Upper Confidence Trees with Short Term Partial Information

Olivier Teytaud, Sébastien Flory

Upper Confidence Trees with Short Term Partial Information

Olivier Teytaud[1] and Sébastien Flory[2]

[1] TAO, Lri, Inria Saclay-IDF, UMR CNRS 8623, Université Paris-Sud
[2] Boostr

Abstract. We show some mathematical links between partially observable (PO) games in which information is regularly revealed, and simultaneous actions games. Using this, we study the extension of Monte-Carlo Tree Search algorithms to PO games and to games with simultaneous actions. We apply the results to Urban Rivals, a free PO internet card game with more than 10 millions of registered users.

1 Introduction

The impact of partial observability in games and planning is studied in several papers, showing in particular that

- Just one player and a random part makes the problem undecidable, even with a finite state space with reachability criteria[10].
- With two players with or without random parts, the problem is EXP, EXPSPACE, 2EXP (i.e. exponential time, exponential space, doubly-exponential time) for the fully observable, no observation, and partially observable case respectivelly for the criterion of deciding whether a 100% winning strategy exists[1]. With exponential horizon, the complexities decrease to EXP, NEXP, EXPSPACE respectively [11].
- With two players without random part, the problem of approximating the best winning probability that can be achieved regardless of the opponent strategy is undecidable[14] by reduction to the one-player randomized case above in the no observation case; the best complexity upper bounds for bounded horizon are 3EXP (for exponential horizon) and 2EXP (for polynomial horizon).

Section 2 presents the frameworks used in this paper: games, acyclic games, games with simultaneous actions, games with hidden information. Section 3 presents a brief overview of computational complexity in games, and provides some new results around the framework of short term hidden information and games with simultaneous actions. Section 4 presents a variant of the Upper Confidence Tree algorithm for games with simultaneous actions. Section 5 presents experimental results on Urban Rivals, a free and widely played game with partial information.

[1] The random part has no impact because we look for a strategy winning with probability 100 %.

2 Frameworks

We consider finite games, represented by finite directed graphs. Each node, also termed a state, is equipped with an observation for player 1, and an observation for player 2. Each state is either of the form $P1$ (meaning that player 1 chooses the next state as a function of his previous observations), or of the form $P2$ (meaning that player 2 chooses the next state as a function of his previous observations), or randomized (the next state is randomly drawn among the states proposed by the directed graph), or simultaneous (both players choose an action as a function of their previous observations and the next state is chosen as a function of these two actions). All actions are chosen from finite sets. A node is fully observable if there's no other node with the same observation for player 1 and no other node with the same observation for player 2. There are leafs which are a win for player 1, leafs which are a win for player 2, leafs which are a draw, and infinite loops are a priori possible. Nodes which are a draw or a win for a player are all leaf nodes; these nodes are fully observable. A game is turn-based if there's no simultaneous actions in it. **Examples:** (i) The rock-paper-scissor game has one node with simultaneous actions, and leafs. (ii) Chess, Draughts, Go, are games with no simultaneous actions. (iii) Bridge, Poker, Scrabble, are games with no simultaneous actions and partial observation. (iv) Urban Rivals is a turn-based game with hidden information; it can be rewritten as a game with no partial information but with simultaneous action (this will be detailed in this paper). (v) The strategies of American football are simultaneously chosen and kept private some time. (vi) In the Pokemon card game (as well as in many similar card games), both players choose their deck simultaneously.

It is known that with no restriction, this setting is undecidable (even if the graph is finite). For example, [10] has shown that with one player only, no observation, random nodes, the probability of winning, when starting in a given node, and for an optimal strategy, is not computable, and even not approximable. [14] has shown that this also holds for two players and no random node.

Some important restrictions simplifying the analysis of games are as follows:

- **Looking for strategies winning with probability** 1 **is much easier**. The existence of strategies winning with probability 1, independently of the opponent, is decidable for 2 players, even in partially observable environments (see [6], showing that this is not true if we have a team of 2 players against a third player).
- **The fully observable setting is always decidable, with complexity reduced by far in the case of limited horizon**; see [13, 11] for more on this for the case in which we consider the existence of strategies winning with probability 1 and [14] for the choice of optimal moves.

In this paper we will investigate the effect of two other possible assumptions: (i) no partial observability, but simultaneous actions; (ii) partial observability, but with hidden information which becomes visible after a bounded number of time steps; these two conditions will be shown nearly equivalent and we will also show that with limited horizon these assumptions have a big impact.

2.1 Bounded Horizon Hidden Information Games (BHHIG)

We define games in BHHIG(H) as games (in the sense above) verifying the following assumptions: (i) the graph is finite; (ii) each node is visited at most once (acyclic graph); (iii) there's no random node and no node with simultaneous actions; (iv) there's no string of length H in the graph containing no fully observable node. The crucial assumption here is the last one.

Remark: We here forbid random nodes. This is in fact not necessary in our analysis and in our algorithms, but it will simplify the discussion.

2.2 Games with Simultaneous Actions (GSA)

We define games in GSA as games (in the sense above) verifying the following assumptions: (i) there's no random node; (ii) there's no partially observable node; (iii) but nodes with simultaneous actions are allowed. The crucial assumption is here the presence of nodes with simultaneous actions. Without such nodes, the solving of such games is well known (see [11, 13, 12] for more on this).

Remark: We here forbid random nodes. This is in fact not necessary in our analysis and in our algorithms (and random nodes can be simulated by nodes with simultaneous actions), but it will simplify the discussion.

2.3 Representing BHHIG as GSA and GSA as BHHIG

In this section, we show a correspondence between GSA and BHHIG(H).

A BHHIG(H) is a GSA. We consider a game G in BHHIG(H), and show how to rewrite it as a game in GSA.

We consider a fully observable node n of G. By the crucial assumption on BHHIG(H), all paths starting at G reach another fully observable node after a length at most H. Let G' be the subset of G covered by these paths (the root is n and leafs are fully observable nodes). Let

- S_1 be the finite set of deterministic strategies that can be chosen by player 1 before reaching another fully observable node and let
- S_2 be the finite set of deterministic strategies that can be chosen by player 2 before reaching another fully observable node.

Then, the subgraph G' can be replaced by a simultaneous node (player 1 chooses a strategy in S_1 and player 2 chooses a strategy in S_2), and the leafs of G'; we then get a node with simultaneous actions. We can do this for all fully observable nodes, and then all partially observable nodes will be removed; this concludes the proof.

A GSA is a BHHIG(H). We consider a game G in GSA, we show that it can be encoded as a BHHIG(H). For this, we just have to encode a node with simultaneous actions as two turns with partial observability, before reaching, again, a node with full observability (therefore the game is BHHIG(2), i.e. $H = 2$). The idea is that one player chooses his action a_1; this action a_1 is not seen by the other player who then chooses a_2; and then both players observe the two actions. This concludes the proof. An example is given in Fig. 1: Rock-Paper-Scissor is classically understood as a player game with simultaneous play, and is here presented as a partially observable turn-based game.

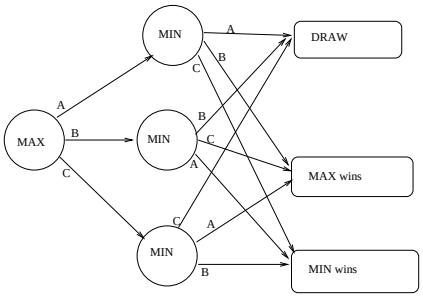

Fig. 1. The Rock-Paper-Scissors game, presented as a partially observable turn-based game: A=rock, B=scissors, C=paper. Here, Min does not see the action chosen by Max; this is clearly equivalent to the classical formulation by simultaneous actions.

3 Complexity of games with simultaneous actions

We have seen how to rewrite a BHHIG(H) as a GSA; we here discuss the complexity of GSA. In order to formalize this complexity, we will consider any representation of a game such that for a polynom $p(.)$;

- A state is described with size $p(n)$;
- For each player, there are at most $p(n)$ legal actions in a given state, and they can be all computed in time $p(n)$;
- The transition from a state and a pair of actions to a new state takes time at most $p(n)$;
- The number of possibles states is $O(\exp(p(n)))$.

The class GSA depends on the chosen polynom p.

Then we claim the following:

Theorem: *Consider a GSA with acyclic graph. Then, the optimal move can be computed in time EXP.*

Proof: The sketch of the proof is as follows. We can sort the nodes in reverse topological order. Then each Bellman value (Nash value, if you prefer) of a node

is computed by solving the matrix game associated to actions in that node, if all Bellman values of later nodes are already known. As the number of nodes is exponential and each matrix game can be solved in polynomial time by linear programming, the overall algorithm solves the problem in time EXP.□

4 Upper Confidence Trees for games with simultaneous actions

We assume in this section that the reader is familiar with the Monte-Carlo Tree Search (MCTS) and Upper Confidence Tree (UCT) literature[4, 7, 9]. We here focus on the experimental application of MCTS to acyclic GSA games.

4.1 The Upper Confidence Tree algorithm

We briefly recall the UCT algorithm in Algo. 1.

Algorithm 1 The UCT algorithm in short.

UCT algorithm.
Input: a game, a state S, a time budget.
Output: an action a.
while Time not elapsed **do**
  $s = S$. // starting a simulation.
  while s is not a terminal state **do**
    Define the score of a legal action a in s as the sum of:
      • its exploitation score: the average reward of past simulations using action a in state s;
      • its exploration score: $\sqrt{\frac{\log(n(s)+2)}{n(s,a)+1}}$, where
        • $n(s)$ is the number of past simulations crossing state s;
        • $n(s,a)$ is the number of past simulations applying action a in state s.
    Choose the action a which has maximum score.
    Let s' be the state reached from s when choosing action a.
    $s = s'$
  end while
  // the simulation is over; it starts at S and reaches a final state.
  Get a reward $r = Reward(s)$ // s is a final state, it has a reward.
  For all states s in the simulation above, let $r_{nbVisits(s)}(s) = r$.
end while
Return the action which was simulated most often from S.

The reader is referred to [7] for more information on UCT; we here focus on the extension of UCT to games with nodes with simultaneous actions, i.e. GSA, in the acyclic case.

4.2 Adapting UCT to the GSA acyclic case

We adapt UCT to acyclic GSA as follows. We use the EXP3 algorithm for GSA nodes (variant of the Grigoriadis-Khachiyan algorithm[5, 2, 1, 3]), leading to a probability of choosing an action of the form $\eta + \exp(\epsilon s)/C$ where η and ϵ are

Algorithm 2 Adapting the UCT algorithm for GSA cases.

UCT algorithm for GSA problems.
Input: a game, a state S, a time budget.
Output: an action a (for each player if the root is in P12, for the player to play otherwise).
Initialize s_1 and s_2 at the null function (equal to 0 everywhere).
while Time not elapsed **do**
  // starting a simulation.
  $s = S$.
  while s is not a terminal state **do**
    if s is a P1 or P2 node **then**
      Define the score of a legal action a in s as in UCT.
      Choose the action a which has maximum score.
      Let s' be the state reached from s when choosing action a.
    else
      Choose action a_1 for player 1 randomly, with action a chosen
        with probability $p_1(a_1, s) = \eta + \exp(\epsilon s_1(a_1, s))/C_1$.
        (C is a normalization so that the sum is 1)
      Choose action a_2 for player 2 randomly, with action a chosen
        with probability $p_2(a_2, s) = \eta + \exp(\epsilon s_2(a_2, s))/C_2$.
        (C is a normalization so that the sum is 1)
      Let s' be the state reached from s when choosing actions a_1, a_2.
    end if
    $s = s'$
  end while
  // the simulation is over; it starts at S and reaches a final state.
  Get a reward $r = Reward(s)$ // s is a final state, it has a reward.
  For all states s in the simulation above, let:

$$s_1(a_1, s) = s_1(a_1, s) + r/p_1(a_1, s),$$
$$s_2(a_2, s) = s_2(a_2, s) + r/p_2(a_2, s).$$

end while
if The root is in 1P or 2P **then**
  Return the action which was simulated most often from S.
else
  Choose action a with probability proportional to its number of simulations.
end if

parameters, s is the estimated sum of rewards for the considered action, and C is the normalization constant. The algorithm is presented in Alg. 2.

We'll see later how to choose ϵ and η; C_1, C_2 are normalization constants (so that the sum of the probabilities of the actions is 1). We did not consider random nodes here, but they could easily be included as well. We do not write explicitly a proof of the consistency of these algorithms, but we guess that the proof is a consequence of properties in [5, 8, 2, 1]. We'll see the choice of constants below.

5 Experiments

We discuss below various experiments we performed for validating or improving our implementation. We compared EXP3 to simpler formulas. We then tested the scalability of the implementation (Section 5.2).

The program was then launched on the website, for playing against humans (Section 5.3). Please keep in mind, in all this section, that for a game like Urban Rivals, based on guessing the opponent's strategy, results on one single game

are noisy; as well as in Poker, it does not make sense to have 80 % of winning rate (as we can see in Go). The numbers we get (average results for one game) are therefore always close to 50%; nonetheless, when considering reasonably long sequences of games, they provide very significant improvements.

5.1 The EXP3 algorithm

We refer to [1] for an introduction to the EXP3 algorithm and variants.

EXP3 vs an ϵ-greedy algorithm. We compared EXP3 as in [1] to a simple η-greedy algorithm, choosing

- any move, randomly and uniformly, with probability $\eta = \min(1, 1.2\sqrt{K/t})$ with K the number of possible actions.
- the move with highest average reward otherwise (when the move is not simulated, it has an infinite average reward).

The probability of the random exploration ($\eta = \min(1, 1.2\sqrt{K/t})$) is chosen in order to match exactly the probability of random exploration in our EXP3 version above. Results were as follows:

Number of simulations per move	Winning rate of the tuned version against the η-greedy-version $\pm$ 2 standard deviation
400	75% $\pm$ 5
800	73% $\pm$ 5
1600	68% $\pm$ 4
3200	72% $\pm$ 4
6400	70% $\pm$ 4

EXP3+UCT vs UCT alone. Our algorithm is based on using EXP3 in nodes with simultaneous actions and UCT in other nodes; this dichotomy is intuitively quite reasonable. However, what happens if we just consider UCT-like formulas everywhere ? We first tested what happens if we replace EXP3 by a simple UCT algorithm for each player, even in nodes with simultaneous actions. We just used the UCT formula with constants as used in nodes with no simultaneous actions. We got 45.8% $\pm$ 1.4% as a success rate against the EXP3 version with 10 000 simulations per move, after having added some random exploration with a fixed probability (otherwise results were very poor) - so with random exploration, UCT is not so far from EXP3 (yet, EXP3 has the advantage, with a speed-up around 2 if we trust the scalability analysis below, and results with UCT could only be obtained at the price of the tuning of a random exploration whereas EXP3 is tuned according to [1]).

Pruning the exploration in EXP3. In UCT-like algorithms, the optimal moves are chosen exponentially more often than other moves. As a consequence, a bandit in UCT can recommend, when all simulations are over, any move with maximal number of simulations - this is clearly consistent. EXP3 has a different goal; as it considers an adversarial case (for us, nodes with simultaneous actions), it must not outputs a single move as a decision, but several moves with their

associated probabilities - this is (in the general case) a mixed strategy, and, unless the game has the particularity of having pure Nash equilibria, there's no good recommendation strategy outputting deterministically a single move.

The standard property of EXP3 is that the Nash is approximated by the empirical frequency; action i should be played with probability proportional to the number of simulations of action i. However, a part of the simulations are pure random exploration (this is the η parameter); could we remove this from the result, before extracting the Nash approximation ? Asymptotically, this effect is negligible, but is there something to win, non-asymptotically ?

In order to test this, we designed a formula sublinear in the maximum number max of simulations of the actions in the root, namely $t' = t^{0.95}$, and kept only actions with a number of simulations at least t'. Results were as follows:

Number of simulations per move	Winning rate of the "pruned" version ± 2× std deviations
50	54.0% ± 4%
100	52.2% ± 4%
200	56.7% ± 4%
400	56.3% ± 4%
800	57.5% ± 4%
1600	58.7% ± 4%
3200	54.6% ± 4%

Results are significant as we here have doubled standard deviations and not standard deviations. The choice of the 0.95 exponent was our first random guess, maybe we can have improvements by a careful tuning. A subtle point must be pointed out, here. These experiments are conducted against our EXP3+UCT algorithm; this is an algorithm which tries to play the Nash equilibrium. Playing against a "Nash" opponent has the advantage that the opponent can not "learn" our weaknesses; therefore, the good results above might hide the fact that our player is less randomized than the original one, and therefore maybe it is possible for a non-Nash opponent to "learn" our (non-asymptotic) lack of randomization. Testing this is difficult however, and we did not see a tendency in this direction from the games we have seen.

Conclusion. We have seen that, on Urban-Rivals, the combination EXP3+UCT works better than UCT+ϵ-greedy algorithms, and significantly better than UCT alone. We could slightly improve the AI by implementing some ideas, and a bit more by brute-force tuning.

5.2 Scalability

We tested the scalability, i.e. the capacity of the program to become stronger when the computation time increases, by testing 2N simulations per move against N simulations per move. We get a constant improvement until 3200 simulations per move. Usually UCT-related programs have a decrease of this quantity; maybe we just did not try with sufficiently many simulations. Results are as follows:

N	Success rate of $2N$ simulations per move versus N simulations per move ± 2 standard deviations
50	0.546 ± 0.03
100	0.556 ± 0.03
200	0.5475 ± 0.03
400	0.594 ± 0.03
800	0.5545 ± 0.03
1600	0.5725 ± 0.03
3200	0.5565 ± 0.03

Fig. 2. Examples of Urban Rivals characters. Characters have different abilities: strong attack (better probability of winning the turn); better strength (more damages in case of won turn). The crucial points is how many "pilz" you use per turn: "more pilz" implies a better probability of winning; the key point is that the choice is the number of pilz is made privately until the end of the turn. At the end of each turn all the hidden information is revealed.

5.3 Games against humans

Urban Rivals (Fig. 2) has 11 millions of registered users. It's a Card Game, related to games like Pokemon or Magic, with Partial Observability, a small number of turns leading to fast games (often less than a minute)[2]. First, each player chooses a deck, which contains four cards (see Fig. 2 for a few examples). The decks are chosen privately, but then shown to the opponent. Each card is equipped with a default strength (a stronger card is more likely to win a fight) and a default power (a card with more power makes more damages to the opponent). At each turn (out of four), one of the players (alternatively) chooses (publicly) one of his four cards, and then chooses (privately) the strength of the attack; the other player chooses publicly one of his cards and the strength. The strength does not come for free - each point is taken from a finite quantity. There is a strong bluff component in Urban Rivals, similarly to Poker: one might use a card with little strength so that the opponent wastes strength. With 200 000 simulations per move, the program reached 1240 ELO the 30th of November, i.e. the top 1.5%, but then decreased to 1144 ELO, i.e. the top 9% ; the precise rank is probably between these two values. A second run after technical improvements the 13th of December is ranked 84^{th} on 8030 players (top 1%) and is still improving.

6 Conclusion

UCT is a major breakthrough in Markov Decision Processes, and PO games are a great challenge. The general case of PO games is undecidable but we here propose a sound extension of UCT to an important subclass of PO games, including games with bounded horizon and simultaneous actions. The resulting algorithm outperformed UCT at Urban-Rivals, and was well ranked on the ELO scale. A further work is the analysis of the parametric complexity (function of H) in BHHIG(H); Urban-Rivals is a nice case thanks to a small H.

[2] A few options are removed from this short description, but they are taken into account in the implementation and do not change the principle.

On the application side, we have not yet a clear understanding of how many games are BHHIG(H) for a reasonnable value of H; "mister X" is a natural other examples. Also, as using a complete memory of observations is probably not that useful, we might consider to which extent usual PO games can be approximate by BHHIG(H) games.

Acknowledgements

The authors are grateful to Grid5000 for providing computational resources around the parallelization of MCTS, and to Dagstuhl and Birs for fruitful seminars.

References

1. J.-Y. Audibert and S. Bubeck. Minimax policies for adversarial and stochastic bandits. In *proceedings of the Annual Conference on Learning Theory (COLT)*, 2009.
2. P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, Los Alamitos, CA, 1995.
3. B. Bouzy and M. Métivier. Multi-agent learning experiments on repeated matrix games. In *ICML*, pages 119–126, 2010.
4. R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, pages 72–83, 2006.
5. M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, Sep 1995.
6. R. A. Hearn and E. Demaine. *Games, Puzzles, and Computation.* AK Peters, 2009.
7. L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.
8. T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
9. C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, 2009.
10. O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34, 2003.
11. M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon markov decision process problems. *J. ACM*, 47(4):681–720, 2000.
12. C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processses. *Mathematics of Operations Research*, 12(3):441–450, 1987.
13. J. Rintanen. Complexity of Planning with Partial Observability. In *Proceedings of ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*, Trento, Italy, June 2003.
14. O. Teytaud. Decidability and complexity in partially observable antagonist coevolution. In *Proceedings of Dagstuhl's seminar 10361*, 2010.