The AIIDE-17 Workshop on
Artificial Intelligence for Strategy Games
WS-17-18

Single Believe State Generation for Handling Partial Observability with MCTS in StarCraft

Alberto Uriarte, Santiago Ontañón
Computer Science Department
Drexel University
{albertouri,santi}@cs.drexel.edu

Abstract

A significant a mount of work exists on handling partial observability for different game genres in the context of game tree search. However, most of those techniques do not scale up to RTS games. In this paper we present an experimental evaluation of a recently proposed technique, *single believe state generation*, in the context of *Star Craft*. We evaluate the proposed approach in the context of a *Star Craft* playing bot and show that the proposed technique is enough to bring the performance of the bot close to the theoretical optimal where the bot can observe the whole game state.

Introduction

It has been shown that partially observable games are often exponentially harder than perfect information games. Specifically, perfect information two-player unbounded-length games have been shown to be **EXPTIME** (Fraenkel and Lichtenstein 1981), while games where there is private information have been shown to be **2-EXPTIME** (Reif 1984). Thus, finding optimal strategies in partially observable games is intractable. This paper extends our previous work on game tree search on partially observable real-time strategy (RTS) games by analyzing the effectiveness of single-believe state estimation techniques in the context of game tree search in StarCraft.

Recent work on game tree search in RTS games has been focused on handling the fact that RTS games have huge branching factors (e.g., (Ontañón et al. 2013; Barriga, Stanescu, and Buro 2015)), or the fact that they are real-time, and thus there is very little time for performing search (e.g., (Churchill and Buro 2013)). However, the problem of partial observability in RTS games has not received sufficient attention, and good approaches to handle this problem still do not exist.

Thus, specifically, this paper explores the idea of *determinization* (from all the possible states that the game can be in, given the player observation, just pick one or a few, and then perform standard game tree search) in the context of StarCraft. In our previous work, we proposed three different determinization techniques specifically designed for RTS games, and evaluated them in a simplified RTS game (μRTS[1]). This paper extends upon such work, by evaluating the performance of some of them in the context of StarCraft. The key idea behind our approach is to use a combination of *memory of past knowledge* (remember the information we observed from the opponent in the past) and an *inference process* (e.g., if we have seen the opponent to have air units, then she must have a space port) to maintain a single *believe state*. This believe state is an estimation of actual game state given the current observations, memory of past events and inference. We present experiments in the context of *Monte Carlo Tree Search* (MCTS) in the domain of StarCraft.

The remainder of this paper is organized as follows. First, we introduce some basic concepts of RTS games and partially observable games. Then, we present our believe state estimation approach. After that, we report an empirical evaluation on StarCraft, showing very promising results, and finally we present related work in the area of *determinization* to handle partially observable games.

Background

This section briefly describes RTS games and then summarizes the existing approaches to handle partial observability in adversarial games.

Real-Time Strategy Games

RTS games are complex adversarial domains, typically simulating battles between a large number of military units, that pose a significant challenge to both human and artificial intelligence (AI) (Buro 2003). Designing AI techniques for RTS games is challenging because of three main reasons::

- *Scale*: they have *huge decision spaces*: the branching factor of a typical RTS game, StarCraft, has been estimated to be on the order of 10^{50} or higher (Ontañón et al. 2013) (for comparison, that of Chess is about 35, and that of Go about 180). Moreover, the state space of StarCraft has been estimated to be at least 10^{1685} (Ontañón et al. 2013), compared to about 10^{47}(Chinchalkar 1996) for Chess and 10^{171}(Tromp and Farnebäck 2006) for Go.
- *Real-time nature*: 1) RTS games typically execute several decision cycles per second, leaving players with a fraction

[1] https://github.com/santiontanon/microrts

Figure 1: A screenshot of StarCraft, showing the area of the game that the player can see, and that that is hidden due to the *fog-of-war*. The right part of the screen is visible, since there are three units (three SCVs). Since there are no player units on the left side, the player cannot see what is there.

of a second to decide the next action, 2) players can issue actions simultaneously, and 3) actions are durative.

- *Partial observability*: due the fog-of-war, players cannot observe the parts of the map that are out of the sight of their units units.

Additionally, some RTS games are also non-deterministic, but we will not deal with this problem in this paper.

The reason for which the branching factor in RTS games is so large is that players control many units, and players can issue multiple *unit-actions* at the same time (one per unit). Thus, the set of possible *player-actions* grows exponentially with the number of units a player controls. Notice that although when humans play these games, due to the limitations of the interface, they cannot issue an arbitrary number of actions per game cycle, when an AI plays these games, the full combinatorics exists. The fact that these games are real-time only exacerbates this problem.

Moreover, the focus of this paper is partial observability. In order to illustrate the idea of the *fog-of-war*, Figure 1 shows a screenshot of StarCraft where we can see the area that is visible to the player, and that that is hidden by the fog-of-war, since there are no units nearby. In the rest of this paper we will use the term *observed state* to refer to the game state as observed by the player (which might no include some enemy units or part of the map, since they are not visible). We will use the term *actual game state* to refer to the underlying game state (which is not observable to the player), and which includes all units in the game. Finally, we will use the term *believe state* to any estimation of the actual game state the a player makes given the observed state.

As in our previous work (Uriarte and Ontañón 2017), we will use the following definition for a two player partially-observable RTS game as a 9-tuple $G = (P, S, Z, O, A, L, T, W, s_0)$, where:

- $P = \{max, min\}$ is the set of players.
- S is the set of possible game states.
- Z is the set of possible observations (i.e., since the game is partially observable, the only thing players can observe are the states in Z).
- $O(p, s) \rightarrow Z$, is the observation function. Given a player $p \in P$ and the current game state $s \in S$, returns the observed state $z_p \in Z$ from a point of view of player p.
- A is the finite set of unit-actions (a) that units can execute. Also remember that we defined α as the set of unit-actions from the same player.
- $L(p, \alpha, s) \rightarrow \{true, false\}$, is a function that returns whether player p can execute player-action α in state s.
- $T(s_t, \alpha_{min}, \alpha_{max}) \rightarrow s_{t+1}$ is the deterministic transition function, that given a state $s_t \in S$ at time t, and the player-actions of each player (α_{min} and α_{max}), returns the state that will be reached at time $t + 1$ (i.e., T is the *forward model* of the game).
- $W : S \rightarrow \{maxwins, minwins, draw, ongoing\}$ is a function that determines the winner of the game, if the game is still ongoing, or if it is a draw.
- $s_0 \in S$ is the initial state.

Additionally, we define $I_p(z) \subseteq S$ as the information set of player p given observation z, which is the set of all states that are indistinguishable for player p given the current observed state z, i.e., $I_p(z) = \{s \in S | O(p, s) = z\}$.

Game Tree Search in Partially Observable Domains

Partially observable games are usually modeled as **extensive-form games**, where the main difference with respect to fully observable games is that instead of considering *states*, we consider *information sets*, i.e. sets of states that are indistinguishable to a player at the time she has to make a decision. Moreover, the game tree complexity of imperfect information games tends to be very high even for simple games, and approaches to compute the optimal *randomized strategy* for them can require an exponential amount of time as a function of the size of the game tree (Kuhn 1950) or at least polynomial in the size of the game tree (Koller and Pfeffer 1995).

For this reason most work on partially observable games (or *imperfect information games*) requires making assumptions that do not hold true for RTS games. For example, some of them assume opponents with fully observability or require searching all the possible states, such as *Best Defence model* (Frank and Basin 1998), *Vector minimaxing* (Frank, Basin, and Matsubara 1998), or *Believe-state AND-OR tree search* (Russell and Wolfe 2005). Zinkevich et al. (Zinkevich et al. 2007) proposed *Counterfactual Regret Minimization* (CFR), an algorithm that converges to the Nash equilibrium without assuming an opponent with full observability, but still requiring sampling all the different states. Lanctot et al. (Lanctot et al. 2009) improved over this by avoiding having to explore the whole state space by using Monte Carlo sampling (*Monte Carlo CFR*, MCCFR). MCCFR has been applied with great success in games with a

short game tree depth, compared to that in RTS games, and where the disambiguation of the information sets only happens at the end, such as Poker or Liar's Dice.

The most common approach to handle partial observability is known as *determinization* (see the related work below for an overview of work in this area). The key idea of *determinization* is to sample a *state* from the current information set and proceed with a perfect information game tree algorithm. This is usually repeated several times, and the action to perform is determined via voting. This has been described as "averaging over clairvoyance" (Russell and Norvig 2009) and as Frank et al. (Frank and Basin 1998) pointed out, it raises several problems:

- *Strategy fusion*: players must behave the same way in states from the same information set,
- *Non-locality*: the search can be "fooled" to pursue a highly rewarded state that cannot actually be reached under some information, and
- *Fake omniscience*: when the player never tries to hide or gain information because she believes that she has perfect information.

Despite these problems, *determinization* works very well in some domains. Long et al. (Long et al. 2010) explained why by defining three properties of game trees that can lead to *strategy fusion* and *non-locality*: 1) probability of another terminal node with the same payoff value (**leaf correlation**), 2) probability that the game will favor a particular player over the other (**bias**), and 3) how quickly the states in an information set shrinks with regard to the depth of the tree (**disambiguation factor**). They found that *determinization* will perform well in games with a very high *disambiguation factor* or with a very high *leaf correlation* combined with a polarized *bias* (i.e., a very low or very high *bias*). In our previous work (Uriarte and Ontañón 2017), we analyzed whether RTS games satisfied these properties by analyzing them in a simplified RTS game (μRTS (Ontañón 2013)), and found that: 1) *leaf correlation* is really high as expected, since the last player's moves usually do not change the outcome of the game, 2) the *bias* is balanced, and 3) the *disambiguation factor* is really low or even negative, this is because in RTS games we can lose information (specially toward the end of a game a losing player will start losing units, thus losing her ability to see the board). So, based on this analysis, it seems that determinization should not work well on RTS games. However, our experiments showed that it indeed worked very well in μRTS. In this paper, we will show that it also works very well in STARCRAFT.

Believe State Generation

Previous work on sampling believe states from the current information set (Parker, Nau, and Subrahmanian 2005; Richards and Amir 2012) requires search processes that could be too computationally expensive for real-time games. Thus, our proposed approach focuses on generating a single believe state that hopefully resembles the actual game state. Our hypothesis is that due to the complexity of RTS games (and due to the fact that decisions need to be made in every single frame), finding the optimal move at every game frame might not be necessary. Therefore, sampling a believe state that approximates the actual game state sufficiently well might be enough to achieve strong gameplay (with respect to the current state of the art). In our previous work (Uriarte and Ontañón 2017) we proposed three techniques to sample such believe state, one of which clearly outperformed the others when evaluated in μRTS. Thus, in this paper, we focused on evaluating the performance of the best of those three techniques in STARCRAFT. This section summarizes such approach.

Let us assume that the initial state is fully observable, i.e., both players know the actual game state for the very first game frame. This assumption is true for board games like Kriegspiel where the initial board configuration is known for both players, or for RTS games where for a given map the initial base locations are known for both players (like in μRTS). It is also true for 2-player StarCraft maps where there are only 2 possible starting locations, and thus each player knows where the other player is starting. For other games like Poker this is not true since the opponent hand is unknown, but those are out of the scope of this work.

With this assumption, our believe state generation strategy called *Perfect memory* (PM) works as follows:

- PM keeps a record U_o of the location of all opponent units that have been observed at any time during the game (including all of those in the fully observable initial game state s_0), but at not currently visible.
- Then, given the current observation z, if the location where a unit u in U_o was last seen is visible but the unit is not there anymore, the location of u is updated to the nearest not observable location (i.e., it assumes the unit has moved, but just the minimum amount as for making the unit not observable).
- Inference mechanism: Additionally, if there is any enemy unit that we have never seen but that is required to explain part of the observation (i.e., if in order for the opponent to have units of a certain type t_1, the opponent must have first build a unit of type t_2, if we observe a unit of type t_1, then for sure we know the opponent has a unit of type t_2), then such units are also added to U_o. Inferred units are added in the closest non-observable location to the opponent's unit that caused the inference.
- Given the current observation z, the believe state is generating by adding all the units of U_o into z.

Intuitively, this strategy just takes the current observation, adds units that we had seen in the past and also adds units that we cannot observe, but that must be there in order to explain the current observation. In order to implement the inference mechanism in STARCRAFT, we exploit the technology tree of the game. And if a unit of a certain type t_1 is observed, all the units that are required for this unit to be produced according o the tech tree are added to U_o.

From a point of view of game theory, this strategy resembles the idea of *memory of past knowledge* (Bonanno 2004).

Finally, notice that in its current form, this strategy can only handle 2-player STARCRAFT maps, since 3+ player

maps would require either maintaining a probability distribution over possible unit locations, or consider multiple believe states and do information set game tree search (Whitehouse, Powley, and Cowling 2011). We plan to explore this possibility in our future work.

Incorporating Believe State Generation into MCTS

The base MCTS approach used in this work is the *Informed MCTSCD* approach of Uriarte and Ontañón 2016a, implemented in the Nova StarCraft bot. Specifically this MCTS approach works as follows:

- The underlying bot is Nova, which controls every aspect of the game, except for military units. Informed MCTSCD controls the movement of all military units in the game.
- Rather than working using the low-level game state of StarCraft, Informed MCTSCD uses an abstracted game state. The map is represented as a graph of regions and chokepoints (generated using BWTA2 (Uriarte and Ontañón 2016b)), and instead of considering each unit individually, units are grouped by type and by area. So, all the units in the same area of the same type are issued the same action by informed MCTSCD.
- The specific configuration we used is to use the *Informed MCTSCD* algorithm, using a *best informed ϵ-greedy sampling* as the tree policy ($\epsilon = 0.2$); a *Squad Action Naive Bayes Model* for the default policy (Uriarte and Ontañón 2016a), and *Alt* policy in nodes where both players can move (Churchill, Saffidine, and Buro 2012); a limit of 2,880 frames for the length of playouts (or simulations); and a *Decreasing DPF model* for the forward model.

Since RTS games are real-time, we perform a search process periodically (each 400 frames), and after each search, the action associated with each unit is updated with the result of the search.

Experimental Evaluation

In order to evaluate our approach, we experimented with executing informed MCTSCD with a computational budget from 1 to 10,000 playouts. We compared three scenarios:

- *Full Observability (cheating)*: in this configuration, we deactivate the fog-of-war, and the game state given to informed MCTSCD is the actual fully observable game state. This configuration is used just for testing, since deactivating the fog-of-war would be considered cheating, in a real StarCraft match.
- *Partial Observability*: in this configuration, the game state given to informed MCTSCD is just what the player can observe. So, for example, at the beginning of the game, the player will not see any opponent units.
- *Partial Observability with Believe State Generation*: in this configuration, the proposed *Perfect Memory* strategy is used to generate a believe state, which is passed on to informed MCTSCD.

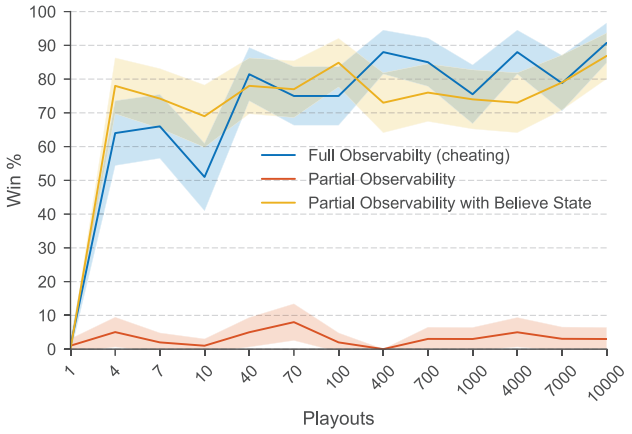

Figure 2: Comparison of Win % and 95% CI using informed MCTSCD with perfect information (cheating) or imperfect information with and without a *Perfect Memory single believe state* against the StarCraft built-in AI.

For each scenario, we ran 100 games against the built-in AI. All games were performed in the Benzene StarCraft tournament map. A general timeout for all the games was set to 28,800 frames. Games going longer than this timeout are considered a draw.

Figure 2 shows the win % average and the 95% confidence interval of informed MCSTCD using perfect information (cheating) or using imperfect information (with and without the proposed *Perfect Memory single believe state* approach) in StarCraft. As expected, when we enable the fog-of-war, informed MCTSCD algorithm without a believe state is not aware of the enemy and it loses most of the games. But once we use the *Perfect Memory single believe state* approach, the win % is similar than when we have perfect information of the game. Although the results seem to show that with less playouts the *single believe state* performs slightly better than the cheating version, and the opposite for a large number of playouts, the differences are not statistically significant. Thus, we can conclude that the proposed *Perfect Memory* believe state estimation approach seems to be enough to bring the performance of the StarCraft playing bot used in our experiments to the same level of performance as if it could observe the whole map. We hypothesize that the slight difference in performance for the larger number of playouts is due to a "disconnection" between the real game state and the abstract game state, which manifests itself more prominently when we give the bot a larger computation budget. An interesting line of future work is to assess whether these results generalize to other bots as well.

Moreover, we would like to emphasize that our system did not lose any game, and the small percentage of games it did not win were ties, where our system could not locate the last remaining enemy units. This happens because some of the regions that BWTA2 divides the map on are larger than the sight range of the units in the region, and thus, there might be unseen areas of a region unbeknownst to the MCTS algorithm. To address this issue, BWTA2 actually generates

a list of *points of visibility* for each region. if we place a unit in each of these points, then we can ensure we are seeing the whole region. Our current implementation of Informed MCTSCD does not exploit this information though.

Related Work

Determinization is by far the most common technique to handle partial observability in game tree search. This section provides a brief overview of the work in this area.

Monte Carlo Sampling (Corlett and Todd 1986): Also known as *Perfect Information Monte Carlo Sampling* (PIMCS), it randomly samples states to apply a perfect information search algorithm like alpha-beta and it returns the best average move of all sampled states. This technique have been applied in games like Bridge (Levy 1989), but it have been shown that the error to find the optimal strategy rapidly approaches to 100% as the depth of the game tree increases (depth = 13) (Frank, Basin, and Matsubara 1997).

Statistical Sampling (Parker, Nau, and Subrahmanian 2005): Parker et al. proposed a statistical sampling for large believe state games like Kriegspiel. More specifically they proposed 4 different samplings: *Last Observation Sampling* (LOS), *All Observation Sampling* (AOS), *All Observation Sampling with Pool* (AOSP) and *Hybrid Sampling* (HS). For each sample state (world) it uses alpha-beta and decides the move that maximizes the score.

Information-set search (Parker, Nau, and Subrahmanian 2010) is game-tree search approach using *information sets* and computing the expected utility (EU) of each *information set*. The EU is computed as the weighted sum of the EUs for each possible move, weighted by the probabilities of a move given all the previous moves (perfect recall) times the EU of applying the move. To make the problem tractable in Kriegspiel, they used the following simplifications:

- The states in each information set are sampled using Monte Carlo sampling.
- The search is limited to a depth and a heuristic evaluation function is used for the EU.
- The probabilities of choosing moves for our opponent (strategy of a player, a.k.a. player modeling) are limited to two options: the opponent knows our pure strategy and chose moves that minimize our EU (**paranoia**); the opponent does not know anything and uses a uniform random distribution of our actions (**overconfidence**).

In their experiments, overconfident opponent model outperformed the paranoid model.

Monte Carlo Tree Search (MCTS) with Simulation Sampling (Ciancarini and Favini 2009): Ciancarini et al. proposed to delay the determinization until the simulation phase of MCTS, showing that they got better results using a heuristic function to get the probability of each type of world.

Determinized MCTS (Whitehouse, Powley, and Cowling 2011): It uses *root parallelization* where each root is a different determinization.

Single Observer Information Set MCTS (SO-ISMCTS) (Cowling, Powley, and Whitehouse 2012): The idea is that on each iteration of MCTS, a random root determinization is made to get the set of legal actions. and information sets as are used as nodes in the game tree. This algorithm makes several assumptions: 1) the same action applied to all the states of an information set transition to the same information set, 2) opponent uses a random move selection on the moves that are not observed, and 3) at each real player turn (i.e., after executing an action in the game state), we can generate the information sets from the current observations. Unfortunately this last assumption is not true for RTS games because the presence of durative actions.

Multi Observer Information Set MCTS (MO-ISMCTS) (Cowling, Powley, and Whitehouse 2012): To solve the second assumption of the previous algorithm, they proposed a search using two ISMCTS simultaneously, one for each player or "point of view". The traversing is done simultaneously but the action is considered from the point of view of each player. In their tests a *Determinized MCTS* works better for games with low probability of *strategy fusion* while MO-ISMCTS works better when there is a high chance of *strategy fusion*.

Although all of these techniques use *determinization* at some degree, those that compact tree nodes by information sets do not usually have the problem of *strategy fusion* or *non-locality*; and only SO-ISMCTS partially avoids *fake omniscience* since none of the players have perfect information but there is not a mechanism to detect and exploit gathering/hiding information actions.

Conclusions

In this paper, we have focused on the problem of performing game tree search in partially observable RTS games. In order to address this problem, we have explored a simple and fast approach to generate a single believe state given the current information set, and experimented with it in the context of STARCRAFT. Our results show that our approach achieves the same performance of having access to the whole game state in STARCRAFT. Therefore, despite the evidence shown in previous work, *determinization* is able to handle RTS games with partially observable game states with without any statistical significant penalty in large domains (STARCRAFT).

This paper is a natural continuation of our previous work (Uriarte and Ontañón 2017), where we showed similar good performance in a different RTS game (μRTS). Thus, we can conclude that single believe state generation is a viable approach for a large class of RTS games.

One aspect that our approach still cannot handle are games where the initial state is not fully observable. For example, when playing on a 3 or 4 player map in STARCRAFT, the initial position of the enemy is unknown. Therefore, we would not be able to add their locations to U_o. We would like to extend out approach to handle a probability distribution of the possible locations of the units in U_o, so that we could account for the fact that we know the enemy started in one of the possible initial base locations. When generating the believe state, these probabilities would be used to sample a single believe state. Moreover, observations should be used to update these probabilities in order to reduce those for positions that are unlikely, and increase the probability

for those positions that are more likely. Moreover, notice that because of the determinization approach used in this paper, this would not result on the system actively exploring to resolve the ambiguity in the information set. In order to do that, we would need to move away from determinization approaches. Additionally, we would like to explore the possibility of applying MCTS to control more tasks than just moving the military units, such as production, and study whether performance decreases. Finally, given that at this point our approach would finally allow for comparing our MCTS approach against other competition bots under STARCRAFT AI competition settings with fog-of-war activated, we would like to evaluate the performance of MCTS-based bots against the top hardcode bots that are currently dominating the competition.

References

Barriga, N. A.; Stanescu, M.; and Buro, M. 2015. Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games. In *AIIDE*.

Bonanno, G. 2004. Memory and perfect recall in extensive games. *Games and Economic Behavior* 47(2):237–256.

Buro, M. 2003. Real-time strategy games: a new AI research challenge. In *IJCAI*, 1534–1535. Morgan Kaufmann Publishers Inc.

Chinchalkar, S. 1996. An upper bound for the number of reachable positions. *ICCA Journal* 19(3).

Churchill, D., and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *CIG*, 1–8. IEEE.

Churchill, D.; Saffidine, A.; and Buro, M. 2012. Fast heuristic search for RTS game combat scenarios. In *AIIDE*. AAAI Press.

Ciancarini, P., and Favini, G. P. 2009. Monte carlo tree search techniques in the game of kriegspiel. In *IJCAI*, volume 9, 474–479.

Corlett, R. A., and Todd, S. J. 1986. A monte-carlo approach to uncertain inference. In *Artificial intelligence and its applications*. John Wiley & Sons, Inc. 127–137.

Cowling, P. I.; Powley, E. J.; and Whitehouse, D. 2012. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2):120–143.

Fraenkel, A. S., and Lichtenstein, D. 1981. Computing a perfect strategy for $n \times n$ Chess requires time exponential in n. *Combinatorial Theory, Series A* 32(2):199–214.

Frank, I., and Basin, D. A. 1998. Search in games with incomplete information: A case study using bridge card play. *Artif. Intell.* 100(1-2):87–123.

Frank, I.; Basin, D.; and Matsubara, H. 1997. Monte-carlo sampling in games with imperfect information: Empirical investigation and analysis. In *Game Tree Search Workshop*.

Frank, I.; Basin, D. A.; and Matsubara, H. 1998. Finding optimal strategies for imperfect information games. In *AAAI/IAAI*, 500–507.

Koller, D., and Pfeffer, A. 1995. Generating and solving imperfect information games. In *IJCAI*, 1185–1193.

Kuhn, H. W. 1950. A simplified two-person poker. *Contributions to the Theory of Games* 1:97–103.

Lanctot, M.; Waugh, K.; Zinkevich, M.; and Bowling, M. 2009. Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems*, 1078–1086.

Levy, D. N. 1989. The million pound bridge program. *Heuristic Programming in Artificial Intelligence* 95–103.

Long, J. R.; Sturtevant, N. R.; Buro, M.; and Furtak, T. 2010. Understanding the success of perfect information monte carlo sampling in game tree search. In *AAAI*.

Ontañón, S. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *AIIDE*. AAAI Press.

Ontañón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game AI research and competition in StarCraft. *TCIAIG* 5(4):293–311.

Parker, A.; Nau, D.; and Subrahmanian, V. 2005. Game-tree search with combinatorially large belief states. In *IJCAI*, 254–259.

Parker, A.; Nau, D. S.; and Subrahmanian, V. S. 2010. Paranoia versus overconfidence in imperfect information games. In Dechter, R.; Geffner, H.; and Halpern, J. Y., eds., *Heuristics, Probability and Causality: a Tribute to Judea Pearl*. College Publications. 63–87.

Reif, J. H. 1984. The complexity of two-player games of incomplete information. *Journal of computer and system sciences* 29(2):274–301.

Richards, M., and Amir, E. 2012. Information set generation in partially observable games. In *AAAI*.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition.

Russell, S., and Wolfe, J. 2005. Efficient belief-state AND-OR search, with application to Kriegspiel. In *IJCAI*, volume 19, 278.

Tromp, J., and Farnebäck, G. 2006. Combinatorics of Go. In *International Conference on Computers and Games*, 84–99. Springer.

Uriarte, A., and Ontañón, S. 2017. Single believe state generation for partially observable real-time strategy games. In *Proceedings of CIG 2017*.

Uriarte, A., and Ontañón, S. 2016a. Improving monte carlo tree search policies in StarCraft via probabilistic models learned from replay data. In *AIIDE*.

Uriarte, A., and Ontañón, S. 2016b. Improving terrain analysis and applications to RTS game AI. In *AIIDE*.

Whitehouse, D.; Powley, E. J.; and Cowling, P. I. 2011. Determinization and information set monte carlo tree search for the card game dou di zhu. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 87–94. IEEE.

Zinkevich, M.; Johanson, M.; Bowling, M. H.; and Piccione, C. 2007. Regret minimization in games with incomplete information. In *NIPS*, 1729–1736.