Online learning and mining human play in complex games

Mihai Sorin Dobre, Alex Lascarides
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB
Scotland
Email: m.s.dobre@sms.ed.ac.uk, alex@inf.ed.ac.uk

***Abstract*—We propose a hybrid model for automatically acquiring a policy for a complex game, which combines online learning with mining knowledge from a corpus of human game play. Our hypothesis is that a player that learns its policies by combining (online) exploration with biases towards human behaviour that's attested in a corpus of humans playing the game will outperform any agent that uses only one of the knowledge sources. During game play, the agent extracts similar moves made by players in the corpus in similar situations, and approximates their utility alongside other possible options by performing simulations from its current state. We implement and assess our model in an agent playing the complex win-lose board game *Settlers of Catan*, which lacks an implementation that would challenge a human expert. The results from the preliminary set of experiments illustrate the potential of such a joint model.**

I. Introduction

This paper is concerned with the problem of finding optimal policies for complex problems that lack any analytic solution. Our approach is to learn from a combination of the agent's own exploration of the game space and from human play. We deploy online learning strategies of [1], [2], specifically we adapt Monte Carlo Tree Search (MCTS) so that it reasons simultaneously from the data generated via simulation and from human corpus data. We aim to show that it is useful to combine online learning with knowledge of what humans did in similar states, and that such a model will outperform any model that uses only one of these information sources.

Learning optimal policies by using reinforcement learning techniques is well established, and in simple games these approaches can find an exact solution [3]. Reinforcement learning has also been useful for learning strategies in domains where the environment is dynamic and partially observable. But for complex games, where the state space is massive, it is impractical to run the number of simulations one needs to derive reliable policies for every state one might encounter. A popular alternative is to use online methods to estimate the expected utility of each of the possible actions (e.g. [4]). However, neither of these or a combination of the two have ever reached the level of an expert human player in complex games, without extraneous methods for choosing among high-level strategies such as hand-written heuristics (e.g. [5]).

The traditional learning methods are rarely trained on general human play. Yet, many agents created to date are the result of extensively analysing what a human would do when faced with the same set of options. There are many techniques for using human knowledge, including: supervised learning through human feedback [6], modelling human opponents to guide the sampling [7], offline learning of a policy from expert play [8], inverse reinforcement learning from human demonstration [9] and writing hand-crafted heuristics based on expert advice [10] (this is also used to inform different search algorithms [5]). These have generally increased the level of play of the deployed agent, but either require a lot of development work or are expensive to run. Furthermore, some of these methods need to create abstractions over the complex game so as to reason with a simpler game [11] and do not take advantage of why moves are made from specific states. In this research we aim to address these two parts of the problem together in such a manner that a solution to one would benefit the other and vice-versa.

Games have been used to model human behaviour in specific scenarios that relate to many daily activities (e.g. logistics, planning or negotiations). The interaction between humans and machines has also received increased attention, especially in the realm of human robot interaction. In addition, there is significant research in creating programs that would have a more human–like approach to solving decision problems in complex environments (e.g. Bot Turing Test the CIG Bot 2K Competition [12]). Despite the large volumes of game data available containing human play (e.g. network games [4]), the state of the art methods are still combined with a large number of hand-coded heuristics in order to raise the level of play of AI agents.

II. Related Work

A. *Work on Settlers of Catan*

Prior work on modelling agents that play Settlers of Catan varies on the extent to which they rely on hand-coded heuristics vs. machine learning, on how large a portion of the game they aim to model, and on the empirical data that informs their approach. The agent released with the JSettlers platform developed by [10] doesn't use any machine learning techniques. Instead it forms a plan based on an estimate as to how quickly the player reaches 10 victory points (and so wins the game). This estimate is computed from hand-coded heuristics. Another approach treats the problem from a multi-agent perspective, separating the original JSettlers heuristics into different agents, each handling different aspects of the game [13].

Pfeiffer [5] presented a method that combines low-level learning mechanisms with hand-crafted, symbolic, high-level heuristics. The heuristics are intended to reflect a prior knowledge of effective moves that a human expert player would choose. The low-level action choices for achieving the higher level (intermediate) goals is then learned via reinforcement learning. The author reported that the system could develop strategies and methods, while winning against some human players, and also observed that the hand-coded heuristics for choosing the high-level actions were critical to success. On the other hand, [14] applied MCTS with only a limited amount of domain knowledge on a simplified version of the game (i.e. it removes the elements of imperfect information and the agent's ability to trade with other players). Their results show a considerable increase in its playing strength compared to the existing heuristics-based strategies for the game. Roelofs [15] wrote another MCTS implementation on the same simplified version of the game and also reported an increase in performance compared with the heuristics-based agent. We intend to refine this work, enhancing the performance of online learning by combining its exploration techniques with automatically acquired knowledge of what a human would do.

B. *Similar models for learning in complex games*

Silver et al. [2] describe a hybrid implementation of an offline (learning) and an online (searching) method that increases the performance of a Go player. In this sense, the agent has a permanent and a transient memory, where the memory is a set of features and parameters used to estimate Q-values of different state–action pairs ($Q(s, a)$). The permanent memory is learnt offline during millions of simulated games and it is only modified over many moves and games. The permanent Q-value is used as a proxy for domain knowledge during the searching part of the algorithm, which in turn learns a local correction to the function. This modification is used to update the permanent memory and is erased after the action that maximises it is executed in the real game. Overall, the agent performed better than an Upper Confidence bounds for Trees (UCT) based approach and it outperformed all previous methods when combined with an alpha–beta heuristic search. This technique is successful because, while GO is complex, it is sufficiently simple for Q-learning to produce decent policies. But Settlers of Catan is more complex than GO, with more options on average and more non-determinism. Consequently, Pfeiffer [5] has shown that a pure Q-learning strategy for Settlers produces very poor policies, which fail to beat simple baselines.

The game of GO is a very popular research framework and some methods incorporate knowledge from records of expert players. Sutskever and Nair [7] train a Convolutional Neural Network over professional games in order to predict how experts play the game, beating the state of the art. Other approaches [16] use a complex combination of online learning, transient learning, expert knowledge and patterns learned offline. Gelly et al. [17] present methods for combining offline knowledge with online knowledge and test their performance on the 9×9 Go domain. Their strongest player integrates prior knowledge learnt offline by the RLGO program [18]: a value function using a linear combination of binary features is learnt offline via self-play and is used to bias the initial search of the UCT algorithm. Neither authors train their models on human data, but use expert games or self-play instead. Even though learning from expert play has proven to be the most effective [19], it is still not guaranteed that all the moves contained are optimal. A method which can filter the moves should perform better when applied on any of the options: self-play, expert play or mixed human play. Furthermore, it would eliminate the need for selecting the optimal moves before the learning process.

The model presented in [1] improves the performance of an agent that plays Civilisation 2 on the basis of Monte-Carlo Search, by extracting from a manual the domain knowledge that is relevant to the current game context and biasing the player's action selection policy according to the manual's recommended actions in those relevant states. This is done in an online fashion by estimating the model's parameters using feedback from the Monte-Carlo simulations performed. The manual can be considered a permanent memory, which is completely trustworthy and never modified (i.e. it is reasonable to assume that advice in an expert manual is good advice). Following the given advice, the search algorithm is biased towards the most similar action that can be taken from the current game context. But a manual gives sparse advice—many game states will bear no or very little relevance to any of the pieces of advice given in the manual. Also, some games do not have such detailed manuals, or if they do, they contain general advice intended for beginners. In contrast, our approach will mine knowledge from a corpus of actual moves expert human players choose in particular states.

III. Experimental Resources

A. *Settlers of Catan*

The domain in which we test our model is Settlers of Catan (for a more detailed set of rules see www.catan.com). This is a multi-player (2–6 players) board game which has won several major awards and has been used as a platform in previous research [10], [14], [15], [20]. This research will focus on the core game for 4 players. It is a win–lose game: 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). The players build roads, settlements and cities on an island represented by the board formed of hexagonal tiles like the one presented in Figure 1. The tiles represent one of the five resources (Clay, Ore, Sheep, Wheat and Wood), desert, water or ports. Each of the resource producing tiles have an associated number between 2 and 12. Players obtain resources via the positions their pieces occupy on the board and dice rolls that start each turn of the game, so the outcomes of specific actions are non-deterministic. 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 at the port (3:1 or 2:1 ratio depending on the port) or through negotiated trades with other players.

The game starts with each player taking turns to place 2 free settlements and 2 free roads. The starting player places one settlement and one adjoining road, then the other players, in a clockwise direction, do the same. The last player also places the second settlement and road, then everyone follows counter-clockwise until the first player has placed the second settlement and road. Each player receives the starting resources based on the hex types adjacent to their second settlement: one for each hex up to a maximum of 3 resources. The method presented in this paper is evaluated on the placement of the second settlement during this set-up period. Our motivation for choosing this phase of the game is the fact that it is a decisive stage. Any player that fails to select good locations on the board will receive fewer resources throughout the game and, as a consequence, will find it difficult to expand to other good locations before the other players. The order in which players place their pieces also has an effect, as better locations are occupied first [14].

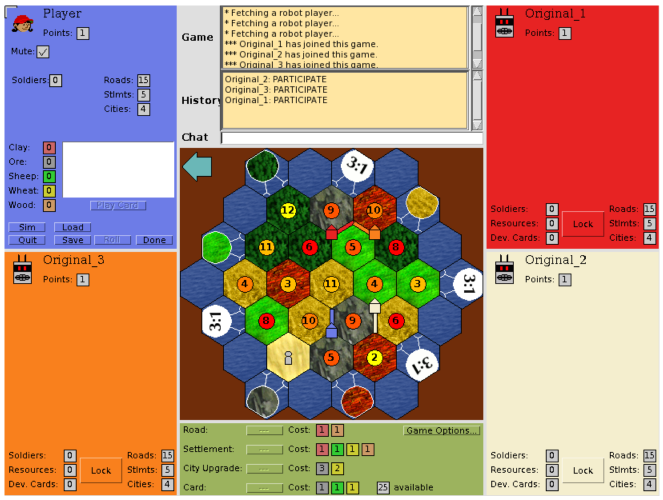
The game contains many special actions which increase the complexity of the game. On a dice roll of 7, any player in possession of more than 7 resources must discard half of them and the current player moves the robber, thus blocking the resource production of the chosen tile. Moving the robber also allows the current player to steal one resource at random from one of the players with a settlement or city adjacent to the chosen location. The unaffected agents cannot see what type of resources were discarded or stolen, so game states are partially observable. During a normal turn, the current player can buy a development card with 1 wheat, 1 sheep and 1 ore. The cards are: 14 knight cards (the owner can move the robber and steal one resource), 5 victory points cards (one victory point for each of them), 2 road building cards (build 2 roads for free), 2 year of plenty cards (get 2 resources of choice) and 2 monopoly cards (when played, the owner receives all the resources of specific type held by everyone else). Any number of cards can be drawn from the deck in exchange for the required resources; however only a single card can be played per turn. Finally, two awards, each having a value of 2 victory points, are given to either the player with the current longest road (i.e. at least 5 road pieces linked together) or the player with the current largest army (i.e. at least 3 knight cards played).

From a game theoretic perspective, Settlers of Catan is a very complex game. It is symmetric as players have the same goal and sequential as each player takes turns to execute actions. However, the fact that other agents can make offers to the current player breaks the turn based pattern. The players are in a race to achieve their goal, but coalitions are very likely to happen as the game progresses. In addition, the game contains elements of imperfect information (not just the opponents' player types, but also the resources that opponents possess, as well as the unplayed development cards) and it is stochastic (dice rolls decide the generation of resources and could decide the following actions).

Due to the large branching factor of some of the actions of the game (rolling dice, moving the robber and discarding) and given the negotiation actions one must employ to gain the necessary resources to achieve one's goal, the game tree has an exponential growth in time making any learning methods that are based only on random walk infeasible. In addition to this, the generation of the board is done by shuffling the 19 land hexes and the 9 port hexes, so the game has a huge state space for its possible initial states. As far as we are aware, there is no fixed solution or always winning strategy, so a player must adapt to the current game configuration and employ a combination of strategies in order to be successful.

B. JSettlers

The starting point for conducting our experiments is an open source implementation called JSettlers (www.jsettlers2.sourceforge.net, [10]). JSettlers is a client–server system: a server maintains the game state and passes messages between each of the players' clients. Clients can be human players or computer agents. The game interface for human players (including the board and each of the player's information) is shown in Figure 1. The JSettlers agent is heuristics based, deploying only hand-crafted rules. We will call the agent that comes with the above open source package, the *original* agent. Guhe and Lascarides [20] have developed an improved symbolic agent which beats the *original* agent. Their improvements include a modified initial placement strategy and a different preference function for choosing the next action; the new agent has an increase of approximately 10% in its number of wins when playing against original agents. We will refer to it as the *ranking* agent.



```
* Fetching a robot player...
* Fetching a robot player...
* Fetching a robot player...
*** Original_1 has joined this game.
*** Original_2 has joined this game.
*** Original_3 has joined this game.
```
```
Original_2: PARTICIPATE
Original_3: PARTICIPATE
Original_1: PARTICIPATE
```

Fig. 1. JSettlers game interface

C. Game rules modifications

Previous implementations [14], [15] simplify the game by removing the elements of imperfect information (such as a lack of knowledge of player resources and unplayed development cards) and do not allow trades among players. As shown in [21], limiting the possibility of a player to trade with others handicaps it, as trading with the bank has a poor trade ratio, while negotiations among players would most often result in a 1 to 1 trade from our observation of the trades executed in our database [22]. Furthermore, changing the visibility of each players' resources may have an unwanted effect on negotiations, as [21] have shown that the belief and memory models affect both the number and the quality of trade offers.

Information such as these human preferences for trading with each other rather than with the bank would not be so useful for improving the agent's decision making in a simplified version of the game. For instance, the game branching factor is drastically reduced by performing the two mentioned simplifications. In such a scenario, Monte Carlo Tree Search would be able to search the game tree in greater depth and execute multiple visits per node at lower depths, resulting in a more accurate estimate of expected utilities from a smaller number of simulations. [14] and [15] prune the game tree by simplifying the game itself—no trades with other players and perfect information. We would like to evaluate the performance of our models on the full game, which contains the full spectrum of strategies.

D. Human Corpus

Our human data is taken from a corpus of humans playing Settlers in a tournament that ran for three seasons [22]. Two of the seasons are split into different leagues based on the participants' previous experience and level of play. In these leagues each player plays multiple games and a detailed ranking system is produced. This gives us the possibility to decide on the confidence in certain players given their performance in the leagues, in addition to their performance during a specific game. In other words, these can be a set of informative features for estimating the utility of an action attested in the corpus. One of the seasons contains 21 games in which players of different levels are mixed together, with each player playing a single game. The corpus consists of 59 games played on different board layouts and containing different number of players (i.e. 2, 3 or 4 players), from a total of 100 participants of mixed experience out of which 75% were male, 25% were female, the average age was 27 (minimum 19, maximum 54). Overall, this corpus ensures the data we have collected contains a large variety of scenarios with a total of approximately 14000 state–action pairs. Despite this, the data is very sparse, and it is highly unlikely that an exact state that is attested in the corpus will be encountered in a future game.

IV. Implementation

Our main objective was to show that a model that uses information extracted from a corpus of human play would have an increased performance over a model that doesn't exploit human data and that we can use human knowledge to directly guide the play of an agent in an online fashion. Moreover, we want to show that relying only on the information extracted from the human corpus is not effective: one needs the agent to perform an online exploration of his current decision problem to overcome the sparse data the corpus provides.

The results in [14] show that Settlers of Catan is a very complex game and that online methods that rely completely on random walk to search the space fail to achieve reasonable play. They succeed only if sufficient time for running simulations is provided and the game's rules are simplified, thus reducing the branching factor of the tree. Given that we will be evaluating our model against a stronger baseline and that we do not want to simplify the game, we fear that running 1000 random simulations (which results in an agent of equal strength to the baseline in [14]) is not enough to defeat our baseline heuristic agent or even generate a decent player to play the full game. Due to the complexity of the game and the fact that we are using a corpus of mixed play, we believe that this small number of simulations will not be enough to filter the noise and prune the bad moves extracted from the corpus. Furthermore, we aim to show that a method which makes use of past play to bias the online learning will reduce the need of running a large number of simulations, which are expensive. But it is impractical to increase the number of simulations to 10000, which might generate an agent able to defeat the baseline and build our model on top of this agent. Such a player could never give a human opponent a satisfying experience, because it would take too long to decide on each move it makes. We were unable to find the exact number of simulations that would result in a player of similar strength as our baseline in the given time frame. Therefore, we decided to implement a more controlled experiment to evaluate the benefits of mining the human corpus, but one that does not sacrifice the complexity of the game.

As mentioned before, we have chosen to verify our hypothesis on one of the most important actions of the game namely, the placement of the second settlement during the initial set-up phase. All agents use the same policy for the remainder of the real game, so we can assess the utility of just this move following different policies. This approach also permits testing the performance on the complete unmodified game, as the heuristic agent handles all aspects of the game (including trading and imperfect information), while during the initial placement set-up everything is observable and we do not need to modify the search algorithms.

A. 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. The algorithm can be separated into four steps (see Figure 2): selection (starting from the root that represents the agent's current state and select the nodes based on a policy until a leaf node is reached), expansion (one or more children are added to the tree), simulation (from the new node the game is played following a simulation policy) and backpropagation (the result is used to update the expected utility of every chosen node in the tree).

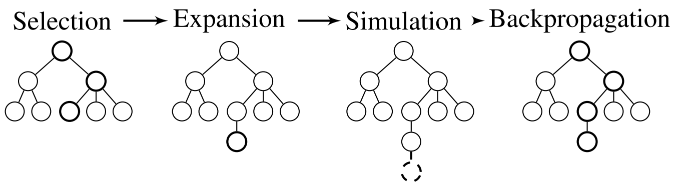


Fig. 2. MCTS algorithm; Repeat the steps n times

UCT is a very successful algorithm for modelling MCTS as it has been shown to have a better action selection procedure than ϵ-greedy methods (in which exploration is achieved choosing uniformly at random from the available options) and overall better performance [23], [24]. The UCT formula is:

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

where $\overline{X_j}$ is the value of the node represented as number of wins out of the number of plays selected via this node, n the number of times the parent node was tried, n_j the number of times the child node was tried and C is a constant used to decide on the level of exploration.

MCTS in its simplest form has been successfully applied on games of perfect information, whereas Settlers of Catan contains hidden information, in which the agent must reason about the relative likelihood of the hidden values to make an optimal decision. Previous implementations simplify the game domain as described before in order to be able to apply the algorithm as it is. But we want to evaluate the performance of our model on the full game, which contains the full spectrum of strategies. The alternative is to adapt the MCTS algorithm by extending it with: determinization (of hidden information) [25] or Information Set Monte-Carlo Tree Search [26]. For now we apply our approach in one of the fully observable stages of the game—namely the initial placement—and we will implement one of the extensions to the MCTS algorithm as future work.

B. Flat Monte-Carlo

As we have chosen to test only on one action of the game, we want for now to avoid having MCTS (see Figure 2) selecting actions based on how good a specific part of a branch continuing from the current node is. We have decided to remove the expansion part of the algorithm turning this into a similar approach as Flat Monte-Carlo (FMC) method [4], which samples each action uniformly at random. FMC has proven to be very strong in the domains of Bridge [27] and Scrabble [28]; these are however much simpler than Settlers. We want to keep the benefits of UCT, so our method will treat the leaves of the tree as a multi-armed bandit. Such an approach is called *Flat Upper Confidence Bounds* (*Flat-UCB*) [4] (see Figure 3).

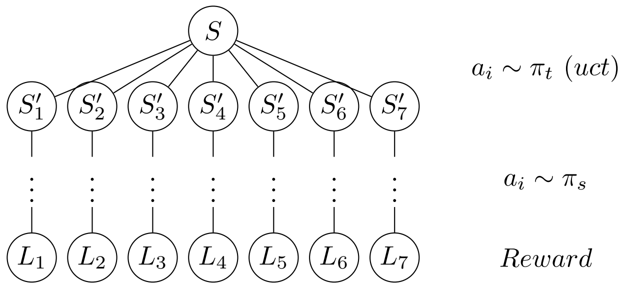


Fig. 3. Flat-UCB algorithm

FMC has been shown to be a weak approach as it does not allow for opponent modelling [29]. Flat-UCB with random simulations has the same issue, because UCT is only used in the tree level (i.e. only from the current player's perspective), while the other players take only random decisions during simulations. To overcome this, we are simulating following an ϵ-greedy policy π_t for all the players (with $\epsilon = 0.2$), in which an action is chosen based on the *ranking* heuristic in $1 - \epsilon$ proportion of time and uniformly at random in the remaining ϵ proportion. Whitehouse et al. [30] have shown that existing heuristics can be re-purposed inside MCTS to retain the personality of the original agent and potentially create a stronger player. In addition, an ϵ-greedy policy will also allow the model to explore the game space in more detail as the *ranking* agent's decisions are deterministic.

Finally we have chosen a fixed number of 1000 roll-outs for the Flat-UCB method, with each roll-out terminating at an end state of the game (i.e. the state where the first player acquires 10 victory points). The action selected to be played in the real game is the one that maximises $\overline{X_j}$ in the UCT formula, with $C = 1$.

C. Assessing relevance

Linear function approximations can give good results in large or continuous domains where the state space is too large for performing the required simulations [2]. Via this process called *generalisation*, reinforcement learning methods learn to take reasonable decisions to similar states to the ones already tried [11]. To perform such approximations, one first needs to transform the game data into a different representation, e.g. feature vectors. This pre-processing has proven to be a very important step in the performance of the final systems [31]. It involves feature selection and feature extraction. The former is the process of creating a subset of features from the raw data set by eliminating any irrelevant or redundant ones. Feature extraction is a special form of dimensionality reduction and could create new features from an existing set, by performing some mathematical calculations. Generalisation over binary features has been successful in many domains, e.g. Go [2], Civilisation 2 [1], Checkers [32] and Backgammon [33]. In general, the features were chosen manually based on the author's domain knowledge. However, [34] show that a simple automatic process using a Genetic Algorithm can greatly improve the agent's performance.

We have chosen to represent states and actions as vectors of numerical features. The vector representing an action is constructed via vector difference between the vector representing the state in which the action is performed and the vector representing the resulting state. Due to the sparsity of our corpus data, we need a set of features that does not necessarily contain the exact description of the state (i.e. coordinates of pieces on the board), but rather specific characteristics or relations (e.g. number of pieces on board, distances between them, access to resources etc). Furthermore, it is very difficult to transpose the actions due to parts of the game configuration varying a lot between games (e.g. hexes on the board are randomised in the beginning). Therefore, our features are intended to capture the abstract properties that best represent the current player's state and focus on what effects his actions may have. The feature vectors corresponding to the states encountered in the corpus are stored alongside the game's data in a database for quick access.

In contrast to previous approaches, we have chosen to underline the importance of similarity of both states and actions. Therefore we will use the vector of features representing these only to compute the relevance to the current options and use this value as a weight when introducing prior knowledge into the tree, while the utility is measured by running simulations. We calculated the relevance of the states and actions by computing their vectors' *cosine distance* (where A and B are the vectors describing the candidate states or actions):

$$R(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \quad (2)$$

D. Informing the search

Having defined the two parts of the model separately, now we will present how they are combined. A popular approach is to have an ϵ-greedy policy that will treat the advice given by the corpus as optimal and sometimes choose an action uniformly at random. UCT has a few benefits over this approach (see Section IV-A) and there are many developed approaches to bias its search. [14] have chosen to seed in the nodes based on their knowledge of Settlers of Catan and the values seeded have been chosen from observations. This approach is dangerous as it can affect UCT's ability to balance exploitation of the most promising nodes with exploration of rarely tried nodes. [4] present various approaches for introducing prior bias in the tree, out of which UCT-Rapid Action Value Estimation (RAVE) performance in previous complex applications stands out [17]. In this experiment we have chosen to seed in a value in a similar way to the approach presented in [14], as we are using Flat-UCB with an ϵ-greedy simulation policy instead of MCTS with random exploration. We are initialising the number of plays and wins of the node.

In general, RAVE or any other form of seeding may introduce some bias as the value of actions also depends on the state from which they are executed. Game actions may have the same high level description (e.g. placing a settlement), but their effect is determined based on the state description (i.e. description of adjacent hexes). To reduce this bias we compute the similarity values of both states and actions and we are averaging over the two. We decide on the nodes to seed by using the state and action similarity value, such that a pair from the corpus will only be used once to bias towards the most similar pair for each decision in the tree (i.e. action selection). In contrast to [17], who use linear function approximation that generalises to the whole space of the game, we have access to a corpus of exact actions taken in specific states.

The resulting value computed by the cosine distance is in the range $[-1, 1]$, where 1 suggests that the candidates are the same and -1 suggests they are opposites. In general, the vectors describing the states are in positive space, so the cosine distance is in $[0, 1]$. Action vectors are the result of the difference between the resulting state and the initial state, so these could contain negative values for different actions of the game. We compute the similarity of the states ($R(s, s')$, where s is the state encountered in the real game and s' is the state encountered in the corpus) and actions ($R(a, a')$, where a is one of the possible actions from the real game and a' is the action taken in the corpus) separately, so the final relevance value of a pair is the average of the two values, and it is bounded in $[-1, 1]$ (see Equation 3). This covers the scenario in which an action encountered in the corpus may be very similar to the current possible action, but it was executed from a very dissimilar state. Let's assume for now that the pairs in the database have already been evaluated and each have a utility value $U(s', a')$. Then the value of the pair in relation to the current state and action option, would be this utility value weighted with the relevance value (see Equation 4).

$$R((s,a),(s',a')) = \frac{R(s,s') + R(a,a')}{2} \tag{3}$$

$$Q(s',a') = R((s,a),(s',a'))U(s',a') \tag{4}$$

Finally, we need to take into account that multiple pairs from the corpus may suggest the same action as the best next move. As a result, the estimated $Q(s, a)$ value is averaged over all suggestions:

$$Q(s,a) = \frac{\sum_{i=0}^{n} Q_i(s',a')}{n} \tag{5}$$

The current experiment only treats the placement of a free settlement, which is always a gain. So the cosine distance of the actions' vectors is also bounded in $[0, 1]$ and so is the final relevance value from Equation 3. For now, we do not estimate the utility of the state–action pairs gathered in the corpus; instead, we trust the decision making of the person who played the action. To reflect this assumption, the utility from Equation 4 is initialised to: $U(s', a') = 1$. Even though our database contains play from novices and experts, we have observed that an average Settlers player will win around 50% of the games when playing against 3 *original* agents (also confirmed by [10]). So the assumption that humans are better at choosing locations for the initial placement than the heuristic agent is naive, but largely reasonable.

Finally, we introduce the information in the tree by setting $\overline{X_j} = Q(s, a)$ (as the value is in the range $[0, 1]$) and $n_j = 10$ in Equation 1. This means that the value of the node is equal to $Q(s, a)$, after 10 visits. Note that the UCT algorithm will try the nodes that were not initialised at least once before the seeded value will influence the search. We want to underline that the value of each state–action pair from the corpus is used only once to influence the most similar pair from the current options. The result is that we are not seeding all the nodes in the tree with a value, but rather we bias the search towards a smaller part of the tree deemed good by the information extracted from the corpus.

V. Results and analysis

We measure the performance of a modified agent by running simulations of 10000 games of 4 players; one of the players is the modified agent and he plays 3 baseline agents. Therefore, a player that is of equal strength to the baseline agent would win 25% of the games. We performed Z-test to test the significance of win rates against the above null hypothesis and we chose a threshold $p < 0.01$, so results between 24-26% are not considered significant. The need of such a large number of simulations is due to the degree of non-determinism in the game (i.e. dice rolls and the board configuration being randomised at the beginning of each game). Therefore, we want to avoid the risk of a non-reproducible results, given the vagaries of the outcomes of small number of trials.

We have run our experiments using as an upper-bound baseline the *ranking* agent briefly presented in Section III-B: i.e. an agent that plays according to a set of sophisticated, but hand-crafted rules. Results against the lower-bound baseline (random player) are not reported in the table: any of the presented agents win 99% of the games against a random agent. The modified agent was replaced with different versions of the Flat-UCB model with the intention to test each addition on its own and then assess the joint model. Furthermore, we explore the effects of biasing the search with the evaluation performed by the *ranking* agent. Therefore we have the following modified agents:

1. Flat-UCB without any prior knowledge;
2. Flat-UCB seeded with prior information extracted from the *ranking* heuristics;
3. Flat-UCB seeded with the Corpus suggested policy;
4. An agent that follows the Corpus suggested policies without performing simulations;

Agent 1 searches the options following the UCT formula (see Equation 1) and chooses the action that yielded the best result during simulations to play in the real game. This player doesn't use any prior information. Agent 4 builds the tree and seeds in the values computed from the corpus. It doesn't perform any simulations, but just chooses the action with the highest value. Agent 3 combines these two approaches. Agent 2 computes the values of actions following the *ranking* heuristic agent estimates and only the 5 nodes corresponding to the best 5 actions are initialised; the tree is seeded in the same manner as for the other two players that make use of prior knowledge. All agents, aside from the last one, perform simulations using the ϵ-greedy policy described in the previous section, where the optimal move is chosen by the *ranking* heuristic. The differences apply only on one action of the game (i.e. the placement of the second settlement during the initial set up), while the remainder of the game is played by each agent as if they were *ranking* agents.

TABLE I. RESULTS

Agent type	Win percentage	Number of games
Ranking (baseline)	*25%*	*10000*
Flat-UCB	28.74%	10000
Flat-UCB seeded with *ranking* heuristics	28.48%	10000
Flat-UCB seeded with Corpus	30.43%	10000
Corpus	19.11%	10000

Table I contains the performance in terms of win percentage of each of these agents over 10000 games, in which the other 3 players are baseline agents. As expected, Flat-UCB shows a large increase in win rate (3.7%) over the baseline and informing the search with a policy that averages over the related advice from the corpus further increases the performance by another 1.7%. These results illustrate the potential value of mining human knowledge for learning optimal strategies in complex domains. The increase in performance may seem small, however the model only tests this on one action of the game. Furthermore, the strong simulation policy could also be a cause for such a large increase in performance resulting from applying the Flat-UCB method compared to the one resulting from extending it with the seeding mechanism. The default random policy or a mixture with an opponent modelling technique should generalise better and show exactly how beneficial the corpus bias is. Unsurprisingly, the agent that doesn't run any simulations performs poorly, decreasing the win rate by 6%. The corpus data is too sparse to be the only source of information for guiding the decision making. The method for assessing the relevance presented above is an approximation and averaging over the given advice only partly filters the noise. Without any roll-outs this is not enough. In future work, we will test this hypothesis by following the same policy throughout the game. This will be made possible once we apply this model to all action types and inform MCTS instead of Flat-UCB.

A. *Note on performance*

We have tested the performance of the model on a typical desktop machine (Intel-I5 hyper-threaded processor with 4GB of RAM). The JSettlers environment runs 10000 games with 4 *ranking* players in approximately 1 hour on a typical desktop machine. Running the 1000 simulations required for the Flat-UCB method with the JSettlers environment takes 5 minutes, which is too slow for an online method. The seeding method requires less than a second to finish and can be further improved. The SmartSettlers model presented in [14] takes around 12 seconds to run 10000 random simulations, which is still too slow for an online method. However, combining 1000 simulations on the SmartSettlers model with the current seeding method would result in approximately 2 seconds required for each decision. We intend to build our approach on top of the SmartSettlers model as such short period of times would also be acceptable for the human–AI experiments which we intend to run in the future.

VI. CONCLUSION

We have presented a method that combines knowledge extracted from a corpus of human play with simulations in a Monte Carlo Tree Search implementation for Settlers of Catan. This is a proof of concept which yields promising results when compared against an agent that relies on only one of the knowledge sources or relies on exploration seeded with hand-coded heuristics for estimating promising portions of the game tree. Furthermore, it exceeds the performance of the currently best symbolic agent for the game, an implementation which was built and whose heuristics were improved over many years.

In the future we aim to show that guiding the Monte Carlo Tree Search method with information taken from the human corpus for each possible action in the game will overcome some of the issues encountered by random search. To achieve this, we need to develop an agent that is capable of handling both the remaining actions (i.e. trading) and the hidden information on its opponent's hands, as the data in our corpus has been collected by observing humans playing the unsimplified game. We would also like to pre-compute the utility of state–action pairs as we believe this will help filter the noise out before running any simulations. Finally, we aim to assess different methods of introducing prior knowledge to the search (e.g. RAVE) as well as their benefits over the current approach. Once we have a model which plays the full game, we will evaluate our agents against human expert players in addition to the method presented.

ACKNOWLEDGMENT

This work is supported by ERC grant 269427 (STAC) and Engineering and Physical Sciences Research Council (EPSRC).

References

[1] 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.

[2] D. Silver, R. S. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 968–975.

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

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

[6] W. B. Knox and P. Stone, "TAMER: Training an Agent Manually via Evaluative Reinforcement," in *IEEE 7th International Conference on Development and Learning*, August 2008.

[7] I. Sutskever and V. Nair, "Mimicking go experts with convolutional neural networks," in *Artificial Neural Networks - ICANN 2008, 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part II*, 2008, pp. 101–110.

[8] G. Tesauro, "Neurogammon : a neural-network backgammon program," IBM US Research Centers (Yorktown,San Jose,Almaden, US), Tech. Rep. RC 15619, 1990.

[9] B. Tastan and G. R. Sukthankar, "Learning policies for first person shooter games using inverse reinforcement learning." in *AIIDE*, V. Bulitko and M. O. Riedl, Eds. The AAAI Press, 2011.

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

[11] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators (Automation and Control Engineering)*, 1st ed. CRC Press, 2010. [Online]. Available: http://www.worldcat.org/isbn/1439821089

[12] P. Hingston, "A new design for a turing test for bots," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, Aug 2010, pp. 345–350.

[13] L. Branca and S. J. Johansson, "Using multi-agent system technologies in settlers of catan bots," in *Proceedings of the International Conference on Computer Games: Artificial Intelligence,*, 2007.

[14] 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.

[15] G. Roelofs, "Monte carlo tree search in a modern board game framework,," research paper available at umimaas.nl., 2012.

[16] G. Chaslot, C. Fiter, J. Hoock, A. Rimmel, and O. Teytaud, "Adding expert knowledge and exploration in monte-carlo tree search," in *Advances in Computer Games, 12th International Conference, ACG 2009, Pamplona, Spain, May 11-13, 2009. Revised Papers*, 2009, pp. 1–13. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12993-3_1

[17] S. Gelly and D. Silver, "Combining online and offline knowledge in uct," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 273–280.

[18] D. Silver, R. S. Sutton, and M. M. 0003, "Reinforcement learning of local shape in the game of go," in *IJCAI*, M. M. Veloso, Ed., 2007, pp. 1053–1058.

[19] J. P. Patist and M. Wiering, "Learning to play draughts using temporal dierence learning with neural networks and databases," in *Benelearn'04: Proceedings of the Thirteenth Belgian-Dutch Conference on Machine Learning*, T. L. A. Nowe and K. Steenhout, Eds., 2004, pp. 72–79.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower bounding klondike solitaire with monte-carlo planning." in *ICAPS*, A. Gerevini, A. E. Howe, A. Cesta, and I. Refanidis, Eds. AAAI, 2009.

[26] 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.

[27] M. L. Ginsberg, "Gib: Imperfect information in a computationally challenging game." *J. Artif. Intell. Res. (JAIR)*, vol. 14, pp. 303–358, 2001.

[28] B. Sheppard, "World-championship-caliber scrabble," *Artificial Intelligence*, vol. 134, no. 12, pp. 241 – 275, 2002.

[29] C. Browne, "The dangers of random playouts," *ICGA Journal*, vol. 34, no. 1, pp. 25–26, 2011.

[30] D. Whitehouse, P. I. Cowling, E. J. Powley, and J. Rollason, "Integrating monte carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game." in *AIIDE*, G. Sukthankar and I. Horswill, Eds. AAAI, 2013.

[31] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[32] H. C. Neto, R. M. da Silva Julia, G. S. Caexeta, and A. R. A. Barcelos, "Ls-visiondraughts: improving the performance of an agent for checkers by integrating computational intelligence, reinforcement learning and a powerful search method," *Appl. Intell.*, vol. 41, no. 2, pp. 525–550, 2014.

[33] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[34] H. Neto and R. Julian, "Ls-draughts - a draughts learning system based on genetic algorithms, neural network and temporal differences," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, Sept 2007, pp. 2523–2529.