Combining Myopic Optimization and Tree Search: Application to MineSweeper

Michèle Sebag, Olivier Teytaud

Combining Myopic Optimization and Tree Search: Application to MineSweeper

Michèle Sebag[1], Olivier Teytaud[1,2]

[1] TAO-INRIA, LRI, CNRS UMR 8623,
Université Paris-Sud, Orsay, France
[2] OASE Lab, National University of Tainan, Taiwan

Abstract. Many reactive planning tasks are tackled by optimization combined with shrinking horizon at each time step: the problem is simplified to a non-reactive (myopic) optimization problem, based on the available information at the current time step and an estimate of future behavior, then it is solved; and the simplified problem is updated at each time step thanks to new information. This is in particular suitable when fast off-the-shelf components are available for the simplified problem - optimality *stricto sensu* is not possible, but good results are obtained at a reasonnable computational cost for highly untractable problems. As machines get more powerful, it makes sense however to go beyond the inherent limitations of this approach. Yet, a brute-force solving of the complete problem is often impossible; we here propose a methodology for embedding a solver inside a consistent reactive planning solver.
Our methodology consists in embedding the solver in an Upper-Confidence-Tree algorithm, both in the nodes and as a Monte-Carlo simulator. We show the mathematical consistency of the approach, and then we apply it to a classical success of the myopic approach: the MineSweeper game.

1 Introduction

Solving a reactive planning problem is a very hard task; e.g. playing optimally at the MineSweeper game is very hard, and planning power plants maintenances[19] is usually done in a similar approximate manner: a move is decided based on current knowledge, with no tree search. This is quite reasonnable as too many uncertainties make tree search unreliable; however, it makes sense to look for the best of both worlds. As computers get stronger and stronger, it makes sense to work on the "real" problem using the approximate model as a preliminary guide. For this, and following [20, 21] (combining heuristics with Upper Confidence Tree), we propose Upper Confidence Tree (UCT) combined with solvers, for progressively boosting a fast solver (constraint satisfaction problems) to an optimal behavior.

UCT is well known for its ability to work on problems with no or almost no expert information; in particular, it does not need any evaluation function. But is quite convenient for using prior knowledge encoded in heuristic strategies.

These heuristic strategies can be used both in the tree part of the algorithm or in the leaf evaluation; in particular, in the leaf evaluation, it takes into account long-term effects by simulating complete runs (as in $TD(1)$ methods, see [5] for more on this).

MineSweeper is solved until 4x4[17], and is NP-complete in the general case[4]. NP-completeness is not that impressive in games, as EXP or 2EXP complete games are not rare; but, in the partially observable case, we have not so many algorithms which give good results in practice. It is widely known that counting the number of solutions to a MineSweeper problem such that a mine appears in a given location is an efficient tool for playing the game; it is directly proportional to the probability of death if playing in this location. Heuristically, one can use such a method (playing a move with minimum probability of death) for approximately solving MineSweeper; however, it is known [8] that such a strategy is suboptimal, i.e. this is not an optimal strategy, whatever may be the computation time.

We use the following classical terminology:

- A Markov Decision Process (MDP) is a set of states, with edges labelled with actions and probabilities; when a run is in state s and an action a is chosen, the next state is s' with probability p if there is an edge from s to s' labelled with a and p. The sum of the probabilities on out-edges of a state s labelled with a same action a must be equal to 1. A strategy for the MDP maps a finite sequence of states to an action (possibly in a stochastic manner). There are leaf-states (with no out-edge), and one state distinguished as the initial state; there are also rewards on edges, so that the overall Markov Decision Process, together with a strategy, leads to a probability distribution on rewards.
- A Partial Observable Markov Decision Process (POMDP) is similar to a MDP, except that each state is equipped with an observation. A strategy is not a function from a finite sequence of states to an action, but a function from a finite sequence of observations to actions.
- The state in a POMDP can refer to the "real" (partially hidden) state, or to the observed part only; we will refer to the "real" partially hidden state as the "complete state" in order to avoid confusion, in particular because in the MineSweeper literature the "state" usually refers to the observations.
- The feasible set is, given a sequence of observations, the set of possible complete states.
- A consistent belief state estimation is an algorithm which exactly samples the uniform probability distribution on the feasible set, given the observation.
- An asymptotically consistent belief state estimation is an algorithm which samples the probability distribution of the hidden state with a distribution converging to the real one as the computational effort runs to infinity.

Section 2 presents the MineSweeper game. Section 3 will present the algorithms analyzed or proposed in this paper (Section 3.4 is a newly designed algorithm for combining belief state estimation and tree search). Section 4 will

discuss mathematical aspects. Section 5 will explain why the MineSweeper game is interesting and present experimental results on it.

As UCT is now a well established part of the game literature, we refer to [9, 13, 16] for more information on it; we only briefly summarize UCT in Fig. 1. Our

- Initialize the memory to one single node, representing the current state; this is the root of the tree that will grow in memory.
- While (there is time left), simulate an episode as follows:
  - First part of the episode:
    - start at the root,
    - select actions using the UCB (Upper Confidence Bound) formula, thanks to counters updated in previous episodes, until
      - you get a state s which is not yet stored in memory (case 1),
      - or a final state in the nodes already stored in memory (case 2).
  - In case 1:
    - build a new node in memory: this node corresponds to state s and there is an edge from s' to s.
    - Second part of the episode (termed Monte-Carlo part)): select actions randomly until you get a final state.
  - Let r be the reward of the episode.
  - For each node in memory that has been part of this episode,
    - Increase (by 1) the counter of the number of simulations.
    - Increase (by r) the counter of the total reward.
- Play the action which is played most often from the root in episodes simulated above.

Fig. 1. Overview of the original UCT algorithm. There are other variants of the rule for choosing the actually made action (last line). There are also plenty of improvements over the initial UCB formula in the episodes; we also use progressive widening and an improved Monte-Carlo part.

UCT is as specified in Table 1. We use double progressive widening, as explained in [7].

2 The MineSweeper game

The MineSweeper game exists on many platforms. At each turn, the player chooses a location in a grid. After the first move, M mines are randomly put on the $h \times w$ board (anywhere except on the chosen location); the player is informed of the number of mines in the 8-neighborhood of his move. From now on, when a player plays in location l, there are two cases:

- There is a mine: then, the game is over.
- There is no mine: then, the player is informed of the number of mines in the 8-neighborhood.

Feature	In our implementation
Node creation	One per simulation
Progressive widening	Double
Leaf evaluation	By Monte-Carlo simulation
Monte-Carlo moves:	
If possible	Single point strategy (SPS) (SPS can sometimes propose riskless moves, otherwise it does not say anything)
Otherwise:	Move with null probability of mines (SPS does not find all such moves; but CSP detects all)
Otherwise, with probability 0.7	Move with minimum probability of mine (computed by CSP)
Otherwise	Randomly draw a hidden state and play a move with no mine for this hidden state (computed by CSP)
Bandit formula	Upper Confidence Bound with variance estimates[1]
Forced moves[23]	Points with null probability of mines (estimated by CSP)

Table 1. Our UCT implementation in one table. Our work is on the belief estimation part and our UCT is just a classical variant. The single point strategy (from PGMS) is detailed later (Section 3.4). CSP refers to Constraint Satisfaction Problems, and SPS refers to the Single Point Strategy.

The game is a win if the player plays the $h \times w - M$ locations with no mine without ever playing a move on a mine. In usual implementations, when a move is played with no mine in the 8-neighborhood, then all the neighbours are automatically played as they are for sure secure moves; this just saves up time.

All our experiments are performed on the MineSweeper game; the MineSweeper game is more complicated than expected at first view[12, 4, 17]. MineSweeper has motivated a lot of research, sometimes for pedagogical reasons[3] (including a nice version aimed at teaching quantum mechanics[10]), or because it is a widely spread game (anyone here who has never played a MineSweeper game ?), or as a model for real problems[11], and as a challenge for machine learning[6] or genetic programming[14].

Minesweeper artificial intelligences fall in two categories:

- Algorithms based on the understanding of the underlying Markov Decision Process, which is quite hard to analyze and solve[17];
- Algorithms based on heuristically choosing the move with lowest probability of immediate death by Constraint Satisfaction Problems; these algorithms are probably the most impressive for real board size and time settings.

The classical approach for playing MineSweeper consists in using Constraint Satisfaction Problems[22] (CSP). It provides a consistent estimate of the probability of immediate death in case of play in a given location; minimizing this is a quite good heuristic. However, this is not optimal[8]; we here propose an approach which is asymptotically optimal by using UCT (a consistent MDP solver) combined with a consistent belief state estimation by a CSP algorithm. Importantly, this work in based on [17], which shows the relevance of CSP as a consistent belief state estimator. Using CSP inside UCT will require more than just the computation of the probability of presence of a mine in each location - we will need the complete probability of a given complete state given observations.

The best published results are usually those of the CSP approach, with 80% (beginner: 9x9, 10 mines), 45% (intermediate: 16x16, 40 mines); and 34% in expert mode (16x30, 99 mines). [18] proposed a so-called limited search method which gets a earthshaking 92.5% (beginner), 67.7% (intermediate).

Following most previous works, we work on a version of the game in which the first move can not be on a mine (the mines are randomly drawn until they are not at the first chosen location). We do not use the variant in which the initial move is on a "zero" location (as the Linux Gnomine version); this looks like a very good idea for reducing the probability of unlucky death, but it is not widely used and therefore we prefer to be consistent with the body of work around MineSweeper.

We point out the importance of the initial move; the CSP approach does not say anything on it, whereas it is critical; we believe that this explains the success of some implementations - even if the algorithm is not better than the CSP approach, and if the authors have implemented a good initial move or a good opening book, then they might get better results than the CSP approach. However, we point out that the initial move is not the only suboptimality of CSP;

as shown in [8], many small patterns lead to suboptimal moves with non-zero probability.

3 Algorithms

Many algorithms for solving POMDP are based on two tools, more or less separated, namely:

- the estimation of the belief state (i.e. the probability distribution on the complete state, including the hidden part);
- the choice of a move, based on this estimated belief state.

These two components are more or less separated, depending on the approaches. In CSP for MineSweeper, the second part is trivial (play the move with lowest probability of being a mine) whereas the first part is exact. Our goal is to do a non-trivial second part. For the second component, we will use a Monte-Carlo Tree Search; following [20], we will generate transitions conditionally to past observations (we need a complete forward model for a UCT implementation).

We here focus on the first component, namely belief state estimation, and discuss several approaches:

- A simple rejection method (Section 3.1);
- Constraint Satisfaction Problem (CSP, Section 3.2); this is an optimal belief state estimation algorithm, but to the best of our knowledge it has never been used inside a tree search; as a consequence, existing CSP algorithms are not optimal; we will here use it as a belief state estimation rather than as a myopic heuristic;
- Naive Constraint Solving plus Markov-Chain Monte-Carlo (Section 3.3); this was tested in [8] but the authors do not report better results than the simple rejection method;
- BSSUCT (Belief State Solver + UCT), a new method presented in Section 3.4.

3.1 Rejection method

The rejection method is the simplest tool for estimating the complete state, given the observable part. It is consistent, i.e. it proposes an exact belief state estimation; it can be slow, but it will use, by definition, a computation time sufficient for ensuring a correct estimate. The pseudo-code is given in Alg. 1.

In spite of its simplicity, it was reported to be the best method in [8]; the interpretation of this result in [8] is that rejection was the only method which was consistent (The Markov-Chain Monte-Carlo method is *asymptotically* consistent only). Following this comment, we propose a method which is consistent, as well as the rejection method, but faster, cf Section 3.4.

Algorithm 1 The rejection method for consistent belief state estimation. C is a parameter such that $C \times Likelihood_{observation}(s) \leq 1$; if the likelihood of observations, given hidden state, is binary (0 or c), as in MineSweeper ($c = 1$ for MineSweeper), then $C = 1/c$ is the optimal choice; it just means that we accept the state if and only if it is consistent with observations.

Input: observations.
Output: a (uniformly sampled among consistent states) complete state.
$s \leftarrow$random complete state
while $random \geq C \times Likelihood_{observations}(s)$ **do**
  $s \leftarrow$random complete state
end while
Return s

3.2 Constraint Satisfaction Problem (CSP)

We here refer to the many implementations based on counting all complete states which are consistent with observations, i.e. the entire feasible set. This provides the exact probability, for each location, to be a mine. The CSP algorithm can be used for exactly estimating the probability of transition to a given state; yet, to the best of our knowledge, it has only be used for estimating the probability for each location to be a mine and for playing this move. Such an algorithm is presented in Fig. 2.

Algorithm 2 The CSP algorithm for playing MineSweeper. The algorithm is intrinsically myopic: it only optimizes the 1-step result (minimum probability of immediate death).

CSP-function for choosing a move on a partially visible board with M mines
for each location l of the board **do**
  $Nb(l) \leftarrow 0$
end for
for each positioning p of the M mines on the board consistent with observations **do**
  for each location l with a mine for p **do**
    $Nb(l) \leftarrow Nb(l) + 1$
  end for
end for
Play move l uniformly chosen among the set of uncovered locations l such that $Nb(l)$ is minimum.

3.3 Naive Constraint Solving plus Markov-Chain Monte-Carlo

Markov-Chain Monte-Carlo (MCMC) is a tool for asymptotically consistently estimating the belief state. Metropolis-Hastings (MH) is a classical version of MCMC. In the case of MineSweeper, or more generally a case in which observations are deterministic and binary as a function of the belief state, and with

a symmetric transition kernel (i.e. the probability of mutating from state x to state x' is equal to the probability of mutating from state x' to state x), MH boils down to an algorithm as shown in Alg. 3.

Algorithm 3 A version of Metropolis-Hastings with symmetric transition kernel in the case of deterministic binary observations (as in MineSweeper). Please note that removing the initial constraint solving does not change the asymptotical properties of MH; this improved initialization is here for saving up time. According to [8] and for the mutation used there, the initial state chosen by constraint solving unfortunately leads to a too big bias and makes the overall algorithm weaker than the simple rejection method.

```
Input: observations.
Output: a (nearly uniformly sampled) complete state.
Find an initial state s by constraint solving, consistent with observations.
Select a number T of iterations by heuristic methods
   // in [8], T depends on the number of UCT-simulations.
for t ∈ [[1,T]] do
  Let s' be a mutation of s
  if s' is consistent with observations then
    s ← s'
  end if
end for
```

Variants of this version were tested in [8], with disappointing results; the authors conclude that the asymptotic consistency of this approach (instead of "real" consistency of the rejection method) was the reason for the moderately good results. Obviously, it is hard to conclude on this point without testing rigorously the infinitely many possible approaches (the many parameters) for MCMC; maybe, there is a version which would make Alg. 3 extremely efficient. However, we want to have a simple methodology, as far as possible independent of the problem; therefore, we will propose in Section 3.4 a rigorously non-asymptotically consistent belief state estimation.

3.4 BSSUCT: Belief State Solver + Upper Confidence Trees

Our variant of UCT is detailed in Table 1. Our UCT uses several modules:

- the single point strategy as in PGMS (`http://www.ccs.neu.edu/home/ramsdell/pgms/`). The basic idea is that if k of the neighbours of a location are mines, and there are n neighbours, m of which are non-mines and p of which are mines and $n-m-p$ are uncovered, then:
  - If $k = n-m-p$, then all uncovered neighbours are mines;
  - If $k = p$, then all uncovered neighbours are not mines.

  Propagating this is usually done in MineSweeper interfaces in the case $k = 0$ (all neighbours are then automatically played without human action for

fastening the gameplay). Implementing the general case is easy and can't hurt (this plays riskless moves only).
- CSP. Constraint Satisfaction is applied for finding all possible hidden states. We use least constraint variables first - whereas most constrained variables should be used first when looking for one single solution, it is better to use least constrained variables first when looking for all solutions. CSP, in reasonnable time, can provide many information:
  - the probability, for each location, that there is a mine;
  - as a consequence, it can point out sure moves, to be played with no risk;
  - the exact probability distribution of the next state (i.e. how will be the board after next move). With this, the POMDP becomes a MDP.

The use of these modules is detailed in Table 1.

4 Theoretical analysis

We show the consistency of our approach (section 4.1) in the generic case and the non-consistency of the classical approach (constraint satisfaction problems) for the MineSweeper game (section 4.2).

4.1 Consistency of the approach

We first discuss the consistency of our UCT approach as a MDP solver.

UCT (Upper Confidence Trees) is a consistent MDP solver[13] in the finite case. Our approach uses heuristics in the evaluation of leafs by Monte-Carlo simulations, but it does not matter for the consistency proof in [13]. We also use heuristics in the tree part of UCT; this is not detailed in [13], but the upper-confidence-bound proof from [15, 2], used in [13], is also independent of heuristics. A last component which differs from [13] is progressive widening[7]; however, progressive widening here has only a non-asymptotic impact because the number of actions and the number of random outcomes is finite. So, if the problem is a MDP, our UCT-based approach is consistent.

This shows that our approach is consistent for solving a MDP.

We then discuss the consistency for the application to MineSweeper. The difficulty is that MineSweeper is not a MDP: there is partial information, and therefore it is a POMDP. However, [17] has shown that the partial information can be exactly estimated by CSP. The classical approach by CSP is not optimal for playing when just used for the heuristic "play the move with lowest probability of immediate death" (as shown in [8] there are simple situations in which this is not optimal), but it is an exact tool for rephrasing a POMDP as a MDP.

As a consequence, our algorithm is a consistent MDP solver, and combined with CSP it is an asymptotically optimal MineSweeper player.

4.2 A simple case in which our algorithm is better than CSP-based approaches

Our algorithm is asymptotically consistent, i.e. finds the optimal strategy if given a sufficient thinking time (it is asymptotically consistent because UCT is asymptotically consistent for solving MDPs and the rejection method provides the right MDP as explained in Section 4.1). Therefore, it finds the optimal strategy for MineSweeper in 3x3 with 7 mines. We here show that CSP, playing uniformly the initial move, can not be optimal in such a case.

Let's see the success rate, depending on the first move, in this small version of MineSweeper:

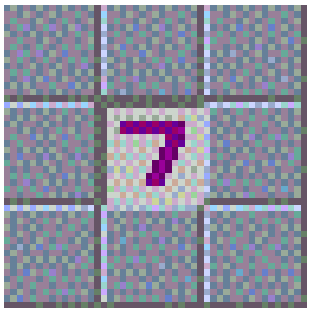

Fig. 2. A bad initial move in 3x3 with 7 mines.

- If we play in the center, we necessarily observe 7; the probability of winning is then $\frac{1}{8}$ as there are 7 mines in the 8 neighbours (see Fig. 2).
- If we play on a side, then we observe:
  - a 5 with probability $\frac{3}{8}$; then the probability of winning is $\frac{1}{3}$;
  - a 4 with probability $\frac{5}{8}$; then the probability of winning is $\frac{1}{5}$.

  This means an overall probability of winning at most $\frac{1}{4} = \frac{3}{8} \times \frac{1}{3} + \frac{5}{8} \times \frac{1}{5}$ when starting on the side.
- If we play in a corner, then we observe:
  - a 2 with probability $\frac{3}{8}$; then the probability of winning is $\frac{1}{3}$;
  - a 3 with probability $\frac{5}{8}$; then the probability of winning is $\frac{1}{5}$.

  This means an overall probability of winning at most $\frac{1}{4} = \frac{3}{8} \times \frac{1}{3} + \frac{5}{8} \times \frac{1}{5}$ when starting in a corner.

An optimal strategy then consists in playing either on a side or in a corner, in order to get a probability $\frac{1}{4}$ of winning.

The CSP algorithm plays uniformly the first move, because all moves have the same probability of death; therefore, it has a probability of winning $\frac{1}{9} \times \frac{1}{8} + \frac{8}{9} \times \frac{1}{4} = \frac{17}{72}$, which is less than the probability $\frac{1}{4}$ of winning obtained by an optimal strategy (and asymptotically reached by BSSUCT).

5 Experiments: MineSweeper

We do validation experiments in the GnoMine Custom mode, as the GnoMine rule (the first move is a zero) makes the problem easier to analyze in a mathematical manner so that we can check the optimality of our algorithm. Then, we switch to the classical MineSweeper rule (the first move is not a mine). We compare our approach to the CSP-PGMS results.

5.1 A Gnomine Custom mode: 15 mines on a 5x5 board

The GnoMine version has this special rule that the location played first has no mine in the neighborhood. This implies that playing in the center leads to a sure win in the configuration 15 mines on a 5x5 board (see Fig. 3). Interestingly, our algorithm finds this optimal strategy (that humans easily find as well); but the important point here is that the CSP method does not say anything about the choice of the first move. If we play randomly and uniformly the first move, then we have a probability $\frac{4}{25}$ of playing in (2,2) (or a symmetry of it), with then a probability of loosing the game at least $\frac{1}{2}$; and we have a probability $\frac{4}{25}$ of playing (2,3) (or a symmetry of it), with then a probability of loosing the game at least $\frac{1}{4}$. This leads to an overall probability of loosing the game at least $\frac{1}{2} \times \frac{4}{25} + \frac{1}{4} \times \frac{4}{25} = \frac{3}{25}$; the real probability is indeed much bigger, as (1,1), (1,2), (1,3) are also very bad moves, but we do not compute the detailed probability - the lower bound $\frac{3}{25}$ on the probability of loosing is enough for showing that the CSP can not reach the optimal play (that our BSSUCT provably reaches).

Given the rules only (no expert knowledge added), our algorithm asymptotically finds the best move, and in 5x5 with 15 mines and gnomine rules it finds it in 5 seconds (in all runs among 500 runs).

Fig. 3. The Gnomine version of the 5x5 board with 15 mines is a sure win: on each of these figures, one can deduce the position of all mines (there is only one non-mine location). This covers all possibles cases by rotation.

5.2 Windows version, various board sizes

We compare our algorithm (BSSUCT) with CSP on various board sizes; the CSP implementation we use is the CSP-PGMS implementation from `http://`

www.ccs.neu.edu/home/ramsdell/pgms/ which plays in corners for the initial moves. Results are presented in Table 2.

Format	CSP-PGMS	BSSUCT
4 mines on 4x4	64.7 %	**70.0% ± 0.6%**
1 mine on 1x3	100 %	100% (2000 games)
3 mines on 2x5	22.6%	**25.4 % ± 1.0%**
10 mines on 5x5	8.20%	9% (p-value: 0.14)
5 mines on 1x10	12.93%	**18.9% ± 0.2%**
10 mines on 3x7	4.50%	**5.96% ± 0.16%**
15 mines on 5x5	0.63%	**0.9% ± 0.1%**

Table 2. Winning rate of our implementation, versus the winning rate of CSP-PGMS. The winning rate for CSP-PGMS is estimated on 100000 games; we provide the standard deviation for our program which is much slower and therefore has non-negligible confidence intervals. Importantly, CSP-PGMCS does not benefit from additional time. The games are played with 10s per move, except for 10 mines in 5x5 (300 seconds per move) and 10 mines in 3x7 (30s per move). All experiments on a 16-cores Xeon 2.93GHz Ubuntu 2.6-32-34 using 1 core only per experiment.

6 Conclusion

We have proposed the use of UCT for difficult decision tasks, with CSP for two simultaneous purposes:

- estimating the belief state, so that a POMDP becomes a MDP;
- suggesting good moves, in a heuristic manner.

This provides a generic methodology for improving a myopic solver by including it into a UCT. We strongly believe that such approaches, using consistent asymptotic behaviors and myopic-guided heuristics, are important for future artificial intelligence, benefiting both from relevant heuristics (which provide tractability) and strong computational power and mathematical consistency analysis (which provide asymptotic optimality). Starting at a reasonnably good strategy and improving it towards optimality by UCT looks like a simple efficient idea.

We have shown the mathematical consistency of the approach, as well as experimental results. The experimental results are on MineSweeper, which is particularly challenging as the CSP approach, which completely neglects long term effects, is very effective as uncertainties are so big that long-term effects are very unreliable - we have chosen a very hard test case and we nonetheless get significant improvements. When the computation time becomes big, CSP does not improve anymore (it has inherent limitations), whereas our approach performs better and better.

For getting impressive results in the game community, we will work on a faster implementation; our CSP module is much slower than e.g. CSP-PGMS. It should

be easy to work on expert modes, provided that we have an implementation as fast as CSP-PGMS for the CSP module. We have already clear improvements on small boards, compared to [8]. As a future work, we will also compare with results of [18].

A crucial good piece of news for our algorithm is that we do not enter any opening move manually (as necessary for good performance with CSP); and we can switch easily to variants of MineSweeper just by changing the implementation of the rules.

Acknowledgements

The authors are grateful to NSC for funding NSC100-2811-E-024-001, to ANR for funding COSINUS program (project EXPLO-RA ANR-08-COSI-004), to L. Simon for fruitful discussions. We are very grateful to L. Simon for very fruitful discussion, and to the authors of PGMS and PGMS-CSP for making their implementations freely available. We also acknowledge gratefully the support by European Program FP7, through the MASH project.

References

1. J.-Y. Audibert, R. Munos, and C. Szepesvari. Use of variance estimation in the multi-armed bandit problem. In *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*, 2006.
2. P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.
3. K. Becker. Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, 17:23–33, December 2001.
4. M. M. Ben-Ari. Minesweeper as an np-complete problem. *SIGCSE Bull.*, 37:39–40, December 2005.
5. D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic Programming*. Athena Scientific, 1996.
6. L. P. Castillo. Learning minesweeper with multirelational learning. In *In Proc. of the 18th IJCAI*, pages 533–538, 2003.
7. 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*, page TBA, Italie, Jan. 2011.
8. A. Couetoux, M. Milone, and O. Teytaud. Consistent belief state estimation, with application to mines. In *Proceedings of the TAAI 2011 conference*, page in press, 2011.
9. 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.
10. M. Gordon and G. Gordon. Quantum computer games: quantum minesweeper. *Physics Education*, 45(4):372, 2010.
11. K. B. Hein and R. Weiss. Minesweeper for sensor networks–making event detection in sensor networks dependable. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 01*, CSE '09, pages 388–393, Washington, DC, USA, 2009. IEEE Computer Society.

12. R. Kaye. Minesweeper is np-complete. *Mathematical Intelligencer*, 22:915, 2000.
13. L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.
14. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Massachussetts, 1994.
15. T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
16. 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.
17. P. Nakov and Z. Wei. Minesweeper, #minesweeper, 2003.
18. K. Pedersen. The complexity of minesweeper and strategies for game playing. *Project report, univ. Warwick*, 2004.
19. ROADEF-Challenge. A large-scale energy management problem with varied constraints. In *http://challenge.roadef.org/2010/*, 2010.
20. P. Rolet, M. Sebag, and O. Teytaud. Optimal active learning through billiards and upper confidence trees in continous domains. In *Proceedings of the ECML conference*, 2009.
21. P. Rolet, M. Sebag, and O. Teytaud. Optimal robust expensive optimization is tractable. In *Gecco 2009*, page 8 pages, Montréal Canada, 2009. ACM.
22. C. Studholme. Minesweeper as a constraint satisfaction problem. *Unpublished project report*, 2000.
23. F. Teytaud and O. Teytaud. On the Huge Benefit of Decisive Moves in Monte-Carlo Tree Search Algorithms. In *IEEE Conference on Computational Intelligence and Games*, Copenhagen, Denmark, Aug. 2010.