State and Action Abstraction for Search and Reinforcement Learning Algorithms

Alexander Dockhorn[0000-0001-8711-7428] and Rudolf Kruse[0000-0003-4981-2758]

Gottfried Wilhelm Leibniz University, 30167 Hannover, Germany
dockhorn@tnt.uni-hannover.de
Otto von Guericke University, 39106 Magdeburg, Germany
rudolf.kruse@ovgu.de

Abstract. Decision-making in large and dynamic environments has always been a challenge for AI agents. Given the multitude of available sensors in robotics and the rising complexity of simulated environments, agents have access to plenty of data but need to carefully focus their attention if they want to be successful. While action abstractions reduce the complexity by concentrating on a feasible subset of actions, state abstractions enable the agent to better transfer its knowledge from similar situations. In this article, we want to identify the different techniques for learning and using state and action abstractions and compare their effect on an agent's training and its resulting behavior.

Keywords: Action Abstraction, State Abstraction, Reinforcement Learning, Search-based Algorithms, Computational Intelligence in Games.

1 Introduction

Decision-making plays an integral part in artificial intelligence (AI) agents. Being confronted with a decision-making problem, possible decisions must be identified first. After that, an agent needs to construct possible action plans and gather information on them. As a result, the agent can weigh the evidence to choose among the available options. After having executed the selected actions, the agent will be able to review its decision based on the observed consequences.

In a reinforcement learning context, we are confronted with a sequential decision-making problem. Here, the agent stays in interaction with its environment and continuously selects actions to maximize its reward over time. Over the course of multiple interactions, the agents gather more information and may revise their previous decisions in later time steps. Furthermore, the learning agent might have to face a large variety of situations and have many actions to choose from for handling any given situation. Thus, effectively re-using the gathered evidence from previous time-steps is key to optimizing the agent's performance over time.

Both, search and reinforcement learning algorithms, can struggle to perform well at training or run-time in either of the two settings. A popular method for improving the agent's performance is to implement abstractions [1] of the problem at hand. We

naturally do this, when designing AI agents with a certain task in mind, by providing it with the information sources and actions we deem to be relevant, thereby restricting the agent to the envisioned setting. Thereby, abstractions help the agent to focus its attention and narrow down the number of relevant alternatives to choose from.

Learning abstractions from experience becomes especially relevant when aiming for more general AI agents [2]. Many AI agents are presented with exactly the information they need to solve the given task. However, detecting the relevant information on your own and abstracting relevant concepts is a non-trivial task but required for becoming proficient in solving more general problems [3], [4].

This survey shall provide an overview of common abstraction methods for state and action spaces of Markov Decision Processes (MDP) [5]. Both, reinforcement learning and search-based algorithms, can make use of such abstractions, to speed up the decision process and thereby often improve the agent's performance. In Section 2 we will summarize preliminaries on MDPs, search, and reinforcement learning algorithms. Section 3 covers methods for state abstraction, whereas Section 4 will focus on action abstractions. Hybrids of both methods will be covered in Section 5. The work will be concluded in Section 6 followed by propositions for future research and applications.

2 Preliminaries

The following sections will summarize preliminaries on MDPs (Section 2.1), reinforcement learning (Section 2.2), and search algorithms (Section 2.3). On this basis, we will summarize the research on state and action abstraction in later Sections.

2.1 Markov Decision Processes

To discuss abstractions in more detail, we first review Markov Decision Processes [5] as a general formalization of a reinforcement learning problem. An MDP is described by a tuple:

$$M = (S, A, T, R) \tag{1}$$

where S is the state space, A the action set, T is the state transition function, and R is the reward function. The state space S consists of all the environment states s that the agent can observe. Action set A consists of all actions a the agent can use to interact with its environment. The state transition function $T(s, a, s')$ represents a probability function, mapping the current state and the agent's action to the probability of the next state s'. Finally, the reward function $R(s, a, s')$ provides the agent with a numerical reward when transitioning from state s to s' via action a.

In the context of an MDP, the agent is in constant interaction with its environment. Every time-step t, the agent observes the state of the environment and chooses an action to execute. As a reaction, the environment will change its state and provide the agent with a reward. The agent's goal is to choose actions such that it maximizes the received reward over time.

In terms of the applications, we will discuss in this work, we want to highlight combinatorial state and action spaces. Combinatorial states are made of multiple

sensory inputs $s = (s_1, ..., s_n)$. Sensors may be part of the same modality (e.g., pixels of a camera-image, or receive data from multiple modalities (e.g., audio, video, tactile sensors, etc.). Making efficient use of multiple sensors and multiple modalities may require extensive pre-processing [6] or fusion algorithms [7]. A similar principle applies to combinatorial action spaces in which each action consists of multiple action components $a = (a_1, ..., a_m)$. Those can be commonly observed in robotics and multi-unit games in which multiple independent entities need to be controlled during every tick of the environment.

2.2 **Reinforcement Learning**

The theory of reinforcement learning algorithms finds its origin in psychology. Conditioning, a form of learning stimulus-response associations, describes the coupling of a neutral stimulus with an unconditional stimulus. Here, the neutral stimulus becomes a conditioned stimulus and triggers a comparable response in the subject (cf. [8]). Reinforcement learning algorithms, adopt this principle to train an agent in arbitrary tasks through continuous feedback. Using a multi-armed bandit, the associative version of the learning problem can be described [9]. Here, the agent learns the average value when using an action without changing the state of the underlying system. If the state is mutable, modeling by Markov decision processes is resorted to [10]. In this, the agent also receives feedback in the form of a numerical reward while at the same time changing the state of its environment.

Here, the learning process is fundamentally different from supervised learning. While the agent is provided with a data set of stimulus-response observations, in reinforcement learning the agent independently explores its environment. Hereby, a balance between the exploration of unknown actions and the exploitation of promising actions must be found continuously. Learning procedures of reinforcement learning can thus be divided into on- and off-policy procedures. While on-policy methods always select the currently best-known action, off-policy methods allow the exploration of actions that have not yet been considered optimal.

Classical methods, also known as tabular methods, such as Temporal Difference Learning [11] or the Monte Carlo Method [12] determine the expected reward of each action or state-action pair based on repeated interaction with the system under observation. If the underlying system is fully known, these values can be determined by Dynamic Programming [13]. In these classical methods, decisions are made based on the learned expected reward. Here, this value must be learned for each action (non-associative), or each state-action pair (associative). This property makes classical learning methods unusable, especially for large state and action spaces.

The use of neural networks in the context of deep reinforcement learning [14], has brought fundamental changes for tackling the learning problem. While before, the observation of the agent and thus the possible state space had to be designed by hand, the use of neural networks allows the internal reshaping of unstructured state observations. Here, training the neural network creates an approximation of the reward signal or an action probability to maximize the reward signal. In contrast to tabular methods, the network-internal transformation of the input signal can also approximate the reward of previously unobserved states. While often done implicitly,

later sections will show how such a network behavior can be actively approached to learn state and action abstractions.

2.3 **Search/Optimization-based Decision-Making**

In contrast to reinforcement learning, search and optimization-based decision-making tries to determine the best action at run-time. Therefore, the agent usually relies on a forward model to determine the outcome of planned actions and optimize an action sequence. In cases where the forward model cannot be accessed, it can be approximated from observation [15–17].

Exhaustive search methods try to simulate all possible action sequences in a structured manner. Classic tree-search algorithms build a tree starting from the current state as the root node. Each simulated action is represented as an edge, connecting the state in which the action has been initialized with the resulting next state. Methods such as breadth-first search and depth-first search [18] do so without any optimization. More efficient algorithms, such as the minimax algorithm [19] and alpha-beta pruning [1] skip infeasible subtrees and have been successfully applied to games with large state spaces (e.g., chess [20]). Nevertheless, they have shown to be infeasible for games such as Go which features a state of approximately 10^{170} states [21].

Due to the large branching factor and the sheer number of possible states, exhaustive search often takes too much time to compute a result. Instead, heuristic search methods can be used to approximate the optimal action given a sample of the gam tree. Methods such as flat Monte Carlo [22] and Monte Carlo Tree Search (MCTS) [23] achieve this by simulating the outcome of multiple action sequences. To focus the search, these methods balance the exploration and exploitation of available options [22], whereas during exploration we aim to find good actions that have not been analyzed yet and during exploitation, we further analyze parts of the game tree that has shown good results so far.

In contrast to tree-search methods, optimization-based methods try to directly optimize an action sequence. Algorithms such as the Rolling Horizon Evolutionary Algorithm (RHEA) [24, 25] do so, by using an evolutionary algorithm to improve multiple candidate action sequences throughout several generations. Once the algorithm terminates, the first action of the best sequence will be applied.

In contrast to reinforcement learning, search- and optimization-based methods can be used without any prior training since necessary evaluations are done at run-time. This made them especially relevant in more general domains, such as general game-playing [26, 27] and general strategy game-playing [28, 29], in which the agent needs to act in a previously unknown environment. Nevertheless, they struggle with optimizing long action sequences and high branching factors. For this reason, they are commonly applied in environments with simple state and action spaces. The performance for more complex scenarios can be improved by methods of state and action abstraction, which will be detailed in the following sections.

3 State Abstractions

In a state abstraction, we intend to reduce the complexity of the decision-making task, by mapping from the history of previously visited states H to a state in the compact state space S' [30]. In early papers, the abstraction has solely been based on a mapping of the current state $s \in S$ to another state in the compact state space $s' \in S'$, whereas $S' \ll S$. Those methods have been designed for MDPs of the first degree, which makes the mapping independent of previously visited states. In either way, the decision space become more compact, and the agent's effort is reduced. For simplicity, we will constrain the following discussion on MDPs of the first degree as well. The following categories of state abstractions have been proposed by Konidaris [2] and served as a guideline to structure this review. We extend his work with an updated summary of the state-of-the-art and an overview of state abstraction in highly structured state spaces as they can be commonly observed in path planning. An overview of discussed methods is presented in Figure 1, whereas each type of method will be discussed in the following sections, respectively.

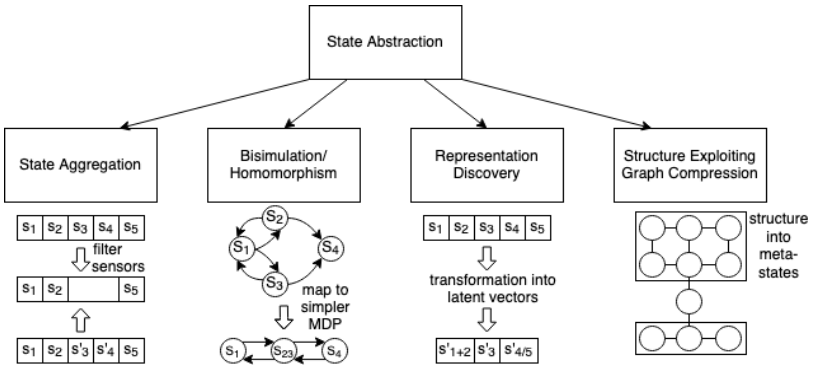

Fig. 1. Overview of the Types of State Abstraction Algorithms.

3.1 Bisimulation Approaches

Bisimulation approaches aim to construct a minimized MDP with similar properties as the original one. Such model minimization approaches are often variations of automaton minimization algorithms. The state space abstraction by Boutilier and Dearden [31] solves a factored MDP by constructing a minimized version based on the impact of propositions on the utility. Due to the existence of multiple factorizations per MDP, the proposed approach is non-deterministic and can result in abstractions of varying quality and size. In any case, the minimization is adequate and computable in polynomial time. The approach by Dean and Givan [32] also uses a factored representation to find the coarsest homogenous refinement of any partition of the MDP's state space. The resulting reduced MDP is minimal in a well-defined sense and can be used to find an optimal policy in the original MDP. Sadly, this process is NP-hard. Simplifications of this process can run in polynomial time but

cannot produce an optimal minimization. In contrast to exact abstractions, an approximate abstraction can often be achieved in polynomial time [33, 34]. Thereby, algorithms balance the coarseness of the abstraction with the quality of the obtained solution from the abstract MDP.

The high complexity of bisimulation approaches often makes them infeasible for very large state spaces. Approximate solutions can reduce the computational complexity but may cause the agent to learn an invalid policy. Often the approximation of the MDP can be iteratively improved at the cost of additional computations. The problem is typically studied in the context of model-based reinforcement learning [35], in which algorithms aim for a good balance between building an internal model and improving their policy. Another solution to the problem is the search for local abstractions of the MDP. In contrast, to the approaches above, local abstractions only model a subgraph of the original MDP. Due to the reduced size of the abstracted graph, such abstractions can be found much faster. Jiang et al. have used this approach in the context of a UCT (MCTS using upper confidence bounds as a selection criterion) to efficiently use the data generated from previously simulated trajectories. Given a small budget of simulations, the local abstractions have been shown to improve the performance of UCT.

3.2 State Aggregation Approaches

In homomorphic state abstraction, we construct a more compact state space S' and a mapping from S to S'. The simplest way to do so is by discarding variables from our state observation [36]. Thereby, we reduce the dimensionality of the state space and often considerably shrink its size. Identifying suitable variables to discard can either be done in communication with domain experts or as a result of feature analysis [37–39]. For example, the approach by Jong and Stone [37] measures the relevance of a variable in terms of its impact on the optimal policy. In case a variable does not significantly change the decisions suggested by the policy, we can safely remove it from our representation. Interesting about this method is that it preserves convergence to the optimal policy, which cannot be ensured in expert-driven abstractions. In contrast to filtering approaches, feature selection can also be done in a bottom-up manner, i.e., starting with no features and adding the ones that improve the model quality the most [40].

Instead of ignoring/selecting state variables to project a state into a subspace, we can also group states into clusters [41–43]. Given the agent's previous trajectories, we can try to identify states with a similar policy, transition, or reward function to create a more compact clustered representation. The aggregation of states into a soft/fuzzy partition can result in more flexibility and model the agent's confidence in the current clustering [43]. This approach will become relevant once again in Section 4.3.

Whereas previous methods have mostly focused on reinforcement learning, state aggregation can also be found in search-based algorithms. Most commonly, stochastic search algorithms aim to group simulations based on their trajectories to evaluate visited states more efficiently [44, 45]. Used approaches are similar to the ones presented above but the feature selection is commonly applied in between iterations of the search, may change over time, and therefore adapt to the current local context of the agent's state space.

3.3 Representation Discovery Approaches

While the previous section discussed methods for reducing the complexity of an existing state space into a representative subspace, representation discovery approaches are used to intrinsically learn a state representation based on the original input. Also homomorphic, representation discovery methods often construct abstracted state spaces that bear not much resemblance but share properties of the original state space. In recent work, deep neural networks, specifically, autoencoders [46], have been used to process raw sensory data. Thereby, the network is learning a compact feature vector in its hidden layers that are subsequently used for decision-making. While being hard to interpret for the human observer in the general domain, analysis of convolutional neural networks for image classification has shown that hidden layers represent increasingly complex features of learned image classes [47]. Resulting latent spaces can often be used as highly compact state abstractions.

A special form of convolution is implemented by graph convolutional networks [48]. Those directly operate on graph structures and can therefore be used to create latent representations of MDPs. A study by Jiang et al. [49] has shown their efficient use in reinforcement learning. Finally, a work by Liu et al. [50] has used meta descriptive statistics (MDS) as supplementary state representations. They compared the MDS approach with abstractions obtained from autoencoders and graph convolutional networks as well as combinations of all three methods. In their evaluation, graph convolutional networks resulted in the best performance among the single methods, whereas a combination of MDS and autoencoders performed best overall.

3.4 Graph Compression Using High-Level Knowledge

An interesting application and challenging research domain of state abstraction algorithms is pathfinding. Here, the agent is tasked to explore an environment and find a (near-)optimal path to a given objective. Thereby, the agent either uses a model of its surroundings or needs to create one while exploring the environment. In either case, the time for planning shall be kept as low as possible while also minimizing the length of the resulting path. Sadly, optimizing for one often worsens the performance of the other since abstractions can reduce the planning time, but in many cases only result in near-optimal solutions [51].

Generally, heuristics can be used to guide the agent's search process [1]. In case the same environment is used for multiple pathfinding queries, it might be useful to preprocess the graph model to solve the queries in a more efficient abstract model of the environment. Many algorithms do so by exploiting the highly regular structure of grid-based path-finding problems. In contrast to arbitrary MDPs, the agent's actions and their meaning remain the same for every state. This allows the usage of simple abstraction heuristics, such as the one used in HPA* [52]. Here, the graph is first decomposed into a map of disjoint square sectors. To plan a path from the current position to the target, we first plan a path from one square sector to another and later refine the path in the original grid map. A multi-layered abstraction and refinement process has been implemented in Partial Refinement A* (PRA*) [53]. Sturtevant later proposed a combination of HPA* and PRA* to decrease the agent's memory

consumption [54]. The efficiency of real-time heuristic search has been further improved by interweaving it with automatic state abstraction. The work by Bulitko et al. [55] has shown huge performance gains over non-abstracting alternatives.

4 Action Abstractions

In terms of large action spaces, algorithms for decision-making struggle with identifying the best actions during the exploration of a vast number of alternatives. Like large state spaces, large action spaces lengthen the training time of reinforcement learning algorithms and the time to explore the game tree in search-based algorithms. The idea of action abstraction is to either narrow the search on a subset of promising candidate actions (Section 4.1) or identify valuable action sequences to be used (Section 4.2). Each of the two approaches (cf. Figure 2) will be discussed in the following sections, respectively.

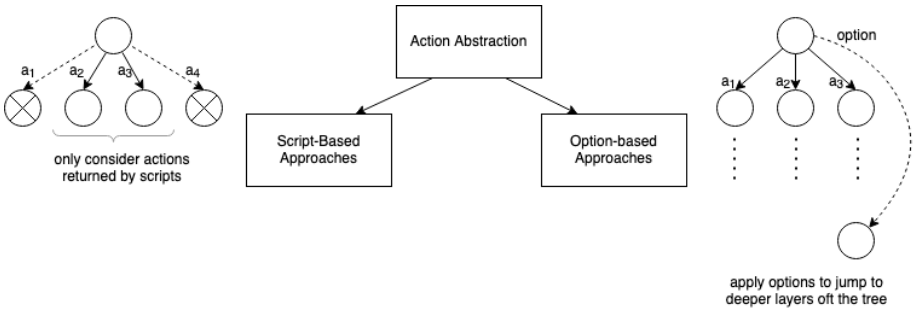

Fig. 2. Overview of the Types of Action Abstraction Algorithms.

4.1 Script-based Action Abstraction

Script-based action abstraction reduces the size of the action space by removing inferior actions or focusing the search on a selection of good candidates. While this approach loses granularity and is dependent on the accuracy of the candidate set, it can drastically speed up the agent's search process. A simple concept to retrieve candidate actions is the use of scripts. A script is a subroutine that returns a candidate action for any current state of the environment.

$$Script\colon S \rightarrow A \tag{2}$$

Thereby, a single script is similar to a deterministic policy. In the context of strategy game AI, scripts have been implemented by a set of rules that focus on a certain strategy, e.g., attacking the closest opponent in the range [56]. Such games often involve multiple entities that can be independently controlled by the agent by selecting an entity's action for each of them. An increasing number of such entities (e.g., units, buildings, etc.) results in an exponentially growing combinatorial action space [57, 58]. Due to the independence of multiple entities, scripts are often constrained to a single entity and its respective actions in the context of multi-unit strategy games. Similar implementations could be thought of for robots for which the

agent controls multiple joints at the same time. In both cases, the output of a script can be restricted to a single component under the agent's control, and therefore, can also be defined by:

$$Entity\ Script\colon\ S \times Entity \rightarrow A_{Entity} \tag{3}$$

The agent's response is then determined by assigning one script to each entity and building the combinatorial action out of the responses by all the scripts.

4.2 Portfolio-based Search Algorithms

While the use of a single script with high performance can already yield a successful agent, the combination of multiple scripts has been shown to improve the overall performance of the agent [29, 56]. Let a portfolio be given by a set of scripts, whereas the abstracted action space is the union of all script responses for the current state of the environment. Portfolio-based algorithms vary in the way they encode and optimize the script selection/assignment. For instance, Portfolio Greedy Search (PGS) [59] has been designed for combinatorial action spaces and optimizes the script assignment to each independent entity. For this purpose, a hill-climbing procedure is used to search for the best combination. In presence of an opponent, the opponent's and the player's script choices are updated iteratively. Moraes et al. [60] have shown that the devised hill-climbing procedure can suffer from non-convergence. To overcome this issue, they proposed a nested greedy search (NGS) in which the best opponent's response is computed in each iteration. Nevertheless, this comes at the drawback of being unable to evaluate many actions due to the increased time complexity.

Other search and optimization algorithms have been used for improving the agent's performance. Portfolio Online Evolution (POE) [61] replaces the hill-climbing procedure with an evolutionary algorithm [62]. Thereby, a candidate solution encodes the script assignment for each controlled entity. Its fitness is determined by simulating the outcome of continuously retrieving actions from a given script assignment. Further on, mutation of a single candidate solution (e.g., randomly replacing a script assignment) and crossover on multiple of these candidate solutions (e.g., uniform crossover of script assignment) is used to search for candidate solutions of higher fitness. In a similar manner to POE, the Portfolio Rolling Horizon Evolutionary Algorithm (PRHEA) [56] uses evolution to optimize script assignments of fixed length (so-called horizon). Each turn, the agent applies the first script of the best candidate solution before reusing the population to initialize the search for the next iteration. While this limits the search depth of the agent, it has shown efficiency in optimizing script assignments in multiple strategy games.

Another layer of abstraction is introduced by Stratified Strategy Selection [63] and Cluster-based UCT [64], which first groups the entities into types or clusters and then assigns one script to each type. Given contextual knowledge of the environment, such layered abstractions can improve the efficiency of the agent's search. Some environments may not allow the enforcement of a type system on all entities or define a suitable abstraction for some of the entities under the agent's control. In these cases, an asymmetric action abstraction allows controlling entities at different layers of

abstraction. While some may be safely abstracted or even grouped into types, others will be restricted to using their original action space. Moraes and Lelis [65] have proposed the methods Greedy Alpha-Beta Search and Stratified Alpha-Beta Search to search in such asymmetric action abstractions. Additionally, they have shown that the optimal strategy derived using an asymmetric action abstraction is at least as good as using a uniform abstraction.

Another view on the search for combinatorial actions is offered by Naïve MCTS [57], which solves it by a combinatorial multi-armed bandit. Hereby, a combination of MCTS and naïve sampling, which considers each entity's action as an independent contribution to the combinatorial action's reward, is used to optimize the actions for each entity. While the naïve assumption might converge to a local optimum, the guided naïve MCTS proposed by Yang and Ontanon [66] only uses the scripts as recommenders during exploration and keeps the original action space in the tree.

Since previously discussed algorithms mostly differ in the way they structure and sample the script space, a unification of PGS, NGS, POE, SSS, and Naïve MCTS has been proposed by Lelis [67]. General Combinatorial Search for Exponential Action Spaces (GEX) generalizes these algorithms by splitting the search process into subprocesses consisting of select and expand, macro-arm sampling, evaluation, and value propagation. Thereby, more instances of portfolio-based search algorithms can be easily generated by choosing implementations of each subprocess.

The usage of deterministic scripts or their portfolios can result in repetitive behavior. To avoid this, the algorithm puppet search [68], [69] introduced scripts with choice points, which query the state's properties to select an action. A search or optimization process can further be used to modify the thresholds used in these choice points to modify the agent's behavior. This can result in a more reactive game-play experience while adjusting the number of choice points can be an efficient way in tuning the strength of the agent. Other strategies for diverse game-play include the optimization of the portfolio composition [56] as well as the adjustment of search parameters [70]. The simplicity of said approaches makes them approachable to game designers who want to achieve more variety in the agent's gameplay.

4.3 **Constructing Higher Level Actions**

Another type of action abstraction can be found in the options framework [71]. Options are a temporal action abstraction of an MDP and represent closed-loop policies for taking actions over a period of time. They can be considered high-level actions that involve a whole sequence of actions. Each option o is given by a tuple

$$o = (I_o, \pi_o, \beta_o) \tag{4}$$

where $I_o \subseteq S$ is the initiation set that describes potential starting states in which the option can be applied, $\pi_o : S \rightarrow A$ is the option policy used for retrieving actions while the option is active, and $\beta_o : S \rightarrow [0, 1]$ is the terminal condition which is returning 1 in case the option has to be stopped after reaching a state s.

During search or execution, agents can choose to apply available options as they would apply an action. In contrast to script-based approaches, options increase the

branching factor of the search tree by adding additional choices. At the same time, they represent a shortcut to deeper layers of the tree, since if an option is once chosen will remain to be active. Nevertheless, it is important to only add options if they have the potential to reduce the agent's training or search time. Otherwise, poorly chosen options have been shown to slow down the learning or search process [72].

A simple option design that has successfully been used in search-based algorithms are macro actions [73], e.g., repeating a chosen action for multiple time-steps [74], [75]. Similarly, more intricate sub-routines or repeating sub-policies can be designed or extracted from successful runs/play-traces [76]. Alternatively, options can be learned e.g., by subgoal identification, whereas a subgoal decomposes the learning problem into two smaller problems [72]. Ways for finding subgoals include the identification of high reward, or high novelty states [77][78], bottlenecks in the state space, or by taking other graph connectivity measures into account [79–81].

Continuous domains, in which the agent is unlikely to be in the same state multiple times, pose special challenges for learning options. Due to the missing repetition of the task and a single never-ending episode, the aim is to learn skills that can be used later. Given a formal problem description, this can be done using problem decomposition [82, 83], in which a task is simplified into multiple sub-tasks.

In case this is not possible, option learning needs to generalize the initial and target states of an option to a set of states which are similar to each other [84]. The similarity of states is exploited in their skill chaining algorithm. Once the goal is reached, the agent starts with searching for an option that starts in the neighborhood of the target state and therefore has a high likelihood of reaching said state. Next, we search for an option that has a high likelihood to reach a state of the previous option's initiation set. The process is repeated and thereby multiple options are chained into a high-level skill. Such skills are especially interesting for lifelong reinforcement learning tasks, in which the agent faces a set of related tasks in environments that are similar to each other [85]. Here, it is even more important to identify options that transfer well between the different tasks and do not bloat up the search space. A work by Brunskill and Li [86] shows that transferrable options can be learned and these are ϵ-optimal for all the tasks the agent may encounter.

5 Conclusion and Future Directions

The papers presented in this survey have shown that abstractions can be used to reduce the complexity of many tasks. Both state and action abstractions not just help to reduce the agent's training time, but also improve the agent's performance and can make it more robust to changes in its environment. At the same time, it is important to study the theoretic bounds of these abstractions to ensure near-optimal behavior. While many papers have focused on a formal analysis of the environment or the a priori optimization of the abstraction, more work needs to be spent on identifying and exploiting suitable abstractions at run-time. Especially, highly dynamic environments may require the agent to add or drop abstractions depending on its current situation.

As a result, the agent needs to act under uncertainty about the accuracy and suitability of its learned abstractions. To our surprise, this uncertainty and its impact on the decision-making problem has not been the focus of attention yet. We believe

that fuzzy decision-making techniques [87] would be key to finding a suitable abstraction or forming a decision by consensus over multiple imperfect abstractions [88]. Additionally, fuzzy models may play an integral part in transferring learned abstractions to new environments, effectively allowing to reuse the model for similar MDPs while keeping track of its reliability [89]. Finally, the rising complexity of tackled decision-making problems results in an increased demand for explainable models [90] that either support the user to understand the learned abstraction [91] or the decisions made by the agent [92].

References

1. Russell, J.S., Norvig, P.: Artificial Intelligence: A Modern Approach. (2003). https://doi.org/10.1017/S0269888900007724.
2. Konidaris, G.: On the necessity of abstraction. Current Opinion in Behavioral Sciences. 29, 1–7 (2019). https://doi.org/10.1016/j.cobeha.2018.11.005.
3. Yang, X., Zhang, G., Lu, J., Ma, J.: A Kernel Fuzzy c-Means Clustering-Based Fuzzy Support Vector Machine Algorithm for Classification Problems With Outliers or Noises. IEEE Transactions on Fuzzy Systems. 19, 105–115 (2011). https://doi.org/10.1109/TFUZZ.2010.2087382.
4. Goertzel, B., Pennachin, C. eds: Artificial General Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-68677-4.
5. Feinberg, E.A., Shwartz, A. eds: Handbook of Markov Decision Processes. Springer US, Boston, MA (2002). https://doi.org/10.1007/978-1-4615-0805-2.
6. Berthold, M.R., Borgelt, C., Höppner, F., Klawonn, F.: Guide to Intelligent Data Analysis. Springer London, London (2010). https://doi.org/10.1007/978-1-84882-260-3.
7. Bloch, I., Hunter, A., Appriou, A., Ayoun, A., Benferhat, S., Besnard, P., Cholvy, L., Cooke, R., Cuppens, F., Dubois, D., Fargier, H., Grabisch, M., Kruse, R., Lang, J., Moral, S., Prade, H., Saffiotti, A., Smets, P., Sossai, C.: Fusion: General concepts and characteristics. International Journal of Intelligent Systems. 16, 1107–1134 (2001). https://doi.org/10.1002/int.1052.
8. Fearing, F., Pavlov, I.P., Anrep, G. V.: Conditioned Reflexes. An Investigation of the Physiological Activity of the Cerebral Cortex, (1929). https://doi.org/10.2307/1134737.
9. Ghavamzadeh, M., Mannor, S., Pineau, J., Tamar, A.: Bayesian reinforcement learning: A survey. (2015). https://doi.org/10.1561/2200000049.
10. Bellman, R.: A Markovian Decision Process. Indiana University Mathematics Journal. 6, 679–684 (1957). https://doi.org/10.1512/iumj.1957.6.56038.
11. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning. 3, 9–44 (1988). https://doi.org/10.1007/BF00115009.
12. Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo Method. John Wiley & Sons, Inc., Hoboken, NJ, USA (2016). https://doi.org/10.1002/9781118631980.
13. Goerzen, C., Kong, Z., Mettler, B.: A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. Journal of Intelligent and Robotic Systems. 57, 65–100 (2010). https://doi.org/10.1007/s10846-009-9383-1.

14. Li, Y.: Deep Reinforcement Learning. ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference Tutorial Abstracts. 19–21 (2018). https://doi.org/10.18653/v1/p18-5007.
15. Dockhorn, A.: Prediction-based Search for Autonomous Game-Playing, (2020).
16. Apeldoorn, D., Dockhorn, A.: Exception-Tolerant Hierarchical Knowledge Bases for Forward Model Learning. IEEE Transactions on Games. 1–14 (2020). https://doi.org/10.1109/TG.2020.3008002.
17. Dockhorn, A., Apeldoorn, D.: Forward Model Approximation for General Video Game Learning. In: Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG'18). pp. 425–432. IEEE (2018). https://doi.org/10.1109/CIG.2018.8490411.
18. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, C.S.: Introduction to Algorithms. The MIT Press, Cambridge (2009).
19. Maschler, M., Solan, E., Zamir, S.: Game Theory. Cambridge University Press, Cambridge (2013). https://doi.org/10.1017/CBO9780511794216.
20. Costalba, M., J. Kiiski, G. Linscott, T. Romstad: Stockfish Chess Engine, http://www.stockfishchess.org/.
21. Tromp, J., Farnebäck, G.: Combinatorics of Go. In: Computers and Games. pp. 84–99 (2007). https://doi.org/10.1007/978-3-540-75538-8_8.
22. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., and Spiliopoulou, M. (eds.) ECML'06 Proceedings of the 17th European conference on Machine Learning. pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/11871842_29.
23. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games. 4, 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810.
24. Gaina, R.D., Lucas, S.M., Perez-Liebana, D.: Rolling horizon evolution enhancements in general video game playing. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG). pp. 88–95. IEEE (2017). https://doi.org/10.1109/CIG.2017.8080420.
25. Gaina, R.D., Liu, J., Lucas, S.M., Pérez-Liébana, D.: Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing. In: Lecture Notes in Computer Science. pp. 418–434 (2017). https://doi.org/10.1007/978-3-319-55849-3_28.
26. Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R.D., Togelius, J., Lucas, S.M.: General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms. (2018).
27. Levine, J., Congdon, C., Ebner, M., Kendall, G.: General video game playing. Dagstuhl Follow-Ups. 1–7 (2013). https://doi.org/10.4230/DFU.VOL6.12191.77.
28. Dockhorn, A., Grueso, J.H., Jeurissen, D., Perez-Liebana, D.: "Stratega": A General Strategy Games Framework. In: Osborn, J.C. (ed.) Joint Proceedings of the AIIDE 2020 Workshops co-located with 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2020). pp. 1–7. CEUR Workshop Proceedings, Worcester (2020).
29. Perez-Liebana, D., Dockhorn, A., Grueso, J.H., Jeurissen, D.: The Design Of "Stratega": A General Strategy Games Framework. (2020).

30. Majeed, S.J.: Abstractions of General Reinforcement Learning. (2021).
31. Boutilier, C., Dearden, R.: Using abstractions for decision-theoretic planning with time constraints. Proceedings of the National Conference on Artificial Intelligence. 2, 1016–1022 (1994).
32. Dean, T., Givan, R.: Model Minimization in Markov Decision Processes. In: In Proceedings of the Fourteenth National Conference on Artificial Intelligence. pp. 106–111 (1997).
33. Dean, T.L., Givan, R., Leach, S.: Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes. 124–131 (2013).
34. Even-Dar, E., Mansour, Y.: Approximate equivalence of markov decision processes. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science). 2777, 581–594 (2003). https://doi.org/10.1007/978-3-540-45167-9_42.
35. Janner, M., Fu, J., Zhang, M., Levine, S.: When to trust your model: Model-based policy optimization. Advances in Neural Information Processing Systems. 32, (2019).
36. Hao, B., Duan, Y., Lattimore, T., Szepesvári, C., Wang, M.: Sparse Feature Selection Makes Batch Reinforcement Learning More Sample Efficient. (2020).
37. Jong, N.K., Stone, P.: State abstraction discovery from irrelevant state variables. IJCAI International Joint Conference on Artificial Intelligence. 752–757 (2005).
38. Kroon, M., Whiteson, S.: Automatic feature selection for model-based reinforcement learning in factored MDPs. 8th International Conference on Machine Learning and Applications, ICMLA 2009. 324–330 (2009). https://doi.org/10.1109/ICMLA.2009.71.
39. Cheng, Z., Ray, L.E.: State abstraction in reinforcement learning by eliminating useless dimensions. Proceedings - 2014 13th International Conference on Machine Learning and Applications, ICMLA 2014. 105–110 (2014). https://doi.org/10.1109/ICMLA.2014.22.
40. Mccallum, A.K.: Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks. From Animals to Animats 4. (2020). https://doi.org/10.7551/mitpress/3118.003.0039.
41. Kheradmandian, G., Rahmati, M.: Automatic abstraction in reinforcement learning using data mining techniques. Robotics and Autonomous Systems. 57, 1119–1128 (2009). https://doi.org/10.1016/j.robot.2009.07.002.
42. Mannor, S., Menache, I., Hoze, A., Klein, U.: Dynamic abstraction in reinforcement learning via clustering. In: Twenty-first international conference on Machine learning - ICML '04. p. 71. ACM Press, New York, New York, USA (2004). https://doi.org/10.1145/1015330.1015355.
43. Singh, S.P., Jaakkola, T., Jordan, M.J.: Reinforcement Learning with Soft State Aggregation. Advances in Neural Information Processing Systems 7. 361–368 (1995).
44. Dockhorn, A., Hurtado-Grueso, J., Jeurissen, D., Xu, L., Perez-Liebana, D.: Game State and Action Abstracting Monte Carlo Tree Search for General Strategy Game-Playing. In: 2021 IEEE Conference on Games (CoG). pp. 1–8. IEEE (2021). https://doi.org/10.1109/CoG52621.2021.9619029.
45. Hostetler, J., Fern, A., Dietterich, T.: State aggregation in Monte Carlo tree search. Proceedings of the National Conference on Artificial Intelligence. 4, 2446–2452 (2014).

46. Lange, S., Riedmiller, M.: Deep auto-encoder neural networks in reinforcement learning. Proceedings of the International Joint Conference on Neural Networks. (2010). https://doi.org/10.1109/IJCNN.2010.5596468.
47. Olah, C., Mordvintsev, A., Schubert, L.: Feature Visualization. Distill. 2, (2017). https://doi.org/10.23915/distill.00007.
48. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings. 1–14 (2017).
49. Jiang, J., Dun, C., Huang, T., Lu, Z.: Graph Convolutional Reinforcement Learning. (2018).
50. Liu, K., Fu, Y., Wu, L., Li, X., Aggarwal, C., Xiong, H.: Automated Feature Selection: A Reinforcement Learning Perspective. IEEE Transactions on Knowledge and Data Engineering. 4347, (2021). https://doi.org/10.1109/TKDE.2021.3115477.
51. Bulitko, V., Sturtevant, N., Lu, J., Yau, T.: Graph Abstraction in Real-time Heuristic Search. Journal of Artificial Intelligence Research. 30, 51–100 (2007). https://doi.org/10.1613/jair.2293.
52. Botea, A., Martin, M., Schaeffer, J., Tg, C.: Near Optimal Hierarchical Path-Finding. Journal of Game Development. 1, 1–30 (2004).
53. Sturtevant, N., Buro, M.: Partial pathfinding using map abstraction and refinement. In: Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference. pp. 1392–1397 (2005).
54. Sturtevant, N., Buro, M.: Improving collaborative pathfinding using map abstraction. Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2006. 80–85 (2006).
55. Bulitko, V., Sturtevant, N., Kazakevich, M.: Speeding up learning in reel-time search via automatic state abstraction. Proceedings of the National Conference on Artificial Intelligence. 3, 1349–1354 (2005).
56. Dockhorn, A., Hurtado-Grueso, J., Jeurissen, D., Xu, L., Perez-Liebana, D.: Portfolio Search and Optimization for General Strategy Game-Playing. In: 2021 IEEE Congress on Evolutionary Computation (CEC). pp. 2085–2092. IEEE (2021). https://doi.org/10.1109/CEC45853.2021.9504824.
57. Ontañón, S.: The combinatorial multi-armed bandit problem and its application to real-time strategy games. Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 58–64 (2013).
58. Churchill, D., Buro, M.: Hierarchical portfolio search: Prismata's robust AI architecture for games with large search Spaces. Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015. 2015-Novem, 16–22 (2015).
59. Churchill, D., Buro, M.: Portfolio greedy search and simulation for large-scale combat in starcraft. IEEE Conference on Computational Intelligence and Games, CIG. (2013). https://doi.org/10.1109/CIG.2013.6633643.
60. Moraes, R.O., Mariño, J.R.H., Mariño, M., Lelis, L.H.S.: Nested-Greedy Search for Adversarial Real-Time Games. In: Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2018. pp. 67–73 (2018).

61. Wang, C., Chen, P., Li, Y., Holmgård, C., Togelius, J.: Portfolio Online Evolution in StarCraft. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. pp. 114–120 (2021).
62. Kruse, R., Borgelt, C., Braune, C., Mostaghim, S., Steinbrecher, M.: Computational Intelligence. Springer London, London (2022).
63. Lelis, L.H.S.: Stratified Strategy Selection for Unit Control in Real-Time Strategy Games. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. pp. 3735–3741 (2017). https://doi.org/10.24963/ijcai.2017/522.
64. Justesen, N., Tillman, B., Togelius, J., Risi, S.: Script- and cluster-based UCT for StarCraft. IEEE Conference on Computational Intelligence and Games, CIG. (2014). https://doi.org/10.1109/CIG.2014.6932900.
65. Moraes, R.O., Levi H. S. Lelis: Asymmetric Action Abstractions for Multi-Unit Control in Adversarial Real-Time Games. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 876–883 (2020).
66. Yang, Z., Ontañón, S.: Guiding Monte Carlo tree search by scripts in real-time strategy games. Proceedings of the 15th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2019. 100–106 (2019).
67. Lelis, L.H.S.: Planning Algorithms for Zero-Sum Games with Exponential Action Spaces: A Unifying Perspective. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. pp. 4892–4898 (2020). https://doi.org/10.24963/ijcai.2020/681.
68. Barriga, N.A., Stanescu, M., Buro, M.: Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. pp. 9–15 (2015).
69. Barriga, N.A., Stanescu, M., Buro, M.: Game tree search based on nondeterministic action scripts in real-time strategy games. IEEE Transactions on Games. 10, 69–77 (2018). https://doi.org/10.1109/TCIAIG.2017.2717902.
70. Perez-Liebana, D., Guerrero-Romero, C., Dockhorn, A., Xu, L., Hurtado, J., Jeurissen, D.: Generating Diverse and Competitive Play-Styles for Strategy Games. In: 2021 IEEE Conference on Games (CoG). pp. 1–8. IEEE (2021). https://doi.org/10.1109/CoG52621.2021.9619094.
71. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence. 112, 181–211 (1999). https://doi.org/10.1016/S0004-3702(99)00052-1.
72. Jong, N.K., Hester, T., Stone, P.: The Utility of Temporal Abstraction in Reinforcement Learning. In: AAMAS ’08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems. pp. 299–306 (2008). https://doi.org/10.5555/1402383.1402429.
73. Pickett, M., Barto, A.G.: PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. Proceedings of the 19th International Conference on Machine Learning (ICML). 506–513 (2002).
74. Powley, E.J., Whitehouse, D., Cowling, P.I.: Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem. 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012. 234–241 (2012). https://doi.org/10.1109/CIG.2012.6374161.

75. Perez, D., Powley, E.J., Whitehouse, D., Rohlfshagen, P., Samothrakis, S., Cowling, P.I., Lucas, S.M.: Solving the Physical Traveling Salesman Problem: Tree Search and Macro Actions. IEEE Transactions on Computational Intelligence and AI in Games. 6, 31–45 (2014). https://doi.org/10.1109/TCIAIG.2013.2263884.
76. Dockhorn, A., Doell, C., Hewelt, M., Kruse, R.: A decision heuristic for Monte Carlo tree search doppelkopf agents. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (2017). https://doi.org/10.1109/SSCI.2017.8285181.
77. Şimşek, Ö., Barto, A.G.: Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Twenty-first international conference on Machine learning - ICML '04. p. 95. ACM Press, New York, New York, USA (2004). https://doi.org/10.1145/1015330.1015353.
78. McGovern, A., Barto, A.G.: Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. Proceedings of the 18th International Conference on Machine Learning. 361–368 (2001).
79. Şimşek, Ö., Wolfe, A.P., Barto, A.G.: Identifying useful subgoals in reinforcement learning by local graph partitioning. ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning. 817–824 (2005). https://doi.org/10.1145/1102351.1102454.
80. Moradi, P., Shiri, M.E., Rad, A.A., Khadivi, A., Hasler, M.: Automatic skill acquisition in reinforcement learning using graph centrality measures. Intelligent Data Analysis. 16, 113–135 (2012). https://doi.org/10.3233/IDA-2011-0513.
81. Metzen, J.H.: Online skill discovery using graph-based clustering. In: Proceedings of the Tenth European Workshop on Reinforcement Learning. pp. 77–88 (2013).
82. Kemke, C., Walker, E.: Planning with Action Abstraction and Plan Decomposition Hierarchies. In: 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. pp. 447–451. IEEE (2006). https://doi.org/10.1109/IAT.2006.99.
83. Sebastia, L., Onaindia, E., Marzal, E.: Decomposition of planning problems. AI Communications. 19, 49–81 (2006).
84. Konidaris, G., Barto, A.: Skill discovery in continuous reinforcement learning domains using skill chaining. Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference. 1015–1023 (2009).
85. Abel, D., Jinnai, Y., Guo, Y., Konidaris, G., Littman, M.L.: Policy and Value Transfer in Lifelong Reinforcement Learning. In: Proceedings of the 35th International Conference on Machine Learning. pp. 1–10. PMLR, Stockholm (2018).
86. Brunskill, E., Li, L.: PAC-inspired option discovery in lifelong reinforcement learning. 31st International Conference on Machine Learning, ICML 2014. 2, 1599–1610 (2014).
87. Blanco-Mesa, F., Merigó, J.M., Gil-Lafuente, A.M.: Fuzzy decision making: A bibliometric-based review. Journal of Intelligent and Fuzzy Systems. 32, 2033–2050 (2017). https://doi.org/10.3233/JIFS-161640.
88. Kacprzyk, J., Zadrozny, S., Nurmi, H., Bozhenyuk, A.: Towards innovation focused fuzzy decision making by consensus. IEEE International Conference on Fuzzy Systems. 2021-July, (2021). https://doi.org/10.1109/FUZZ45933.2021.9494531.
89. Onisawa, T., Kacprzyk, J. eds: Reliability and Safety Analyses under Fuzziness. Physica-Verlag HD, Heidelberg (1995). https://doi.org/10.1007/978-3-7908-1898-7.

90. Alonso Moral, J.M., Castiello, C., Magdalena, L., Mencar, C.: Explainable Fuzzy Systems. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-71098-9.
91. Owsiński, J.W., Stańczak, J., Opara, K., Zadrożny, S., Kacprzyk, J.: Reverse Clustering. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-69359-6.
92. Chakraborti, T., Sreedharan, S., Kambhampati, S.: The emerging landscape of explainable automated planning & decision making. IJCAI International Joint Conference on Artificial Intelligence. 2021-January, 4803–4811 (2020). https://doi.org/10.24963/ijcai.2020/669.

Authors

Alexander Dockhorn is Juniorprofessor for Computer Science at the Gottfried Wilhelm Leibniz University Hannover. Following his dissertation at the Otto-von-Guericke University of Magdeburg (OVGU), he worked at the Queen Mary University of London and later came back to the OVGU. During this time, he studied artificial intelligence in general strategy games and began working on abstraction methods for simulation-based search agents in games. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and currently serves as the chair of the IEEE Games Technical Committee and the Summer School Subcommittee. From 2017 to 2020 he was organizing the Hearthstone AI competition to foster comparability of AI agents in complex card games. See his webpage for a complete list of projects and publications: https://adockhorn.github.io/

Rudolf Kruse is an Emeritus Professor for Computer Science at the Otto-von-Guericke University of Magdeburg Germany. He obtained his Ph.D. and his Habilitation in Mathematics from the Technical University of Braunschweig in 1980 and 1984 respectively. Following a stay at the Fraunhofer Gesellschaft, he joined the Technical University of Braunschweig as a professor of computer science in 1986. From 1996-2017 he was the leader of the Computational Intelligence Group in Magdeburg. He has co-authored 15 monographs and 25 books as well as more than 400 peer-refereed scientific publications in various areas. He is Fellow of the Institute of Electrical and Electronics Engineers (IEEE), Fellow of the International Fuzzy Systems Association (IFSA), and Fellow of the European Association for Artificial Intelligence (EURAI/ECCAI). His group is successful in various industrial applications, see his webpage www.is.ovgu.de/Team/Rudolf+Kruse.html. His research interests include data science and intelligent systems.