Predicting Dynamic Difficulty

Olana Missura and Thomas Gärtner
University of Bonn and Fraunhofer IAIS
Schloß Birlinghoven
52757 Sankt Augustin, Germany
{olana.missura,thomas.gaertner}@uni-bonn.de

ABSTRACT

Motivated by applications in electronic games as well as teaching systems, we investigate the problem of dynamic difficulty adjustment. The task here is to repeatedly find a game difficulty setting that is neither 'too easy' and bores the player, nor 'too difficult' and overburdens the player. The contributions of this paper are (*i*) formulation of difficulty adjustment as an online learning problem on partially ordered sets, (*ii*) an exponential update algorithm for dynamic difficulty adjustment, (*iii*) a bound on the number of wrong difficulty settings relative to the best static setting chosen in hindsight, and (*iv*) an empirical investigation of the algorithm when playing against adversaries.

1. INTRODUCTION

Difficulty adjustment is common practice in many traditional games. Consider, for instance, the handicap in golf or the handicap stones in go. The case for dynamic difficulty adjustment in electronic games, however, has been made only recently [7]. The aim is to develop games that provide challenges of the "right" difficulty, i.e., such that players are stimulated but not overburdened. Naturally, what is the right difficulty depends on many factors and can not be fixed once and for all players. For that, we investigate how general machine learning techniques can be employed to automatically adjust the difficulty of games. A general technique for this problem has natural applications in the huge markets of computer and video games but can also be used to improve the learning rates when applied to serious games.

The traditional way in which games are adjusted to different users is by providing them with a way of controlling the difficulty level of the game. To this end, typical levels would be 'beginner', 'medium', and 'hard'. Such a strategy has many problems. On the one hand, if the number of levels is small, it may be easy to choose the right level but it is unlikely that the difficulty is then set in a very satisfying way. On the other hand, if the number of levels is large, it is more likely that a satisfying setting is available but finding it becomes more difficult. Furthermore, creating the appropriate game settings for each of these levels is a difficult and time-consuming task.

In this paper, we formalise dynamic difficulty adjustment as a meta-game between a *master* and a *player* in which the *master* tries to predict a game state with the most appropriate difficulty. As the *player* is typically a human with changing performance depending on many hidden factors as well as luck, no assumptions about the *player* can be made.

Note that the *player* is oblivious to the existence and her participation in this meta-game. Her actions and responses in the actual (electronic or traditional) game she is playing provide the *master* with the information necessary to make its own moves, i.e. to change the difficulty level appropriately. It should be noted here that while in general this information should tell the *master* whether the current difficulty level is too easy, just right, or too difficult, in practice its realisation depends very much on a particular game (e.g. the puzzles are solved too quickly, the player is losing her lives too fast, and so on). Therefore, we do not consider the question of what exactly should be included in this information; the assumption is only that somehow the *master* has an access to it.

The difficulty adjustment meta-game is played on a partially ordered set which reflects the 'more difficult than'-relation on the set of game states. To understand where the partially ordered set comes from consider the following simplified example: Let there be two ways into a castle, one with a locked door and another one guarded by several guards. For a player who is good with a lockpick but not so good with the fighting the first one is easy and the second one is difficult, while for a player who is a good fighter but doesn't even have a lockpick the first one is difficult and the second one is easy. In general these two game states are not comparable.

Note that while for each particular player and an instance of gameplay the game states may build a totally ordered set, the perceived difficulty depends not only on the game-specified skills (lockpicking, fighting, etc.), but also on the intrinsic player qualities (strategic thinking, patience, etc.), which are difficult or impossible to include into the model. Therefore, we state that the game states build a partially ordered set.

On this partially ordered set, the problem of difficulty adjustment reduces to predicting an unknown vertex cut between the 'too difficult' and the 'too easy' states. To the best of our knowledge, in this paper, we provide the first thorough theoretical treatment of dynamic difficulty adjustment as a prediction problem.

The contributions of this paper are: We formalise the learning problem of dynamic difficulty adjustment (in Section 2) and propose a novel learning algorithm for dynamic difficulty adjustment (in Section 4). For this algorithm we then give a bound on the number of proposed difficulty settings that were not just right (in Section 5). The bound limits the number of mistakes the algorithm can make relative to the best static difficulty setting chosen in hindsight. For the bound to hold, no assumptions whatsoever need to be made on the behaviour of the player. Last but not least we empirically study the behaviour of the algorithm under various circumstances (in Section 6). In particular, we investigate the performance of the algorithm 'against' statistically distributed players by simulating the players as well as 'against' adversaries by asking humans to try to trick the algorithm in a simplified setting. Implementing our algorithm into real games and testing it with real players is left to future work.

2. PROBLEM FORMULATION

To be able to theoretically investigate dynamic difficulty adjustment, we view it as a game between a *master* and a player, played on a partially ordered set modelling the 'more difficult than'-relation. The game is played in turns where each turn has the following elements:

1. the game *master* chooses a difficulty setting,
2. the *player* plays one 'round' of the game in this setting, and
3. the game *master* experiences whether the setting was 'too difficult', 'just right', or 'too easy' for the player.

The *master* aims at making as few as possible mistakes, that is, at choosing a difficulty setting that is 'just right' as often as possible. In this paper, we aim at developing an algorithm for the *master* with theoretical guarantees on the number of mistakes in the worst case while not making any assumptions about the *player*.

To simplify our analysis, we make the following, rather natural assumptions:

- the set of difficulty settings is finite;
- in every round, the (hidden) difficulty settings respect the partial order, that is,
  - no state that is 'more difficult than' a state which is 'too difficult' can be 'just right' or 'too easy' and
  - no state that is 'more difficult than' a state which is 'just right' can be 'too easy'; and
- in every round, for every pair of difficulty settings, one of which is 'too difficult' and one of which is 'too easy', there exists a difficulty setting in between that is 'just right'.

Even with these natural assumptions, in the worst case, no algorithm for the *master* will be able to make even a single correct prediction. As we can not make any assumptions about the *player*, we will be interested in comparing our algorithm theoretically and empirically with the best statically chosen difficulty setting, as is commonly the case in online learning [3]. If gameplay determines a subset of the difficulty settings that are meaningful, the *master* algorithm is restricted to choose from this subset and we should also compare to the best static in each of these subsets. For this paper, we make the restricting assumption that each subset forms a chain, that is, it is linearly ordered.

3. RELATED WORK

As of today there exist a few commercial games with well designed dynamic difficulty adjustment. These commonly employ heuristics and as such suffer from the typical disadvantages like requiring extensive testing, not being transferable easily to other games, etc). Instead of heuristics, in this paper, we aim at universal mechanism for dynamic difficulty adjustment: An online algorithm that takes as an input game-specific ways to modify difficulty as well as the current player's in-game history and produces as an output an appropriate difficulty modification. The game history includes the player's previous actions, performance, reactions, . . . and the difficulty modifications can either be an explicitly given finite set or an implicitly defined infinite set.

Both, artificial intelligence researchers and the game developers community, display interest in the problem of automatic difficulty scaling. Different approaches can be seen in the work of R. Hunicke and V. Chapman [10], R. Herbich and T. Graepel [9], Danzi et al [7], and others. Since perceived difficulty and preferred difficulty are subjective parameters, the dynamic difficulty adjustment algorithm should be able to choose the "right" difficulty level quickly for any particular player. Existing work in player modelling in computer games [13, 5, 12] demonstrates the power of utilising player models to create the games or in-game situations of high interest and satisfaction for the players.

As can be seen from these examples the problem of dynamic difficulty adjustment in video games was attacked from different angles, but a unifying and theoretically sound approach is still missing. To the best of our knowledge this work contains the first theoretical formalisation of dynamic difficulty adjustment as a learning problem.

Under the assumptions described in Section 2, we can view the partially ordered set as a directed acyclic graph, at each round labelled by three colours (say, red, for 'too difficult' green for 'just right', and blue for 'too easy') such that

- for every directed path in the graph between two equally labelled vertices, all vertices on that path have the same colour,

- there is no directed path from a green vertex to a red vertex and none from a blue vertex to either a red or a green vertex, and
- on every directed path from a red to a blue vertex, there is a green vertex, that is, the green vertices form a vertex cut.

The colouring is allowed to change in each round as long as it obeys the above rules. The *master*, i.e., the learning algorithm, does not see the colours but must point at a green vertex as often as possible. The feedback that the *master* receives corresponds to the true color of the vertex he pointed at.

This setting is related to learning directed cuts with membership queries. For learning directed cuts, i.e., monotone subsets, Gärtner and Garriga [8] provided algorithms and bounds for the case in which the labelling does not change over time. They then showed that the intersection between a monotone and an anti-monotone subset in not learnable. This negative result is not applicable in our case, as the feedback we receive is more powerful. They furthermore showed that directed cuts are not learnable with traditional membership queries if the labelling is allowed to change over time. This negative result also does not apply to our case as the aim of the *master* is "only" to point at a green vertex as often as possible and as we are interested in a comparison with the best static vertex chosen in hindsight.

If we ignored the partial order inherent in the difficulty settings, we will be in a standard multi-armed bandit setting [2]: There are K arms, to which an unknown adversary assigns loss values on each iteration (0 to the 'just right' arms, 1 to all the others). The goal of the algorithm is to choose an arm on each iteration to minimize its overall loss. The difficulty of the learning problem comes from the fact that only the loss of the chosen arm is revealed to the algorithm. This setting was studied extensively in the last years, see [11, 6, 4, 1] and others. The standard performance measure is the so-called 'regret': The difference of the loss acquired by the learning algorithm and by the best static arm chosen in hindsight. The best known to-date algorithm that does not use any additional information is the Improved Bandit Strategy (called IMPROVEDPI in the following) [3]. The upper bound on its regret is of the order $\sqrt{KTln(T)}$, where T is the amount of iterations. IMPROVEDPI will be the second baseline after the best static in hindsight (BSIH) in our experiments.

4. ALGORITHM

In this section we give an exponential update algorithm for predicting a vertex that corresponds to a 'just right' difficulty setting in a finite partially ordered set $(\mathcal{K}, \succ)$ of difficulty settings. The partial order is such that for $i, j \in \mathcal{K}$ we write $i \succ j$ if difficulty setting i is 'more difficult than' difficulty setting j. The set of chains that the environment may choose one from at each round is denoted by $\mathcal{C}$ and the learning rate of the algorithm by β. The response that the *master* algorithm can observe o_t is $+1$ if the chosen difficulty setting was 'too easy', 0 if it was 'just right', and -1 if it was 'too difficult'. The algorithm maintains a belief w of each vertex being 'just right' and updates this belief if the observed response implies that the setting was 'too easy' or 'too difficult'.

Algorithm 1 PARTIALLY-ORDERED-SET MASTER (POSM) for Difficulty Adjustment

```
Require: parameter $\beta \in (0,1)$, $K$ difficulty Settings $\mathcal{K}$,
    partial order $\succ$ on $\mathcal{K}$, $C$ subsets $\mathcal{C}$ of $\mathcal{K}$ each forming a
    chain in the partially ordered set $(\mathcal{K}, \succ)$, and a sequence
    of observations $o_1, o_2, \dots$
 1: $\forall k \in \mathcal{K}$ : let $w_1(k) = 1$
 2: for each turn $t = 1, 2, \dots$ do
 3:     PICK a chain $c_t \in \mathcal{C}$
 4:     $\forall k \in c_t$ : let $A_t(k) = \sum_{x \in c_t : x \succeq k} w_t(k)$
 5:     $\forall k \in c_t$ : let $B_t(k) = \sum_{x \in c_t : x \preceq k} w_t(k)$
 6:     PREDICT $k_t = \text{argmax}_{k \in c_t} \min\{B_t(k), A_t(k)\}$
 7:     OBSERVE $o_t \in \{-1, 0, +1\}$
 8:     if $o_t = +1$ then
 9:         $\forall k \in \mathcal{K}$ : let $w_{t+1}(k) = \begin{cases} \beta w_t(k) & \text{if } k \succeq k_t \\ w_t(x) & \text{otherwise} \end{cases}$
10:     end if
11:     if $o_t = -1$ then
12:         $\forall k \in \mathcal{K}$ : let $w_{t+1}(k) = \begin{cases} \beta w_t(k) & \text{if } k \preceq k_t \\ w_t(x) & \text{otherwise} \end{cases}$
13:     end if
14: end for
```

The main idea of Algorithm 1 is that for each round, once a chain has been chosen in line 3 of the algorithm, we want to make sure we can update as much belief as possible. The significance of this will be clearer when looking at the theory in the next section. To ensure it, we compute for each setting k the belief 'above' k on the chain as well as 'below' k on the chain. That is, A_t in line 4 of the algorithm collects the belief of all settings on the chain that are known to be 'more difficult' and B_t in line 5 of the algorithm collects the belief of all settings on the chain that are known to be 'more easy' than k. Depending on the observation, we will be able to either update the believes above or below the chosen setting and as we are considering a worst case scenario, we can only guarantee to update an amount of belief larger than or equal to $\min\{B_t(k), A_t(k)\}$. To achieve the best performance, we choose the k that gives us the best worst case guarantee in line 6 of the algorithm.

The way in which a chain is picked in line 3 will typically depend on the game at hand. In absence of further assumptions or knowledge about the game, however, one possible strategy is to sample uniformly at random a chain from the minimum path cover of the partially ordered set.

5. THEORY

We will now show a bound on the number of inappropriate difficulty settings that are proposed, relative to the number of mistakes the best static difficulty setting makes on each chain. We denote the number of mistakes of our *master* algorithm until time T on chain c by m^c and the minimum number of times a statically chosen difficulty setting on chain c would have made a mistake (counting only those rounds in which chain c was chosen) until time T by M_c. Summed over all chains, this is the number of mistakes that the best static in hindsight (BSIH) makes and that we will also compare to empirically in the next section. For all $c \in \mathcal{C}$ we denote the amount of belief on every chain by $W_t^c = \sum_{x \in c} w_t(x)$.

We will first show that in each iteration in which our algorithm proposes an inappropriate difficulty setting, we update at least half of the weight of the chosen chain. That is, we will first show that $\max_{k \in c_t} \min\{A_t(k), B_t(k)\} \geq W_t^{c_t}/2$. For that, we choose

$$i = \underset{k \in c_t}{\operatorname{argmax}}\{B_t(k) \mid B_t(k) < W_t/2\}$$

and

$$j = \underset{k \in c_t}{\operatorname{argmin}}\{B_t(k) \mid B_t(k) \geq W_t/2\} .$$

This way, we obtain $i, j \in c_t$ for which $B_t(i) < W_t/2 \leq B_t(j)$ and which are consecutive, that is, $\nexists k \in c : i \prec k \prec j$. Such i, j exist and are unique as $\forall x \in \mathcal{K} : w_t(x) > 0$. We then have $B_t(i) + A_t(j) = W_t^{c_t}$ and thus also $A_t(j) > W_t/2$. This immediately implies

$$W_t/2 \leq \min\{A_t(j), B_t(j) \leq \max_{k \in c_t} \min\{A_t(k), B_t(k)\}$$

proving our original statement.

For each mistake on a chain c_t, the weight on that chain changes such that

$$W_{t+1}^{c_t} \leq W_t^{c_t}/2 + \beta W_t^{c_t}/2 = W_t^{c_t}(1+\beta)/2 .$$

We thus have $W_T^c \leq W_0^c(1+\beta)^{m_c}/2^{m_c}$. As the length of the chain is bounded by the number of difficulty settings we get

$$W_T^c \leq K(1+\beta)^{m_c}/2^{m_c} .$$

The weight of each difficulty setting is updated only if the observed response implies that this setting was not 'just right'. Therefore, β^{M_c} is a lower bound on the weight of one difficulty setting on this chain and hence $W_T^c \geq \beta^{M_c}$. Solving

$$\beta^{M_c} \leq K(1+\beta)^{m_c}/2^{m_c}$$

for m_c, we obtain

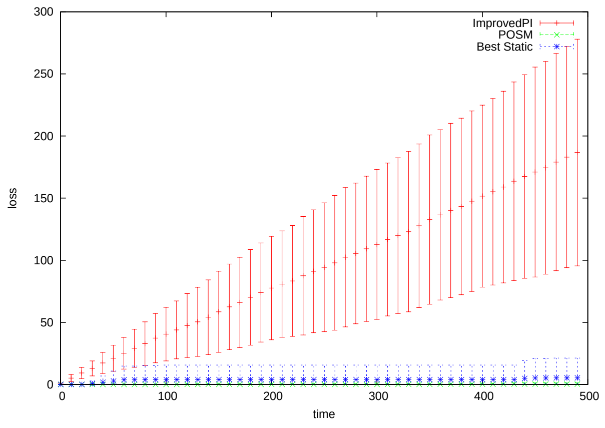
$$m_c \leq \left\lfloor \frac{\log_2 K + M_c \log_2 1/\beta}{\log_2 \frac{2}{1+\beta}} \right\rfloor .$$

Note, that this bound is essentially the same as the bound for the full information setting [3] despite much weaker information being available in our case.

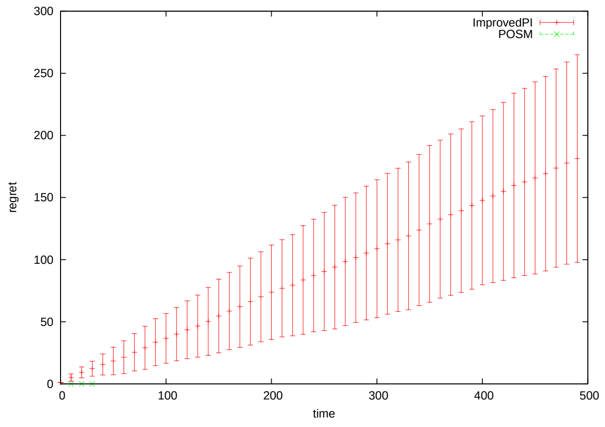
6. EXPERIMENTS

We limit our experiments to a case of a linear order on difficulty settings, in other words, a single chain. Even though it is a simplified scenario, this situation is rather natural for games and it demonstrates the power of our algorithm.

We performed two sets of experiments: simulating a game against a stochastic environment as well as using human players to provide our algorithm with a non-oblivious adversary. To evaluate the performance of our algorithm we have chosen two baselines. The first one is the best static difficulty setting in hindsight BSIH: It is the difficulty that a player would pick if she knew her skill level perfectly well in advance and could choose the difficulty only once. As it uses information in hindsight, BSIH has an 'unfair' advantage over any 'real' and fair algorithm. Therefore we also implemented and compare to a state-of-the-art fair competitor, the IMPROVEDPI algorithm [3].



(a) Loss.

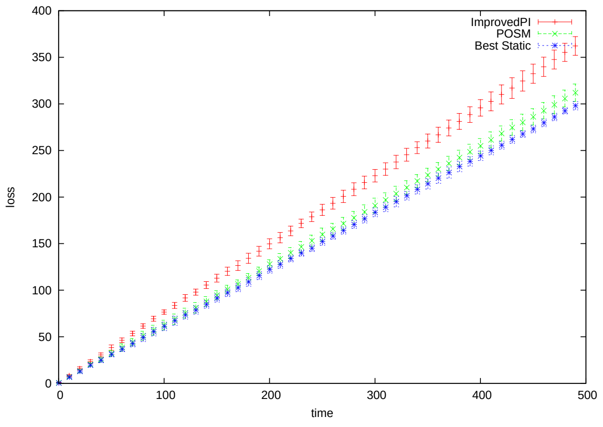


(b) Regret.

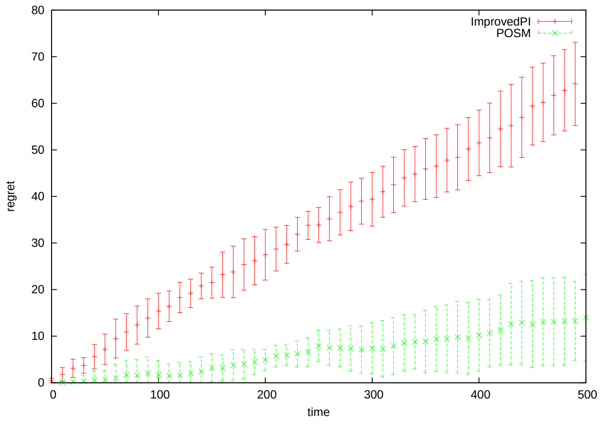
Figure 1: Stochastic adversary, 'smooth' setting

The labels form three distinct zones on this chain: first there are all states that are 'too easy', then come all the states that are 'just right', and then all the states that are 'too difficult'. Let us call the set of vertices with 'just right' labels the zero-zone (because in the corresponding loss vector their components are equal to zero).

In both, the stochastic and adversarial scenarios, we consider two different settings, 'smooth' as well as 'non-smooth' changes. They differ in the way the borders of the zero-zone change over time. In the 'non-smooth' setting we don't place any restrictions on them, while in the 'smooth' setting these borders are allowed each to move only by one vertex at a time. These two settings represent two extreme situations: one player changing her skills gradually with time is changing the zero-zone 'smoothly'; different players with different skills for each new challenge will make the zero-zone jump all over the chain. In a more realistic scenario the zero-zone would change 'smoothly' most of the time, but sometimes it would perform jumps (representing a change of the player). We will consider this 'mixed' setting in our future work.



(a) Loss.



(b) Regret.

Figure 2: Stochastic adversary, 'non-smooth' setting

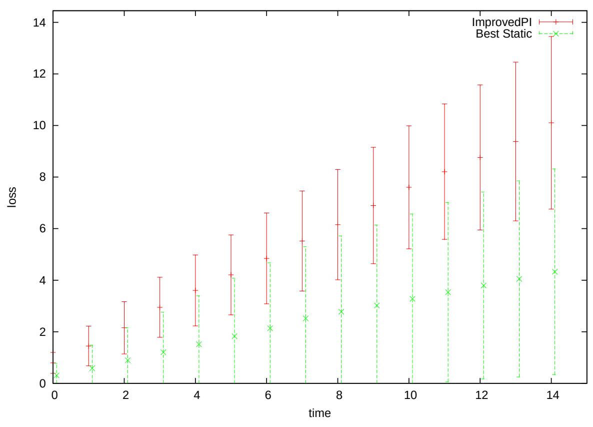
6.1 Stochastic Adversary

In the first set of experiments we performed, the adversary is stochastic: On every iteration each of the borders of the zero-zone changes with probability 1/2. In both, the 'smooth' and the 'non-smooth' setting, the game is played on a chain of length 100 for 500 iterations. The game is repeated 10 times. The resulting mean and standard deviation of loss and regret are shown in Figure 1 for the 'smooth' setting and in Figure 2 for the 'non-smooth' setting.

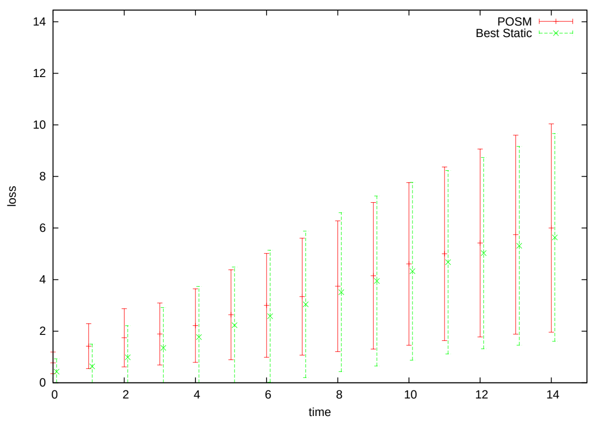
Note that in the 'smooth' setting Posm is outperforming Bsih (the best static arm chosed in hindsight) and, therefore, its regret is negative. Furthermore, in the more difficult 'non-smooth' setting, Posm performance is still very close to that of Bsih. Both of these results demonstrate that in a stochastic environment Posm is a good choice to predict a vertex from the 'just right' zone. While Bsih is a baseline that can not be implemented as it requires to foresee the future, Posm is a correct algorithm for dynamic difficulty adjustment. Therefore it is surprising that Posm performs almost as good as Bsih or even better.

6.2 Evil Adversary

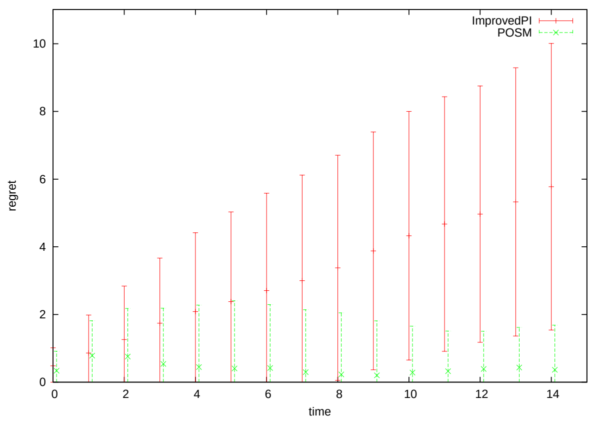
While the experiments in our stochastic environment show encouraging results, of real interest to us is the situation



(a) Games vs ImprovedPI, loss values.



(b) Games vs Posm, loss values of posm.

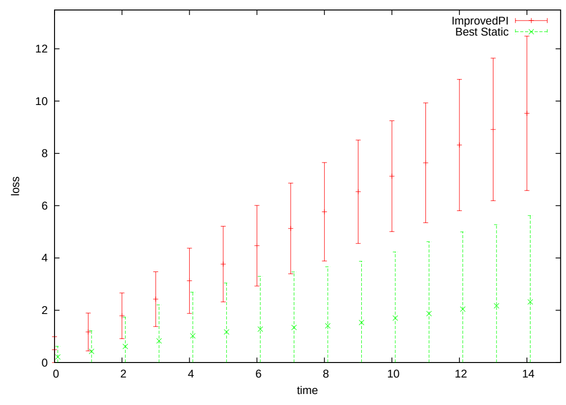


(c) Regret.

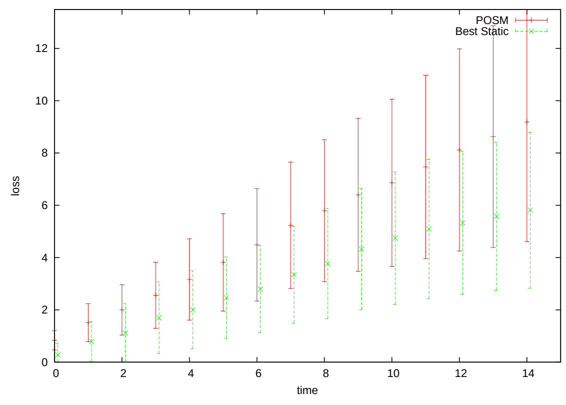
Figure 3: Evil adversary, 'smooth' setting.

where the adversary is 'evil', non-stochastic, and furthermore, non-oblivious. In dynamic difficulty adjustment the algorithm will have to deal with people, who are learning and changing in hard to predict ways.

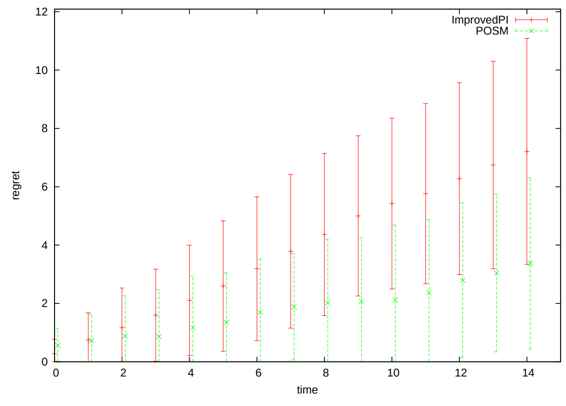
To simulate this situation, we decided to use people as adversaries. Just as in dynamic difficulty adjustment players are not supposed to be aware of the mechanics, our methods and goals were not disclosed to the testing persons. Instead



(a) Games vs IMPROVEDPI, loss values.



(b) Games vs POSM, loss values.



(c) Regret.

Figure 4: Evil adversary, 'non-smooth' setting.

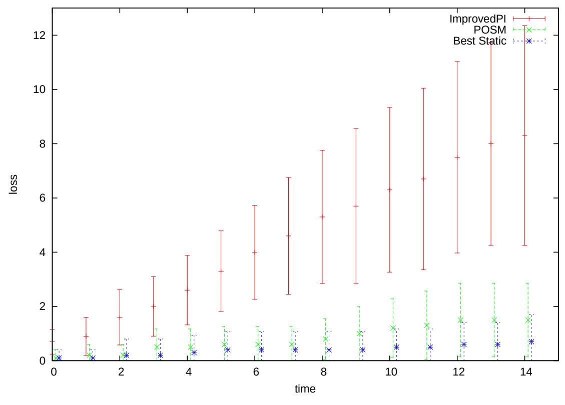
they were presented with a modified game of cups: On every iteration the casino is hiding a coin under one of the cups; after that the player can point at two of the cups. If the coin is under one of these two, the player wins it.

Behind the scenes the cups represented the vertices on the chain and the players' choices were setting the lower and upper borders of the zero-zone. If the algorithm's prediction was wrong, one of the two cups is picked at random and the coin is placed underneath. If the prediction was correct, no coin was awarded.

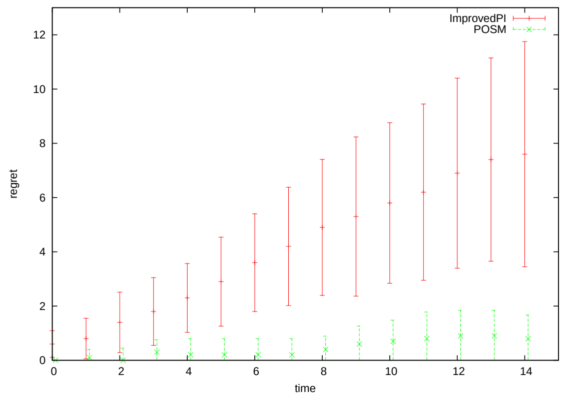
Unfortunately, using people in such experiments places severe limitations on the size of the game. They can only handle short chains and short games before getting bored. In our case we restricted the length of the chain to 8 and the length of each game to 15.

Again, we created the 'smooth' and 'non-smooth' setting by placing or removing restrictions on how players were allowed to choose their cups. To each game either IMPROVEDPI or POSM was assigned. The results for the 'smooth' setting are in Figure 3 and the results for the 'non-smooth' setting are in Figure 4. Note, that due to the fact that this time different games were played by IMPROVEDPI and POSM, we have two different plots for their corresponding loss values.

We can see that in the 'smooth' setting again the performance of POSM is very close to that of BSIH. In the more difficult 'non-smooth' one the results are encouraging. Note, that the loss of BSIH appears to be worse in games played by POSM. A plausible interpretation is that players had to follow more difficult (less static) strategies to fool POSM to win their coins. Nevertheless, the regret of POSM is small even in this case.



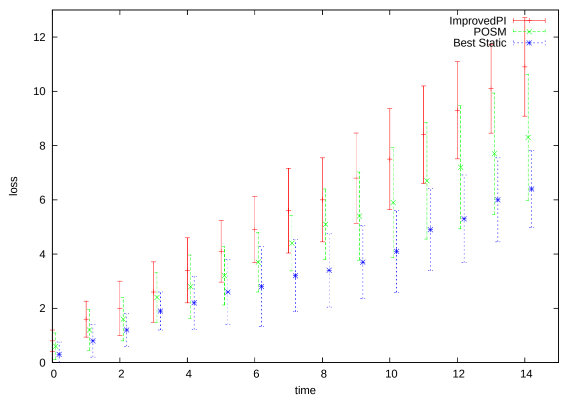
(a) Loss.



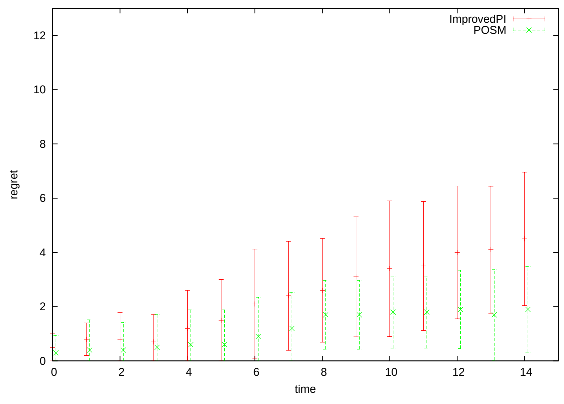
(b) Regret.

Figure 5: Stochastic adversary, 'smooth' setting, short game

For comparison we also show the performance of all algorithms vs stochastic adversary in the games of length 15 in Figures 5 and 6.



(a) Loss.



(b) Regret.

Figure 6: Stochastic adversary, 'non-smooth' setting, short game

7. CONCLUSIONS

In this paper we formalised dynamic difficulty adjustment as a prediction problem on partially ordered sets and proposed a novel online learning algorithm Posm for dynamic difficulty adjustment. Using this formalisation, we were able to prove a bound on the performance of Posm relative to the best static difficulty setting chosen in hindsight Bsih. To validate our theoretical findings empirically, we performed a set of experiments, comparing Posm and another state-of-the-art algorithm to Bsih in two settings (a) simulating the player by a stochastic process and (b) simulating the player by humans that are encouraged to play as adverserially as possible. These experiments showed that Posm performs very often almost as well as Bsih and, more surprisingly, sometimes even better. As this is better than the behaviour suggested by our mistake bound, there seems to be a gap between the theoretical and empirical performance of our algorithm. In future work we will on the one hand investigate this gap, aiming to provide better bounds by, perhaps, making stronger but still realistic assumptions. On the other hand, we will implement Posm in a range of computer games as well as teaching systems to observe its behaviour in real application scenarios.

8. ACKNOWLEDGEMENTS

Part of this work was supported by the German Science Foundation (DFG) under 'GA 1615/1-1'. The authors thank their co-workers and reviewers for their valuable comments and suggestions. Special thanks also go to all 'adversaries'.

9. REFERENCES

[1] J. Abernethy, E. Hazan, and A. Rakhlin. Competing in the dark: An efficient algorithm for bandit linear optimization. 2008.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:322, 1995.

[3] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[4] N. Cesa-Bianchi, Y. Mansour, and G. Stoltz. Improved second-order bounds for prediction with expert advice. *Machine Learning*, 66:321–352, 2007. 10.1007/s10994-006-5001-7.

[5] D. Charles and M. Black. Dynamic player modeling: A framework for player-centered digital games. In *Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, 2004.

[6] V. Dani and T. P. Hayes. Robbing the bandit: less regret in online geometric optimization against an adaptive adversary. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 937–943, New York, NY, USA, 2006. ACM.

[7] G. Danzi, A. H. P. Santana, A. W. B. Furtado, A. R. Gouveia, A. Leitão, and G. L. Ramalho. Online adaptation of computer games agents: A reinforcement learning approach. *II Workshop de Jogos e Entretenimento Digital*, pages 105–112, 2003.

[8] T. Gärtner and G. C. Garriga. The cost of learning directed cuts. In *Proceedings of the 18th European Conference on Machine Learning*, 2007.

[9] R. Herbrich, T. Minka, and T. Graepel. Trueskill$^{\text{tm}}$: A bayesian skill rating system. In *NIPS*, pages 569–576, 2006.

[10] R. Hunicke and V. Chapman. AI for dynamic difficulty adjustment in games. *Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence*, 2004.

[11] H. McMahan and A. Blum. Online geometric optimization in the bandit setting against an adaptive adversary. In J. Shawe-Taylor and Y. Singer, editors, *Learning Theory*, volume 3120 of *Lecture Notes in Computer Science*, pages 109–123. Springer Berlin / Heidelberg, 2004.

[12] O. Missura and T. Gärtner. Player Modeling for Intelligent Difficulty Adjustment. In *Discovery Science*, pages 197–211. Springer, 2009.

[13] J. Togelius, R. Nardi, and S. Lucas. Making racing fun through player modeling and track evolution. In *SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, pages 61–70, 2006.