Consistent Belief State Estimation, with Application to Mines

Adrien Couetoux, Mario Milone, Olivier Teytaud

Consistent Belief State Estimation, with Application to Mines

Adrien Couëtoux[1], Mario Milone[1], Olivier Teytaud[1,2]
1. TAO, Inria, Lri, Univ. Paris-Sud, UMR CNRS 8623, France
2. Dept. of Computer Science and Information Engineering, National University of Tainan, Taiwan

***Abstract*—Estimating the belief state is the main issue in games with Partial Observation. It is commonly done by heuristic methods, with no mathematical guarantee. We here focus on mathematically consistent belief state estimation methods, in the case of one-player games. We clearly separate the search algorithm (which might be e.g. alpha-beta or Monte-Carlo Tree Search) and the belief state estimation. We basically propose rejection methods and simple Monte-Carlo Markov Chain methods, with a time budget proportional to the time spent by the search algorithm on the situation at which the belief state is to be estimated; this is conveniently approximated by the number of simulations in the current node. While the approach is intended to be generic, we perform experiments on the well-known Mines game, available on most Windows and Linux distributions. Interestingly, it detects non-trivial facts, e.g. the fact that the probability of winning the game is not the same for different moves, even those with the same probability of immediate death. The rejection method, which is slow but has no parameter and which is consistent in a non-asymptotic setting, performed better than the MCMC method in spite of tuning efforts.**

***Index Terms*—Partially Observable Markov Decision Processes; Belief State Estimation; Monte-Carlo Tree Search; Upper Confidence Trees; Mines game.**

I. Introduction: belief state estimation, from Mines to mathematics

In Section I-A we present the Mines games, which is convenient as an experimental testbed and for introducing notations. In Section I-B we present the formalism of Markov Decision Processes and Partially Observable Markov Decision Processes. In Section I-C we formalize the problem of belief state estimation.

A. The Mines game

Consider the simple Mines game, which is well known for his free clones on several platforms (e.g. on Linux or Windows). Mines look like a simple game, sometimes equipped with a "hint" system, which helps you for finding a good move. However, this hint system is usually unable to solve more than simple cases, or cheats by using hidden information. Indeed, an exact choice of optimal move is far from being simple.

The rules of the game are as follows. This is a one player game. There are N mines, located randomly on a $p \times q$ grid. Mines are not visible. At each move, the player chooses one location in the grid. If there is no mine at this location, then the number of mines in the 8-neighborhood is displayed to the player; otherwise, the game is over. Usually, the score is the time before complete solving (i.e. all non-mines locations are played at least once) - the smaller, the better. No score in case of game over by playing on a mine.

We will here consider as score the number of (unique) moves before the game is over, divided by $pq - N$ (so that 1 means complete solving). Figure 1 shows that the game is not trivial: which move would you play here ?

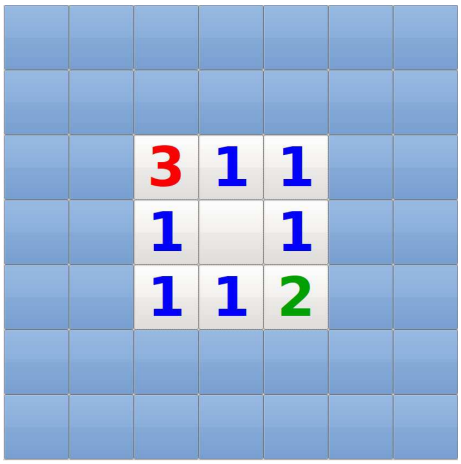

Fig. 1. A case in which choosing the next move is non-trivial.

All players know many cases in which a move can be played for free, because it is sure that there is no mine here - and many players know that sometimes, you can not be sure and must play with a non-zero probability of loosing the game. But not many players know that in such cases, sometimes, some locations are more likely than others to be mines. Specific examples, as preliminaries for the discussion on Belief State estimation below, are given in Fig. 2.

Some details on the rules: The Windows version of the game ensures that the first move is not on a mine (the mines are randomly distributed after the first move). Some versions on the game (Gnomine, default version on many Linux distributions) moreover ensure that the number of mines in the 8-neighborhood is 0. In our tests we will only consider the Gnomine version, which makes the game more subtle (the player is less likely to loose very early without having any chance of playing anything subtle).

B. Markov Decision Processes and Partial Observation

Let's now consider Markov Decision Processes (MDP) and Partially Observable MDP (POMDP) from a more abstract point of view.

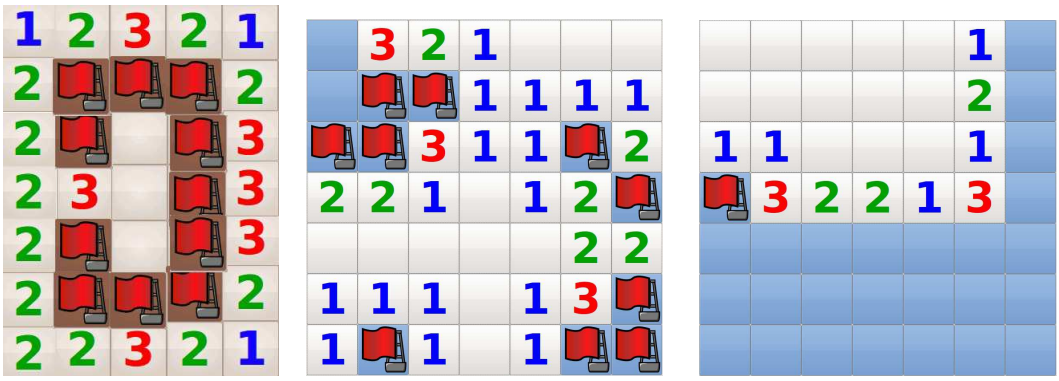

Fig. 2. Left: here, you can deduce that one of the remaining locations is a mine. If you play in the middle location, you have an expected number of (unique) moves before loosing which is, if you play perfectly, 1 (the three outcomes, loosing immediately, or loosing after 1 move, or completely solving the game, are equally likely) - whereas playing the top or bottom unknown location gives an expected number of (unique) moves $4/3$ (with probability $\frac{1}{3}$, it's an immediate loss - otherwise, it's a complete solving). Middle: a situation with 50% probability of winning. Right: this situation is difficult to analyze mathematically; our program immediately sees that the top-right location (0,6) is a mine. However, this could easily (and faster) be found by a branch-and-bound optimization. More importantly, it also sees that the location just below (1,6) is good; but the real good point is that it can say which locations are more likely to lead to a long-term win than others.

A MDP or POMDP is a directed graph, each edge of each being equipped with a label (action) and a number (probability); the edge between vertex x and vertex y has label m and number r if the probability of switching from state x to state y, when choosing action m, is r. The sum of probabilities in outgoing edges from a state x and equipped with a label m should be 1. By definitions, legal actions in x are labels which can be found on at least one outgoing edge from x. Nodes with no outgoing edge are termed terminal nodes and are equipped with a reward (in some definitions, edges are equipped with a reward, or non-terminal nodes; this will make no difference for this work).

Additionally, in a POMDP, each node x is equipped with an observation o_x. A MDP can be seen as a POMDP in which the observation is equal to the state or to a unique state identifier. A (possibly random) sequence of actions $a_1, \ldots, a_t, \ldots$ defines naturally a random sequence of states $x_1, \ldots, x_t, \ldots$ and observations $o_1, \ldots, o_t$; each state and observation is obtained from the previous one using actions and probabilities of transition.

A strategy is a (possibly random) mapping from a sequence of pairs (actions,observations) to actions. In a MDP, observations are uniquely determined as a function of states; so, equivalently, the strategy in a MDP can be defined as a mapping from sequences of pairs (actions,states) to actions (one can note that it is known that the mapping can depend only on the last state without loss of performance in optimal strategies). Basically and informally, a MDP is therefore a POMDP in which the player always knows in which state she is. Given a MDP or POMDP, and a strategy, a random sequence of states and observations is defined, as well as a random reward. The Mines game is a priori a POMDP; mines are not visible. Yet, it can be rephrased as a MDP; we'll see this in the next section.

C. POMDP transformed into MDP: the problem of belief state estimation

A POMDP can be rephrased as a MDP. This can be done as follows. Consider a POMDP P. The MDP M rephrasing P is built as follows:

- The state space of M is the set of sequences of pairs (actions,observations) in P;
- In M, action a in state $((a_1, o_1), \ldots, (a_t, o_t))$ leads to $((a_1, o_1), \ldots, (a, o_{t+1}))$ where o_{t+1} is the random observation obtained when applying action a in state s where s is the t^{th} state, randomly drawn according to observations $(o_1, \ldots, o_t)$ and actions $(a_1, \ldots, a_t)$.

This transformation (originating in [1]) has the advantage of being consistent; a good strategy for M is a good strategy for P - the distribution of rewards is exactly the same. It has the drawback that it leads to a much bigger state space; possibly, M is infinite whenever P is quite small. Moreover, the conditional law above can be very hard to sample - we have to sample the next observation, conditionally to all past observations. The key part of this sampling is the sampling of s_t, the state after $t-1$ actions and observations, conditionally to past actions and observations. This is known as the problem of *belief state estimation*.

There are several methods for this:

- Simple rejection methods: randomly sample s and reject s unless it is accepted, which happens with probability proportional to its likelihood conditionally to observations. This is very slow, but mathematically consistent. This is summarized in Alg. 1.
- Markov-Chain Monte-Carlo (MCMC) [2] can be used as well; e.g. Metropolis-Hastings, the simplest version of MCMC.
- Randomly extend the observations in order to get a complete state; this is possible in some games, and widely used in e.g. phantom-games or dark chess which are a quite difficult challenge in terms of partial

Algorithm 1 The rejection method for estimating the belief state.

while True **do**
  Randomly draw x
  Let $r \leftarrow$ likelihood of x (given observations)
  Randomly draw u in $[0, 1]$.
  if $u \leq r$ **then**
    Break.
  end if
end while
Return x

observability[3], [4]. This is certainly not consistent in general (i.e. this approximation of M is not equivalent to P), unless the random choice of the completion in a very specific manner - this is certainly not usual.

The purpose of this paper is to propose new consistent methods for belief state estimation. The main claim of the paper is that, even on an a priori simple game like Mines, a rigorous belief state estimation can provide significant improvements on a simple constraint satisfaction problem. By "consistent" methods, we mean methods which, at least at the limit of infinite computational power, lead to perfect estimates; POMDP algorithms based on this algorithm should, as a consequence, be consistent in the sense that they find, at the limit of an infinite computational power, an optimal strategy; this will be formalized in the rest of this paper.

II. New methods and their consistency

In section II-A we present Monte-Carlo Tree Search (that will be used in our experiments),

A. The rigorous approach: Monte-Carlo Tree Search with rejection

All our experiments are performed using a Monte-Carlo Tree Search (MCTS) implementation, from the Mash project. It uses the Upper Confidence Tree (UCT) formula from [5]. MCTS became famous in the game of Go [6], [7], [8], [9], and UCT is a classical variant of it.

However, we need something for estimating transition probabilities - as our MDP comes from a POMDP setting (see section I-C), this is not straightforward. We have to merge the MCTS algorithm with the rejection method (Alg. 1).

The algorithm is as shown in Alg. 2. As this paper is more devoted to belief state estimation than to UCT, we refer to [5] for more information on UCT. $n(s)$ is the number of simulations including state s, and $n(s, a)$ is the number of simulations including action a in state s; they are all initialized to 0. Indeed, the implementation that we use (from the Mash project) is slightly more sophisticated and uses progressive widening[10], [11]; this is not relevant for this paper.

B. Faster methods

As mentioned above, it is somehow natural to use Metropolis-Hastings as an algorithm for sampling the current state conditionally to past observations; just choose an initial point, and then randomly, at each iteration, choose a mutation of it (by some transition kernel), accepting it or not thanks to the Metropolis-Hastings formula. With a symmetric transition kernel, the algorithm boils down to Alg. 3, i.e. the Metropolis version.

This can be further extended by introducing inheritance: the initial point is not randomly chosen in the domain, but is extracted from the last Metropolis run (we have one Metropolis run for each transition of the MCTS algorithm); while it is probably possible to have more sophisticated algorithms, we just decided to proceed as shown in Alg. 4.

Algorithm 2 The BSR-UCT algorithm (UCT with Belief State estimation by Rejection method). This is a mathematically consistent algorithm for finite horizon problems (i.e. asymptotically in the computation time it finds an optimal strategy), because UCT is consistent on MDPs with finite horizon and the rejection method is consistent for transforming a POMDP into a MDP.

BSR-UCT algorithm.
Input: a POMDP, a state S, a time budget.
Output: an action a.
while time budget permits **do**
  $s = S$. *// starting a simulation*
  while s is not a terminal state and $n(s) > 0$ **do**
    For action a legal in S
      Compute $Q^{\oplus}_{UCT}(s, a)$
    $Q^{\oplus}_{UCT}(s, a) = Q_{UCT}(s, a) + \sqrt{\frac{\log(1+n(s))}{1+n(s,a)}}$
    Select action a with maximal $Q^{\oplus}_{UCT}(s, a)$
    Randomly draw complete state z conditionally to observations s, by Alg. 1.
    Let s' be a state reached from z when choosing action a.
    $s \leftarrow s'$
  end while
  while Time horizon is not reached **do**
    Select action a uniformly in $\mathcal{A}$ *// random episode*
    Let s' be the state reached from s when choosing action a.
    $s = s'$
  end while
  Let R be the reward in s
  For all visited (s, a) during the above simulation,
    Update the average reward:
    $Q_{UCT}(s, a) \leftarrow Q_{UCT}(s, a) + \frac{1}{n(s,a)+1}[R(s) - Q_{UCT}(s, a)]$.
    Increment $n(s, a)$
  Let $n(s) = \sum_a n(s, a)$
end while
Return the action a which was simulated most often from S.

III. Experimental results

We first performed some sanity check, as discussed in Fig. 2 (right). We then tested the approach above on the Mines games, with the rule that the initial location can not be close to a mine (we ensure, as in the Linux/Gnome implementation, that there's no mine in the 8-neighborhood of the first move). We experimented two board sizes, as follows.

The BSR-UCT version of the algorithm uses the MCTS implementation from the Mash project, based on the UCT formula, with a rejection algorithm for estimating the transition probabilities (see Alg. 2). The fastened version is the one proposed in Alg. 4, with as initial point the final point of the last run of Metropolis' algorithm in the same MCTS run

Algorithm 3 The Metropolis algorithm, specialized to our case. The number of trials, for us, is simply the number of simulations in the current state; this means that the computational effort for rigorously estimating the belief state in a node is increasing linearly with the number of simulations in the node.

Choose an initial state x
while my number of trials is not exhausted **do**
  Randomly draw x', with a distribution $p(x)$ such that the probability of drawing x' from x is the same as the probability of drawing x from x'.
  Let $r \leftarrow$ likelihood of x' divided by likelihood of x. (given observations)
  Randomly draw u in $[0,1]$.
  if $u \leq r$ **then**
    $x \leftarrow x'$
  end if
end while
Return x.

Algorithm 4 The Metropolis algorithm with initial point, for use inside a Monte-Carlo Tree Search implementation.

Let x be the last state of the last call to this function; if first call, randomly choose x.
while I have time left **do**
  Randomly draw x', with a distribution $p(x)$ such that the probability of drawing x' from x is the same as the probability of drawing x from x'.
  Let $r \leftarrow$ likelihood of x' divided by likelihood of x. (given observations)
  Randomly draw u in $[0,1]$.
  if $u \leq r$ **then**
    $x \leftarrow x'$
  end if
end while
Return x.

and with transition kernel a random choice of each Mine with probability inverse of the length of x, namely:

$$\begin{aligned} \forall i &\in \{1,\dots,length(x)\}, \\ x'_i &= random \text{ with probability } 1/length(x); \\ x'_i &= x_i \text{ otherwise.} \end{aligned}$$

The number of Metropolis transitions for estimating the belief state in s is equal to the number of MCTS simulations through the node s. We tested many variants of formulas above with no better results.

A. Small board: 4x4, 6 mines

In 4x4, the Mines game, with the constraint as in Gnomine that the first move has no mine in its neighborhood, is a forced win. The computer essentially reaches perfect play at 6 seconds per move. The algorithm is provably optimal at the limit of a large number of simulations because (i) the consistency of the transformation of the POMDP into a MDP is ensured by the rejection method (ii) UCT is consistent in the limit of a large number of simulations.

Thinking time per move	Average result of BSR-UCT	Average result of fast version
0.1	0.933333	0.778788
0.2	0.933333	0.851515
0.4	0.963636	0.857576
0.8	0.963636	0.854545
1.6	0.993939	0.857576
3.2	0.987879	0.869697
6.4	1	0.845455
12.8	1	0.839394

TABLE I
LEFT: RESULTS OF BSR-UCT ON THE SMALL BOARD; IF THE FIRST MOVE IS WELL CHOSEN (THIS IS CRUCIAL), AND NOT SIMPLY UNIFORMLY DRAWN ON THE BOARD, THEN THE MINE GAMES IN 4X4 WITH 6 MINES IS A FORCED WIN. IMPORTANTLY, A SIMPLE CONSTRAINT SOLVER COULD NOT TELL US THAT THE FIRST MOVE SHOULD BE IN THE CENTRAL SQUARE OF THE BOARD AND WOULD NOT REACH 100% OF SUCCESS. RIGHT: THE FASTER METHOD DOES NOT HAVE A PROPER BELIEF STATE ESTIMATION, IT DOES NOT CONVERGE TO OPTIMALITY.

We provide experimental results in Table I, averaged over 30 runs on a 2.83 GHz core.

B. Big board: 7x7, 11 mines

We provide in Table II experimental results on the bigger size 7×7, with 11 mines. Each run is performed only once, we will average results later on; it is on the same 2.83 GHz core as above. This board size with 11 mines is not trivial at all; the good results for long thinking time are not easy to reproduce for humans. Fig. 2 (middle) shows a case in which we have 50% probability of winning.

We compare the two columns of row numbers from Table II in Fig. 3. Basically, the fast version is a trade-off - it provides a lower quality approximation of the belief state (it is only consistent asymptotically in the number of iterations of the Metropolis algorithm), but this approximation is faster, leading to greater numbers of simulations.

IV. CONCLUSIONS

Rigorously estimating the belief state is a big challenge in partial observation problems. This turns out to be important even in simple games like Mines: Fig. I shows that approximate belief state estimation is necessary for exact optimal play.

There are several tools for estimating belief states. First, there are non-asymptotic approaches, which provide a consistent belief state estimation: typically, rejection methods. A second tool is based on asymptotically consistent approaches, like MCMC. We tested MCMC in a simple settings:

- the Metropolis version, with symmetrical transition kernel;
- number of iterations of MCMC in state s proportional to the number of visits of state s by MCTS;
- inheritance (the initial state of the MCMC is the last sample from the previous MCMC).

Thinking time per move	Average result (BSR-UCT)	Average result (fast)
0.001	0.789474	0.447368
0.00118921	0.289474	0.631579
0.00141421	0.657895	0.289474
0.00168179	0.236842	0.842105
0.002	0.473684	0.763158
0.00237841	0.289474	0.447368
0.00282843	0.210526	0.526316
0.00336359	0.157895	0.236842
0.004	0.342105	0.315789
0.00475683	0.315789	0.421053
0.00565685	0.552632	0.552632
0.00672717	0.263158	0.921053
0.008	0.447368	0.894737
0.00951366	0.263158	0.736842
0.0113137	0.368421	0.473684
0.0134543	0.526316	0.578947
0.016	0.315789	0.526316
0.0190273	0.789474	0.736842
0.0226274	0.315789	0.236842
0.0269087	0.368421	0.763158
0.032	0.473684	0.578947
0.0380546	0.526316	0.763158
0.0452548	0.289474	0.394737
0.0538174	0.263158	0.736842
0.064	0.236842	0.710526
0.0761093	0.684211	0.394737
0.0905097	0.394737	0.552632
0.1	0.778788	0.933333
0.107635	0.263158	0.447368
0.128	0.473684	0.315789
0.152219	0.342105	0.736842
0.181019	0.605263	0.394737
0.2	0.851515	0.933333
0.215269	0.631579	0.657895
0.256	0.157895	0.710526
0.304437	0.763158	0.631579
0.362039	0.631579	0.473684
0.4	0.857576	0.963636
0.430539	0.526316	0.526316
0.512	0.315789	0.342105
0.608874	1	0.236842
0.724077	0.763158	0.526316
0.8	0.854545	0.963636
0.861078	0.394737	0.552632
1.024	0.657895	0.342105
1.21775	1	0.263158
1.44815	0.447368	0.447368
1.6	0.857576	0.993939
1.72216	1	0.552632
2.048	0.605263	0.631579
2.4355	0.342105	0.552632
2.89631	0.5	0.447368
3.2	0.869697	0.987879
3.44431	0.421053	0.447368
4.096	0.263158	0.710526
4.87099	1	0.5
5.79262	0.631579	0.394737
6.4	0.845455	1
6.88862	0.684211	0.684211
8.192	0.605263	0.526316
9.74198	0.657895	0.578947
11.5852	0.368421	0.657895
12.8	0.839394	1
13.7772	0.552632	0.473684
16.384	0.868421	0.763158
19.484	1	0.657895
23.1705	1	0.526316
27.5545	0.736842	0.684211

TABLE II
RESULTS ON THE BIG-BOARD; AS WE WILL SEE IN FIG. 3 (PRESENTING MOVING AVERAGES OF THE NOISY RESULTS IN THIS TABLE), THE FASTENED ALGORITHM (USING AN APPROXIMATE BELIEF STATE ESTIMATION) IS AN IMPROVEMENT OVER THE EXACT SLOW BELIEF STATE ESTIMATION ONLY WITH VERY SMALL THINKING TIME.

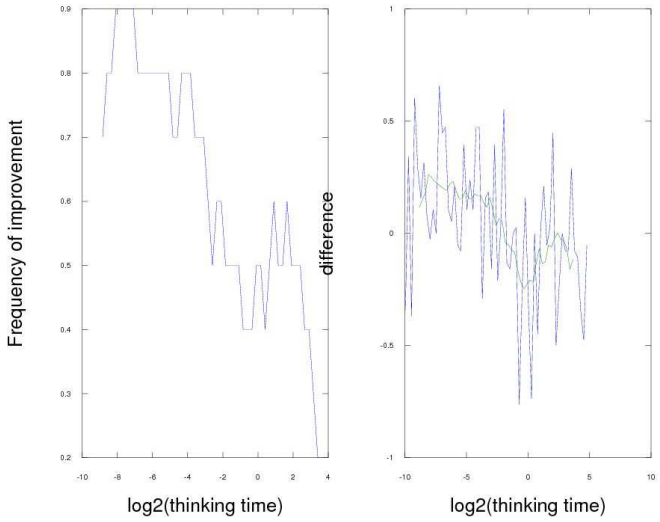

Fig. 3. Left: indicator function of improvement (of fastened version compared to the BSE-UCT version), with a moving average of window size 10: we see that the fast version is better for small thinking time, but is outperformed for large thinking time. Right: performance of the fast method minus performance of BSE-UCT: there is improvement only with small thinking time (the moving average with window size 10 is also shown on the plot). The average of the results for thinking time $\leq .256$ (first half) and $> .256$ (second half) are statistically significantly > 0 and < 0 respectively.

We tested variants of it with no better results. The slow rejection method was in fact better, as it has no bias in its sampling. It is clear that a specialized approach, using constraint propagation methods for exactly computing the probability of death, is natural for Mines; e.g., counting for each possible next location the number of hidden states compliant with observations and leading to an immediate death, as an exact evaluation of the probability of death. This approach is simple and would be computationally much faster. We point out however that this would *not* lead to optimal results: for example, in 4x4 mines with 6 mines, the first move, with this approach, can be played anywhere on the domain (all moves have the same probability of death, which is zero!), whereas our approach plays optimally, which means in this small case 4x4 with 6 mines and for the initial move *in the central square*. That might be surprising for humans, but the best strategy does *not* only consists in playing a move with minimum probability of immediate death (see Fig. 2), and our algorithm, in spite of drawbacks (it's slow for easy cases), has the strength of, asymptotically, playing optimally these subtle cases. This example (first move in 4x4 board) might look like an artificial counter-example, but Fig. 2 (left) shows that such non-trivial phenomena also occur late in the game. This second example can be sovled manually, but it is certainly hard for humans to solve cases like Fig. 2 (right) without a tool like our algorithm, and constraint programming does not provide a solution to this.

Moreover, an important feature of our experiments is that they are all problem-independent; we tested on Mines but

- we did not use any Mines-specific trick and
- we ensure long-term optimality, and not only that the

probability of immediate death is minimum (for an example of this subtle difference which can be mathematically analyzed exactly, see Fig. 2, left).

This paper did not get good results for asymptotic approaches for estimating conditional distributions; this does not mean that such asymptotic approaches can not do a good job, with a different implementation or a different parametrization. Nonetheless, we conclude that the simple rejection method has the advantage of being directly consistent; it will take as much time as necessary for providing a rigorous estimate. We do not conclude that only the rejection method should be considered, but we conclude that comparing with the rejection method or other slow but consistent methods might be a good idea in POMDP. For sure, in many real-world cases, the rejection method is just impossible, making it hard to propose the rigorous way we use here.

We guess that such a rigorous approach might be important in two-player games as well, at least with moderate size; this is a first natural further work. This is far more complicated as in two-player games we have no direct estimate of the likelihood. Moreover, in many games, the state space is huge, making it hard to get rid of heuristics[12], [3], [4].

As a second further work, a direct application of this work is also the test on industrial POMDPs; we guess that neglecting the PO part or handling it in a computationally fast but statistically irrelevant way might lead to serious troubles in real world cases.

REFERENCES

[1] K. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[2] W. R. Gilks, *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, December 1995. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0412055511

[3] T. Cazenave, "A phantom-go program," in *ACG*, 2006, pp. 120–125.

[4] T. Cazenave and J. Borsboom, "Golois wins phantom go tournament," *ICGA Journal*, vol. 30, no. 3, pp. 165–166, 2007.

[5] L. Kocsis and C. Szepesvari, "Bandit based Monte-Carlo planning," in *15th European Conference on Machine Learning (ECML)*, 2006, pp. 282–293.

[6] 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*, pp. 72–83, 2006.

[7] G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," in *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, P. Wang *et al.*, Eds. World Scientific Publishing Co. Pte. Ltd., 2007, pp. 655–661. [Online]. Available: papers\pMCTS.pdf

[8] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *ICML '07: Proceedings of the 24th international conference on Machine learning*. New York, NY, USA: ACM Press, 2007, pp. 273–280.

[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. [Online]. Available: http://hal.inria.fr/inria-00369786/en/

[10] R. Coulom, "Computing elo ratings of move patterns in the game of go," in *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.

[11] A. Couetoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous Upper Confidence Trees," in *LION'11: Proceedings of the 5th International Conference on Learning and Intelligent OptimizatioN*, Italie, Jan. 2011, p. TBA. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00542673/en/

[12] S.-J. Yen, S.-Y. Chiu, and I.-C. Wu, "Modark wins chinese dark chess tournament," *ICGA Journal*, vol. 33, no. 4, pp. 230–231, 2010.