Combining Gameplay Data with Monte Carlo Tree Search to Emulate Human Play

Sam Devlin, Anastasija Anspoka, Nick Sephton, Peter I. Cowling
Digital Creativity Labs, Department of Computer Science, University of York, UK

Jeff Rollason
AI Factory Ltd., Pinner, Middlesex, UK

Abstract

Monte Carlo Tree Search (MCTS) has become a popular solution for controlling non-player characters. Its use has repeatedly been shown to be capable of creating strong game playing opponents. However, the emergent playstyle of agents using MCTS is not necessarily human-like, believable or enjoyable. AI Factory Spades, currently the top rated Spades game in the Google Play store, uses a variant of MCTS to control non-player characters. In collaboration with the developers, we collected gameplay data from 27,592 games and showed in a previous study that the playstyle of human players significantly differed from that of the non-player characters. This paper presents a method of biasing MCTS using human gameplay data to create Spades playing agents that emulate human play whilst maintaining a strong, competitive performance. The methods of player modelling and biasing MCTS presented in this study are generally applicable to digital games with discrete actions.

Introduction

It is now feasible to collect vast quantities of high granularity data from digital games (Bohannon 2010). This ease of access to gameplay data has given rise to an increased use of data mining approaches during various stages of game development (El-Nasr, Drachen, and Canossa 2013). Simultaneously, Monte Carlo Tree Search (MCTS) has become a popular solution for Artificial Intelligence (AI) in digital games. Its use has repeatedly been shown to be capable of creating strong game playing opponents. However, the emergent playstyle of agents using MCTS is not necessarily human-like, believable or enjoyable. This paper presents a method of biasing MCTS using human gameplay data to create agents that emulate human play whilst maintaining a strong, competitive performance.

To demonstrate the method, we apply it to a digital implementation of the traditional card game Spades based on gameplay data collected from human players of AI Factory Spades; the leading commercial implementation available for Android devices. In previous work with the game's developers, we analysed these gameplay traces and showed significant differences in the playstyle of the non-player characters controlled by their current implementation of MCTS compared to human players (Cowling et al. 2015). In this paper, using the methodology proposed, we empirically demonstrate that it is feasible to emulate human play by indirect imitation whilst still maintaining equivalent play strength to an unbiased agent.

Furthermore, we argue that the methodology we have applied could feasibly be applied to any digital game currently using MCTS provided gameplay trace data is collected and the agent's actions are (or could be modelled as) discrete. In particular, we note that the common usage of tech or progression trees in many modern games provides a suitable potential application for this method. Creating agents that emulate human play can inform game design (Zook, Harrison, and Riedl 2015) and, given potential applications of MCTS in non-game contexts (e.g. simulations, decision support systems or operations research), the same methodology could again be applied to provide more human-like agents.

The remainder of the paper is organised as follows. The next section covers relevant previous work on this topic. The two sections following this detail the key contribution of this paper; our methodology of combining gameplay data and MCTS to emulate human play whilst maintaining playing strength of the agent. The paper then ends with a discussion on the underlying assumptions and conclusions of this study.

Background

This section will start with a brief background on MCTS. We will then discuss previous related work on player modelling and imitation learning, to emphasise the place of our contribution within the existing literature. The section ends with discussion of our prior work and ongoing collaboration with AI Factory.

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a sequential decision making algorithm which runs a large number of playouts from the root state in order to provide statistical information about the relative strength of the moves available from that state. The statistical information gathered is used online to determine the direction of tree expansion. The nature of MCTS means that knowledge of game strategy is not required, however such heuristic knowledge is often used to improve performance (Browne et al. 2012)

MCTS has been a focus for research since its invention in 2006 (Chaslot et al. 2006; Coulom 2007; Kocsis and Szepesvári 2006), and has shown state of the art performance in a number of games, most notably Go (Gelly et al. 2012; Silver et al. 2016). It creates an asymmetrical decision tree, with specific focus upon building towards the stronger decisions. The basic MCTS algorithm, which is executed once for each iteration in the provided budget is:

- Selection: A child node of the root node is selected using a *tree policy*. Selection continues until the node selected is a terminal node, or a node which has unexpanded children.
- Expansion: If the selected node is non-terminal, one child node is randomly selected for expansion.
- Simulation: A *default policy*, often random, creates a playout of the game from the expanded node to the end.
- Back-propagation: The result of the playout is back-propagated through the tree until it reaches the root.

One of the most commonly used MCTS tree policies is *Upper Confidence Bound for Trees (UCT)* (Kocsis and Szepesvári 2006). UCT treats the selection of a node for expansion as a multi-arm bandit problem: each node in a partial search tree is assigned a value that describes its average expected reward, calculated from previous iterations. On each iteration the algorithm selects a node that maximises:

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

where $\overline{X}_j$ is the average expected reward of the child node j, C is an exploration constant, n is the number of times the parent node has been visited, and n_j is the number of times the child node j has been visited. This formula allows control of the balance between exploitation and exploration by setting the constant C.

A wide variety of MCTS variants and enhancements have been investigated (Browne et al. 2012). One specific variant of relevance to this work is *Information Set Monte Carlo Tree Search (ISMCTS)* (Cowling, Powley, and Whitehouse 2012), a variant designed to handle games of imperfect information. The principle difference between MCTS and ISMCTS is that a new determinization of the root game state is created on each iteration, meaning iterations are split between different determinations of the game with a distribution which approximates the likelihood of that determinization occurring. The work of this paper is based on ISMCTS.

Emulating Human Play

A lot of previous research has been focused on using gameplay traces to learn a model for predicting the behaviour of a human player. Synnaeve and Bessiere (2011) created a Bayesian model for predicting the opening of opponents in the real-time strategy game StarCraft by learning the parameters of the model from game logs with labeled openings. Dereszynski et al. (2011) used replays from StarCraft to learn a Hidden Markov Model for predicting high-level strategic behaviour. These models were integrated into the existing agents in order to improve their ability to predict the strategy of a human player, but the models were not used to bias the agents towards a human play style. Similarly, the recent high profile success of Google DeepMind's AlphaGo also involved modelling human play from a large dataset of expert games (Silver et al. 2016). However, AlphaGo's goal was to maximise game playing ability and the resultant playstyle was frequently commented on to be unconventional. In contrast, our priority in this work was on the emulation of human play. In this work, the resultant play strength of the agent was a secondary objective and one we aimed to maintain equivalent performance in, not maximise.

Another area of interest, more closely related to our goal, is developing an agent that imitates a human player. Togelius et al. (2007) divide behaviour imitation into two categories: direct and indirect. Direct imitation involves using a supervised learning method on a dataset that contains a list of states in a game, together with the action that a human player has taken in that state. The result of the training process is a model that chooses an action to perform given a description of the state of the game. Once the model has been trained, it cannot be modified, so the resulting agent is incapable of learning from its mistakes, which is one of the main critiques of direct modeling. Another problem was found by Togelius et al. (2007) when they used direct modeling to develop an AI agent for a racing game: whenever the controller was in a state that it had not seen before, it was not able to react in a sensible way, leading to a car crash that it could not recover from. Despite these problems, there have been a number of attempts to create a human-like agent using direct imitation. Thurau et al. (2007) used Bayesian imitation learning to teach an agent to navigate through a maze in Quake II using recordings of human players. The results of their experiments showed that the agents appear human-like, but their functionality is limited to imitating movement and would not be able to play against a human player. This again differs from the method we present in this paper, as our intention was to develop an agent that whilst human-like could act rationally in states it had not previously observed and could feasibly be deployed within the game.

Alternatively, indirect imitation involves modifying the fitness function of an existing agent. Unlike direct imitation, this approach biases an agent towards a certain behaviour instead of fully specifying it, which overcomes the problem of the agent being unable to act when in an unfamiliar state. This approach is the closest in relation to the method we propose. An existing example of this trained agents to play Super Mario Bros using both direct and indirect imitation methods (Ortega et al. 2013). The results of their experiments show that the agents with indirect imitation were perceived as human-like significantly more often than the agents with direct imitation, and also performed better. However, these agents did not perform as well as the agents that were trained just to play well, without any concern for believability. Therefore, there is a compromise in previous work between maximising the performance of an agent and making it appear more human-like. As we will show in the remainder of the paper, the approach we propose is capable of both playing in a more human-like way and maintaining a play strength equivalent to an unbiased MCTS agent.

AI Factory Spades

AI Factory[1] is a UK-based independent game developer, currently specialising in implementations of classic board and card games for Android mobile devices. Spades is a four-player trick-taking card game, especially popular in the USA but played worldwide (Pagat 2016). Spades is a partnership game, with North and South in coalition against East and West. Spades has some similarities with, but slightly simpler rules than, the game of Bridge. AI Factory's implementation of Spades has been downloaded more than 5 million times, with an average review score of $4.4/5$ from more than $180,000$ ratings on the Google Play store[2]. The game is a single-player implementation, in which the user plays with an AI partner against two AI opponents.

The results reported in this paper are based on data collected between 1st April 2013 and 12th November 2013 from a random sample of approximately $\frac{1}{32}$ of players representing 690 unique players and $27,592$ full games. Specifically, upon completion of a game, the following information is captured via Google Analytics:

- The player's anonymised identifier and country;
- Historical win and loss counts for each game level and for each AI character;
- The version number of the game;
- The random seed used for this game (the same pseudo-random number generator is used for card deals and for ISMCTS simulations);
- Parameters for the chosen AI players;
- Rule settings for this game;
- The final score;
- The sequence of bidding and trick play moves.

In previous work we collaborated with AI Factory to implement ISMCTS-based AI players (Whitehouse et al. 2013) and then analysed the data collected to gain insights into the playstyles of both the AI and human players (Cowling et al. 2015). One conclusion from the previous analysis was that in certain contexts the AI players were acting significantly different to human players, as evidenced by differences in the distributions of abstract moves chosen given the game state. Following on from this observation, and assuming that an AI that more closely emulates human play would be more enjoyable, our aim in this work was to reduce this difference in the distribution of moves chosen. This could be done by direct imitation using a probabilistic model of the human moves but, as discussed earlier, this could result in undesirable behaviour if the agent needs to make a decision in a new context. Furthermore, as the difficulty of the game was deemed to be appropriate given the previous analysis of player win rates (Cowling et al. 2015) and reviews of the game on the App store, maintaining a play strength equivalent to the existing AI was also necessary.

[1] http://www.aifactory.co.uk

[2] https://play.google.com/store/apps/details?id=uk.co.aifactory.spadesfree&hl=en

Therefore, we propose an indirect imitation solution with the dual aims of:

1. Reducing the difference in the distribution of abstract moves chosen by human and AI players;
2. Maintaining a comparable play strength to the existing ISMCTS agents.

In the next section we will cover our approach to modelling the human play, which we will then use in the section afterwards to bias an ISMCTS agent.

Modelling Human Play

To model human play the method proposed by Xiao and Müller (2016) was used as a foundation. However, to reduce the number of parameters, we removed the pairwise interactions between features from their model ranking the relative strength of moves. As our method of bias involves calculating the strength of each move in the search tree, using a large number of parameters could significantly increase the response time of an agent which would be unsuitable for deployment in a mobile game such as AI Factory Spades.

Formally, the strength E_m of a move m is calculated as a sum of weights w of the features f that apply to the move:

$$E_m = \sum_{f \in m} w_f \tag{2}$$

where the features f are domain specific, but listed in the Appendix for completeness of the report. Given N moves $\{m^1, m^2, ..., m^N\}$, the probability of a human player choosing a move m^i is calculated using an exponential model:

$$P(m^i \text{ is chosen}) = \frac{\exp(E_{m^i})}{\sum_{j=1}^{N} \exp(E_{m^j})} \tag{3}$$

The goal of the algorithm is to find the parameters of the model that maximise the likelihood of the training data. To achieve this, the following loss function is used for state s_j :

$$l_j = -\ln \frac{\exp(E_{m_j^*})}{\sum_{i=1}^{|\Gamma(s_j)|} \exp(E_{m_j^i})} = -E_{m_j^*} + \ln \sum_{i=1}^{|\Gamma(s_j)|} \exp(E_{m_j^i}) \tag{4}$$

where $\Gamma(s_j)$ is the set of all legal moves in the state s_j, and m_j^* is the move chosen by the human player. This model was fit by stochastic gradient decent, with validation-based early stopping (Prechelt 1998). As stochastic gradient descent may require multiple passes through the data, the samples in the training set were shuffled before each pass to avoid biasing the gradient. The training process was performed with learning rate $\alpha = 0.003$ and batch size $n = 200$. The gradient of the loss function l_j for the parameter θ_f of the feature f is calculated as:

$$\nabla_j \theta_f = -\mathbb{I}(f \in m_j^*) + \frac{\sum_{i=1}^{|\Gamma(s_j)|} \exp(E_{m_j^i}) \mathbb{I}(f \in m_j^i)}{\sum_{i=1}^{|\Gamma(s_j)|} \exp(E_{m_j^i})} \tag{5}$$

where $\mathbb{I}(f \in m_j^i)$ equals 1 if the feature f applies to the move m_j^i, and 0 otherwise.

For each card played by a human player, a competition was created, in which all moves that were legal in that game state were viewed as contestants, and the move chosen by the player was declared a winner of this competition. The dataset of competitions was further divided into 3 subsets: training data (189,770 competitions), validation data (63,200 competitions), and test data (63,200 competitions). The mean test accuracy of the model from 5 repeats of the training process was 44.96%. For comparison, the test accuracy of a randomly guessing model was 29.39%. As our aims were emulating human play, these intermediate results are included for completeness of the report and to emphasise that it is not necessary to maximise the predictive accuracy of the player model for the resultant playstyle of the agent to more closely resemble human play.

In the next section we will discuss the method of integrating this model into the ISMCTS agent and evaluate the combined system of modelling human play and biasing ISMCTS against our dual aims.

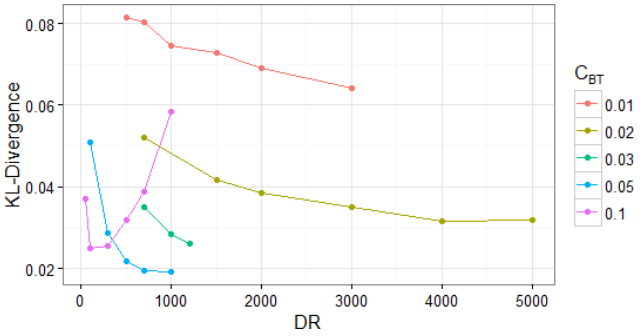
Biasing Monte Carlo Tree Search

To incorporate the model of human play, the formula for calculating the UCT score in the baseline ISMCTS agent was modified to include a bias term based on the Bradley-Terry value. The updated UCT score of a node i was calculated as:

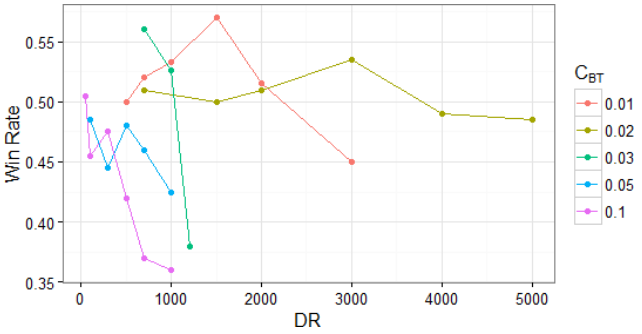
$$UCT(i) = \overline{X}_i + 0.7\sqrt{\frac{\ln n}{n_i}} + C_{BT}\sqrt{\frac{DR}{n + DR}}P(m_i) \quad (6)$$

where $P(m_i)$ is the probability of the move corresponding to the node i being chosen by a human player, calculated using Equation 3. C_{BT} is a parameter that controls the influence of the bias on the UCT score, and DR controls the rate at which the influence of the bias decreases with the number of parent node visits. This form of integrating the model of human play was based upon the method of knowledge bias, a technique used previously by Ikeda and Viennot (2014) to integrate prior knowledge into a MCTS agent for the game Go. To set the parameters C_{BT} and DR, we performed an exploratory analysis (illustrated in Figure 1) of 26 combinations of settings and obtained the difference in abstract move distribution compared to human play measured by the Kullback-Leibler (KL) divergence (Kullback 1968) and the win rate compared to a baseline ISMCTS player over 200 games. Given our dual aims, settings that minimise the KL-divergence and maximise the win rate are desireable.

For $C_{BT} = 0.1$ there seems to be a curvilinear relationship between DR and KL-divergence: for DR in range $[50, 100]$ the value of KL-divergence decreases rapidly, but for DR in range $[100, 1000]$ it starts to increase instead. This behaviour could be caused by the bias having too strong an effect on the UCT score. The Bradley-Terry model was not intended to be used as a stand-alone algorithm for describing agent's behaviour, but rather as an algorithm for biasing an existing agent. If it is given too much weight in the UCT equation, the agent may start to perform poorly. Even though the bias term is supposed to bias an agent towards human-like behaviour, using a stronger bias may result in larger KL-divergence. This situation is similar to a novice player trying to play purely by imitating professional players. She/he



(a) Difference in move distribution compared to human play



(b) Win Rate (%) against a baseline ISMCTS player

Figure 1: The effect of the bias parameters C_{BT} and DR on:

may perform moves that are generally considered good, but without having an appropriate strategy in mind and without understanding the state of the board he/she is likely to lose. As the value of DR increases, the weight of the bias starts to decrease at a slower rate, resulting in the bias having a stronger effect on agent's behaviour. Eventually the effect becomes too strong and KL-divergence starts to increase.

For other values of C_{BT} the relationship is negative: as the value of DR increases, the value of KL-divergence decreases. It may be possible to reduce KL-divergence further by choosing larger values of DR but, considering Figure 1(b), this configuration is likely to result in a win rate of less than 50%. Given our secondary aim of maintaining an equivalent playing strength (i.e. a 50% win rate) larger values of DR were not tested. Of all the parameter settings that achieved a win rate of at least 50%, $C_{BT} = 0.03$ and $DR = 1000$ produced the smallest KL-divergence and were therefore considered as the best configuration for the bias.

After selecting the bias parameters, we conducted further experiments to provide an unbiased evaluation of the effect of the bias on the performance and playing style of the baseline agent. Ten sets of 50 games were played between a partnership of baseline, unbiased ISMCTS agents and a partnership of biased ISMCTS agents, and for each set the win rate, average response time, and the probability distribution of abstract moves of each agent type were recorded. The biased player had an average win rate of 49.6% with 95% confidence interval of $\pm 3.4\%$. This indicates that the performance of the biased agent is comparable to the performance

Table 1: KL-divergence in the distribution of abstract moves chosen by the AI players compared to human players

Baseline ISMCTS	AIFactory Spades	Biased Agent
0.1142	0.0981	0.0291

of the baseline ISMCTS agent. The average response time of the biased player in each set of games was within ± 30 ms of the response time of the baseline agent, so again they are comparable in terms of our secondary aim of maintaining equivalent performance.

Given that our secondary objective of maintaining equivalent playing strength has been demonstrated, we now evaluate the primary objective of emulating human play by minimising the difference in moves chosen. Specifically, Table 1 shows the KL-divergence between the distribution of the abstract moves chosen by the AI players (the baseline ISMCTS agent we modified, the ISMCTS agent deployed in AI Factory Spades and the biased agent we created for this study) compared to the human players. These values are given to quantify the reduction in difference in playstyle between AI players and human players. The results show a large improvement caused by including bias, as smaller values represent behaviour closer to the human distribution and, therefore, closer to emulating human play.

To provide a more detailed look at the specific changes between the previously deployed agent and the biased agent we have proposed in this paper, Figure 2 displays the mean frequency and standard deviation each abstract move is played by the AIFactory Spades ISMCTS agent (referred to as Original AI Player), our biased player, and human players. In these graphs an abstract move represents the effect of a card that is played given the context the card is played in. For Spades, this is more informative than considering the specific card played as the same card can have significantly different effect on the game dependent on the cards already played. In particular we note the large changes in usage of the abstract move type Follow Duck Other[3] in Figure 2(c) and Follow Duck Highest in Figure 2(d).

These results empirically demonstrate that our biased player also satisfies our aims of reducing the difference in the distribution of abstract moves chosen by human and AI players. Compared to both the AIFactory Spades ISMCTS agent and a baseline ISMCTS agent, the biased player has a distribution of abstract moves that is much closer to the distribution of abstract moves played by human players. In addition, both win rate and response time of the biased player are comparable to those of the baseline player, indicating that it is possible to emulate human play without having a negative impact on performance.

[3]Due to space constraints, we have chosen not to give more details on each of the abstract moves as they are domain specific and we want to emphasise the general applicability of our methodology. Readers interested in this specific application can find a richer description of the abstract moves in Cowling et al. (2015).

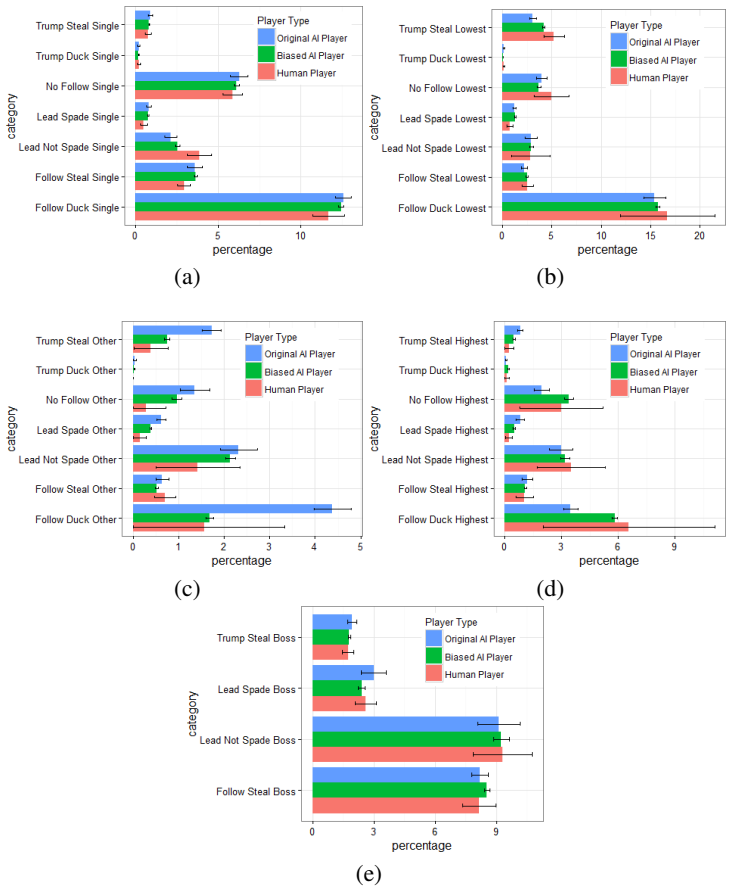


Figure 2: Comparison of abstract move frequencies among human players, AIFactory Spades ISMCTS agent and our biased agent. (a)-(e) show, within each category, how frequently each agent plays their (a) only, (b) lowest, (d) highest non-boss, (e) boss, or (c) any other card in that category.

Discussion

Before concluding this paper, it is important for us to consider three underlying assumptions in this study each of which could become topics for future work. Firstly, the modifications proposed here were applied to a baseline ISMCTS agent and not directly to the AIFactory ISMCTS implementation. This was partially to focus the experimentation on the modifications being made and to isolate the cause of the effect in changing the agents playstyle, but was also due to necessarily restricted access to the source code of the commercial game. Similarly the version of Spades used was a command line implementation of the core game mechanics to enable rapid iteration of experiments and not the AIFactory codebase. Therefore, we have assumed throughout this study that the effect of the modifications on the ISMCTS player in our implementation of Spades will be similar to the effect if applied in future work to the ISMCTS agent in the commercial game. We feel this is a justifiable assumption given that previous work on our implementation of Spades resulted in similar beneficial gains when transferred to the

commercial game (Whitehouse et al. 2013).

Secondly, we have amalgamated all gameplay data into one model of human play. This assumes that there is only one playstyle that all humans use in the game. Whilst this is a large assumption, we justified this internally due to the outcomes of an early study in this work that attempted to find clusters of differing playstyle amongst the gameplay traces. Specifically, we generated decision tree (Breiman et al. 1984) models of each individual player's behaviour and then clustered these models using K-medoids (Park and Jun 2009) and a partial tree kernel (Moschitti 2006). The results clearly showed 4 distinct clusters but, on inspection of the resultant medoid trees, no significant qualitative difference in the playstyle of each cluster was observed. Therefore, our conclusion at this time is that amalgamating all playstyles is potentially the cause of the relatively low accuracy of our player model but, because the resultant agent behaviour sufficiently match our dual objectives in this work, this assumption is currently justifiable for this application. In future work, we may explore alternative methods of clustering game behaviour data (Bauckhage, Drachen, and Sifa 2015) (e.g. graph edit distance (Robles-Kelly and Hancock 2005)). In applications where such clusterings are available, this could produce multiple different agents each of which emulate a unique human playstyle.

Finally, as is common for research on commercial game AI, our motivation is ultimately to create agents that are fun to play with or against. In this work we have assumed that agents that more closely emulate human play are more enjoyable to play with. This hypothesis is often underpinning work on imitation learning in game contexts, but at this time is relatively under studied and requires further research to validate. Therefore, our contribution is currently specifically to emulation of human play but could be extended in future work to conclude effects on enjoyment provided thorough evaluation to suitable metrics (Mekler et al. 2014; Fang et al. 2010).

Conclusions

In conclusion, we have presented a method for biasing MCTS with gameplay trace data to emulate human play. Our specific study used data from AIFactory Spades, a commercial mobile game that uses ISMCTS to control non-player characters. Previous work had shown that the existing non-player characters acted significantly different to human players, but this study empirically demonstrates that adding a bias towards models of human play can reduce this difference whilst maintaining play strength equivalent to unmodified MCTS agents. Furthermore, provided the availability of human gameplay traces and an existing MCTS solution, this method could feasibly be applied to any game with a discrete action space.

Acknowledgments.

This work was conducted in the Digital Creativity Labs (www.digitalcreativity.ac.uk), jointly funded by the EPSRC, AHRC and InnovateUK under grant no EP/M023265/1.

Appendix: Model Features

The table below lists the features used in the model of human play and their resultant weights. The features were originally chosen as an exhaustive coverage of the abstract moves that a player can make. These were then extended for this work to capture gameplay states where there was the most significant differences in the distribution of human moves compared to agent moves. Interested readers can find a richer description of their implementation in Cowling et al. (2015).

Feature	Weight
Lead Non Spade Boss	1.62
Lead Spade Boss	1.17
Trump Steal Boss	−0.03
Follow Steal Boss	1.12
Lead Non Spade Highest	−0.06
Lead Spade Highest	−1.05
Trump Duck Highest	0.05
Trump Steal Highest	−0.63
No Follow Highest	0.72
Follow Duck Highest	−1.88
Follow Steal Highest	0.36
Lead Non Spade Lowest	−0.34
Lead Spade Lowest	−0.28
Trump Duck Lowest	−0.15
Trump Steal Lowest	1.56
No Follow Lowest	−0.10
Follow Duck Lowest	0.16
Follow Steal Lowest	0.75
Lead Non Spade Other	−1.27
Lead Spade Other	−1.60
Trump Duck Other	−0.51
Trump Steal Other	−1.14
No Follow Other	−1.90
Follow Duck Other	−1.86
Follow Steal Other	0.28
Lead Non Spade Single	1.18
Lead Spade Single	0.63
Trump Duck Single	0.10
Trump Steal Single	1.17
No Follow Single	0.88
Follow Duck Single	0.04
Follow Steal Single	1.03
Steal Lowest If Last Turn	0.63
Duck Highest When Needing Tricks	−0.39
Duck Lowest When Boss Cards Are in Hand	−0.47
Did Not Trump Steal	−0.92
No Follow Highest With Potential Boss	−0.49
Steal Unwanted Tricks	−1.23
No Follow Lowest When Needing Tricks	1.22
Lead With Suit Of Smallest Size	0.66
Follow Duck Highest For Unwanted Tricks	2.21
Follow Duck Highest With a Small Value Card	1.19

References

Bauckhage, C.; Drachen, A.; and Sifa, R. 2015. Clustering game behavior data. *IEEE Transactions on Computational Intelligence and AI in Games* 7(3):266–278.

Bohannon, J. 2010. Game-miners grapple with massive data. *Science* 330(6000):30–31.

Breiman, L.; Friedman, J.; Stone, C. J.; and Olshen, R. A. 1984. *Classification and regression trees*. CRC press.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Chaslot, G. M. J.-B.; Saito, J.-T.; Bouzy, B.; Uiterwijk, J. W. H. M.; and van den Herik, H. J. 2006. Monte-Carlo Strategies for Computer Go. In *Proceedings of the Benelux Conference on Artificial Intelligence*, 83–91.

Coulom, R. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of the 5th International Conference on Computers and Games, LNCS 4630*, 72–83.

Cowling, P. I.; Devlin, S.; Powley, E. J.; Whitehouse, D.; and Rollason, J. 2015. Player preference and style in a leading mobile card game. *IEEE Transactions on Computational Intelligence and AI in Games* 7(3):233–242.

Cowling, P.; Powley, E. J.; and Whitehouse, D. 2012. Information set Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2):120–143.

Dereszynski, E. W.; Hostetler, J.; Fern, A.; Dietterich, T. G.; Hoang, T.-T.; and Udarbe, M. 2011. Learning probabilistic behavior models in real-time strategy games. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

El-Nasr, M. S.; Drachen, A.; and Canossa, A. 2013. *Game analytics: Maximizing the value of player data*. Springer Science & Business Media.

Fang, X.; Chan, S.; Brzezinski, J.; and Nair, C. 2010. Development of an instrument to measure enjoyment of computer game play. *International Journal of Human-Computer Interaction* 26(9):868–886.

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

Ikeda, K., and Viennot, S. 2014. Efficiency of static knowledge bias in Monte-Carlo Tree Search. In *Computers and Games*. Springer. 26–38.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Kullback, S. 1968. *Information theory and statistics*. Courier Corporation.

Mekler, E. D.; Bopp, J. A.; Tuch, A. N.; and Opwis, K. 2014. A systematic review of quantitative studies on the enjoyment of digital entertainment games. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, 927–936. ACM.

Moschitti, A. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *Machine Learning: ECML 2006*. Springer. 318–329.

Ortega, J.; Shaker, N.; Togelius, J.; and Yannakakis, G. N. 2013. Imitating human playing styles in Super Mario Bros. *Entertainment Computing* 4(2):93–104.

Pagat. 2016. Spades. http://www.pagat.com/boston/spades.html. Accessed: May 20th 2016.

Park, H.-S., and Jun, C.-H. 2009. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications* 36(2):3336–3341.

Prechelt, L. 1998. Early stopping-but when? In *Neural Networks: Tricks of the trade*. Springer. 55–69.

Robles-Kelly, A., and Hancock, E. R. 2005. Graph edit distance from spectral seriation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27(3):365–378.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Synnaeve, G., and Bessiere, P. 2011. A Bayesian model for opening prediction in RTS games with application to StarCraft. In *IEEE Conference on Computational Intelligence and Games*, 281–288.

Thurau, C.; Paczian, T.; Sagerer, G.; and Bauckhage, C. 2007. Bayesian imitation learning in game characters. *International journal of intelligent systems technologies and applications* 2(2-3):284–295.

Togelius, J.; De Nardi, R.; and Lucas, S. M. 2007. Towards automatic personalised content creation for racing games. In *2007 IEEE Symposium on Computational Intelligence and Games*, 252–259.

Whitehouse, D.; Cowling, P. I.; Powley, E. J.; and Rollason, J. 2013. Integrating Monte Carlo Tree Search with knowledge-based methods to create engaging play in a commercial mobile game. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Xiao, C., and Müller, M. 2016. Factorization ranking model for move prediction in the game of Go. In *AAAI Conference on Artificial Intelligence*.

Zook, A.; Harrison, B.; and Riedl, M. O. 2015. Monte-carlo tree search for simulation-based strategy analysis. In *Proceedings of the 10th Conference on the Foundations of Digital Games*.