Exploiting Action Categories in Learning Complex Games

Citation for published version:
Dobre, M & Lascarides, A 2018, Exploiting Action Categories in Learning Complex Games. in IEEE Technically Sponsored Intelligent Systems Conference (IntelliSys 2017). IEEE, London. UK, pp. 729-737. DOI: 10.1109/IntelliSys.2017.8324210

Digital Object Identifier (DOI):
10.1109/IntelliSys.2017.8324210

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
IEEE Technically Sponsored Intelligent Systems Conference (IntelliSys 2017)

Exploiting Action Categories in Learning Complex Games

Mihai S. Dobre
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB
Scotland
Email: m.s.dobre@sms.ed.ac.uk

Alex Lascarides
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB
Scotland
Email: alex@inf.ed.ac.uk

***Abstract*—This paper presents a model for planning in a highly complex game, where certain action *types* are more common than others and cyclic behaviour can also easily arise. These issues are addressed by exploiting the inherent structure among the possible options to enhance the online learning algorithm: sampling during Monte Carlo Tree Search becomes a two step process, by first sampling from a distribution over the *types* of legal actions followed by sampling from individual actions of the chosen type. This policy drastically reduces the breadth of the rollout as well as its depth by avoiding redundant sampling behaviour. The result is a large increase in both the performance and efficiency of the model. Another contribution of this paper is assessing the benefits of a parallel implementation and afterstates in complex games. Evaluation is done via agent simulations in the board game *Settlers of Catan*. The resulting agent is the first based on purely online learning strategies that can handle the full set of legal actions of the game. The evaluation shows that our model outperforms previous state-of-the-art agents while taking decisions in a time threshold tolerated by human opponents.**

Keywords*—model-based reinforcement learning; Monte Carlo Tree Search; complex games; hierarchical sampling; online learning; Settlers of Catan;

I. Introduction

Learning optimal policies by using reinforcement learning is well established, and in simple games these approaches can find an exact solution [1]. But for games where the search space is massive, deriving reliable policies for every state one might encounter is infeasible. A popular alternative is to use *online* methods to estimate the expected utility of the legal actions every time a state is encountered, without keeping the result in memory. Monte Carlo Tree Search (MCTS) [2] has become a popular online planning method, with considerable success in complex games such as Go [3]. MCTS addresses the vast dimensionality by dynamically building the game tree as it encounters new states [4].

Despite these characteristics, the performance of MCTS degrades with the length of the planning horizon, because the complexity of the planning problem increases exponentially (this is known as the curse of history). One solution for handling this problem is to define a hierarchy subroutine, also known as Macro actions, which are composed of a sequence of primitive actions. Dietterich [5] introduced the MAXQ framework and proposed the Taxi problem for evaluating the performance of hierarchical planners. Vien and Toussaint [6] have extended the MCTS planning framework accordingly, via a hierarchy of pre-defined subtasks that in turn reduce the set of policies that can be considered. As a result, the computational cost shrinks considerably, with the effect proportional to the length of the macro actions [7].

These hierarchical models assume that subgoals can be easily identified by the developers [5]. They haven't been applied to very complex games where it is impossible to exhaustively define all possible subgoals. If they were, it would only partly reduce the computational cost, due to long rollouts. He, Brunskill and Roy [7] show that macro-actions can be automatically generated given a set of parameters (such as the number of macro-actions and their maximum length). But our aim is to create an agent whose decision time a human opponent can tolerate, so any such generative methods would have to be applied offline. Furthermore, macro actions impose a structure on the task. So, defining some but not all of the possible subgoals limits the possible policies that the agent can learn, with the inherent risk that the optimal policy isn't among them.

There are also well developed ways of dealing with a large branching factor of some complex games, such as grouping the moves in MCTS according to some characteristic and combining their statistics in the tree. This approach is known as Move Groups, in which the moves are clustered based on a specific characteristic defined in advance [8]. Another very similar method is to group the chance actions that can be executed from one state into a single group, which is called node-groups [9], [10]. All these grouping techniques have only been applied to the selection part of MCTS (that is the question of which next action among all the possible next actions the algorithm should compute an expected utility for). In other complex games, splitting the decision into multiple steps and choosing the order of the steps have greatly increased the performance of the algorithm [11], [12]. All of the methods described above place a large burden on the developer, who must have detailed knowledge of the game characteristics: e.g. which kinds of actions tend to contribute to which goals.

In this paper we present a novel approach to sampling the decision space of a complex planning problem, which increases both performance and efficiency. Instead of defining macro-actions or grouping actions based on a specific metric, we exploit the natural hierarchy of the actions as it is defined in the environment. We therefore sample from the set of legal action *types* before sampling the specifics of each type during rollouts in MCTS. This results in a reduced branching factor and a reduced depth by avoiding redundant behaviour. Such an approach has numerous benefits, particularly in domains where the sets of distinct action types varies a lot in size but their respective benefits are generally comparable. We evaluate the performance of the model against two state-of-the art baselines in the board game *Settlers of Catan*. Our model is the first purely online learning implementation that is able to handle all legal actions of the game and it outperforms all previous implementations.

II. Related Work

A. Hierarchical planning models

Many existing models exploit a domain's hierarchy to reduce the number of policies an RL algorithm needs to consider. Dietterich's [5] MAXQ framework, which we mentioned earlier, recursively decomposes the overall value function into a collection of value functions for the subtasks, subsubtasks and so on. Experiments on the taxi problem show that the MAXQ model can converge much faster than Q-learning. Dietterich however, provides a detailed discussion on how this approach relies heavily on careful reasoning when designing certain parts of the hierarchical models, such as defining the subgoals or the use of state abstraction. In a game as complex as Settlers of Catan, the smallest mistakes may cause such an approach to fail.

Vien and Toussaint [6] present an extension of the UCT and POMCP methods [13] for planning in hierarchical MDPs and POMDPs. They test performance on the taxi problem and a partially-observable version of Pacman, reporting significant improvements over the initial models. However, these domains are small enough that the agent can store information on the environment between runs. This isn't feasible in a game the size of Settlers of Catan without a very powerful abstraction method. Another advantage of keeping a Q-table over runs is that the algorithm can benefit from intermediate or discounted rewards. But the intermediate rewards are hard to define in Catan (the current players' score is not necessarily a good indicator in the first half of the game) and the length of the game makes it impossible to discount the simple return of a 1 for a win or 0 for a loss. Finally, exhaustively defining subgoals in Settlers of Catan is impossible given its complexity. Due to being a multi-player game, great care must be given in case subtasks can be blocked by the opponents. This requires methods for checking if the subgoals are achievable and fine-tuning parameters, such as the depth of the search.

Instead of defining a subroutine hierarchy, our model takes advantage of the way the actions group naturally into distinct categories or *types*. Our approach ensures a uniform distribution over the types of actions when exploring the search space, such that the employed policy will not be biased towards preferring the type of action whose set of (concrete) options vastly outnumbers those of another type. This model is related to Hierarchical Bayes, where a prior distribution over the hyperparameters yields a more expressive model [14]. The hierarchy that we have defined is similar to a semantic hierarchy which can be used to reduce the search space, e.g. in computer vision, such a semantic hierarchy can be exploited to acquire labels more efficiently during annotation tasks [15].

B. Monte Carlo Tree Search

Move Groups is a proposed method for reducing the branching of the tree by grouping moves based on a specific characteristic [8]. Therefore an extra decision node is introduced at each level of the tree and the statistics of the children nodes are combined at the parent node. This approach works well when there is some correlation between the moves such that the information on one move gained during the search generalises to the other moves belonging to the same group. In the past, certain similarity metrics (e.g. Manhattan distance) have been used to generate groups that are not necessarily disjoint, i.e. one move may belong to several groups. Move groups has been applied to the stochastic game of Chinese Dark Chess to represent all the revealing actions that can be executed in one state by a single node. This is referred to as node-groups [9], [10].

Other methods split a move into several parts and allow the MCTS to decide on each sequentially [11], [12], [16]. This approach requires expert knowledge of the domain to choose the parts as well as the order these are executed in. Cowling et al. [16] mention that the important decisions should be higher in the tree. Therefore, in their implementation on the game Magic: The Gathering, they chose the order of these decisions based on the mana cost of each card. Due to how the statistics of these action parts are stored in a tree structure, the secondary parts are "grouped" and have as parent one of the former parts of the action. It also greatly reduces the branching factor of the tree. So it roughly resembles the Move Group method, but these are two different approaches.

There are also many methods for improving the rollouts in the MCTS algorithm, but mostly these make use of existing rule-based implementations or hand-crafted heuristics to bias the search. Previous methods employed an existing agent to search the game [17], [18] or define their own rules based on expert knowledge [16], [19]. These methods are sometimes referred to as pseudorandom games [11]. The rules used in previous implementations need to be devised in advance and may not generalise to all cases in the game. Furthermore, the resulting policy is a deterministic policy that could get stuck in a loop.

Instead of defining similarity metrics or modifying the decision process as presented in the existing literature, our method exploits how the actions are naturally grouped into categories by the game rules. Furthermore, we apply this grouping to the sampling stage of the MCTS algorithm instead of the selection step. Different to the heuristic rollouts previously employed, our rollouts follow a stochastic policy. In the two step sampling that we describe in section V, where we first sample the type and then sample from that type's distinct options, there is a shift in the probability mass towards the type of actions that are less likely to be selected due to the small number of options. In this paper we use a uniform distribution to select action types and to select the following options. It can easily be extended with domain knowledge by weighting the types and their options accordingly.

C. Settlers of Catan

Prior work on modelling agents that play *Settlers of Catan* varies on: (a) the extent to which they rely on hand-coded heuristics vs. machine learning, (b) on how large a portion of the game they aim to model, and (c) on the empirical data that informs their approach. The agent released with the JSettlers platform [20] doesn't use any machine learning. Instead, it forms a decision tree based on a symbolic estimate as to how quickly the player reaches 10 victory points (and so wins the game). This estimate is computed from hand-coded heuristics. Guhe and Lascarides [21] improved the original JSettlers heuristics to create what we call the *Stac* agent. To our knowledge this is the strongest rule-based Settlers of Catan agent and, like ours, it can play according to the game's complete set of options.

Pfeiffer [22] presented a method that combines low-level reinforcement learning mechanisms with hand-crafted, high-level heuristics. The heuristics are intended to reflect human expert policies on medium-term goals. The author reported that the system could win against some human players, and also observed that the very complex hand-coded heuristics for choosing the high-level actions were critical to success. On the other hand, Szita, Chaslot and Spronck [23] developed the *SmartSettlers* agent that applies MCTS with only a limited amount of domain knowledge on a simplified version of the game, removing the agent's ability to negotiate and trade resources with other players. Their results show a considerable increase in playing strength compared to the original JSettlers agent [20].

We will evaluate our agent's performance against the SmartSettlers agent (on the reduced game) and the Stac agent (on the full game). We will also compare our agent against a version of itself that doesn't exploit the action type hierarchy.

III. Resources

A. Settlers of Catan

Settlers of Catan is a multi-player win-lose board game. We focus on the core game for 4 players. The players build roads, settlements and cities on the board, which is formed of hexagonal tiles. The first player to reach 10 victory points wins the game. One obtains victory points in a variety of ways (e.g. a settlement is worth 1 point and a city is worth 2 points). Board tiles represent one of the five resources (Clay, Ore, Sheep, Wood and Wheat), desert, water or ports. Each of the resource producing tiles has an associated number between 2 and 12 (but not 7). Players obtain resources via the location of their buildings and dice rolls that start each turn of the game. One needs different combinations of resources to build different pieces (e.g. a road costs 1 clay and 1 wood). In addition to dice rolls, players can acquire resources through trades with the bank (at a 4:1 ratio), or with a port if they have built a settlement or city there (3:1 or 2:1, depending on the port) or through negotiated trades with other players. There are many special actions which increase the complexity of the game, such as: *(i)* playing development cards that each give different advantages, *(ii)* moving the robber and stealing resources from other players and *(iii)* gaining victory points via the longest road or largest army (see www.catan.com for details).

B. Game analysis

Settlers of Catan incorporates a clear structure to its actions. These can be grouped into several types, e.g. trade actions or build road actions. The cardinality of one class almost always dominates those of others (i.e. there are always more trade options). Many other games share this characteristic - e.g. Civilisation, Diplomacy, Battlestar Galactica to name a few. The way our model avoids MCTS's inherent bias towards choosing an action of a dominant type can be applied to these games too.

From a game theoretic perspective, Settlers of Catan is a very complex game. In addition to the incomplete information (i.e. the opponent's policies), the game contains elements of imperfect information (the resources that opponents possess and the unplayed development cards) and it is stochastic (dice rolls determine the players' resources). It has a large branching factor e.g. there's a wide range of negotiation actions one can employ to trade the necessary resources with others. The generation of the board is done by shuffling the 19 land hexes and the 9 port hexes, so the game has a huge space of possible initial states ($\approx 1.2 * 10^{15}$ compared to 1 for the game of Go). As far as we are aware, there is no analytic solution. Table I contains approximations of the branching and depth factors, given 3 different sampling policies. The results corresponding to the human and agent policies have been averaged over 60 human games [24] and 1000 simulated games respectively. Due to how these games have been logged, trades are considered as a single exchange action and the preceding negotiations are not taken into account, hence the depths in Table I are smaller than in reality.

TABLE I. Average branching and depth. Many of the human games have 2 or 3 players, hence the smaller depth.

Policy	Branch	Depth
Heuristic	69	234
Human	63	152
Random	64	11639

Game Modifications: First of all, we make the players hands visible, but we keep the order of the deck of the development cards hidden and treat the action of buying a development card as a chance event. This does not modify the set of legal actions, but it turns the game into one of perfect information. The baseline agents that we use to evaluate our approach will therefore also have perfect information. The game's imperfect information is a problem that we intend to handle in a future version of our player and it is unrelated to this paper's contribution.

The purpose of the following modifications is to level the playing field between the evaluated agents and the baselines. Since the rule-based agents only consider 1:1, 1:2 and 2:1 resource trades, our MCTS agent's set of legal actions against Stac opponents is also limited to these. There are still a large set of possible trades (up to 540). Finally, since SmartSettlers agents don't trade, our MCTS agent playing against them doesn't trade either. Otherwise, the SmartSettlers agent would be disadvantaged; limiting an agent's trade capability handicaps it [25].

IV. Monte Carlo Tree Search

MCTS is a planning method for finding optimal solutions that combines random sampling of the decision space with the precision of a search tree [2], [4]. The high-level structure is presented in Algorithm 1.

```
create $n$ root node;
while within computational budget do
    $n \leftarrow$ TREE_POLICY($n$);
    $r \leftarrow$ ROLLOUT_POLICY($n$);
    BACKPROPAGATION($n, r$);
end
return BEST_ACTION($n$);
```

Algorithm 1: The basic MCTS algorithm

Upper Confidence bounds for Trees (UCT) is a very successful node selection criterion used in the tree policy, with an overall better performance than ϵ-greedy methods [26], [27]. During the tree traversal phase of MCTS, the child nodes with the highest UCT value are selected. The UCT value of a node j is computed as shown in Equation 1, where $\overline{X_j}$ is the value of the node represented as number of wins out of the number of plays via this node, n the number of times its parent node was tried, n_j the number of times it was tried and C is a constant used to decide on the level of exploration. The standard tree policy is shown in Algorithm 2.

$$UCT_j = \overline{X_j} + C\sqrt{\frac{2\ln n}{n_j}} \quad (1)$$

```
while $n$ is non-terminal do
    if $n$ is a leaf node then
        return EXPAND($n$);
    else
        $n \leftarrow$ SELECT_UCT($n$);
    end
end
return $n$;
```

Algorithm 2: Standard TREE_POLICY

The search starts from the root node corresponding to the current state in the game and slowly builds up a tree by adding one node per iteration. A new node is added in the expansion step of the algorithm, which is executed when a leaf node is encountered during the tree policy. The rollout step follows, which utilises the game model to play the game until a terminal state is reached. The policy used (π_a), chooses an action uniformly at random from the list of legal actions and is called the default policy. Finally, the statistics stored in the nodes are updated based on the result of the rollout. When the computational budget limit is reached, the best action is executed in the real game. There are multiple methods for choosing this action: max child, robust child [4]. For our implementation we select the action that yields the highest reward average ($\overline{X_j}$) to be played in the real game.

MCTS can be extended to multi-player games, by keeping track of the statistics for each player in the nodes and modelling the turn change in the game logic. Non-deterministic actions can be handled by introducing chance nodes, where the following move is a nature move. These select the outcome of the player's action based on a distribution that's determined by the game rules (e.g. the player rolls a die and nature chooses one of the 6 outcomes via an even distribution). There are several extensions that support reasoning about partially-observable moves and imperfect information, but we leave this to future work.

```
$s \leftarrow n$.getState();
while $s$ is non-terminal do
    $a \sim \pi_a(s)$);
    $s \leftarrow$ performAction($s, a$);
end
return reward for $s$;
```

Algorithm 3: Standard ROLLOUT_POLICY

A. Parallelising MCTS

We wish to balance the need of exploring the game space to yield decent strategies with the need to decide on the next move within a time interval that human opponents would tolerate. We achieve this by exploiting efficiency gains afforded by parallelising MCTS. There are three alternatives: leaf parallelisation, root paralellisation and tree parallelisation [28]. We have chosen tree parallelisation, which keeps a single copy of the tree that is shared by all threads. Our implementation synchronises the update and expansion steps of the algorithm, while storing the tree in a synchronised data structure. We have also added virtual loss to discourage multiple threads from taking the same path through the tree. A lock-free version implementation is straightforward to write, but we have not observed any major efficiency improvements. This could be due to the relatively small number of threads compared to other implementations such as AlphaGo [3] and large variance in game length causing the threads to be in different parts of the tree or steps of the algorithm most of the time.

V. MCTS with the Action Type Hierarchy

The base MCTS algorithm relies on Monte Carlo sampling of the space to ensure that an accurate estimation of the current state's value can be performed given a sufficient computational budget. The default sampling policy assumes a uniform distribution over the legal actions given the current state. These actions could belong to the same class of actions for simple games or could belong to one of many classes in more complex games. In the case where there is a single class, or if the number of action options belonging to each class is similar, the uniform sampling over all the legal options would ensure a sufficiently unbiased estimation. But, if the cardinality of different classes is very different, the resulting policy is more likely to execute an action that belongs to a dominant class. The resulting estimation would be accurate only if the MCTS agent's opponents in the real game have a similar policy (i.e. one where they tend to perform actions from a dominant class). This is highly unlikely if there isn't a huge benefit to executing one of the options from the dominant class over those of other classes.

Large complex games, such as most of the multi-player board games or video games (e.g the Civilisation series, Diplomacy, Battlestar Galactica), generally have a large branching factor and present a clearly defined structure in the rules of the game. This structure permits the classification of states and actions into types, which learning algorithms can exploit. Furthermore, in most games, the number of actions in each type varies widely: e.g. in turn-based games, the end turn type cardinality will almost always be smaller than the other classes. In complex games with this characteristic, not only will learning be biased towards action types of large cardinality, but a policy that prefers the actions belonging to the dominant type(s) also creates redundant or cyclic behaviour (e.g. the trade type actions in Settlers of Catan). This is caused by either the policy avoiding the action types that are needed to finish the game or the dominant type presenting options that revert the state back to an earlier one (e.g. the move action of an agent in a maze like environment). Moreover, such a policy may be a weak policy if the action type is not sufficient: e.g. in Catan, trades are necessary to gain access to scarce resources, but executing other action types is required to win the game.

Turning the Monte Carlo sampling into a two step process – where first the action type is sampled followed by the action description from the options belonging to the class – would address all the above limitations. In Settlers of Catan for example, road building is a different type of action to city building; but the specific description of such actions include where to place the piece on the board. Accordingly, our model first chooses between building a road or a city, followed by the location from the legal options for the chosen type. This approach will also reduce the branching factor of the game: only the options belonging to one type are listed as options and the number of classes is inherently smaller than the number of all options. Finally, it allows for more game specific sampling tricks, e.g. if the trade type is selected in Catan, we can sample resources from the two participant's hands directly instead of sampling from the larger space of all the trade options.

Table II contains the list of types that a player encounters in what we call *normal phase* of the Settlers of Catan game [29]. Most of the game is spent in this phase in which players usually have to decide between multiple types of actions. The table also includes the probability of options belonging to a type being present, the average number of options belonging to the type when this is present and the maximum number of options for each type computed over all the decisions made in this phase during 10k games. These games were generated using the standard random policy of selecting uniformly at random from all legal actions indifferent of their type. The high values for the trade actions indicates that players have to generally decide between many trade options and a few options belonging to other classes. A uniform at random policy over this set is much more likely to choose a trade action. This also explains why the depth of games generated following the random policy is greater than the ones using a more reasonable one such as the rule-based agent's policy (see Table I). As mentioned before, Settlers of Catan and other similar turn-based games include the end turn type that will always have a single option and it will always be available. In some situations, ending the turn might be desirable over other options (e.g. a player can build a road, but may want to keep its resources so it can build a settlement in the following turn). This small cardinality compared to the other types, means that the action is unlikely to be tried if other types are also available. Finally, the random policy is more likely to build a road over other pieces. We checked the end state of random games and the board is usually completely occupied with roads.

TABLE II. THE LIST OF ACTION TYPES DURING THE NORMAL PHASE OF THE GAME, THE PROBABILITY OF THE TYPE OF ACTION BEING AVAILABLE, THE AVERAGE NUMBER OF OPTIONS WHEN IT IS AVAILABLE AND THE MAXIMUM NUMBER OF OPTIONS. COLLECTED FROM 10K GAMES PLAYED USING THE STANDARD RANDOM OVER ALL OPTIONS POLICY.

Action type	Probability	Average	Maximum
Build road	0.0547	5.48	19
Build settlement	0.0331	2.82	13
Build city	0.0122	2.71	5
Buy development card	0.0114	1	1
Trade with opponent	0.9982	64.8	512
Trade with bank/port	0.2361	4.47	20
Play knight card	0.0001	16.16	29
Play monopoly card	0.0010	3.92	5
Play discovery card	0.0005	15	15
Play free road card	0.0023	6.35	34
End turn	1	1	1

A. Extending the Rollout policy

We now present the formal details. Let T be the set of action types and $t \in T$. a is an action option from the set A_t of options belonging to type t. n is a tree node and s is the corresponding game state. Algorithm 4 shows how the rollout policy can incorporate a step of selecting the action type based on a policy π_t then select the action description based on policy π_a. In the simple case, which is also what we evaluate in our experiments, these two policies select uniformly at random from the available options. This would be sufficient to address the concerns presented in the previous section. However, this model can easily be combined with an opponent model or a better sampling strategy if one is available, just by training a set of parameters that define the distribution over types and the distribution over action descriptions for a given type. Just as a Hierarchical Bayes model defines richer priors, this separation allows a more expressive opponent model to be implemented and permits more interesting combinations of Monte Carlo planning with data driven models. We will refer to the agent that implements the presented sampling strategy as TypedMCTS, while the standard MCTS method will be known as MCTS.[1]

```
$s \leftarrow n$.getState();
**while** $s$ *is non-terminal* **do**
    $T \leftarrow$ ListLegalActionTypes($s$);
    $t \sim \pi_t(s, T)$);
    $A_t \leftarrow$ ListLegalActionsOfType($s, t$);
    $a \sim \pi_a(s, A_t)$);
    $s \leftarrow$ performAction($s, a$);
**end**
**return** *reward for* $s$;
```

Algorithm 4: Extended ROLLOUT_POLICY

VI. Introducing Negotiations and Trades

We now describe how negotiations and trades are modelled and how the planning method interfaces with the real game. This is clearly a challenging task. Negotiations are composed of the actions of offer, reject and accept. When a player

[1] The code will be released at https://github.com/sorinMD/MCTS.

accepts another's offer, the exchange of resources is executed. We are not allowing our MCTS agent to make offers during an opponent's turn due to timing issues and the complexity of handling such behaviour. Instead it must choose between accepting or rejecting. However our results show that our agent with these limits still outperforms the Stac agent, which doesn't have these limits.

In addition to increasing the complexity of the search space, the planned offer may fail to result in a trade in the real game (the opponent can reject the offer). Despite MCTS's assumption that the opponents play optimally (i.e. they try to maximise their expected return), our evaluation section shows that a large number of offers get refused in the real game. The usual application of MCTS is to plan and select one best action to execute in the game, then repeat for each state. But here there is a risk of cyclic behaviour: the planning model may choose the same futile trade offer repeatedly.[2] A solution is to implement a tabu list that keeps track of the failed offers while replanning, but this would increase complexity. Replanning from the same state would replicate the same effort several times and would slow down the experiments. Our solution is to rank all available actions according to their value, which was already estimated in the first search, and try to execute these actions in decreasing order. The agent makes a new plan when an action is successful (indifferent of the action's type). If it receives a counter-offer to any of its offers, our agent treats it as a reject action to speed up the negotiations in the real game. If the received counter-offer exists in the list of evaluated actions, it will be considered by our agent only if nothing better is successful.

Counter-offers are completely ignored during the search. The list of offers is exhaustive, so any counter-offer will already be contained in this list. Therefore, this action would not bring any benefit aside from increasing the depth of the tree. In future work, we will explore this option again in combination with reasoning about player types.

As explained before, cyclic behaviour may appear during sampling: the same state may appear following a sequence of trade actions. This is especially problematic when the new nodes haven't been explored enough and uncertainty in their current value is high. Due to the tree policy of selecting the action with the highest UCT value, the current search could get stuck in an infinite loop. Again a tabu list or certain restrictions to disallow this cyclic behaviour seem like the obvious solution. However we have chosen a much simpler and cheaper alternative, which takes advantage of how the virtual loss is used during multi-threading. Instead of adding a virtual loss per thread, we increment a counter of virtual losses every time a thread enters a node. In time, this cyclic behaviour will be discouraged. This counter is reset during updates. To avoid impairing the UCT's ability to balance between exploration and exploitation, we only update a node once per rollout no matter how many times it was accessed during the rollout.

In addition to trades, the cyclic behaviour can be generated by other moves in the game and by the stochastic effect of certain actions. This creates a risk of generating a large tree containing only repeating sequences of actions. Furthermore, a state can be encountered via different paths in the tree. This is a characteristic encountered in many large complex games. To address these concerns we use a transposition table [8] to keep the tree small by sharing the information between nodes.

VII. Afterstates

The initial UCT selection criterion was proposed in [30] as an application of the Upper Confidence Bounds (UCB) algorithm to trees. UCB is used for selecting the next arm to pick in bandit based problems, therefore the natural translation would be to apply UCT to select the next action, instead of the child node. The literature has since branched out in two methods: the one presented in section IV and a method for selecting the actions as shown in equation 2 (e.g. [8], [3]). $Q(s,a)$ is the action value computed as the number of wins out of the number of plays when action a was chosen in state s, $N(s)$ is the number of times s was encountered before and $N(s,a)$ is the number of times action a was selected in state s. These two approaches would be equivalent if the nodes encountered in the tree are unique (i.e. the corresponding states are unique) and all the actions are deterministic. This is not the case in complex games and our test environment is no exception.

$$UCT_a = Q(s,a) + C\sqrt{\frac{2\ln N(s)}{N(s,a)}} \tag{2}$$

As mentioned before, a state can be reached by executing different sequences of actions. Moreover, executing different actions in different states could result in the same outcome state. The outcome state is known in the literature as an *afterstate* [1] or a *post-decision state* [31] (see Figure 1 for a simple illustration). These states account for non-deterministic actions since they represent states immediately after a decision was made but right before any of the stochastic effects are performed. In addition to separating the deterministic effect from the stochastic effect of an action, the state space (as well as the afterstate space) is much smaller than that of the state-action pairs. Therefore using the value of the states could be more economical and efficient than using the action value [31]. Another benefit of afterstates is that different state-action pairs produce the same outcome state, so their value must be the same [1]. By using the afterstate value in the computation as in Equation 1, the results of the rollouts following either of the two pairs are shared.

The best model presented in this paper uses the UCT computation as presented in Equation 1 in combination with chance nodes when actions are non-deterministic, thus being equivalent to defining post-decision states. An empirical comparison to the one presented in Equation 2 is done in the final results subsection.

VIII. Results

A. General performance

We will briefly mention the performance of the multi-threaded version of the agent on a server with Intel Xeon quad-core CPUs at 3GHz frequency. Our model requires a single CPU and less than 4GB of RAM. Multiple identical CPUs are

[2] This issue could potentially be avoided with an accurate opponent model, if available.

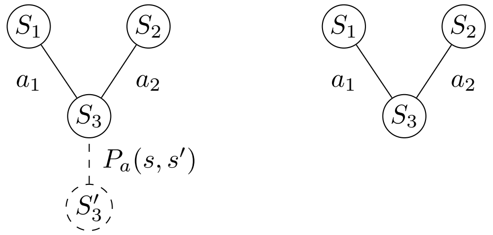

Fig. 1. Simple example of afterstates or post-decision states. The first graph from the left accounts for the stochastic effect or other unknown effects of the environment (including the opponent model). Node S_3 from the first graph is the state prior to the stochastic effect of the action. Two different non-deterministic actions executed in different states, could have the same afterstate if their set of successor states and transition probabilities are the same. The second graph shows the simplified case when the actions are deterministic.

used when the number of threads is over 4. Table III shows the effects of introducing the second step in the sampling policy on the depth and time of the random rollouts. Despite the extra sampling step, the gains are massive and the times presented in Table IV would be tolerated by human opponents. As expected, the more iterations are run, the more useful it becomes to increase the number of threads. We didn't evaluate the effect of the number of threads on the agent's win rate, so we fixed this number to 4 for our MCTS agents. In the interest of space we will not present the time required for running the full MCTS algorithm with the default rollout policy. To illustrate the differences, a game between 4 players, where one is the standard (untyped) MCTS agent and 3 are Stac, finishes in 17 minutes. In comparison, a game with one TypedMCTS agent and 3 Stac agents finishes in under 3 minutes.

Note that running 10k of the typed MCTS algorithm is cheaper than just running 10k of random rollouts. We believe this is caused by the focused approach of the tree selection policy in combination with the importance of the free initial placement stage. As a last general experiment, we have tuned the exploration parameter C and noticed that a value of 1 is better than any higher values.

TABLE III. COMPARING THE TYPED SAMPLING AGAINST THE SINGLE STEP SAMPLING OVER 10K CATAN GAMES. THE TIME SPENT IS IN MILLISECONDS. BOTH METHODS ARE SINGLE-THREADED.

Policy	time	depth
Default	583384	11639
Typed	8530	420

TABLE IV. TIME IN MILLISECONDS REQUIRED FOR THE MCTS ALGORITHM WITH TYPED SAMPLING OVER 100 CATAN GAMES.

Rollouts	Threads			
	1	4	8	16
10k	5252	1717	1187	1192
30k	23021	5615	4046	3539
40k	30289	7170	5787	4943
50k	31736	8636	7242	6394

B. Agent Evaluation

We are evaluating our agents in the JSettlers environment; similar to the approach in [21]. The performance of an agent is measured by running simulations of 2000 games between 4 players: one of the players is the (modified) agent we are evaluating and the other 3 are baseline agents. So, a player that is of equal strength to the baseline agent would win 25% of the games. We tested the significance of win rates against this null hypothesis using the z-test and a threshold $p < 0.01$. This makes any win rate between 22.5-27.5% not significantly different from the null hypothesis (i.e. a win rate of 25%). Since all results presented below are significant, we will not include the z scores in the tables.

Table V contains the performance of the TypedMCTS agent against 3 (state of the art, hand crafted, rule-based) Stac agents when varying the number of rollouts. If trades between players were not allowed, the performance of the TypedMCTS agent would cap at 43% with 30k rollouts. With trades, the game is much more complex and we can still observe an increase even at 50k rollouts, but at the expense of increasing the decision time. For the remainder of the experiments we fixed the number of rollouts of our MCTS agents to 10k, unless otherwise specified.

TABLE V. WIN RATES OF THE TYPEDMCTS AGENT AGAINST 3 STAC AGENTS, GIVEN THE NUMBER OF ROLLOUTS THE TYPEDMCTS AGENT CAN PERFORM. TRADES BETWEEN PLAYERS ARE ALLOWED.

5k	10k	20k	30k	50k
22.2%	33.8%	45.65%	51.27%	53.37%

We have observed that the MCTS agents make on average a large number of offers and this number is only slightly reduced as the number of rollouts is increased. Only 15% of the offers get accepted by the opponents. Since such a behaviour may seem annoying to a human opponent, we introduced a limit on the number of offers allowed before a different action type must be executed. Table VI shows the performance and number of offers a TypedMCTS agent makes when it plays against 3 Stac agents. Since the planning method is not aware of this limit, we expected the performance of the agent to be reduced. However this was only the case when less than 5 trades are allowed before a different move must be made. Despite making half the number of offers when compared to the unlimited agent, the agent that is limited to 10 trades had an increased win percentage. This result could indicate several things. One of them is that agents, like humans, miscalculate the equilibria during negotiations. Secondly, it is likely that the number of rollouts is insufficient to adequately differentiate between trade actions and the other options. Otherwise, we believe the agent would make fewer offers before executing a different action type. Looking at the games collected in the corpus by [24], a decent human player makes on average far fewer offers compared to novice players. Another possible cause for the reduced performance when the number of offers are unlimited could be that the Stac opponents may benefit from our agent's eagerness to trade. Even though the Stac agents use rule-based policies and would only accept trades that are relevant to their current plan, proposing so many exchanges increases the chances of making one that is relevant to their plan. We believe a human player would be able to exploit the unlimited agent's behaviour, so we intend to explore introducing this limit into the planning algorithm in a future version of the agent.

TABLE VI. WIN RATES AND AVERAGE NUMBER OF OFFERS PER GAME OF THE TYPEDMCTS AGENT GIVEN THE NUMBER OF OFFERS IT IS ALLOWED TO MAKE BEFORE EXECUTING A DIFFERENT ACTION TYPE.

Offers allowed	3	5	10	unlimited
Win Rate	30.45%	33.8%	36.1%	33.8%
No. offers	93	148	258	527

Table VII shows the performance of our TypedMCTS agent against the two current state of the art agents, Stac and SmartSettlers. The table contains the performance of the modified agents specified on the first column of the table. To clarify, TypedMCTS wins 33.8% of the games versus 3 Stac agents, while one Stac agent wins only 15.18% of the games against 3 TypedMCTS. Evaluating the agent both ways ensures against the possibility of the modified agent winning due to having a different policy to the three baseline agents. To level the playing field between the two planning agents, we wanted to limit the time each agent can take to deliberate before executing an action. Unfortunately, SmartSettlers doesn't contain an option to set this limit, so we had to limit the number of rollouts instead. 10k rollouts for our agent takes approximately the same amount of time as 2k rollouts for the SmartSettlers one.

TABLE VII. PERFORMANCE OF THE TYPEDMCTS AGENT WITH 10K ROLLOUTS AGAINST THE 2 STATE OF THE ART AGENTS: STAC AND SMARTSETTLERS. TRADING IS ALLOWED IN THE GAMES AGAINST STAC, BUT NOT IN THE GAMES AGAINST SMARTSETTLERS.

Modified	Baseline		
	Stac	SmartSettlers	TypedMCTS
Stac	–	–	15.18%
SmartSettlers	–	–	18.67%
TypedMCTS	33.8%	36.39%	–

The final set of experiments presented in this section will evaluate the benefits of the action type hierarchy over the vanilla MCTS method. Due to the expensive planning of the standard MCTS method, we run games only against our proposed model and the Stac agent. Also, the legal trades options have been reduced to only 1 for 1 exchanges for this experiment. Otherwise, the standard MCTS method would take to long to finish the search, given the time required to perform random rollouts as shown in Table III. We have performed this experiment twice: once when the rollout was fixed to 10k for both planning methods (Table VIII); and the second where a budget limit of 1.5 seconds up to a maximum of 50k rollouts was introduced (Table IX). As before, the modified agents are specified on the first column and the corresponding baseline on the second row.

TABLE VIII. PERFORMANCE OF THE MCTS AGENTS WITH AND WITHOUT THE TYPE CATEGORISATION AND A LIMIT OF 10K ROLLOUTS.

Modified	Baseline		
	Stac	MCTS	TypedMCTS
MCTS	22.34%	–	9.58%
TypedMCTS	33.8%	54.3%	–

TABLE IX. PERFORMANCE OF THE MCTS AGENTS WITH AND WITHOUT THE TYPE CATEGORISATION AND A LIMIT OF 1.5 SECONDS (UP TO A MAXIMUM OF 50K ROLLOUTS).

Modified	Baseline		
	Stac	MCTS	TypedMCTS
MCTS	6.55%	–	0.81%
TypedMCTS	27.98%	88.57%	–

C. Afterstates

All our MCTS agents so far compute the UCT value using afterstates, i.e. Equation 1. We now evaluate an agent that doesn't use afterstates in computing the UCT value, i.e. Equation 2. We will refer to the agent that employs the latter as *TypedMCTS-NA* and we compare it against the best MCTS agent presented in the previous sections, i.e. *TypedMCTS*. As before, the Stac agents were used as a baseline and the two variations are also pitched against each other. Trades are allowed in this experiment, since all agents can handle these actions. We have tuned the exploration parameter of the *TypedMCTS-NA* agent and observed that the best value for this agent is 2. Table X outlines the large benefit of afterstates, as the TypedMCTS-NA agent doesn't perform significantly different to the Stac agents and is much weaker than the TypedMCTS. This result backs up the hypothesis that using the value of the outcome state during UCT calculations can reduce the complexity of the space.

TABLE X. PERFORMANCE OF THE MCTS ALGORITHM WITH AND WITHOUT AFTERSTATES. BOTH MCTS METHODS PERFORMED 10K ROLLOUTS PER DECISION AND USED THE TYPED ROLLOUT POLICY.

Modified	Baseline		
	Stac	TypedMCTS-NA	TypedMCTS
TypedMCTS-NA	9.23%	–	3.42%
TypedMCTS	33.8%	71%	–

IX. POSSIBLE EXTENSIONS

In addition to various improvements mentioned throughout the paper, one possible extension is to integrate the action type category in the tree policy as shown in Algorithm 5. π_t^* can either be a selection policy based on an opponent model or could employ the standard selection based on the UCT value. If the latter is chosen, the UCT value of a type can be computed by selecting the maximum from the options belonging to the corresponding type, resulting in an algorithm equivalent to the standard single step selection. Alternatively, a discounted reward or an average of the values of all the options belonging to the action type can be employed. Such a selection strategy gives precedence to choosing the action type over choosing the action description and the performance may depend on the environment description. The expansion part of MCTS may also need to add the type nodes to the tree. This approach is very similar to the standard move groups [8], except that the groups are chosen based on the type of action and therefore are disjoint.

```
while $n$ is non-terminal do
    if $n$ is not expanded then
        return EXPAND($n$);
    else
        $s \leftarrow n$.getState();
        $T \leftarrow$ ListLegalActionTypes($s$);
        $t \sim \pi_t^*(s, T)$);
        $A_t \leftarrow$ ListLegalActionsOfType($s, t$);
        $a \leftarrow$ SELECT_UCT($s, A_t$);
        $s \leftarrow$ performAction($s, a$);
        $n \leftarrow$ Tree.getNode($s$);
    end
end
return $n$;
```

Algorithm 5: Extended TREE_POLICY

X. CONCLUSION

We have presented a method of enhancing the MCTS framework with action type categories to yield the state of the art player for a very complex game. The results show that an agent based on this method is stronger than the existing state of the art implementations and an equivalent MCTS process without action types. Furthermore, the proposed approach increases the efficiency of the planning method such that the decision time is cut to a reasonable threshold for human opponents. The technique presented in this paper can be easily used in any game that presents similar characteristics. As future work, we intend to evaluate the presented method on other complex games.

ACKNOWLEDGMENTS

We thank the reviewers for their helpful suggestions. This work is supported by ERC grant 269427 (STAC) and Engineering and Physical Sciences Research Council (EPSRC).

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement learning: An introduction.* Cambridge University Press, 1998.

[2] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *In: Proceedings Computers and Games 2006*. Springer-Verlag, 2006.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[4] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, March 2012.

[5] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res. (JAIR)*, vol. 13, p. 227303, 2000.

[6] N. A. Vien and M. Toussaint, "Hierarchical monte-carlo planning," in *Proc. of The Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 15)*, 2015.

[7] R. He, E. Brunskill, and N. Roy, "Puma: Planning under uncertainty with macro-actions." in *AAAI*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[8] B. E. Childs, J. H. Brodeur, and L. Kocsis, "Transpositions and move groups in monte carlo tree search." in *CIG*, P. Hingston and L. Barone, Eds. IEEE, 2008, pp. 389–395.

[9] N. Jouandeau and T. Cazenave, "Monte-carlo tree reductions for stochastic games," in *Technologies and Applications of Artificial Intelligence, 19th International Conference, TAAI 2014, Taipei, Taiwan, November 21-23, 2014. Proceedings*, 2014, pp. 228–238.

[10] ——, "Small and large MCTS playouts applied to chinese dark chess stochastic game," in *Computer Games - Third Workshop on Computer Games, CGW 2014, Held in Conjunction with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18, 2014, Revised Selected Papers*, 2014, pp. 78–89.

[11] J. Kloetzer, H. Iida, and B. Bouzy, "The monte-carlo approach in amazons," in *Proc. Comput. Games Workshop,Amsterdam, Netherlands*, 2007, pp. 113–124.

[12] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search." *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.

[13] D. Silver and J. Veness, "Monte-carlo planning in large pomdps." in *NIPS*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2164–2172.

[14] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[15] J. Deng, O. Russakovsky, J. Krause, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Scalable Multi-Label Annotation," in *CHI*, 2014.

[16] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering." *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 4, no. 4, pp. 241–257, 2012.

[17] M. Dobre and A. Lascarides, "Online learning and mining human play in complex games," in *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*, Tainan, Taiwan, 2015.

[18] S. Branavan, D. Silver, and R. Barzilay, "Learning to win by reading manuals in a monte-carlo framework," *Journal of Artificial Intelligence Research*, vol. 43, pp. 661–704, 2012.

[19] G. Chaslot, C. Fiter, J.-B. Hoock, A. Rimmel, and O. Teytaud, "Adding expert knowledge and exploration in monte-carlo tree search," in *Proceedings of the 12th International Conference on Advances in Computer Games*, ser. ACG'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–13.

[20] R. Thomas, "Real-time decision making for adversarial environments using a plan-based heuristic," Ph.D. dissertation, Department of Computer Science, Northwestern University, 2004.

[21] M. Guhe and A. Lascarides, "Game strategies in *the settlers of catan*," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, Dortmund, 2014.

[22] M. Pfeiffer, "Machine learning applications in computer games," Master's thesis, Institute for Theoretical Computer Science, Graz University of Technology, 2003.

[23] I. Szita, G. Chaslot, and P. Spronck, "Monte-carlo tree search in settlers of catan," in *Advances in Computer Games*, H. van den Herik and P. Spronck, Eds. Springer, 2010, pp. 21–32.

[24] S. Afantenos, N. Asher, F. Benamara, A. Cadilhac, C. Degremont, P. Denis, M. Guhe, S. Keizer, A. Lascarides, P. M. Oliver Lemon, S. Paul, V. Rieser, and L. Vieu, "Developing a corpus of strategic conversation in the settlers of catan," in *Proceedings of the 1st Workshop on Games and NLP*, Kanazawa, Japan, 2012.

[25] M. Guhe, A. Lascarides, K. O'Connor, and V. Rieser, "Effects of belief and memory on strategic negotiation," in *Proceedings of the 17th Workshop on the Semantics and Pragmatics of Dialogue (DialDam)*, Amsterdam, 2013.

[26] N. R. Sturtevant, "An analysis of uct in multi-player games." *ICGA Journal*, vol. 31, no. 4, pp. 195–208, 2008.

[27] R.-K. Balla and A. Fern, "Uct for tactical assault planning in real-time strategy games," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 40–45.

[28] G. Chaslot, M. H. M. Winands, and H. J. van den Herik, "Parallel monte-carlo tree search." in *Computers and Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, Eds., vol. 5131. Springer, 2008, pp. 60–71.

[29] M. Dobre and A. Lascarides, "Combining a mixture of experts with transfer learning in complex games," in *Proceedings of the AAAI Spring Symposium: Learning from Observation of Humans*, Stanford, USA, 2017.

[30] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proceedings of the 17th European Conference on Machine Learning*, ser. ECML'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 282–293.

[31] C. Szepesvári, *Algorithms for Reinforcement Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.