Single Believe State Generation for Partially Observable Real-Time Strategy Games

Alberto Uriarte and Santiago Ontañón
Computer Science Department, Drexel University
Philadelphia, PA, USA 19104
Email: {albertouri,so367}@drexel.edu

***Abstract*—Real-Time Strategy (RTS) games pose a big challenge due their large branching factor and real-time nature. This challenge is even bigger if we consider partially observable RTS games due to the *fog-of-war*. This paper focuses on extending Monte Carlo Tree Search (MCTS) algorithms for RTS games to consider partially observable settings. Specifically, we investigate sampling a single believe state consistent with a perfect memory of all the past observations in the current game, and using it to perform MCTS. We evaluate the performance of this approach in the μRTS game simulator, showing that the performance of this approach is only between 8%-15% lower than if we could observe the entire game state (e.g., by cheating).**

I. Introduction

Games with imperfect information are usually exponentially harder than games with perfect information. Two-player unbounded-length games where players have perfect information, i.e., those that can be modeled using *alternating Turing machine* (ATM) such as Chess, are **EXPTIME** [1]; while if there is private information, i.e., games that require *private alternating Turing machine* (PATM) such as *PRIVATE-PEEK*, they are **2-EXPTIME** [2]. Therefore, any attempt to try to find an optimal strategy will be intractable. This paper approaches the problem of solving games with imperfect information and large branching factors by generating a single believe state as close as possible to the real game state that can be used as the starting point for search by any game tree search algorithm, instead of the partially observed game state.

Real-Time Strategy (RTS) games pose unique challenges than other games: 1) they have a huge branching factor [3] and hence a huge believe state, 2) they are *real-time*, leaving very little time to make a decision, and 3) they have durative actions. Thus, the problem of handling partial observability in RTS games has not received yet sufficient attention, and current research in game tree search in partially observable domains has focused in games with smaller branching factors as described in Section II-B.

This paper proposes to use a combination of a *memory of past knowledge* and an *inference process* to maintain a single *believe state*, which is an estimation of the most probable game state given the current partially observable game state. We present experiments in the context of *Monte Carlo Tree Search* (MCTS) in the domain of μRTS[1].

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

The remainder of this paper is organized as follows. Section II introduces RTS games and game tree search for imperfect information games. Section III presents our single believe state generation approach. Section IV presents an empirical evaluation of the proposed approach, and finally Section V reviews the common approach of *determinization* to handle games of imperfect information.

II. Background

This section begins with an overview of RTS games and a formal definition of a generic RTS game. Finally it reviews the research done in imperfect information games.

A. 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) [4]. Designing AI techniques for RTS games is challenging because:

- 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 [3] (for comparison, that of Chess is about 35, and that of Go about 180).
- They are *real-time*, which means that: 1) RTS games typically execute at 10 to 50 decision cycles per second, leaving players with just a fraction of a second to decide the next action, 2) players can issue actions simultaneously, and 3) actions are durative.
- They are partially observable due the fog-of-war. So we cannot observe the parts of the map that are out of the sight of our 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 actions at the same time (one per unit). We will refer to those actions as *unit-actions* (a). A *player-action* (α) is the set of unit-actions that one player issues simultaneously in a given game cycle. Thus players issue only one *player-action* at any given time (which will consist of zero or more unit-actions). Notice that, even if unit-actions are durative, each player issues exactly one player-action at each decision cycle.

Figure 1 is a game state snapshot from a μRTS game that illustrates the idea of *fog-of-war*. There are two players, *max*

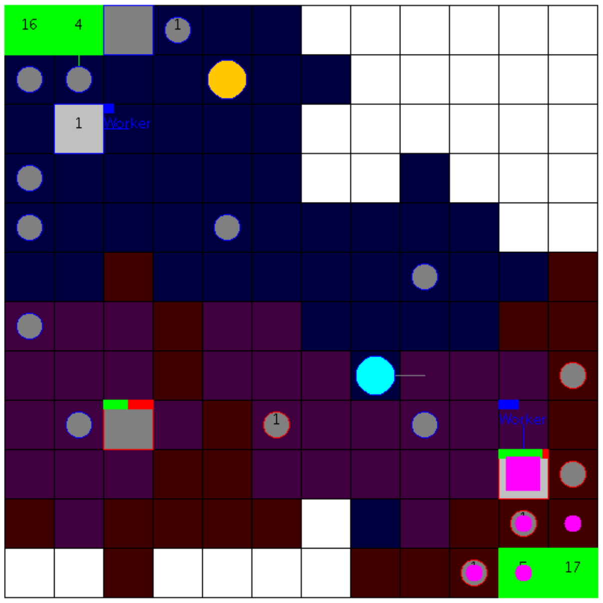

Fig. 1. A screenshot of the μRTS simulator. Square units correspond to "bases" (light grey, that can produce workers), "barracks" (dark grey, that can produce military units), and "resources mines" (green, from where workers can extract resources to produce more units), the circular units correspond to workers (small, dark grey) and military units (large, yellow or light blue).

(shown in blue), and *min* (shown in red). Squares with a white background are the locations not observable for any player, squares in blue are the ones observable by *max* player, squares in red are observable by *min* player, and violet squares are the locations observable by both players. In Figure 1 *max* player can only observe two *min*'s units (the red building in the bottom-left side and the worker in the bottom-center), while *min* player can see three *max*'s workers (the two near the barracks and one near the base). Figure 1 also shows the *believe state* for player *max* as magenta dots, showing that *max* is predicting correctly the location of two workers but missing the other two.

In the remainder of this paper, we will use the following definition for an RTS game. A partially-observable RTS game is 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 that given a player $p \in P$ and the current game state $s \in S$, returns the observable game 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 observation z, i.e., $I_p(z) = \{s \in S | O(p, s) = z\}$.

B. Game Tree Search in Partially Observable Domains

Zermelo's theorem [5] says that an optimal **deterministic strategy** (a.k.a. a *pure strategy*) can be found computing the Nash equilibrium for perfect information games, but in imperfect information games deterministic strategies can be exploited by the opponent, rendering them suboptimal. This was confirmed by Kuhn [6] showing that the optimal strategy for a simplified poker game is indeed a **randomized strategy** (a.k.a. a *behavior strategy*).

Imperfect information games are usually modeled as **extensive-form games**, they are like perfect information games but with *information sets* to combine all the states (a.k.a. *worlds*) that are indistinguishable to a player at the time she has to make a decision. Unfortunately, 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* can be exponential on the size of the game tree [6] or at least polynomial in the size of the game tree [7]. For this reason most researchers tried approximation algorithms or they made assumptions related to their particular game that not always hold true for RTS games. Some of them assume an opponent with full observability and they require to visit all the possible states, like *Best Defence model* [8], *Vector minimaxing* [9], or *Believe-state AND-OR tree search* [10]. Zinkevich et al. [11] presented *Counterfactual Regret Minimization* (CFR), an algorithm that converges to the Nash equilibrium and does not assume an opponent with full observability, but still needs to sample all the different states. Lanctot et al. [12] improved the previous algorithm to avoid sampling all the states 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 used simplification is known as *determinization* (see Section V for an overview of work in this area). The idea of *determinization* is to sample a *world* from the 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" [13] and as Frank et al. [8] pointed out, it raises many problems: 1) players must

TABLE I
LEAF CORRELATION, BIAS AND DISAMBIGUATION FACTOR IN EACH OF THE TESTED MAPS.

	8x8	**12x12**
Leaf Correlation	0.994	0.987
Bias	0.510	0.511
Disambiguation Factor	-0.010	0.046

behave the same way in states from the same information set (**strategy fusion**) and 2) the search can be "fooled" to pursue a highly rewarded state that cannot be reached under some information sets (**non-locality**). Additionally, in the case of imperfect information RTS games, *determinization* leads to **fake omniscience**, which happens 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. [14] 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*).

We analyzed these three properties for μRTS with two maps: one of size 8×8 and one of size 12×12. Computing the exact values by exploring the whole game tree is not feasible for RTS games. Therefore we approximate the values exploring the game tree with a *RandomBias* AI (explained in Section IV). Table I shows the approximated values after 10000 games in the 8×8 map and 2000 games in the 12×12 map, where we can do the following observations: 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*[2] is really low or even negative, this is because in RTS games we can lose information. Figure 2 shows the average disambiguation factor for a winning player, for a losing player, and for any player, computed as the difference in the number of non-observable tiles (positive means that the number of non-observable tiles reduces, and negative that it grows). The figure shows how at the beginning of the game with a 12×12 map, both players start gaining information, but this gain decays over time. Also, we see that the loser player tends to lose information toward the end of a game (which is normal, since a losing player will start losing units fast at the end of the game).

In this paper we specifically focus on the problem of generating a **single believe state** for games with durative actions and time constrains such as RTS games. Parker et al. [15] proposed 4 different sampling strategies for large information sets, but they require to perform depth-first search over the space of observations, something that could be expensive in domains with time constrains. Richards and Amir [16] improved the performance of the previous approach by modeling the problem as a Constrain Satisfaction Problem (CSP). In our approach, instead of generating a pool of valid believe states, we try to generate a single believe state as close as possible to the real game state.

[2]Since calculating the size of the information set in a RTS game is complex, due to their size, we used the number of non-observable tiles in the map as a proxy; the assumption is that the size of the information set is exponential in the size of non-observable tiles.

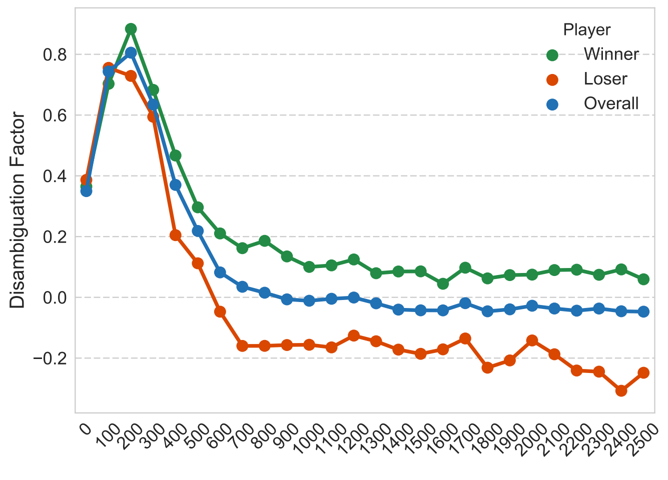

Fig. 2. Disambiguation factor over time in a 12x12 map.

TABLE II
IMPERFECT INFORMATION GAME PROPERTIES FOR DIFFERENT GAMES.

	Poker	**Kriegspiel**	**μRTS**
Initial board configurations	1,081*	1	1
Observable moves	YES	NO	YES
Hidden moves	NO	YES	YES
Exposing moves	NO	NO	YES
Gathering moves	NO	YES	YES

Partially observable RTS games have the following properties: 1) the set of **initial board configurations** is small; this is because players know which map they are playing on, and the only non-observable information at the start is where did the opponent player start. For example, in STARCRAFT there is a fixed set of "start locations" in each map, and thus a player only needs to determine in which of the possible start locations did the opponent start. In μRTS each map defines a given start location for both players, and thus the initial game state is known to both players. 2) **Observable moves** are those that are fully observable to all players. 3) Players can perform **hidden moves** for units that are under the fog-of-war. 4) **Exposing moves** occur when a player executes an action that reveals information to her opponent (like moving a unit out of the opponent's fog-of-war). 5) **Information gathering moves** are those that make some opponent information observable.

*Number of possible hands for a two players Texas hold'em game after the flop.

Table II shows a comparison of imperfect information game properties for different games.

III. Single Believe State Generation for RTS games

Although some work exists in the literature concerning handling partial observability in RTS games (such as Weber et al. [17] particle model to predict the locations of opponent's units under the fog-of-war in StarCraft), to the best of our knowledge this paper is the first attempt to deal with the partial observability in RTS games in the context of game tree search. Specifically, in this paper we explore how far we can go with only sampling one of the possible worlds in a large information set. Our hypothesis is that since the pace of RTS games is very fast, we do not need to find the optimal strategy every frame, therefore sampling a game state that approximates the actual game state sufficiently well, will be enough to converge towards a strong gameplay.

We assume that both players know the initial board configuration, i.e., s_0 is fully observable. 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). For other games like Poker this is not true since the opponent hand is unknown, but those are out of the scope of this paper. With this assumption we propose three different strategies for sampling a single state, which we call the *believe state* from the current information set:

- **Goal Seeker** (GS): Given an initial state s_0 with perfect information, this strategy records the location of each opponent unit that cannot move, U_{nm}. Then, in order to generate a believe state s_b from the current observation z, all the units in U_{nm} are added to z if they are not already there. If a unit present in the memory is destroyed from the game state, then it is removed from the memory as well. This can be seen as an extreme version of the *overconfidence* player model from [18], where the opponent always chooses to do nothing. We call this strategy the "goal seeker", since it has the effect of just remembering where the opponent base is, and thus, tends to go toward the enemy base right away.
- **Imperfect Memory** (IM): Given an initial state s_0 fully observable, this strategy records the location of **all** opponent units into a record U_o. Then, given a current observation z, if the location of a unit in U_o is visible, the unit is removed from U_o, otherwise we add the unit into z. This method basically adds to the current observation the "last known enemy unit location" of all the units that we cannot currently see, but that were observed at some point. We call this "imperfect memory" since if we saw a unit in a position x, which then became unobservable, if we observe x again, but there is no unit there, then we forget that we ever was a unit there.
- **Perfect Memory** (PM): This strategy acts like IM but adds an inference mechanism. The inference mechanism is used in two situations: 1) when the location of a unit in U_o is not visible, it updates the unit's location in U_o with the closest not observable location from the unit's location in the memory (i.e., it assumes the unit has moved, but just the minimum amount of move as for making the unit not observable); and 2) it adds units that we have never seen but they are 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). Inferred units are added in the closest non-observable location to the opponent's unit that caused the inference. In other words, this sampling never forgets a unit that it has seen (hence the name "perfect memory"), it also tries to guess their location, and it infers units that cannot be seen but that must be there.

From a point of view of game theory, these samplings capture the *memory of past knowledge* [19] at different degrees of accuracy.

IV. Experimental Evaluation

In order to evaluate the performance of each sampling strategy, we used the open-source μRTS game [20]. We ran experiments in six different maps: three standard game maps where each player starts with one base and one worker but with different map sizes (1BW8x8, 1BW10x10, 1BW12x12) and three maps where each player starts with four bases and four workers (4BW8x8, 4BW10x10, 4BW12x12). Given the small size of the maps being used for evaluation, the default visibility ranges of some of the units in μRTS were too large (basically the whole map was almost visible at the start in 8x8 maps). So, we reduced the visibility range of bases to 3, and of workers to 2. In our experiments, we used the following AIs:

- *RandomBiased*: It selects one of the possible player-actions at random, but with 5 times more probability of selecting an attack or a harvest action than any other action.
- *Partial Observability Worker Rush (POWorkerRush)*: It is a hard-coded strategy that constantly produces "Workers" to attack the nearest target or if there is not enemies they move to the nearest not visible location. It only uses one worker to mine resources.
- *Partial Observability Light Rush (POLightRush)*: Like *POWorkerRush* but it builds a barracks, and then constantly produces "Light" military units instead of "Workers".
- *ϵ-Greedy MCTS* (ϵ-MCTS): As a baseline for game tree search we selected the well known MCTS with an ϵ-greedy sampling strategy (ϵ = 0.25 as shown to be best in [20]) as a tree policy, and a *RandomBiased* default policy to simulate 200 game cycles. The evaluation function used was: the sum of the cost in resources of all the player units in the board weighted by the square root of the fraction of hit-points left, then subtract the same sum for

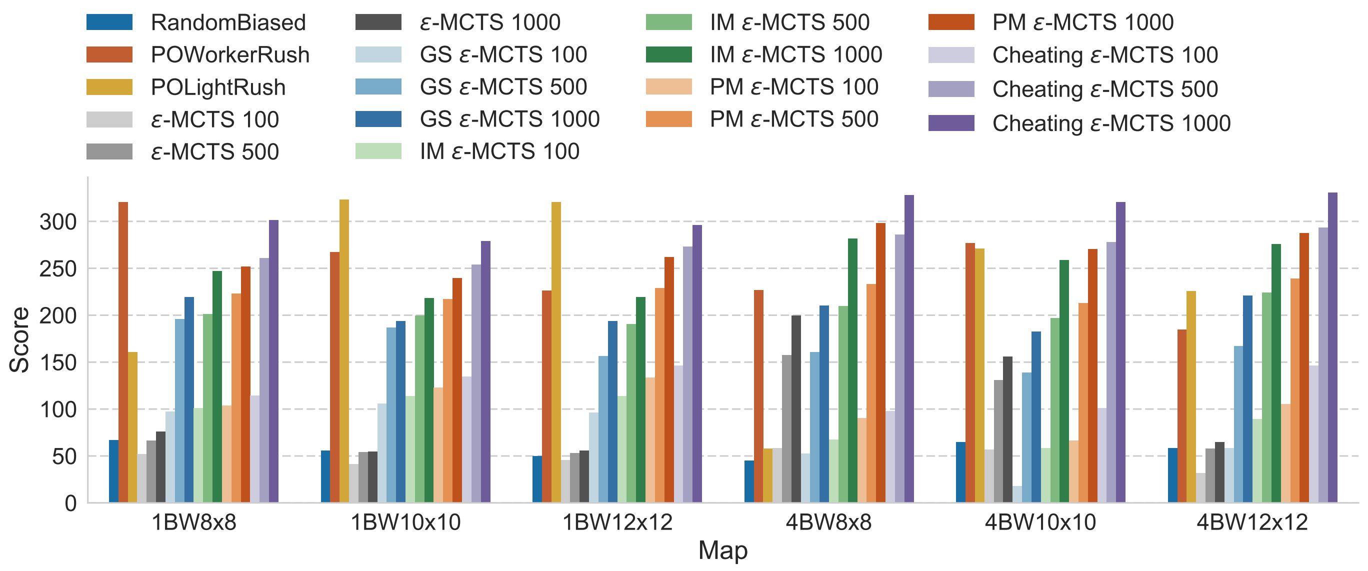

Fig. 3. Accumulated score (wins + 0.5 × ties) obtained by each AI in each of the different μRTS maps (maximum score would be 340, meaning winning every single game).

the opponent player. We tested with different number of playouts per cycle (100, 500 and 1,000). Although there are better game tree search algorithms for RTS games like Portfolio GReedy Search [21], LSI [22] NaiveMCTS [20] or Informed MCTS [23], [24] we wanted to test the performance of a partially observable game state with a simple and well known game tree search algorithm since the performance difference between the different single believe state samplings should be analog for other search tree algorithms. This AI uses directly the partially observable state, i.e., if there is no enemies in the partially observable state the AI thinks it won and it might return an arbitrary action.

- *GS ϵ-MCTS*: Like ϵ-MCTS AI but at each frame the partially observable game state is enhanced with the Goal Seeker sampling.
- *IM ϵ-MCTS*: Like ϵ-MCTS AI but at each frame the partially observable game state is enhanced with the Imperfect Memory sampling.
- *PM ϵ-MCTS*: Like ϵ-MCTS AI but at each frame the partially observable game state is enhanced with the Perfect Memory sampling.
- *Cheating ϵ-MCTS*: Like ϵ-MCTS AI but this time instead of a partially observable state we receive a fully-observable game state. Notice that this amounts to cheating, since the opponent still receives a partially-observable game state. The purpose of using this AI is to offer an upper bound of the performance that can be expected out of ϵ-MCTS.

For each pair of AIs (3 hard-coded and 5 MCTS, with 3 different playout budgets each, results in 18 different AIs, and a total of $\binom{18}{2} = 153$ match-ups), we ran 20 games per map and per match-up (each AI plays 20 games as player 1 and another 20 games as player 2 in the same map against the same opponent) in 6 different maps, resulting in a total of $153 \times 20 \times 6 = 18360$ games. We limited each game to 3000 cycles (5 minutes), after which we considered the game a tie.

A. Results

Figure 3 shows the summarized results of our experiments. For each map and for each AI, we show a "score", calculated as $wins + 0.5 \times ties$. From this Figure 3 we can make the following observations:

- **Hard-coded strategies**: These strategies perform very well in maps with a single base and worker, since that's the scenario for which they were designed. The main drawback of hard-coded strategies is that they cannot handle all type of situations. This can be seen in maps with 4 bases where *POWorkerRush* and *POLightRush* do not know how to exploit all the bases and they perform poorly against MCTS. We can also observe how the performance of *POWorkerRush* decreases by the size of the map, while *POLightRush*'s performance increases. This shows how these strategies lack generality (they perform very well in some situations, outperforming all other techniques, but they under-perform in others).
- **Number of playouts**: As expected, the performance of MCTS based AIs increases if we perform more playouts (100, 500 and 1000 were tested in our experiments). Showing a large improvement between 100 and 500 but a small one between 500 and 1000. For the purposes of our evaluation we fixed the number of playouts for comparison. But keep in mind that for RTS games the budget is usually limited by time and not by playouts, therefore MCTS AIs that can perform more playouts in the same amount of time will perform in practice better

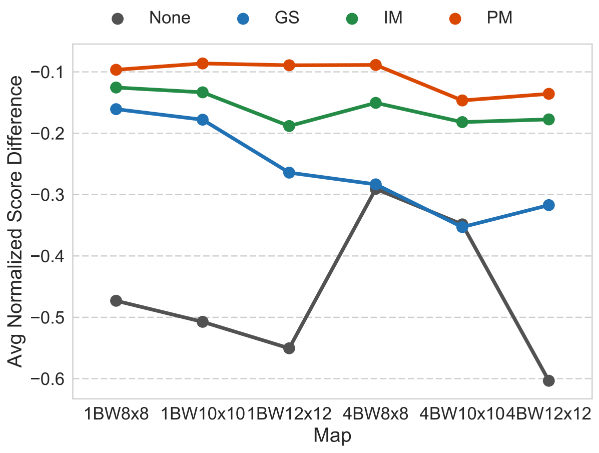

Fig. 4. Avg normalized score difference of each believe state sampling respect cheating.

than other theoretically stronger MCTS AIs that need more time to finish a playout.

- **Size of non-observable map positions**: The partially observable game state property in RTS games comes from the fog-of-war that establishes that we cannot observe the parts of the map that are out of sight of our units. Therefore if we are able to position all our units to cover all the map, the game state becomes full observable. This property can be seen in 4BW8x8 map where we already begin with spread units that can cover almost the entire small map. Hence the good performance of ϵ-*MCTS* in the 4BW8x8 map.
- **Single believe state generation performance**: As we can see in Figure 3, even the simple GS sampling is better than not using any believe state estimation at all. IM is better than GS and PM is better than IM ($none < GS < IM < PM < cheating$). Moreover, we can see that the performance of PM is very close to that of cheating in most scenarios (the largest difference is seen in the 4BW10x10 map).
Figure 4 shows a deeper analysis of how accurate are the believe state estimations made by GS, IM and PM by computing the average normalized score ($wins + 0.5 \times ties$) difference for each believe state sampling against cheating (perfect information). It reveals that PM only suppose a 8%-15% performance decreasing while without any sampling can be a 50% performance penalty. Another observation is that while the performance penalty of PM is more constant across the maps, the performance of the other believe state samplings is more sensitive to the map size.

Analyzing the game replays we also observed that PM exhibits and emergent behavior of units trying to find a previously seen enemy. However this is an illusion of gathering information or scouting since the game tree search believes it has perfect information and tries to attack the unit in the believe state, and when it approaches the believe state keeps relocating the position of the unit in the believe state. But this is also the key of the higher performance of PM since it will search for remaining opponent units in the map, while other sampling will stop searching and hence ending more games in a tie.

We also analyzed the similarity between the generated believe state and the real game state. We used a measure inspired in the Jaccard index (a well known similarity measure between sets: the size of their intersection divided by the size of their union). Given the full observable game state s (s is a set of units), a believe state s_p from the point of view of player p, and given the opponent player q, the similarity between s and s_p in estimating the units of q is defined as:

$$J(q, s, s_p) = \frac{unitDist(q, s, s_p)}{units(q, s_p) + units(q, s) - unitDist(q, s, s_p)}$$

Where $units(q, s)$ is the number of units of player q in s, and $unitDist(q, s, s_p)$ is the sum of normalized distances between the units of player q that exist both in s and s_p. This measure gives us the notion of how well are we guessing the opponent's units. Figure 5 shows the average Jaccard index over time and the 95% confidence interval (CI) for each believe state sampling and for each map. As we can see PM outperforms the other believe state sampling strategies by a large margin. Notice how for maps bigger than 8×8 the prediction accuracy of all samplings starts to decrease quickly at the beginning of the game until combat is engaged (usually at frame 250-300) to increase again until frame 600-750 and finally decrease slowly until the end. To avoid this "depression" at the beginning of the game, humans players usually send a scout to the enemy's base to gather information. Also, small maps or maps with units covering a good portion of the map from the initial state (1BW8x8, 4BW8x8, 4BW10x10) do not exhibit the "depression", meaning that from the beginning we almost have perfect information and we keep losing information over time (specially for the loser player). The fact that the 95% CI grows over time is because we have less samples of long games, especially for small maps.

V. Related Work

As mentioned above, the most common technique to handle partial information in game tree search is *determinization*. We review work in this area in this section.

Monte Carlo Sampling [25]: 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 [26], 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) [27].

Statistical Sampling [15]: 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

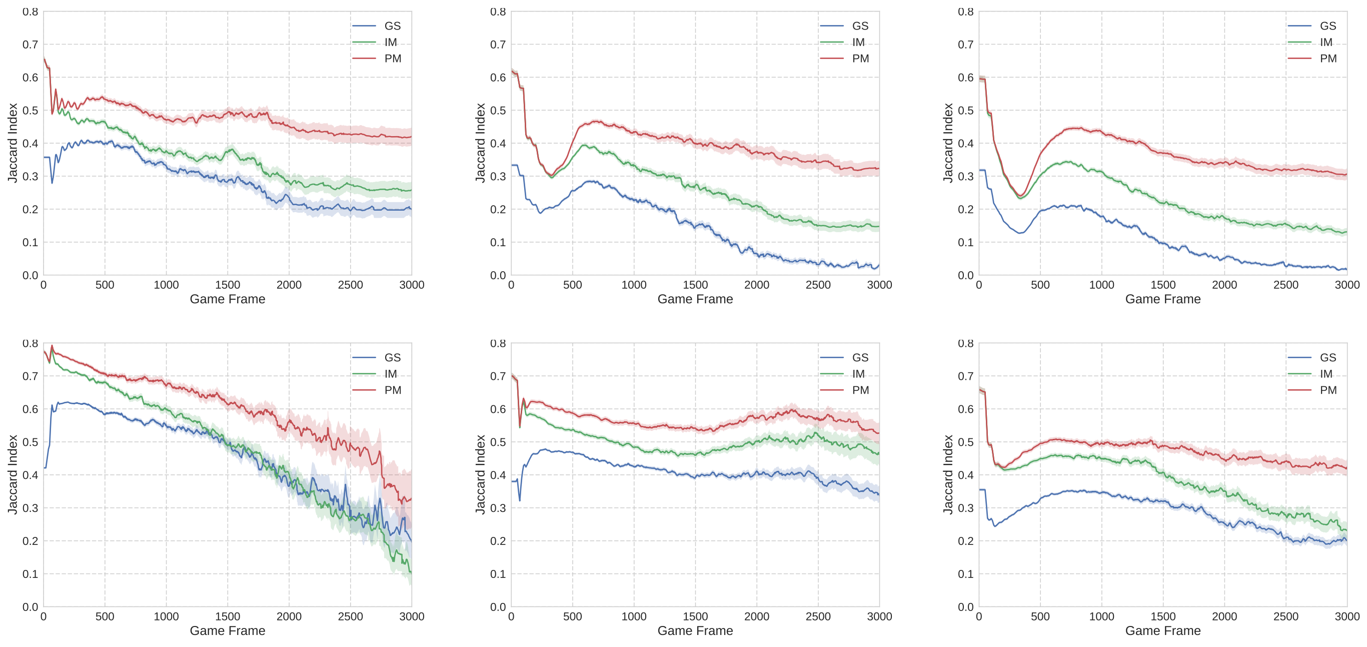

Fig. 5. Average Jaccard index between the believe state and the real game state (1 means perfect matching, 0 means nothing in common) over time (game frames) and 95% CI. Top row are the maps with one base: 1BW8x8 (top left), 1BW10x10 (top center) and 1BW12x12(top right); bottom row are maps with four bases: 4BW8x8 (bottom left), 4BW10x10 (bottom center), and 4BW12x12(bottom right).

Hybrid Sampling (HS). For each sample state (world) it uses alpha-beta and decides the move that maximizes the score.

Information-set search [18]: Parker et al. proposed a game-tree search 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 [28]: 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 [29]: It uses *root parallelization* where each root is a different determinization.

Single Observer Information Set MCTS (SO-ISMCTS) [30]: The idea is that on each iteration of the MCTS it makes a random root determinization to get the set of legal actions and it uses information sets as nodes in the game tree. This algorithm makes several assumptions: 1) the same action (a) applied to all the states of an information set (I) transition to the same information set (I'), 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) [30]: 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.

VI. Conclusions

This paper presented a comparison of strategies for single believe state generation for partially observable RTS games. RTS games pose a particular imperfect information game problem where the search space is larger than other imperfect information games like Kriegspiel, the time between turns is really short and the information sets can grow really fast. For these reasons we limited ourselves to sampling a single believe state as close as possible to the real game state.

Experimental results indicate that the *Perfect Memory* believe state sampling strategy only decreases the performance of a game tree search algorithm (like MCTS) by 8%-15% compared to having access to the whole game state (cheating).

As part of our future work we want to incorporate the concept of information sets into game tree search algorithms for RTS games in order to being able to generate gathering/hiding information actions (which would never be generated with the proposed approach). We also want to relax the assumption that we have perfect information of the initial state to a set of possible initial game states. This can be seen for example as if we know that the opponent will have one base in one of the possible initial base locations, but we do not know which one. Lastly, we would like to update the believe state taking into account the fact that actions are durative. For example, if a barracks in the believe state starts training a marine, even if we do not see the barracks any more, we can infer that in a certain amount of time, the opponent will have a marine available.

References

[1] A. S. Fraenkel and D. Lichtenstein, "Computing a perfect strategy for $n \times n$ Chess requires time exponential in n," *Combinatorial Theory, Series A*, vol. 32, no. 2, pp. 199–214, 1981.

[2] J. H. Reif, "The complexity of two-player games of incomplete information," *Journal of computer and system sciences*, vol. 29, no. 2, pp. 274–301, 1984.

[3] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *TCIAIG*, vol. 5, no. 4, pp. 293–311, 2013.

[4] M. Buro, "Real-time strategy games: a new AI research challenge," in *IJCAI*. Morgan Kaufmann Publishers Inc., 2003, pp. 1534–1535.

[5] U. Schwalbe and P. Walker, "Zermelo and the early history of game theory," *Games and economic behavior*, vol. 34, no. 1, pp. 123–137, 2001.

[6] H. W. Kuhn, "A simplified two-person poker," *Contributions to the Theory of Games*, vol. 1, pp. 97–103, 1950.

[7] D. Koller and A. Pfeffer, "Generating and solving imperfect information games," in *IJCAI*, 1995, pp. 1185–1193.

[8] I. Frank and D. A. Basin, "Search in games with incomplete information: A case study using bridge card play," *Artif. Intell.*, vol. 100, no. 1-2, pp. 87–123, 1998.

[9] I. Frank, D. A. Basin, and H. Matsubara, "Finding optimal strategies for imperfect information games," in *AAAI/IAAI*, 1998, pp. 500–507.

[10] S. Russell and J. Wolfe, "Efficient belief-state AND-OR search, with application to Kriegspiel," in *IJCAI*, vol. 19, 2005, p. 278.

[11] M. Zinkevich, M. Johanson, M. H. Bowling, and C. Piccione, "Regret minimization in games with incomplete information." in *NIPS*, 2007, pp. 1729–1736.

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

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

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

[15] A. Parker, D. Nau, and V. Subrahmanian, "Game-tree search with combinatorially large belief states," in *IJCAI*, 2005, pp. 254–259.

[16] M. Richards and E. Amir, "Information set generation in partially observable games." in *AAAI*, 2012.

[17] B. G. Weber, M. Mateas, and A. Jhala, "A particle model for state estimation in real-time strategy games," in *AIIDE*, 2011, pp. 103–108.

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

[19] G. Bonanno, "Memory and perfect recall in extensive games," *Games and Economic Behavior*, vol. 47, no. 2, pp. 237–256, 2004.

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

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

[22] A. Shleyfman, A. Komenda, and C. Domshlak, "On combinatorial actions and cmabs with linear side information," in *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. IOS Press, 2014, pp. 825–830.

[23] S. Ontañón, "Informed monte carlo tree search for real-time strategy games," in *CIG*, 2016.

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

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

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

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

[28] P. Ciancarini and G. P. Favini, "Monte carlo tree search techniques in the game of kriegspiel." in *IJCAI*, vol. 9, 2009, pp. 474–479.

[29] D. Whitehouse, E. J. Powley, and P. I. Cowling, "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*. IEEE, 2011, pp. 87–94.

[30] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.