420

Anand A, Grover A, Mausam M, Singla P (2015) ASAP-UCT: abstraction of state–action pairs in UCT. In: IJCAI

Anand A, Noothigattu R, Singla P (2016) OGA-UCT: on-the-go abstractions in UCT. In: Proceedings of the twenty-sixth international conference on international conference on automated planning and scheduling, pp 29–37

Anthony T, Tian Z, Barber D (2017) Thinking fast and slow with deep learning and tree search. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30. Curran Associates, Inc

Arneson B, Hayward RB, Henderson P (2010) Monte Carlo tree search in Hex. IEEE Trans Comput Intell AI Games 2(4):251–258

Asta S, Karapetyan D, Kheiri A, Özcan E, Parkes AJ (2016) Combining Monte Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. Inf Sci 373:476–498

Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed Bandit problem. Mach Learn 47(2–3):235–256

Awais M, Mohammadi HG, Platzner M (2018) An MCTS-based framework for synthesis of approximate circuits. In: 2018 IFIP/IEEE international conference on very large scale integration (VLSI-SoC). IEEE, pp 219–224

Ba S, Hiraoka T, Onishi T, Nakata T, Tsuruoka Y (2019) Monte Carlo Tree Search with variable simulation periods for continuously running tasks. In: 2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI). IEEE, pp 416–423

Bai A, Wu F, Chen X (2013) Bayesian mixture modelling and inference based Thompson sampling in Monte Carlo Tree Search. In: Proceedings of the advances in neural information processing systems (NIPS), pp 1646–1654

Baier H, Cowling PI (2018) Evolutionary MCTS for multi-action adversarial games. In: 2018 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Baier H, Kaisers M (2020) Guiding multiplayer MCTS by focusing on yourself. In: 2020 IEEE Conference on Games (CoG). IEEE, pp 550–557

Baier H, Mark HW (2013) Monte Carlo Tree Search and minimax hybrids. In: 2013 IEEE conference on Computational Intelligence in Games (CIG). IEEE, pp 1–8

Baier H, Winands MH (2012) Beam Monte Carlo Tree Search. In: 2012 IEEE conference on Computational Intelligence and Games (CIG). IEEE, pp 227–233

Baier H, Winands MH (2014) MCTS-minimax hybrids. IEEE Trans Comput Intell AI Games 7(2):167–179

Baier H, Winands MH (2015) Time management for Monte Carlo Tree Search. IEEE Trans Comput Intell AI Games 8(3):301–314

Baier H, Sattaur A, Powley EJ, Devlin S, Rollason J, Cowling PI (2018) Emulating human play in a leading mobile card game. IEEE Trans Games 11(4):386–395

Barriga NA, Stanescu M, Buro M (2014) Parallel UCT search on GPUs. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–7

Barriga NA, Stanescu M, Buro M (2017) Combining strategic learning and tactical search in Real-Time Strategy Games. In: Proceedings, the thirteenth AAAI conference on artificial intelligence and interactive digital entertainment (AIIDE-17)

Benbassat A, Sipper M (2013) EvoMCTS: enhancing MCTS-based players through genetic programming. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8

Best G, Cliff OM, Patten T, Mettu RR, Fitch R (2019) Dec-MCTS: decentralized planning for multi-robot active perception. Int J Robot Res 38(2–3):316–337

Bondi E, Oh H, Xu H, Fang F, Dilkina B, Tambe M (2020) To signal or not to signal: exploiting uncertain real-time information in signaling games for security and sustainability. In: Proceedings of the thirty-fourth AAAI conference on artificial intelligence, pp 1369–1377

Bravi I, Khalifa A, Holmgard C, Togelius J (2016) Evolving UCT alternatives for general video game playing. In: International joint conference on artificial intelligence, pp 63–69

Bravi I, Khalifa A, Holmgård C, Togelius J (2017) Evolving game-specific UCB alternatives for general video game playing. In: European conference on the applications of evolutionary computation. Springer, pp 393–406

Browne C (2012) A problem case for UCT. IEEE Trans Comput Intell AI Games 5(1):69–74

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

Brügmann B (1993) Monte Carlo Go. Technical report. Citeseer

Caceres-Cruz J, Arias P, Guimarans D, Riera D, Juan AA (2014) Rich vehicle routing problem: survey. ACM Comput Surv (CSUR) 47(2):1–28

Campbell M, Hoane AJ Jr, Hsu Fh (2002) Deep blue. Artif Intell 134(1–2):57–83

Cazenave T (2015) Generalized rapid action value estimation. In: 24th International conference on artificial intelligence, pp 754–760

Cazenave T (2016) Playout policy adaptation with move features. Theor Comput Sci 644:43–52

Cazenave T, Jouandeau N (2007) On the parallelization of UCT. In: Proceedings of CGW07, pp 93–101

Cazenave T, Jouandeau N (2008) A parallel Monte Carlo Tree Search algorithm. In: International conference on computers and games. Springer, pp 72–80

Chang NY, Chen CH, Lin SS, Nair S (2018) The big win strategy on multi-value network: an improvement over AlphaZero approach for 6×6 Othello. In: Proceedings of the 2018 international conference on machine learning and machine intelligence, pp 78–81

Chaslot GMB, Winands MH, van den Herik HJ (2008a) Parallel Monte Carlo Tree Search. In: International conference on computers and games. Springer, pp 60–71

Chaslot GMB, Hoock JB, Perez J, Rimmel A, Teytaud O, Winands MH (2009) Meta Monte Carlo Tree Search for automatic opening book generation. In: Proceedings of the 21st international joint conference on artificial intelligence, Pasadena, California, pp 7–12

Chaslot GMJ, Winands MH, Herik HJVD, Uiterwijk JW, Bouzy B (2008) Progressive strategies for Monte Carlo Tree Search. N Math Nat Comput 4(03):343–357

Chen KH (2012) Dynamic randomization and domain knowledge in Monte Carlo Tree Search for Go knowledge-based systems. Knowl Based Syst 34:21–25

Chen X, Zhang X, Wang W, Wei W (2016) Multi-objective Monte Carlo Tree Search based aerial maneuvering control. In: 2016 IEEE Chinese guidance, navigation and control conference (CGNCC), pp 81–87

Choe JSB, Kim J (2019) Enhancing Monte Carlo Tree Search for playing Hearthstone. In: 2019 IEEE conference on games (CoG), pp 1–7

Churchill D, Buro M (2013) Portfolio greedy search and simulation for large-scale combat in Starcraft. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8

Clary P, Morais P, Fern A, Hurst JW (2018) Monte Carlo planning for agile legged locomotion. In: ICAPS, pp 446–450

Coulom R (2006) Efficient selectivity and backup operators in Monte Carlo Tree Search. In: International conference on computers and games. Springer, pp 72–83

Coulom R (2007) Computing "elo ratings'' of move patterns in the game of Go. ICGA J 30(4):198–208

Cowling PI, Powley EJ, Whitehouse D (2012) Information set Monte Carlo Tree Search. IEEE Trans Comput Intell AI Games 4(2):120–143

Cowling PI, Ward CD, Powley EJ (2012) Ensemble determinization in Monte Carlo Tree Search for the imperfect information card game Magic: The Gathering. IEEE Trans Comput Intell AI Games 4(4):241–257

Cowling PI, Whitehouse D, Powley EJ (2015) Emergent bluffing and inference with Monte Carlo Tree Search. In: 2015 IEEE conference on computational intelligence and games (CIG), pp 114–121

De Waard M, Roijers DM, Bakkes SC (2016) Monte Carlo Tree Search with options for General Video Game Playing. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Demediuk S, Tamassia M, Raffe WL, Zambetta F, Li X, Mueller F (2017) Monte Carlo Tree Search based algorithms for dynamic difficulty adjustment. In: 2017 IEEE conference on computational intelligence and games (CIG). IEEE, pp 53–59

Di Palma S, Lanzi PL (2018) Traditional wisdom and Monte Carlo Tree Search face-to-face in the card game Scopone. IEEE Trans Games 10(3):317–332

Drake P, Uurtamo S (2007) Move ordering vs heavy playouts: where should heuristics be applied in Monte Carlo Go. In: Proceedings of the 3rd North American game-on conference. Citeseer, pp 171–175

Fang F, Stone P, Tambe M (2015) When Security Games go green: designing defender strategies to prevent poaching and illegal fishing. In: Proceedings of the twenty-fourth international joint conference on artificial intelligence. AAAI Press, pp 2589–2595

Farooq SS, Oh IS, Kim MJ, Kim KJ (2016) StarCraft AI competition report. AI Mag 37(2):102–107

Feldman Z, Domshlak C (2014) Simple regret optimization in online planning for Markov decision processes. J Artif Intell Res 51:165–205

Finnsson H, Björnsson Y (2008) Simulation-based approach to General Game Playing. AAAI 8:259–264

Finnsson H, Björnsson Y (2010) Learning Simulation Control in General Game-Playing Agents. AAAI 10:954–959

Finnsson H, Björnsson Y (2011) Cadiaplayer: search-control techniques. Künstl Intell 25(1):9–16

Frank I, Basin D (1998) Search in games with incomplete information: a case study using Bridge card play. Artif Intell 100(1–2):87–123

Frydenberg F, Andersen KR, Risi S, Togelius J (2015) Investigating MCTS modifications in General Video Game Playing. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 107–113

Furtak T, Buro M (2013) Recursive Monte Carlo search for imperfect information games. In: 2013 IEEE conference on computational intelligence in games (CIG), pp 1–8

Gabor T, Peter J, Phan T, Meyer C, Linnhoff-Popien C (2019) Subgoal-based temporal abstraction in Monte Carlo Tree Search. In: IJCAI, pp 5562–5568

Gaina RD, Liu J, Lucas SM, Pérez-Liébana D (2017a) Analysis of Vanilla rolling horizon evolution parameters in general video game playing. In: European conference on the applications of evolutionary computation. Springer, pp 418–434

Gaina RD, Lucas SM, Perez-Liebana D (2017b) Rolling horizon evolution enhancements in general video game playing. In: 2017 IEEE conference on computational intelligence and games (CIG). IEEE, pp 88–95

Gaina RD, Perez-Liebana D, Lucas SM, Sironi CF, Winands MH (2020) Self-adaptive rolling horizon evolutionary algorithms for general video game playing. In: 2020 IEEE conference on games (CoG). IEEE, pp 367–374

Gaina RD, Devlin S, Lucas SM, Perez D (2021) Rolling horizon evolutionary algorithms for general video game playing. IEEE Trans Games. https://doi.org/10.48550/arXiv.2003.12331

Gao C, Hayward R, Müller M (2017) Move prediction using deep convolutional neural networks in Hex. IEEE Trans Games 10(4):336–343

Gao C, Müller M, Hayward R (2018) Three-head neural network architecture for Monte Carlo Tree Search. In: Twenty-seventh international joint conference on artificial intelligence, pp 3762–3768

Gao C, Takada K, Hayward R (2019) Hex 2018: MoHex3HNN over DeepEzo. J Int Comput Games Assoc 41(1):39–42

Gaudel R, Hoock JB, Pérez J, Sokolovska N, Teytaud O (2010) A principled method for exploiting opening books. In: International conference on computers and games. Springer, pp 136–144

Gaymann A, Montomoli F (2019) Deep neural network and Monte Carlo Tree Search applied to fluid–structure topology optimization. Sci Rep 9(1):1–16

Gedda M, Lagerkvist MZ, Butler M (2018) Monte Carlo methods for the game Kingdomino. In: 2018 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Gelly S, Silver D (2011) Monte Carlo Tree Search and rapid action value estimation in computer Go. Artif Intell 175(11):1856–1875

Gelly S, Wang Y (2006) Exploration exploitation in Go: UCT for Monte Carlo Go. In: NIPS: neural information processing systems conference on-line trading of exploration and exploitation workshop, Canada. https://hal.archives-ouvertes.fr/hal-00115330

Gelly S, Kocsis L, Schoenauer M, Sebag M, Silver D, Szepesvári C, Teytaud O (2012) The grand challenge of computer Go: Monte Carlo Tree Search and extensions. Commun ACM 55(3):106–113

Genesereth M, Thielscher M (2014) General game playing. Synth Lect Artif Intell Mach Learn 8(2):1–229

Ghallab M, Nau D, Traverso P (2004) Automated planning: theory and practice. Elsevier, Burlington

Gholami S, McCarthy S, Dilkina B, Plumptre AJ, Tambe M, Driciru M, Wanyama F, Rwetsiba A, Nsubaga M, Mabonga J et al (2018) Adversary models account for imperfect crime data: forecasting and planning against real-world poachers. In: Proceedings of the 17th international conference on autonomous agents and multiagent systems, pp 823–831

Godlewski K, Sawicki B (2021) Optimisation of MCTS Player for The Lord of the Rings: the card game. Bull Pol Acad Sci Tech Sci. https://doi.org/10.48550/arXiv.2109.12001

Goodman J (2019) Re-determinizing MCTS in Hanabi. In: 2019 IEEE conference on games (CoG). IEEE, pp 1–8

Goodman J, Lucas S (2020) Does it matter how well I know what you're thinking? Opponent modelling in an RTS game. In: 2020 IEEE congress on evolutionary computation (CEC). IEEE, pp 1–8

Graepel T, Goutrie M, Krüger M, Herbrich R (2001) Learning on graphs in the game of Go. In: International conference on artificial neural networks. Springer, pp 347–352

Graf T, Platzner M (2014) Common fate graph patterns in Monte Carlo Tree Search for computer Go. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–8

Graf T, Platzner M (2015) Adaptive playouts in Monte Carlo Tree Search with policy-gradient reinforcement learning. In: Advances in computer games. Springer, Cham, pp 1–11

Graf T, Platzner M (2016) Adaptive playouts for online learning of policies during Monte Carlo Tree Search. Theor Comput Sci 644:53–62

Gudmundsson SF, Björnsson Y (2013) Sufficiency-based selection strategy for MCTS. In: Twenty-third international joint conference on artificial intelligence. Citeseer

Guerrero-Romero C, Louis A, Perez-Liebana D (2017) Beyond playing to win: diversifying heuristics for GVGAI. In: 2017 IEEE conference on computational intelligence and games (CIG). IEEE, pp 118–125

Guo X, Singh S, Lee H, Lewis RL, Wang X (2014) Deep learning for real-time Atari game play using offline Monte Carlo Tree Search planning. In: Advances in neural information processing systems, pp 3338–3346

Hennes D, Izzo D (2015) Interplanetary trajectory planning with Monte Carlo Tree Search. In: Twenty-fourth international joint conference on artificial intelligence. Citeseer

Horn H, Volz V, Pérez-Liébana D, Preuss M (2016) MCTS/EA hybrid GVGAI players and game difficulty estimation. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

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

Huang SC, Arneson B, Hayward RB, Müller M, Pawlewicz J (2013) MoHex 2.0: a pattern-based MCTS Hex player. In: International conference on computers and games. Springer, pp 60–71

Hufschmitt A, Mehat J, Vittaut JN (2015) MCTS playouts parallelization with a MPPA architecture. In: 4th Workshop on general intelligence in game-playing agents, GIGA 2015

Hunicke R (2005) The case for dynamic difficulty adjustment in games. In: Proceedings of the 2005 ACM SIGCHI international conference on advances in computer entertainment technology, pp 429–433

IEEE-CoG (2020) Call for competitions. In: IEEE conference on games. https://ieee-cog.org/2020/competitions

Ihara H, Imai S, Oyama S, Kurihara M (2018) Implementation and evaluation of information set Monte Carlo Tree Search for Pokémon. In: 2018 IEEE international conference on systems, man, and cybernetics (SMC), pp 2182–2187

Ikeda K, Viennot S (2013a) Efficiency of static knowledge bias in Monte Carlo Tree Search. In: International conference on computers and games. Springer, pp 26–38

Ikeda K, Viennot S (2013b) Production of various strategies and position control for Monte Carlo Go-entertaining human players. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8

Ilhan E, Etaner-Uyar AŞ (2017) Monte Carlo Tree Search with temporal-difference learning for General Video Game Playing. In: 2017 IEEE conference on computational intelligence and games (CIG). IEEE, pp 317–324

Imagawa T, Kaneko T (2015) Enhancements in Monte Carlo Tree Search algorithms for biased game trees. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 43–50

Ishihara M, Ito S, Ishii R, Harada T, Thawonmas R (2018) Monte Carlo Tree Search for implementation of dynamic difficulty adjustment fighting game AIs having believable behaviors. In: 2018 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Jacobsen EJ, Greve R, Togelius J (2014) Monte Mario: platforming with MCTS. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation, pp 293–300

Jain M, Tsai J, Pita J, Kiekintveld C, Rathi S, Tambe M, Ordóñez F (2010) Software assistants for randomized patrol planning for the LAX Airport Police and the Federal Air Marshal Service. Interfaces 40(4):267–290

Jia J, Chen J, Wang X (2020) Ultra-high reliable optimization based on Monte Carlo Tree Search over Nakagami-m Fading. Appl Soft Comput 91:106244

Joppen T, Moneke MU, Schröder N, Wirth C, Fürnkranz J (2017) Informed hybrid game tree search for General Video Game Playing. IEEE Trans Games 10(1):78–90

Juan AA, Faulin J, Jorba J, Caceres J, Marquès JM (2013) Using parallel and distributed computing for real-time solving of vehicle routing problems with stochastic demands. Ann Oper Res 207(1):43–65

Junghanns A (1998) Are there practical alternatives to alpha–beta? ICGA J 21(1):14–32

Justesen N, Tillman B, Togelius J, Risi S (2014) Script- and cluster-based UCT for Starcraft. In: 2014 IEEE conference on computational intelligence and games, pp 1–8

Kao KY, Wu IC, Yen SJ, Shan YC (2013) Incentive learning in Monte Carlo Tree Search. IEEE Trans Comput Intell AI Games 5(4):346–352

Kartal B, Hernandez-Leal P, Taylor ME (2019) Action guidance with MCTS for deep reinforcement learning. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 15, pp 153–159

Kartal B, Hernandez-Leal P, Taylor ME (2019b) Action guidance with MCTS for deep reinforcement learning. arxiv: 1907.11703

Karwowski J, Mańdziuk J (2015) A new approach to security games. In: Artificial intelligence and soft computing—14th international conference, ICAISC 2015, proceedings, Part II, lecture notes in computer science, Zakopane, Poland, vol 9120. Springer, pp 402–411

Karwowski J, Mańdziuk J (2016) Mixed strategy extraction from UCT tree in security games. In: ECAI 2016—22nd European conference on artificial intelligence—including prestigious applications of artificial intelligence (PAIS 2016), frontiers in artificial intelligence and applications, The Hague, The Netherlands, vol 285. IOS Press, pp 1746–1747

Karwowski J, Mańdziuk J (2019) A Monte Carlo Tree Search approach to finding efficient patrolling schemes on graphs. Eur J Oper Res 277(1):255–268

Karwowski J, Mańdziuk J (2019b) Stackelberg Equilibrium Approximation in general-sum extensive-form games with double-oracle sampling method. In: Proceedings of the 18th international conference on autonomous agents and multiagent systems, AAMAS '19, Montreal, QC, Canada. International Foundation for Autonomous Agents and Multiagent Systems, pp 2045–2047

Karwowski J, Mańdziuk J (2020) Double-oracle sampling method for Stackelberg Equilibrium Approximation in general-sum extensive-form games. In: The thirty-fourth AAAI conference on artificial intelligence, AAAI 2020, NY, USA. AAAI Press, New York, pp 2054–2061

Karwowski J, Mańdziuk J, Zychowski A (2020) Anchoring theory in Sequential Stackelberg Games. In: Proceedings of the 19th international conference on autonomous agents and multiagent systems, AAMAS '20, Auckland, New Zealand. International Foundation for Autonomous Agents and Multiagent Systems, pp 1881–1883

Keehl O, Smith AM (2018) Monster Carlo: an MCTS-based framework for machine playtesting Unity games. In: 2018 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Keehl O, Smith AM (2019) Monster Carlo 2: integrating learning and tree search for machine playtesting. In: 2019 IEEE conference on games (CoG). IEEE, pp 1–8

Keller T, Eyerich P (2012) PROST: probabilistic planning based on UCT. In: ICAPS, pp 119–127

Khalifa A, Isaksen A, Togelius J, Nealen A (2016) Modifying MCTS for human-like General Video Game Playing. In: IJCAI, pp 2514–2520

Kim MJ, Kim KJ (2017) Opponent modeling based on action table for MCTS-based fighting game AI. In: 2017 IEEE conference on computational intelligence and games (CIG). IEEE, pp 178–180

Kishimoto A, Schaeffer J (2002) Transposition table driven work scheduling in distributed game-tree search. In: Proceedings of the 15th conference of the Canadian Society for Computational Studies of Intelligence on advances in artificial intelligence. AI '02. Springer, London, pp 56–68

Kocsis L, Szepesvári C (2006) Bandit based Monte Carlo planning. In: Proceedings of the 17th European conference on machine learning, ECML'06. Springer, Berlin, pp 282–293

Kuipers J, Plaat A, Vermaseren J, van den Herik H (2013) Improving multivariate Horner schemes with Monte Carlo Tree Search. Comput Phys Commun 184(11):2391–2395. http://www.sciencedirect.com/science/article/pii/S0010465513001689

Kurzer K, Zhou C, Zöllner JM (2018) Decentralized cooperative planning for automated vehicles with hierarchical Monte Carlo Tree Search. In: 2018 IEEE intelligent vehicles symposium (IV). IEEE, pp 529–536

Lanctot M, Winands MH, Pepels T, Sturtevant NR (2014) Monte Carlo Tree Search with heuristic evaluations using implicit minimax backups. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–8

Liang X, Wei T, Wu IC (2015) Job-level UCT search for solving Hex. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 222–229

Lisỳ V, Lanctot M, Bowling M (2015) Online Monte Carlo counterfactual regret minimization for search in imperfect information games. In: Proceedings of the 2015 international conference on autonomous agents and multiagent systems, pp 27–36

Liu YC, Tsuruoka Y (2015) Regulation of exploration for simple regret minimization in Monte Carlo Tree Search. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 35–42

Lizotte DJ, Laber EB (2016) Multi-objective Markov decision processes for data-driven decision support. J Mach Learn Res 17:211:1-211:28

Lorentz R (2016) Using evaluation functions in Monte Carlo Tree Search. Theor Comput Sci 644:106–113

Lowerre BT (1976) The harpy speech recognition system. Technical report. Department of Computer Science, Carnegie-Mellon University, Pittsburgh

Lucas SM, Samothrakis S, Perez D (2014) Fast evolutionary adaptation for Monte Carlo Tree Search. In: European conference on the applications of evolutionary computation. Springer, pp 349–360

Maia LF, Viana W, Trinta F (2017) Using Monte Carlo Tree Search and Google Maps to improve game balancing in location-based games. In: 2017 IEEE conference on computational intelligence and games (CIG). IEEE, pp 215–222

Mandai Y, Kaneko T (2016) LinUCB applied to Monte Carlo Tree Search. Theor Comput Sci 644:114–126

Mańdziuk J (2018) MCTS/UCT in solving real-life problems. In: Advances in data analysis with computational intelligence methods—dedicated to Professor Jacek Żurada. Springer Studies in Computational Intelligence, vol 738, pp 277–292

Mańdziuk J (2019) New shades of the vehicle routing problem: emerging problem formulations and computational intelligence solution methods. IEEE Trans Emerg Top Comput Intell 3(3):230–244

Mańdziuk J, Nejman C (2015) UCT-based approach to capacitated vehicle routing problem. In: International conference on artificial intelligence and soft computing. Springer, pp 679–690

Mańdziuk J, Świechowski M (2016) Simulation-based approach to Vehicle Routing Problem with traffic jams. In: 2016 IEEE symposium series on computational intelligence, SSCI 2016, Athens, Greece. IEEE, pp 1–8

Mańdziuk J, Świechowski M (2017) UCT in capacitated vehicle routing problem with traffic jams. Inf Sci 406:42–56

Mirsoleimani SA, Plaat A, Van Den Herik J, Vermaseren J (2015) Scaling Monte Carlo Tree Search on Intel Xeon Phi. In: 2015 IEEE 21st international conference on parallel and distributed systems (ICPADS). IEEE, pp 666–673

Mirsoleimani SA, van den Herik HJ, Plaat A, Vermaseren J (2018) A lock-free algorithm for parallel MCTS. In: ICAART—10th international conference on agents and artificial intelligence, pp 589–598

Moraes RO, Marino JR, Lelis LH, Nascimento MA (2018) Action abstractions for combinatorial multi-armed bandit tree search. In: AIIDE, pp 74–80

Nelson MJ (2016) Investigating vanilla MCTS scaling on the GVG-AI game corpus. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–7

Neto T, Constantino M, Martins I, Pedroso JP (2020) A multi-objective Monte Carlo Tree Search for forest harvest scheduling. Eur J Oper Res 282(3):1115–1126

Nguyen KQ, Thawonmas R (2012) Monte Carlo Tree Search for collaboration control of ghosts in MS Pac-Man. IEEE Trans Comput Intell AI Games 5(1):57–68

Ontanón S (2016) Informed Monte Carlo Tree Search for real-time strategy games. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Ontañón S, Synnaeve G, Uriarte A, Richoux F, Churchill D, Preuss M (2013) A survey of real-time strategy game AI research and competition in Starcraft. IEEE Trans Comput Intell AI Games 5(4):293–311

Painter M, Lacerda B, Hawes N (2020) Convex hull Monte Carlo Tree-Search. In: Proceedings of the international conference on automated planning and scheduling, vol 30, pp 217–225

Park H, Kim KJ (2015) MCTS with influence map for General Video Game Playing. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 534–535

Paruchuri P, Pearce JP, Marecki J, Tambe M, Ordonez F, Kraus S (2008) Playing games for security: an efficient exact algorithm for solving Bayesian Stackelberg games. In: Proceedings of the 7th international joint conference on autonomous agents and multiagent systems, vol 2, pp 895–902

Patra S, Mason J, Kumar A, Ghallab M, Traverso P, Nau D (2020) Integrating acting, planning, and learning in hierarchical operational models. In: Proceedings of the international conference on automated planning and scheduling, vol 30, pp 478–487

Pepels T, Winands MH (2012) Enhancements for Monte Carlo Tree Search in MS Pac-Man. In: 2012 IEEE conference on computational intelligence and games (CIG). IEEE, pp 265–272

Pepels T, Winands MH, Lanctot M (2014) Real-time Monte Carlo Tree Search in MS Pac-Man. IEEE Trans Comput Intell AI games 6(3):245–257

Perez D, Samothrakis S, Lucas S, Rohlfshagen P (2013) Rolling horizon evolution versus tree search for navigation in single-player real-time games. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation, pp 351–358

Perez D, Samothrakis S, Lucas S (2014) Knowledge-based fast evolutionary MCTS for general video game playing. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–8

Perez-Liebana D, Samothrakis S, Togelius J, Schaul T, Lucas SM, Couëtoux A, Lee J, Lim CU, Thompson T (2016) The 2014 General Video Game Playing competition. IEEE Trans Comput Intell AI Games 8(3):229–243

Perick P, St-Pierre DL, Maes F, Ernst D (2012) Comparison of different selection strategies in Monte Carlo Tree Search for the Game of Tron. In: 2012 IEEE conference on computational intelligence and games (CIG). IEEE, pp 242–249

Pettit J, Helmbold D (2012) Evolutionary learning of policies for MCTS simulations. In: Proceedings of the international conference on the foundations of digital games, pp 212–219

Pinto IP, Coutinho LR (2018) Hierarchical reinforcement learning with Monte Carlo Tree Search in computer fighting game. IEEE Trans Games 11(3):290–295

Plaat A (2014) MTD(F), a minimax algorithm faster than NegaScout. arXiv preprint arXiv:1404.1511

Powley EJ, Whitehouse D, Cowling PI (2013a) Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8

Powley EJ, Whitehouse D, Cowling PI (2013b) Monte Carlo Tree Search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8

Preuss M, Gunter R (2015) Conference Report on IEEE CIG 2014. IEEE Comput Intell Mag 10(1):14–15

Preuss M, Risi S (2020) A games industry perspective on recent game AI developments. Künstl Intell 34(1):81–83

Rabin S (2013) Game AI Pro: collected wisdom of game AI professionals. CRC Press, Boca Raton

Robles D, Rohlfshagen P, Lucas SM (2011) Learning non-random moves for playing Othello: improving Monte Carlo Tree Search. In: 2011 IEEE conference on computational intelligence and games (CIG'11). IEEE, pp 305–312

Sabar NR, Kendall G (2015) Population based Monte Carlo Tree Search hyper-heuristic for combinatorial optimization problems. Inf Sci 314:225–239

Santos A, Santos PA, Melo FS (2017) Monte Carlo Tree Search experiments in Hearthstone. In: 2017 IEEE conference on computational intelligence and games (CIG), pp 272–279

Sarratt T, Pynadath DV, Jhala A (2014) Converging to a player model in Monte Carlo Tree Search. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–7

Schaefers L, Platzner M (2014) Distributed Monte Carlo Tree Search: a novel technique and its application to computer Go. IEEE Trans Comput Intell AI Games 7(4):361–374

Schaeffer J (1989) The history heuristic and alpha–beta search enhancements in practice. IEEE Trans Pattern Anal Mach Intell 11(11):1203–1212

Schaeffer J, Burch N, Björnsson Y, Kishimoto A, Müller M, Lake R, Lu P, Sutphen S (2007) Checkers is solved. Science 317(5844):1518–1522. https://science.sciencemag.org/content/317/5844/1518

Segler MH, Preuss M, Waller MP (2018) Planning chemical syntheses with deep neural networks and symbolic AI. Nature 555(7698):604–610

Sephton N, Cowling PI, Powley E, Slaven NH (2014) Heuristic move pruning in Monte Carlo Tree Search for the strategic card game Lords of War. In: 2014 IEEE conference on computational intelligence and games, pp 1–7

Sephton N, Cowling PI, Powley E, Whitehouse D, Slaven NH (2014) Parallelization of information set Monte Carlo Tree Search. In: 2014 IEEE congress on evolutionary computation (CEC). IEEE, pp 2290–2297

Sephton N, Cowling PI, Slaven NH (2015) An experimental study of action selection mechanisms to create an entertaining opponent. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 122–129

Sharma A, Harrison J, Tsao M, Pavone M (2019) Robust and adaptive planning under model uncertainty. In: Proceedings of the international conference on automated planning and scheduling, vol 29, pp 410–418

Shi F, Soman RK, Han J, Whyte JK (2020) Addressing adjacency constraints in rectangular floor plans using Monte Carlo Tree Search. Autom Constr 115:103187

Silver D, Sutton RS, Müller M (2012) Temporal-difference search in computer Go. Mach Learn 87(2):183–219. https://doi.org/10.1007/s10994-012-5280-0

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

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

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. https://science.sciencemag.org/content/362/6419/1140

Sironi CF, Winands MH (2016) Comparison of rapid action value estimation variants for General Game Playing. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Sironi CF, Winands MH (2019) Comparing randomization strategies for search-control parameters in Monte Carlo Tree Search. In: 2019 IEEE conference on games (CoG). IEEE, pp 1–8

Sironi CF, Liu J, Winands MH (2018) Self-adaptive Monte Carlo Tree Search in General Game Playing. IEEE Trans Games 12(2):132–144

Soemers DJ, Sironi CF, Schuster T, Winands MH (2016) Enhancements for real-time Monte Carlo Tree Search in General Video Game Playing. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8

Soemers DJ, Piette E, Stephenson M, Browne C (2019) Learning policies from self-play with policy gradients and MCTS value estimates. In: 2019 IEEE conference on games (CoG). IEEE, pp 1–8

Subramanian K, Scholz J, Isbell CL, Thomaz AL (2016) Efficient exploration in Monte Carlo Tree Search using human action abstractions. In: Proceedings of the 30th international conference on neural information processing systems, NIPS, vol 16

Świechowski M (2020) Game AI competitions: motivation for the imitation game-playing competition. In: 2020 Federated conference on computer science and information systems (FedCSIS), vol 21. IEEE, pp 155–160

Świechowski M, Mańdziuk J (2014) Self-adaptation of playing strategies in general game playing. IEEE Trans Comput Intell AI Games 6(4):367–381

Świechowski M, Mańdziuk J (2016a) A hybrid approach to parallelization of Monte Carlo Tree Search in general game playing. In: Challenging problems and solutions in intelligent systems. Springer, pp 199–215

Świechowski M, Mańdziuk J (2016) Fast interpreter for logical reasoning in general game playing. J Logic Comput 26(5):1697–1727

Swiechowski M, Ślęzak D (2018) Granular games in real-time environment. In: 2018 IEEE international conference on data mining workshops (ICDMW). IEEE, pp 462–469

Świechowski M, Kasmarik K, Mańdziuk J, Abbass H (2015) Human–machine cooperation loop in game playing. Int J Intell Syst 8:310–323

Świechowski M, Park HS, Mańdziuk J, Kim KJ (2015) Recent advances in general game playing. Sci World J 2015:986262

Świechowski M, Mańdziuk J, Ong Y (2016) Specialization of a UCT-based general game playing program to single-player games. IEEE Trans Comput Intell AI Games 8(3):218–228

Świechowski M, Tajmajer T, Janusz A (2018) Improving Hearthstone AI by combining MCTS and supervised learning algorithms. In: 2018 IEEE conference on computational intelligence and games (CIG), pp 1–8

Syed O, Syed A (2003) ARIMAA–a new game designed to be difficult for computers. ICGA J 26(2):138–139

Szita I, Chaslot G, Spronck P (2009) Monte Carlo Tree Search in settlers of Catan. In: Advances in computer games, pp 21–32

Tak MJ, Winands MH, Bjornsson Y (2012) N-grams and the last-good-reply policy applied in General Game Playing. IEEE Trans Comput Intell AI Games 4(2):73–83

Tak MJ, Lanctot M, Winands MH (2014) Monte Carlo Tree Search variants for simultaneous move games. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–8

Tak MJW, Winands MHM, Bjornsson Y (2012) N-grams and the last-good-reply policy applied in general game playing. IEEE Trans Comput Intell AI Games 4(2):73–83

Takada K, Iizuka H, Yamamoto M (2019) Reinforcement learning to create value and policy functions using minimax tree search in Hex. IEEE Trans Games 12(1):63–73

Tang Z, Zhao D, Shao K, Lv L (2016) ADP with MCTS algorithm for Gomoku. In: 2016 IEEE symposium series on computational intelligence (SSCI). IEEE, pp 1–7

Tesauro G (1994) TD-Gammon, a self-teaching Backgammon program, achieves master-level play. Neural Comput 6(2):215–219

Teytaud F, Teytaud O (2010) Creating an upper-confidence-tree program for Havannah. In: Advances in computer games. Springer, pp 65–74

Thielscher M (2010) A general game description language for incomplete information games. In: Proceedings of the AAAI conference on artificial intelligence, vol 24

Thompson WR (1933) On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika 25(3/4):285–294

Trutman M, Schiffel S (2015) Creating action heuristics for General Game Playing agents. In: Computer games. Springer, pp 149–164

Tversky A, Kahneman D (1974) Judgment under uncertainty: heuristics and biases. Science 185(4157):1124–1131

Uriarte A, Ontanón S (2014) High-level representations for game-tree search in RTS games. In: Tenth artificial intelligence and interactive digital entertainment conference

Uriarte A, Ontanón S (2016) Improving Monte Carlo Tree Search policies in Starcraft via probabilistic models learned from replay data. In: Twelfth artificial intelligence and interactive digital entertainment conference

Uriarte A, Ontanón S (2017) Single believe state generation for partially observable Real-Time Strategy Games. In: 2017 IEEE conference on computational intelligence and games (CIG), pp 296–303

Vallati M, Chrpa L, Grześ M, McCluskey TL, Roberts M, Sanner S et al (2015) The 2014 International Planning Competition: progress and trends. AI Mag 36(3):90–98

Van den Broeck G, Driessens K, Ramon J (2009) Monte Carlo Tree Search in poker using expected reward distributions. In: Asian conference on machine learning. Springer, pp 367–381

Vien NA, Toussaint M (2015) Hierarchical Monte Carlo planning. In: AAAI, pp 3613–3619

Vodopivec T, Šter B (2014) Enhancing upper confidence bounds for trees with temporal difference values. In: 2014 IEEE conference on computational intelligence and games. IEEE, pp 1–8

Waledzik K, Mańdziuk J (2011) Multigame playing by means of UCT enhanced with automatically generated evaluation functions. In: Artificial general intelligence—4th international conference, AGI 2011, lecture notes in computer science, Mountain View, CA, USA, vol 6830. Springer, pp 327–332

Waledzik K, Mańdziuk J (2014) An automatically generated evaluation function in General Game Playing. IEEE Trans Comput Intell AI Games 6(3):258–270

Waledzik K, Mandziuk J (2018) Applying hybrid Monte Carlo Tree Search methods to risk-aware project scheduling problem. Inf Sci 460–461:450–468

Waledzik K, Mańdziuk J, Zadrozny S (2014) Proactive and reactive risk-aware project scheduling. In: 2014 IEEE symposium on computational intelligence for human-like intelligence, CIHLI 2014, Orlando, FL, USA. IEEE, pp 94–101

Waledzik K, Mańdziuk J, Zadrozny S (2015) Risk-aware project scheduling for projects with varied risk levels. In: 2015 IEEE symposium series on computational intelligence. IEEE, pp 1642–1649

Wang FY, Zhang JJ, Zheng X, Wang X, Yuan Y, Dai X, Zhang J, Yang L (2016) Where does AlphaGo go: from Church–Turing thesis to AlphaGo thesis and beyond. IEEE/CAA J Autom Sin 3(2):113–120

Wang J, Zhu T, Li H, Hsueh C, Wu I (2015) Belief-state Monte Carlo Tree Search for phantom games. In: 2015 IEEE conference on computational intelligence and games (CIG), pp 267–274

Wang K, Perrault A, Mate A, Tambe M (2020) Scalable game-focused learning of adversary models: data-to-decisions in network security games. In: Seghrouchni AEF, Sukthankar G, An B, Yorke-Smith N (eds) Proceedings of the 19th international conference on autonomous agents and multiagent systems, AAMAS '20, Auckland, New Zealand, 9–13 May 2020. International Foundation for Autonomous Agents and Multiagent Systems, pp 1449–1457

Wijaya TK, Papaioannou TG, Liu X, Aberer K (2013) Effective consumption scheduling for demand-side management in the smart grid using non-uniform participation rate. In: 2013 Sustainable Internet and ICT for sustainability (SustainIT). IEEE, pp 1–8

Winands MH, Bjornsson Y, Saito JT (2010) Monte Carlo Tree Search in lines of action. IEEE Trans Comput Intell AI Games 2(4):239–250

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Wu F, Ramchurn SD, Jiang W, Fischer JE, Rodden T, Jennings NR (2015) Agile planning for real-world disaster response. In: Proceedings of the 24th international joint conference on artificial intelligence (IJCAI), Buenos Aires, Argentina, pp 132–138

Wu TR, Wu IC, Chen GW, Wei Th, Wu HC, Lai TY, Lan LC (2018) Multilabeled value networks for computer Go. IEEE Trans Games 10(4):378–389

Yang B, Wang L, Lu H, Yang Y (2020) Learning the Game of Go by scalable network without prior knowledge of Komi. IEEE Trans Games 12(2):187–198

Yang Z, Ontañón S (2018) Learning map-independent evaluation functions for Real-Time Strategy Games. In: 2018 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–7

Yee T, Lisỳ V, Bowling MH, Kambhampati S (2016) Monte Carlo Tree Search in continuous action spaces with execution uncertainty. In: IJCAI, pp 690–697

Zhang S, Buro M (2017) Improving Hearthstone AI by learning high-level rollout policies and bucketing chance node events. In: 2017 IEEE conference on computational intelligence and games (CIG), pp 309–316

Zhou H, Gong Y, Mugrai L, Khalifa A, Nealen A, Togelius J (2018) A hybrid search agent in Pommerman. In: Proceedings of the 13th international conference on the foundations of digital games, pp 1–4

Zhuang Y, Li S, Peters TV, Zhang C (2015) Improving Monte Carlo Tree Search for dots-and-boxes with a novel board representation and artificial neural networks. In: 2015 IEEE conference on computational intelligence and games (CIG). IEEE, pp 314–321

Zook A, Harrison B, Riedl MO (2019) Monte Carlo Tree Search for simulation-based strategy analysis. arXiv preprint. arXiv:1908.01423