Multi-scale Bayesian modeling for RTS games: an application to StarCraft AI

Gabriel Synnaeve (gabriel.synnaeve@gmail.com) Pierre Bessière (pierre.bessiere@college-de-france.fr)

***Abstract*—This paper showcases the use of Bayesian models for real-time strategy (RTS) games AI in three distinct core-components: micro-management (units control), tactics (army moves and positions), and strategy (economy, technology, production, army types). The strength of having end-to-end probabilistic models is that distributions on specific variables can be used to inter-connect different models at different levels of abstraction. We applied this modeling to StarCraft, and evaluated each model independently. Along the way, we produced and released a comprehensive dataset for RTS machine learning.**

***Index Terms*—Bayesian modeling, RTS AI, real-time strategy, video games, StarCraft, tactics, micro-management**

I. Introduction

Research on video games rests in between research on real-world robotics and research on simulations or theoretical games. Indeed, artificial intelligences (AIs) evolve in a simulated world (no sensors and actuators problems) that is also populated with human-controlled agents and/or other AI agents on which we often have no control. Thus, video-games constitutes a good middle-ground for experimenting with robotic-inspired and cognitively-inspired techniques and models. Moreover, the gigantic complexity of RTS AI pushes researchers to try different approaches than for strategic board games (Chess, Go...).

We will first show how the complexity of game AI (and particularly RTS AI) is several order of magnitudes larger than those of board games. Thus, abstractions and simplifications are necessary to work on the complete problem. We will then explain how building abstractions with Bayesian modeling is one possible framework to deal with game AI's complexity by dealing efficiently with uncertainty and abstraction. Then, we will successively present our three hierarchical abstraction levels of interconnected models: micro-management, tactical, and strategic Bayesian models. We will see how to do reactive units control, and how to take objectives from a tactical model. Then we will show how to infer the opponent's tactics using knowledge of our strategic prediction. Finally, we will do a detailed analysis of an army composition model.

II. RTS AI Problem

A. Difficulty

RTS is a sub-genre of strategy games where players need to build an economy (gathering resources and building a base) and military power (training units and researching technologies) in order to defeat their opponents (destroying their army and base). From a theoretical point of view, the main differences between RTS games and traditional board games are that RTS have simultaneous moves (all players can move at the same time and as much units as wanted), durative actions (taking some time to complete), incomplete information (due to the "fog-of-war": the player can only see the dynamic state of the world/map where they have units), sometimes non-deterministic (only slightly for StarCraft), and the players need to act in "real-time" (24 game frames per second for StarCraft). As a metaphor, RTS games are like playing simultaneous moves Chess while playing the piano to move pieces around the board. More information about StarCraft gameplay can be found in [1] and in pp.59-69 of [2].

Traditional (non-video) game AI takes roots in solving board strategy games. In those games, the complexity of the game can be captured by the computational complexity of the tree search in a "min-max like" algorithm, which is defined by the branching factor b and the depth d of the tree. For instance for Chess [3]: $b \approx 35, d \approx 80$. Table I gives an overview of such a complexity (first column) for several games and game genres. For video games, we estimate the human difficulty from the players choices and actions (except for RTS for which we do both the strict computational complexity and the human difficulty): b is the number of possible actions each time the player takes an action, and d/min is the average number of (discrete, not counting mouse movements as continuous trajectories) actions per minute (APM). Table I also shows a qualitative analysis of the amount of partial information, randomness, hierarchical continuity (how much an abstract decision constrains the player's actions), and temporal continuity (how much previous actions constrain the next actions).

B. Method

We operate two kinds of simplifications of this very complex problem of full-game real-time strategy AI. On the one hand, we simplify decisions by taking into account their sequentiality. We consider that a decision taken at a previous time $t-1$ (softly) "prunes" the search of potential actions at time t, for a given level of reasoning (given level of abstraction). This corresponds to doing a Markovian hypothesis in probabilistic modeling. For instance, as shown (in red, left-to-right arrows) on Fig. 1, a tactical decision to attack from the front (F) is more likely followed by a hit-and-run (H) than an attack from the back (B) or a sneaky infiltration (I). On the other hand, we decide of hierarchical levels of abstractions at which

G. Synnaeve was (during this work) with the LSCP at ENS / EHESS / CNRS, Paris, France. He is now at Facebook AI Research, Paris, France.

P. Bessière is with the CNRS/Sorbonne Univ./UPMC/ISIR, Paris, France.

TABLE I: Computational complexity of different game genres

quantization in increasing order: no, negligible, few, some, moderate, much

Game	Combinatory	Partial information	Randomness	Hierarchical continuity	Temporal continuity
Chess	$b \approx 35; d \approx 80$	no	no	some	few
Go	$b \approx 30 - 300; d \approx 150 - 200$	no	no	some	moderate
Limit Poker	$b \approx 3; d/hour \in [20 \dots 240]$	much	much	moderate	few
Time Racing	$b \approx 50 - 1,000; d/min \approx 60+$	no	no	much	much
Team FPS	$b \approx 100 - 2,000; d/min \approx 100$	some	some	some	moderate
FFPS duel	$b \approx 200 - 5,000; d/min \approx 100$	some	negligible	some	much
MMORPG	$b \approx 50 - 100; d/min \approx 60$	few	moderate	moderate	much
RTS	d/min(=APM) ≈ 300	much	negligible	moderate	some
human	$b \approx 200; d \approx 7,500$				
full complexity	$b \approx 30^{60}$; $d \approx 36,000$				

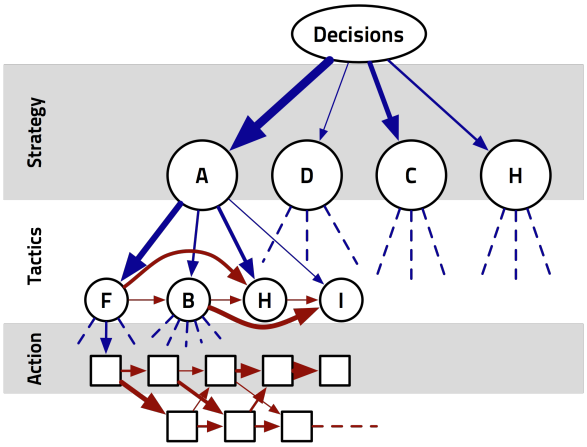

Fig. 1: Sequential (horizontal, red) and hierarchical (vertical, blue) decision constraints. At the strategic level: A, D, C, H respectively stand for attack, defend, collect, hide ; while at the tactical level: F, B, H, I respectively stand for front, back, hit-and-run, infiltrate. The squares correspond to actionable (low level) decisions, like moving a unit or making it attack a target.

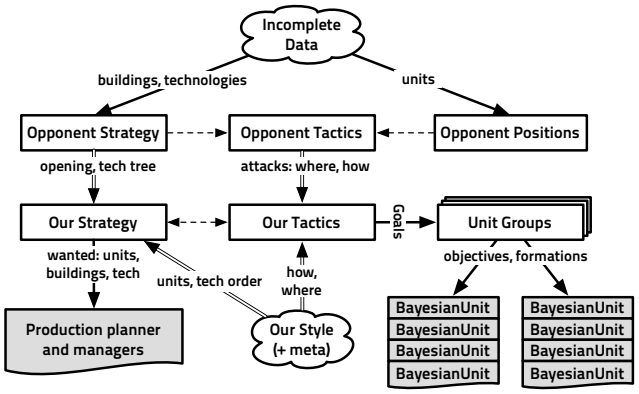

Fig. 2: Information-centric view of the architecture of our StarCraft bot's major components. Arrows are labeled with the information or orders they convey: dotted arrows are conveying constraints, double lined arrows convey distributions, plain and simple arrows convey direct information or orders. The gray parts perform game actions (as the physical actions of the player on the keyboard and mouse).

we should take decisions that impact the level below, pruning the hierarchical decisions according to what is possible. For instance, as shown (in blue, top-down arrows) on Fig. 1, if our strategic decision distribution is more in favor of attacking (A) instead of defending (D), collecting (C) or hiding (H), this constrains the subsequent tactics too. We will see that these levels of abstractions are easily recoverable from the rules/structure of the game.

So, we decided to decompose RTS AI in the three levels which are used by the gamers to describe the game: *strategy*, *tactics*, *micro-management*. These levels are shown from left to right in the information-centric decomposition of our StarCraft bot in Fig. 2. Parts of the map not in the sight range of the player's units are under fog-of-war, so the player has only partial information about the enemy buildings, technologies and army (units positions). The way by which we expand the tech tree, the specific units composing the army, and the general stance (aggressive or defensive) constitute what we call *strategy* (left part of Fig. 2). At the lower level (bottom right in Fig. 2), the actions performed by the player (human or not) to optimize the effectiveness of its units is called *micro-management*. In between lies *tactics*: where to attack, and how. A good human player takes much data in consideration when choosing: are there flaws in the defense? Which spot is more worthy to attack? How much am I vulnerable for attacking here? Is the terrain (height, chokes) to my advantage? The concept of strategy is a little more abstract: at the beginning of the game, it is closely tied to the build order and the intention of the first few moves and is called the *opening*, as in Chess. Then, the long term strategy can be partially summed up by a few indicators: initiative (is the player leading or adapting) and the "technology advancement vs. army production vs. economical growth" distribution of resources.

C. Bayesian programming

Probability is used as an alternative to classical logic and we transform incompleteness (in the experiences, observations or the model) into uncertainty [4]. We now present Bayesian programs (BP), a formalism that can be used to describe entirely any kind of Bayesian model, subsuming Bayesian networks and Bayesian maps, equivalent to probabilistic factor graphs [5]. There are mainly two parts in a BP, the **description** of how to compute the joint distribution, and the **question(s)** that it will be asked.

The description consists in extracting the relevant *variables* $\{X^1, \ldots, X^n\}$ and explaining their dependencies by *decomposing* the joint distribution $\mathrm{P}(X^1 \ldots X^n | \delta, \pi)$ with existing preliminary knowledge π and data δ. The *forms* of each term of the product specify how to compute their distributions: either parametric forms (laws or probability tables, with free parameters that can be learned from data δ) or recursive questions to other Bayesian programs.

Answering a question is computing the distribution $\mathrm{P}(Searched|Known)$, with $Searched$ and $Known$ two disjoint subsets of the variables.

$$\begin{aligned}&\mathrm{P}(Searched|Known)\\ &= \frac{\sum_{Free} \mathrm{P}(Searched,\ Free,\ Known)}{\mathrm{P}(Known)}\\ &= \frac{1}{Z} \times \sum_{Free} \mathrm{P}(Searched,\ Free,\ Known)\end{aligned}$$

$$BP \begin{cases} Desc. \begin{cases} Spec.(\pi) \begin{cases} Variables \\ Decomposition \\ Forms\ (Parametric\ or\ Program) \end{cases} \\ Identification\ (based\ on\ \delta) \end{cases} \\ Question \end{cases}$$

Bayesian programming originated in robotics [6] and evolved to all sensory-motor systems [7]. For its use in cognitive modeling, see [8], and for its first use in first-person shooters, see [9], for Massively Multi-Player Online Role-Playing Games, see [10].

III. Micro-management

The problem at hand is the optimal control of units in a (real-time) huge adversarial actions space (collisions, accelerations, terrain, damages, areas of effects, foes, goals...). Our approach is related to potential fields for navigation [11], and influence maps for maneuvering [12]. We treat the problem in a decentralized fashion, to be able to specify or learn a policy for each agent, e.g. like in CLASS$_{QL}$ [13]. By opposition to (Monte Carlo) tree search (MCTS) approaches [14], our approach does not search for an optimal battle control. There is little doubt that "UCT considering durations" [14] with a good evaluation function would win against our bot in small scale flat-terrain battles. Nevertheless, the goal of our approach is to be robust to all the cases of real (cliffs, ramps, clutter) and large in-game battles, while making it simple to plug the higher level (tactical) order in our units control. A more in-depth study of previous works on this problem can be found in [1] or in pp.74-76 of [2]. More details about this section can also be found in [15].

A. Model

For micro-management, the magic word is "focus fire". The quicker you destroy enemy units, the less they will have time to damage your army. For that, we use a heuristic based ordering of the target for each unit. This can be achieved by using a data structure (a bidirectional map), shared by all our units engaged in the battle, that stores the damages corresponding to future allied attacks for each enemy units. Whenever a unit will fire on a enemy unit, it registers there the future damages on the enemy unit. As attacks are not all instantaneous and there are reload times we can move our units effectively to avoid damage or close-in on their targets during the downtime. Except for this sharing of information and for collision maps, we decided to control our units independently.

Based on this targeting heuristic, we design a very simple finite-state machine (FSM) based unit: when the unit is not firing, it will either flee damages if it has taken too much damages and/or if the differential of damages is too strong, or move to be better positioned in the fight (which may include staying where it is). In this simple unit, the *flee()* function just tries to move the unit in the direction of the biggest damages gradient (towards lower potential damages zones). The *fightMove()* function tries to position the units better: in range of its priority target, so that if the priority target is out of reach, the behavior will look like: "try to fire on target in range, if it cannot (reloading or no target in range), move towards priority target". As everything is driven by the firing heuristic (that we will also use for our Bayesian unit), we call this AI the Heuristic Only AI (HOAI).

The difference between a simple "HOAI" presented above and Bayesian units are in *flee()* and *fightMove()* functions. These functions are performed by deciding where to go according to a Bayesian program, shown in Fig. 3. The main random variables of this model are:

- $Dir_{i \in [\![1 \ldots n]\!]} \in \{True, False\}$: at least one variable for each atomic direction the unit can go to. $\mathrm{P}(Dir_i = True) = 1$ means that the unit will certainly go in direction i ($\Leftrightarrow \vec{d_i}$). For example, in StarCraft we use the 24 atomic directions (48 for the smallest and fast units as we use a proportional scale) plus the current unit position (stay where it is) as shown in Fig. 4.
- $Obj_{i \in [\![1 \ldots n]\!]} \in \{True, False\}$: adequacy of direction i with the objective (given by the tactical model described in the next section). In our StarCraft AI, we use the scalar product between the direction i and the objective vector (output of the pathfinding) with a minimum value (0.3 in *move* mode for instance) so that the probability to go in a given direction is proportional to its alignment with the objective.
  - For *flee()*, the objective is set in the direction which flees the potential damages gradient (corresponding to the unit type: ground or air).
  - For *fightMove()*, the objective is set by the units group either to retreat, to fight freely or to march aggressively towards the goal.
- $Dmg_{i \in [\![1 \ldots n]\!]} \in \{no, low, medium, high\}$: potential damage value in direction i, relative to the unit base health points, in direction i. In our StarCraft AI, this is directly drawn from two constantly updated potential damage maps (air, ground).
- $A_{i \in [\![1 \ldots n]\!]} \in \{free, small, big\}$: occupation of the direction i by an allied unit. The model can effectively use many values (other than "occupied/free") because directions may be multi-scale (for instance we indexed the scale on the size of the unit) and, in the end, small and/or fast units have a

much smaller footprint, collision wise, than big and/or slow. In our AI, instead of direct positions of allied units, we used their linear interpolation at "$\frac{\text{dist}(unit,\vec{d_i})}{unit.speed}$" frames later (i.e. the time it takes the unit to go to $\vec{d_i}$).

- $E_{i\in[\![1\ldots n]\!]} \in \{free, small, big\}$: occupation of the direction i by an enemy unit. As above.
- $Occ_{i\in[\![1\ldots n]\!]} \in \{free, building, staticterrain\}$: Occupied, repulsive effect of buildings and terrain (cliffs, water, walls).

Bayesian program
- Description
  - Specification (π)
    - $Variables$
    - $Dir_{1:n}, Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n}$
    - $Decomposition$
    - $\mathrm{P}(Dir_{1:n}, Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n}) =$
    - $\prod_{i=1}^{n} \left[\mathrm{P}(Dir_i)\mathrm{P}(Obj_i|Dir_i)\mathrm{P}(Dmg_i|Dir_i)\right.$
    - $\left.\mathrm{P}(A_i|Dir_i)\mathrm{P}(E_i|Dir_i)\mathrm{P}(Occ_i|Dir_i)\right]$
    - $Forms$
    - $\mathrm{P}(Dir_i)$: prior on directions (crossing policy)
    - $\mathrm{P}(XYZ_i|Dir_i)$: probability tables
  - $Identification$
  - reinforcement learning or hand specified
- $Question$
- fight moving/fleeing :
- $\mathrm{P}(Dir_{1:n}|Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n})$

Fig. 3: Bayesian program of the *flee()* and *fightMove()* behaviors. This is mainly a Bayesian sensors fusion model.

The $\mathrm{P}(XYZ_i|Dir_i)$ probability tables in Fig. 3 were hand specified to be linearly or quadratically repulsive or attractive dependent on the case. We kept it simple and evaluated them like that, because that is what we used in our bot. However, these tables could be learned so that we use much more tailored policies. One way to learn them would be to use (hierarchical) reinforcement learning as in [16] (on Wargus) or [17] (on StarCraft), but we would have to learn different parameters for different battle scenarios (that [17] started to do for small-scale combats). More about the learning step can be found in pp.89-92 of [2].

From there, the unit can either go in the most probable Dir_i or sample through them. We describe the effect of this choice in the next section. A simple Bayesian fusion from 3 sensory inputs is shown in Fig. 4, in which the final distribution on Dir peaks at places avoiding damages and collisions while pointing towards the goal.

B. Results

We produced three different AI to run experiments with, along with the original AI (OAI) from StarCraft:

- Heuristic only AI (HOAI), as described above: this AI shares the target selection heuristic with our Bayesian AI models and will be used as a baseline reference to avoid the bias due to the target selection heuristic.
- Bayesian AI picking best (BAIPB): this AI follows the model of section and selects the most probable Dir_i as movement.

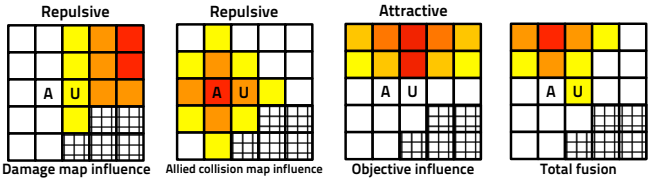

Fig. 4: Simple example of Bayesian fusion from 3 sensory inputs (damages, collisions avoidance, and goal attraction). The grid pattern represents statically occupied terrain, the unit we control is in U, an allied unit is in A. Red represents the highest probabilities, white the lowest. The result is displayed on the rightmost image, where our unit should try and go to the red square.

- Bayesian AI sampling (BAIS): this AI follows the model of section and samples through Dir_i according to their probability (i.e. it samples a direction in the Dir distribution).

The experiments consisted in having the AIs fight against each others on a micro-management scenario with mirror matches of 12 and 36 ranged ground units (Dragoons). In the 12 units setup, the units movements during the battle are easier (less collision probability) than in the 36 units setup. In these special maps, we instantiate only one units group manager and as many Bayesian units as there are units to control. The results are presented in Table II.

12 units	OAI	HOAI	BAIPB	BAIS
OAI	(50%)			
HOAI	59%	(50%)		
BAIPB	93%	97%	(50%)	
BAIS	93%	95%	76%	(50%)
36 units	**OAI**	**HOAI**	**BAIPB**	**BAIS**
OAI	(50%)			
HOAI	46%	(50%)		
BAIPB	91%	89%	(50%)	
BAIS	97%	94%	97%	(50%)

TABLE II: Win ratios over at least 200 battles of OAI, HOAI, BAIPB and BAIS in two mirror setups: 12 and 36 ranged units. Top: 12 units (12 vs 12) setup. Bottom: 36 units (36 vs 36) setup. Read line vs column: for instance HOAI won 59% of its matches against OAI in the 12 units setup.

These results show that our heuristic (HAOI) is comparable to the original AI (OAI), perhaps a little better, but induces more collisions as we can see its performance diminish a lot in the 36 units setup vs OAI. For Bayesian units, the "pick best" (BAIPB) direction policy is very effective when battling with few units (and few movements because of static enemy units) as proved against OAI and HOAI, but its effectiveness decreases when the number of units increases: all units are competing for the best directions (to $flee()$ or $fightMove()$) and they collide. The sampling policy (BAIS) has way better results in large armies, and significantly better results in the 12 units vs BAIPB. BAIPB may lead our units to move inside the "enemy zone" a lot more to chase priority targets (in *fightMove()*) and collide with enemy units or get kill. Sampling entails that the competition for the best directions is distributed

among all the "good enough" positions, from the units point of view.

As for our Bayesian units in practice within the bot, they were able to deal with most situations efficiently. For instance, we easily obtained a "kiting" (hit-and-run) behavior with fast and ranged units, as in [18], when the $Objective$ set by the tactical level was not to retreat or to go/pass through the opponent's army.

IV. Tactics

The problem is to predict where, when, and how the opponent can attack us, and, similarly, predict where, when and how we should attack them. Our approach is built on terrain analysis methods, in particular [19] that extracted choke points and regions of StarCraft maps for a pruned Voronoi diagram. Tactical analysis often uses particle filtering to track opponent units, as in [20], or [21] more specifically for StarCraft. We took a slightly different approach by evaluating and tracking forces at the level of discrete regions, and making our model symmetrical so that we use it to take decisions. A more in-depth study of previous works on this problem can be found in [1] or in pp.96-97 of [2]. More details about this section can be found in [22] and [23].

A. Model

We used regions from [19] along with choke-centered sub-regions (see [2] or [23] for details) as our basic spatial unit. We considered 4 main types of attacks: ground attacks (most common), aerial attacks (units that can attack flying units are rare, and flying units can cross all terrain), invisible attacks (which cannot be defended without detection), and drops (using flying transports, most often "backstabbing"). For each region, we used tactical (relative distance to armies), economical (relative distance to mining/production) and defense (against all type of attacks: ground, air, invisible) scoring heuristics. With n regions, we can extract the following random variables:

- $A_{1:n} \in \{true, false\}$, A_i: attack in region i or not?
- $E_{1:n} \in \{no, low, high\}$, E_i is the discretized economical value of the region i for the defender. We choose 3 values: *no* workers in the regions, *low*: a small amount of workers (less than half the total) and *high*: more than half the total of workers in this region i.
- $T_{1:n} \in discrete\ levels$, T_i is the tactical value of the region i for the defender. Basically, T is proportional to the proximity to the defender's army and the size[1] of the defender's army. In benchmarks, discretization steps are $0, 0.05, 0.1, 0.2, 0.4, 0.8$ (log_2 scale): basically from "no military influence" to "very close to most the defender's army".
- $TA_{1:n} \in discrete\ levels$, TA_i is the tactical value of the region i for the attacker (as above but for the attacker instead of the defender).
- $B_{1:n} \in \{true, false\}$, B_i tells if the region belongs (or not) to the defender. $\mathrm{P}(B_i = true) = 1$ if the defender has a base in region i and $\mathrm{P}(B_i = false) = 1$ if the attacker has one. Influence zones of the defender can be measured (with uncertainty) by $\mathrm{P}(B_i = true) \geq 0.5$ and vice versa. In fact, when uncertain, $\mathrm{P}(B_i = true)$ is proportional to the distance from i to the closest defender's base (and vice versa).
- $H_{1:n} \in \{ground, air, invisible, drop\}$, H_i: in predictive mode: how we will be attacked; in decision-making: how to attack, in region i.
- $GD_{1:n} \in \{no, low, med, high\}$: ground defense (relative to the attacker power) in region i, result from a heuristic: *no* defense if the defender's army is $\geq 1/10th$ of the attacker's, *low* defense above that and under half the attacker's army, *medium* defense above that and under comparable sizes, *high* if the defender's army is bigger than the attacker.
- $AD_{1:n} \in \{no, low, med, high\}$: same for air defense.
- $ID_{1:n} \in \{no\ detector, one\ detector, several\}$: invisible defense, equating to numbers of detectors.
- $TT \in [\emptyset, building_1, building_2, building_1 \wedge building_2, techtrees, \dots]$: all the possible technological trees for the given race. For instance $\{pylon, gate\}$ and $\{pylon, gate, core\}$ are two different Tech Trees, these come from the strategic level explained in the next section.
- $HP \in \{ground, ground \wedge air, ground \wedge invis, ground \wedge air \wedge invis, ground \wedge drop, ground \wedge air \wedge drop, ground \wedge invis \wedge drop, ground \wedge air \wedge invis \wedge drop\}$: **h**ow **p**ossible types of attacks, directly mapped from TT information. This variable serves the purpose of extracting all that we need to know from TT and thus reducing the complexity of a part of the model from n mappings from TT to H_i to one mapping from TT to HP and n mapping from HP to H_i. Without this variable, learning the co-occurrences of TT and H_i is sparse in the dataset. In prediction, with this variable, we make use of what we can infer on the opponent's strategy [24], [25], in decision-making, we know our own possibilities (we know our tech tree as well as the units we own).

We will not detail the full Bayesian model (see [22], pp.93-117 of [2]), but we will explain how we can very easily learn its main probability tables. For each battle in r we had one observation for: $\mathrm{P}(e_r, t_r, ta_r, b_r | A = true)$, and $\#regions - 1$ observations for the i regions which were not attacked: $\mathrm{P}(e_{i \neq r}, t_{i \neq r}, ta_{i \neq r}, b_{i \neq r} | A = false)$. For each battle of type t we had one observation for $\mathrm{P}(ad, gd, id | H = t)$ and $\mathrm{P}(H = t | HP = hp)$. By learning with a Laplace's law of succession [4], we allow for unseen event to have a non-zero probability. Note also that, due to the map-independence of our model, we can learn the parameters using different maps, and even do inference on maps which were never seen.

Following this, the probability to attack a region i (better even if it is recomputed for every game starting from a flat prior) is given by:

$$\mathrm{P}(A_i = true) = \frac{1 + n_{battles}(i)}{2 + \sum_{j \in regions} n_{battles}(i)}$$

The joint probability for a region to have a given economic value, tactical values, belonging to the defender and being

[1] "size" as in the sum of the values of units, with $v(unit) = minerals\_value + \frac{4}{3} gas\_value + 50 supply$, see pp.98-99 of [2] for details.

attacked is given by:

$$\mathrm{P}(E=e,T=t,TA=ta,B=b|A=True)=$$
$$\frac{1+n_{battles}(e,t,ta,b)}{|E|\times|T|\times|TA|\times|B|+\sum_{E,T,TA,B}n_{battles}(E,T,TA,B)}$$

While the joint probability of what are the aerial, ground defense, and detectors when an attack of type h happens is given by the co-occurrences:

$$\mathrm{P}(AD=ad,GD=gd,ID=id|H=h)=$$
$$\frac{1+n_{battles}(h,hp)}{|H|+\sum_{H}n_{battles}(H,hp)}$$

For a given region i, we can ask the probability to attack (or be attacked) here:

$$\mathrm{P}(A_i|e_i,t_i,ta_i,b_i)\propto\mathrm{P}(e_i,t_i,ta_i,b_i|a_i)\mathrm{P}(a_i)$$

And we can ask how we think that will happen:

$$\mathrm{P}(H_i|ad_i,gd_i,id_i)\propto$$
$$\sum_{TT,HP}\mathrm{P}(ad_i,gd_i,id_i|h_i)\mathrm{P}(h_i|HP)\mathrm{P}(HP|TT)\mathrm{P}(TT)$$

Where $\mathrm{P}(HP|TT)$ simply says if TT allows for $HP=h$ (0 or 1), and TT comes from the tech tree predictive model of the next section.

B. Results

We downloaded 7649 uncorrupted 1v1 replays from professional gamers leagues and international tournaments of StarCraft, from specialized websites. We then ran them using BWAPI[2] and dumped units' positions, pathfinding and regions, resources, orders, vision events, for attacks: types, positions, outcomes. This yield out more than $177,000$ battles. Basically, every BWAPI event, plus attacks, were recorded, the dataset and its source code are freely available[3]. More information about how this dataset was produced and what it contains can be found in [26].

An in-depth analysis of the results of the learning (showing that the model concur with human expertise) is provided in [2]. We show in Fig. 5 the probability of an attack happening in a region depending on the defender's tactical and economical value of this region. This concurs with game experience: the strategy is either to face the opponent's army (and crush it) or to undermine their economy and reinforcements.

To measure fairly the prediction performance of such a model, we applied "leave-100-out" cross-validation from our dataset: we set aside 100 games of each match-up for testing (with more than 1 battle per match) and train our model on the rest (see Table III). We look at the prediction 30 seconds before the attack, because that is the average time it would take to move a ground army from the middle of the map to anywhere (on a big map, cross the map on a small one). It also gives some time to build static defenses. We write match-ups XvY with X and Y the first letters of the factions involved

[2]http://code.google.com/p/bwapi/

[3]http://snippyhollow.github.com/bwrepdump/

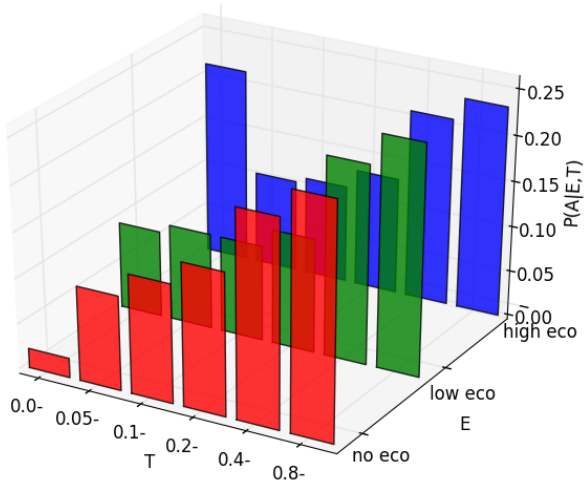

Fig. 5: $\mathrm{P}(A)$ for varying values of E and T, summed on the other variables, for Terran in TvT. Zones with no economy are in red bars, with a low economy in green and the principal economy in blue. The main difference along this economical axis comes at the lowest tactical values of regions (for the defenser) at $T<0.05$ (noted $T=0.0$) and showcases sneaky attacks to unprotected economical regions.

(Protoss, Terran, Zerg). Note that mirror match-ups (PvP, TvT, ZvZ) have fewer games but twice as many attacks from a given faction (it is twice the same faction).

Raw results of predictions of positions and types of attacks 30 seconds before they happen are presented in Table III: for instance the bold number (38.0) corresponds to the percentage of good positions (regions) predictions (30 sec before event) which were ranked 1st in the probabilities on $A_{1:n}$ for Protoss attacks against Terran (PvT).

- The measures on *where* corresponds to the percentage of good prediction and the mean probability for given ranks in $\mathrm{P}(A_{1:n})$ (to give a sense of the shape of the distribution): in average, the first prediction is attacked more that 1 out of 3 times. If we take the top 2, the prediction is correct more than half of the time.
- The measures on *how* corresponds to the percentage of good predictions for the most probable $\mathrm{P}(H_{attack})$ and the ratio of such attack types in the test set for given attack types. We particularly predict well ground attacks (trivial in the early game, less in the end game) and, interestingly, Terran and Zerg drop attacks (which are deadly). We think it is mainly due to the specific tech trees they require, and because they are quite frequent (so we have enough data to learn a robust model from).
- The *where & how* row corresponds to the percentage of good predictions for the maximal probability in the joint $\mathrm{P}(A_{1:n},H_{1:n})$: considering *only the most probable attack*, according to our model, we can predict *where* **and** *how* an attack will occur in the next 30 seconds $\approx$ 1/4th of the time.

The mean number of regions by map is 19, so a random *where* (attack destination) picking policy would have a correctness of 1/19 (5.23%), and even a random policy taking the high frequency of ground attacks into account would at

TABLE III: Results summary for multiple metrics at 30 seconds before attack, including the percentage of the time that it is rightly what happened (% column). Note that most of the time there is a very high temporal continuity between what can happen at time $t+30sec$ and at time $t+31sec$. For the *where* question, we show the four most probable predictions, the "Pr" column indicates the mean probability of the each bin of the distribution. For the *how* question, we show the four types of attacks (Ground, Air, Invisible, Drop), their percentages of correctness in predictions (%) and the ratio of a given attack type against the total numbers of attacks ($\frac{type}{total}$). The percentage of good predictions of *ground* type attacks in PvT is 98.1%, while ground type attacks, in this match-up, constitute 54% (ratio of 0.54) of all the attacks. The *where & how* line corresponds to the correct predictions of both *where* and *how* simultaneously (as most probables). NA (not available) is in cases for which we do not have enough observations to conclude sufficient statistics.

%: good predictions Pr=mean probability		Protoss						Terran						Zerg					
		P		T		Z		P		T		Z		P		T		Z	
total # games		445		2408		2027		2408		461		2107		2027		2107		199	
measure	rank	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr
where	1	40.9	.334	**38.0**	.329	34.5	.304	35.3	.299	34.4	.295	39.0	0.358	32.8	.31	39.8	.331	37.2	.324
	2	14.6	.157	16.3	.149	13.0	.152	14.3	.148	14.7	.147	17.8	.174	15.4	.166	16.6	.148	16.9	.157
	3	7.8	.089	8.9	.085	6.9	.092	9.8	.09	8.4	.087	10.0	.096	11.3	.099	7.6	.084	10.7	.100
	4	7.6	.062	6.7	.059	7.9	.064	8.6	.071	6.9	.063	7.0	.062	8.9	.07	7.7	.064	8.6	.07
measure	type	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$	%	$\frac{type}{total}$
how	G	97.5	0.61	98.1	0.54	98.4	0.58	100	0.85	99.9	0.66	76.7	0.32	86.6	0.40	99.8	0.84	67.2	0.34
	A	44.4	0.05	34.5	0.16	46.8	0.19	40	0.008	13.3	0.09	47.1	0.19	14.2	0.10	15.8	0.03	74.2	0.33
	I	22.7	0.14	49.6	0.13	12.9	0.13	NA	NA	NA	NA	36.8	0.15	32.6	0.15	NA	NA	NA	NA
	D	55.9	0.20	42.2	0.17	45.2	0.10	93.5	0.13	86	0.24	62.8	0.34	67.7	0.35	81.4	0.13	63.6	0.32
total		76.3	1.0	72.4	1.0	71.9	1.0	98.4	1.0	88.5	1.0	60.4	1.0	64.6	1.0	94.7	1.0	67.6	1.0
where & how (%)		32.8		23		23.8		27.1		23.6		30.2		23.3		30.9		26.4	

most be $\approx$ 1/(19*2) correct. For the location only (*where* question), we also counted the mean number of different regions which were attacked in a given game (between 3.97 and 4.86 for regions, depending on the match-up, and between 5.13 and 6.23 for choke-dependent regions). The ratio over these means would give the prediction rate we could expect from a *baseline heuristic* based solely on the location data: a heuristic which knows totally in which regions we can get attacked and then randomly select in them. These are attacks that actually happened, so the number of regions a player have to be worried about is at least this one (or more, for regions which were not attacked during a game but were potential targets). This *baseline heuristic* would yield (depending on the match-up) prediction rates between 20.5 and 25.2% for regions, versus our 32.8 to 40.9%.

To conclude about this tactical model, the results of the attack types (*how*) prediction are very good, in part because we make use of the technology prediction model (presented in the next section) with the distribution on TT. Even though it is better than a robust heuristic, the quality of the prediction of the position (*where*) of the attack can still be improved. In particular, we could track the opponent units (even under the fog-of-war) using e.g. particle filtering as in [21], or at the level of regions as pp.161-165 in [2].

V. Strategy

We now consider the part of strategy that infers the strategy of the opponent. There are similar works on StarCraft's strategy prediction. [27] presented "a data mining approach to strategy prediction" and performed supervised learning (from buildings features) on labeled StarCraft replays. We worked with the same dataset as they did, but we wanted to be able to deal with incomplete information (due to the fog-of-war), and to have building blocks (build trees / tech trees) for other models. [28] used an HMM which states are extracted from (unsupervised) maximum likelihood on the dataset. The HMM parameters are learned from unit counts (both buildings and military units) every 30 seconds and Viterbi inference is used to predict the most likely next states from partial observations. [29] studied the impact of a realistic fog-of-war [30] augmented the C4.5 decision tree and nearest neighbour with generalized exemplars (also used by [27]) with a Bayesian network on the buildings. Their results confirm ours: the predictive power is strictly better and the resistance to noise far greater than without encoding probabilistic estimations of the build tree.

A. Technology Tree Model

We start by predicting the technologies available to the opponent (the "tech tree"), to be able to feed this information to the tactical model, and to adapt our own strategy and technology. A major subpart of the tech tree is the build tree, and all the technologies or units are produced from buildings. The rules of the games are such that we cannot build some advanced buildings without the previous level/layer of technology/buildings. We will take advantage of that, showing how the strategic rules of the game should be used to build strategic abstractions.

The variables of this model are:

- BuildTree: $BT \in \{\emptyset, \{building_1\}, \{building_2\}, \{building_1 \wedge building_2\}, \dots\}$: all the possible building trees for the given race. For instance $\{pylon, gate\}$ and $\{pylon, gate, core\}$ are two different $BuildTrees$.
- Observations: $O_{i \in [\![1 \dots N]\!]} \in \{0,1\}$, O_k is $1/true$ if we have seen (observed) the kth building (it can have been destroyed, it will stay "seen"). Otherwise, it is $0/false$.
- $\lambda \in \{0,1\}$: coherence variable (restraining $BuildTree$ to possible values with regard to $O_{1:N}$)
- Time: $T \in [\![1 \dots P]\!]$, time in the game (1 second resolution).

Bayesian program
- Description
  - Specification (π)
    - $Variables$
    - $T, BT, O_1 \dots O_N, \lambda$
    - $Decomposition$
    - $\mathrm{P}(T, BT, O_1 \dots O_N, \lambda)$
    - $= \mathrm{P}(\lambda|BT, O_{[\![1\dots N]\!]})\mathrm{P}(T|BT)\prod_{i=1}^{N}\mathrm{P}(O_i)\mathrm{P}(BT)$
    - $Forms$
    - $\mathrm{P}(\lambda|BT, O_{[\![1\dots N]\!]}) = functional\ Dirac$ (coherence)
    - $\mathrm{P}(T|BT = bt) = discrete\ \mathcal{N}(\mu_{bt}, \sigma^2_{bt})$
  - $Identification$ (by maximum likelihood)
  - $(\mu_{bt}, \sigma_{bt}) = \arg\max_{\mu,\sigma} \mathrm{P}(T|BT = bt; \mu, \sigma^2)$
- $Question$
- $\mathrm{P}(BT|T = t, O_{1:N} = o_{1:N}, \lambda = 1) \propto \mathrm{P}(t|BT)\mathrm{P}(BT)$
- $\mathrm{P}(\lambda|BT, o_{1:N})\prod_{i=1}^{N}\mathrm{P}(o_i)$

Fig. 6: Bayesian program of the tech-tree prediction model.

Learning the model's parameters is just a matter of counting the co-occurrences of build-trees at their timings, to fit normal distribution, as shown in Fig. 6.

B. Results (build tree prediction)

All the results presented in this section represent the nine match-ups (races combinations) in 1 versus 1 (duel) of StarCraft. We worked with a dataset of 8806 replays ($\approx$ 1000 per match-up) of skilled human players, and we performed cross-validation with 9/10th of the dataset used for learning and the remaining 1/10th of the dataset used for evaluation.

The fully detailed analysis and results of this model can be found in pp.133-138 of [2] or in [25]. More generally, across all match-ups, without noise in the observations:

- the average distance from the most probable ("best") build tree to the real one is 0.535 building.
- the average distance from each value bt of the distribution on BT, weighted by its own probability ($\mathrm{P}(bt)$), to the real one is 0.87 building.

The **robustness to noise** is measured by the distance of the current estimation to the real build tree with increasing levels of noise (random dropping of observations), as shown at the top of Fig. 7. The **predictive power** of our model is measured by the number of next buildings for which we have "good enough" prediction of future build trees. "Good enough" being the maximum distance of the whole build tree that we can tolerate between our prediction for the future and what happens in practice, as shown at the bottom of Fig. 7.

Overall, this model has proven itself to be a solid building block for strategic reasoning, both in such specific studies of its performance, and in the case of our StarCraft bot.

C. Openings

We can quite simply add "openings" (early game strategy and tactics) to this tech tree model by adding an Op random variable with discrete values ($\approx$ 6 for per faction), and learn

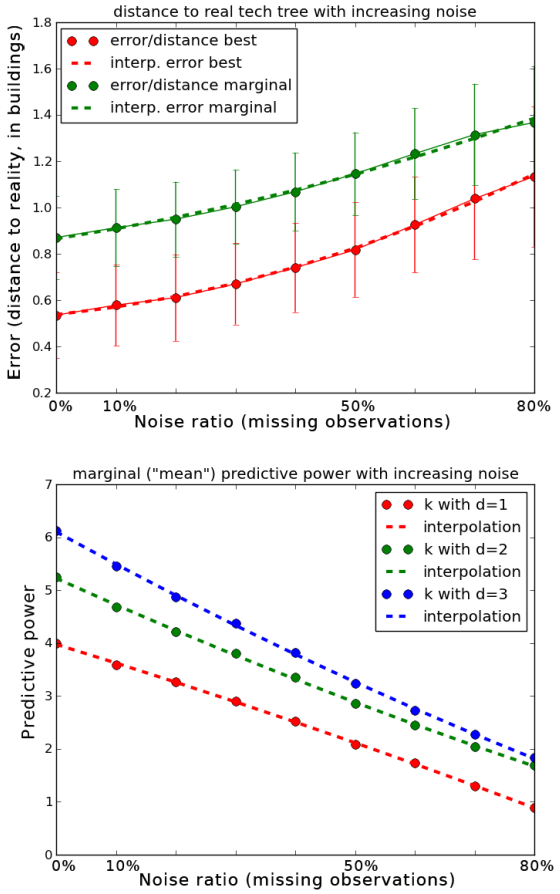

Fig. 7: Evolution of our metrics with increasing noise, from 0 to 80%. The top graphic shows the increase in distance between the predicted build tree, both most probable ("best") and marginal ("mean") and the actual one. The bottom graphic shows the decrease in predictive power: numbers of buildings ahead (k) for which our model predict a build tree closer than a fixed distance/error (d).

their co-occurrences with BT as such:

$$\mathrm{P}(BT = bt|Op^t = op) = \frac{1 + \mathrm{count}(bt, op)}{|BT| + \mathrm{count}(op)}$$

For example for Terran, a possible discretization of the openings could be (from [27]: "Bio" (aggressive rush), "TwoFactory" (strong push), "VultureHarass" (hit-and-run with invisible mines), "SiegeExpand" (economical advantage), "Standard" (versatile), "FastDropship" (tactical drop sneaky attack), and "Unknown" for all the edge cases. The full detail of this model (and all the values that Op can take) is given in [24] and pp.123-147 of [2].

D. Results (openings)

In Fig. 8, we show the evolution of the prediction of the opening during a TvP game (thus with a Terran opponent), with more and more buildings shown during the game.

This model gives very good results, over all match-ups (details can be found pp.145 of [2]), depending on the metric

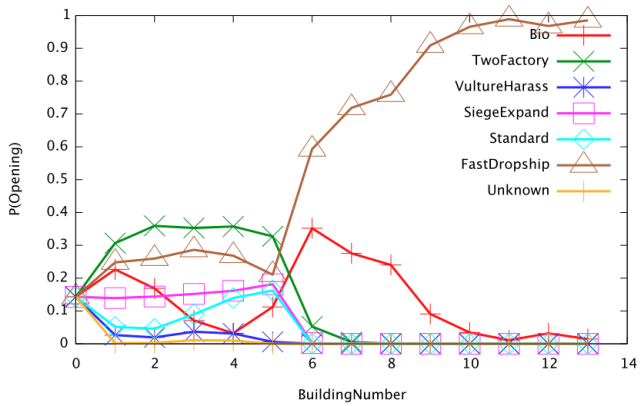

Fig. 8: Evolution of $\mathrm{P}(Opening)$ with increasing observations in a TvP match-up, with Weber's labeling. The x-axis corresponds to the construction of buildings.

that we use, the prediction probabilities are:

- at 5 minutes, 62-68% correct with Weber's labels, 63-67% with our labels.
- at 10 minutes, 73-78% correct with Weber's labels, 63-75% with our labels.
- at 15 minutes, 69-77% correct with Weber's labels, 63-75% with our labels.

We also proceeded to analyze the strengths and weaknesses of openings against each others. For that, we learned the $\mathrm{P}(Win = true|Op^t_{player1}, Op^t_{player2})$ probability table with Laplace's rule of succession. In practice, not all openings are used for one race in each of its 3 match-ups. Table IV shows some parts of this $\mathrm{P}(Win = true|Op^t_{player1}, Op^t_{player2})$ ratios of wins for openings against each others. This analysis can serve the purpose of choosing the right opening as soon as the opponent's opening was inferred.

Zerg — Protoss	TwoGates	FastDT	ReaverDrop	Corsair	DragoonsRanged
Speedlings	0.417	0.75	NED	NED	0.5
Lurkers	NED	0.493	NED	0.445	0.533
FastMutas	NED	0.506	0.5	0.526	0.532

Terran — Protoss	FastDT	ReaverDrop	Corsair	DragoonsRanged
TwoFactories	0.552	0.477	NED	0.578
RaxExpand	0.579	0.478	0.364	0.584

TABLE IV: Opening/Strategies labels counted for victories against each others for the PvZ (top, on 1408 games) and PvT (bottom, on 1657 games) match-ups. NED stands for Not Enough Data to conclude a preference/discrepancy towards one opening. The results should be read as win rates of columns openings vs lines openings, e.g. FastDT wins 75% vs Speedlings.

We now want to push further this idea of comparing the values of the distributions over strategic abstraction for each of the player further than openings. Few models have incorporated army compositions in their strategy abstractions, except sparsely as an aggregate or boolean existence of unit types. Most strategy abstractions are based on build trees (or tech trees), although a given set of buildings can produce different armies. What we will present here is complementary to these strategic abstractions and should help the military situation assessment.

E. Army Clustering Model

The idea behind armies clustering is to give one "composition" label for each army depending on its composing ratio of the different unit types. Giving a "hard" (unique) label for each army does not work well because armies contain different components of unit types combinations. For instance, a Protoss army can have only a "Zealots+Dragoons" component, but it will often just be one of the components (sometimes the backbone) of the army composition, augmented for instance with "High Templars+Archons".

Because a hard clustering is not an optimal solution, we used a Gaussian mixture model (GMM), which assumes that an army is a mixture (i.e. weighted sum) of several (Gaussian) components. The variables are:

- $C \in [\![c_1 \dots c_K]\!]$, our army clusters/components (C). There are K units clusters and K depends on the race (the mixture components are not the same for Protoss/Terran/Zerg).
- $U \in ([0,1] \dots [0,1])$ (length N), our N dimensional unit types (U) proportions, i.e. $U \in [0,1]^N$. N is dependent on the race and is the total number of unit types. For instance, an army with equal numbers of $Zealots$ and $Dragoons$ (and nothing else) is represented as $\{U_{Zealot} = 0.5, U_{Dragoon} = 0.5, \forall ut \neq Zealot|Dragoon\ U_{ut} = 0.0\}$, i.e. $U = (0.5, 0.5, 0, \dots, 0)$ if $Zealots$ and $Dragoons$ are the first two components of the U vector. So $\sum_i U_i = 1$ whatever the composition of the army.

For the M battles, the armies compositions are independent across battles, and the unit types proportions vector (army composition) is generated by a mixture of Gaussian components and thus U_i depends on C_i.

$$\mathrm{P}(U_{1\dots M}, C_{1\dots M}) = \prod_{i=1}^{M} \mathrm{P}(U_i|C_i)\mathrm{P}(C_i)$$

$$\mathrm{P}(U_i|C_i = c) = \mathcal{N}(\mu_c, \sigma_c^2)$$
$$\mathrm{P}(C_i) = Categorical(K, p_C)$$

We learned the Gaussian mixture models (GMM) parameters with the expectation-maximization (EM) algorithm on 5 to 15 mixtures with spherical, tied, diagonal and full co-variance matrices. We kept the best scoring models (by varying the number of mixtures) according to the Bayesian information criterion (BIC) [31].

For the ith battle (one army with units u), we can infer the distribution over the armies clusters with:

$$\mathrm{P}(C_i|U_i = u) = \mathrm{P}(C_i)\mathrm{P}(U_i = u|C_i)$$

In a battle, there are two armies (one for each players), we can thus apply this clustering to both the armies. If we have K clusters and N unit types, the opponent has K' clusters and N' unit types. We introduce EU and EC, respectively with the same semantics as U and C but for the enemy. In a given battle, we observe u and eu, respectively our army composition and the enemy's army composition. We can ask

$\mathrm{P}(C|U=u)$ and $\mathrm{P}(EC|EU=eu)$. As StarCraft unit types have strengths and weaknesses against other types, we can learn which clusters should beat other clusters (at equivalent investment) as a probability table. We use Laplace's law of succession ("add-one smoothing") by counting and weighting according to battles results ($c > ec$ means "c beats ec", i.e. we won against the enemy):

$$\mathrm{P}(C=c|EC=ec) = \frac{1+\mathrm{P}(c)\mathrm{P}(ec)\mathrm{count}_{\text{battles}}(c>ec)}{K+\mathrm{P}(ec)\mathrm{count}_{\text{battles with}}(ec)}$$

F. Results (army clustering)

We used the same dataset as for the tactical model to learn all the parameters and perform the benchmarks (by setting 100 test matches aside and learning on the remaining of the dataset). First, we analyze the posteriors of clustering only one army and then we evaluated the clustering as a mean to predict outcomes of battles.

1) Posterior analysis: Figure 9 shows a parallel plot of *army compositions*. We removed the less frequent unit types to keep only the 8 most important unit types of the PvP match-up, and we display a 8 dimensional representation of the army composition, each vertical axis represents one dimension. Each line (trajectory in this 8 dimensional space) represents an army composition (engaged in a battle) and gives the percentage of each of the unit types. These lines (armies) are colored with their most probable mixture component, which are shown in the rightmost axis. We have 8 clusters (Gaussian mixtures components): this is not related to the 8 unit types used as the number of mixtures was chosen by BIC score. Expert StarCraft players will directly recognize the clusters of typical armies, here are some of them:

- Light blue corresponds to the "Reaver Drop" tactical squads, which aims are to transport (with the flying Shuttle) the slow Reaver (zone damage artillery) inside the opponent's base to cause massive economical damages.
- Red corresponds to a typical army that is played in PvP (lots of Dragoons, supported by Reaver and Shuttle).
- Green corresponds to a High Templar and Archon-heavy army: the gas invested in such high tech units makes it that there are less Dragoons, and thus proportionally more Zealots (which cost no gas).
- Purple corresponds to Dark Templar ("sneaky", as Dark Templars are invisible) special tactics (and opening).

We also look at the clusters' dynamics during the games: Fig. 10 showcases the dynamics of clusters components: $\mathrm{P}(EC^t|EC^{t+1}$, for Zerg (vs Protoss) for Δt of 2 minutes. The diagonal components correspond to those which do not change between t and $t+1$ ($\Leftrightarrow t+2$minutes), and so it is normal that they are very high. The other components show the shift between clusters. For instance, the first line seventh column (in (0,6)) square shows a brutal transition from the first component (0) to the seventh (6). This is the switch in production to Mutalisks (mid-level advanced flying units) from a previously very low-tech army (Zerglings).

2) A soft rock-paper-scissors: We then used the learned $\mathrm{P}(C|EC)$ table to estimate the outcome of the battle. For that, we used battles with limited *disparities* (the maximum strength

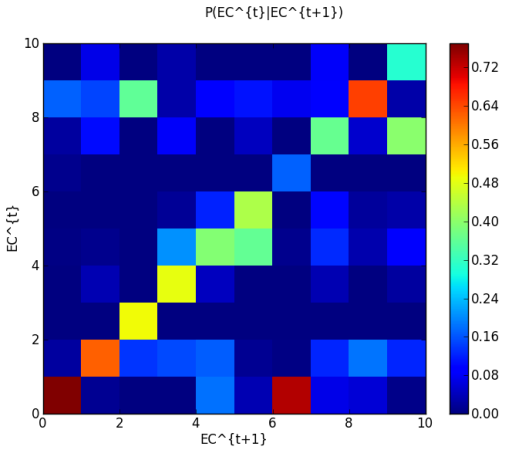

Fig. 10: Dynamics of clusters: $\mathrm{P}(EC^t|EC^{t+1})$ for Zerg, with $\Delta t = 2$ minutes

ratio of one army over the other) of 1.1 to 1.5. Note that the army which has the superior forces numbers has **more than a linear advantage** over their opponent (because of focus firing[4]), so a disparity of 1.5 is very high. For information, there is an average of 5 battles per game at a 1.3 disparity threshold, and the numbers of battles (used) per game increase with the disparity threshold.

We also made up a baseline heuristic, which uses the sum of the values of the units (as in the tactical model) to decide which side should win. If we note $v(unit)$ the value of a unit, the heuristic computes $\sum_{unit} v(unit)$ for each army and predicts that the winner is the one with the biggest score. Of course, we recall that a random predictor would predict the result of the battle correctly 50% of the time.

A summary of the main metrics is shown in Table V, the first line can be read as: for a forces disparity of 1.1, for Protoss vs Protoss (first column),

- considering only military units
  - the heuristic predicts the outcome of the battle correctly 63% of the time.
  - the probability of a clusters mixture to win against another ($\mathrm{P}(C|EC)$) without taking the forces sizes into account, predicts the outcome correctly 54% of the time.
  - the probability of a clusters mixture to win against another, taking also the forces sizes into account ($\mathrm{P}(C|EC) \times \sum_{unit} v(unit)$), predicts the outcome correctly 61% of the time.
- considering only all units involved in the battle (military units, plus static defenses and workers): same as above.

Results are given for all match-up (columns) and different forces disparities (lines). The last column sums up the means on all match-ups, with the whole army (military units plus static defenses and workers involved), for the three metrics.

We can see that predicting battle outcomes (even with a high disparity) with "just probabilities" of $\mathrm{P}(C|EC)$ (without taking the forces into account) gives relevant results as they are always above random predictions. Note that this is a very high

[4]Efficiently micro-managed, an army 1.5 times superior to their opponents can keep much more than one third of the units alive.

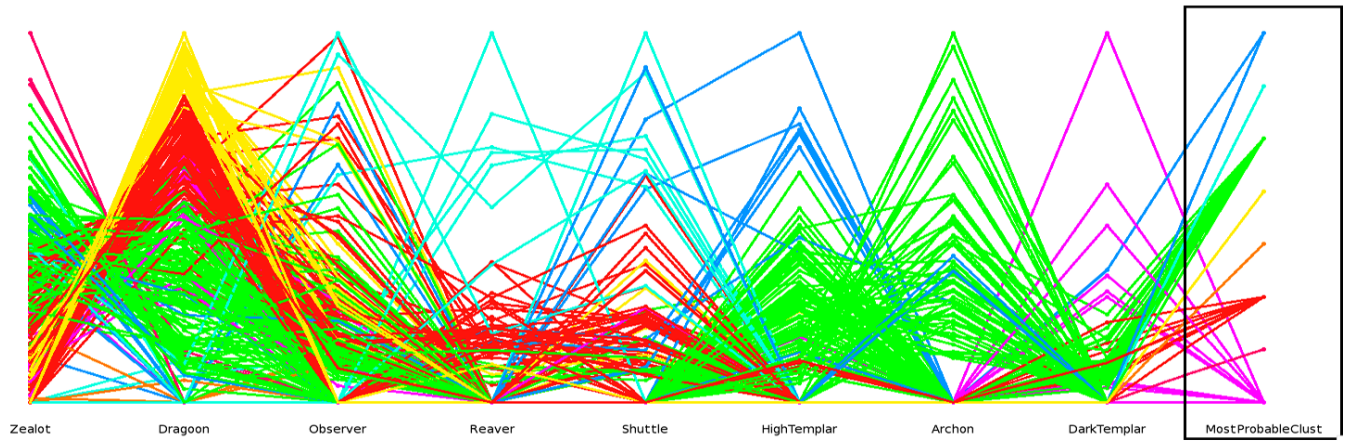

Fig. 9: Parallel plot of a small dataset of Protoss (vs Protoss, i.e. in the PvP match-up) army clusters on most important unit types (for the match-up). Each normalized vertical axis represents the percentage of the units of the given unit type in the army composition (we didn't remove outliers, so most top vertices (tip) represent 100%), except for the rightmost (framed) which links to the most probable GMM component. Note that several traces can (and do) go through the same edge.

forces disparity	scores in %	PvP m	PvP ws	PvT m	PvT ws	PvZ m	PvZ ws	...	mean ws
1.1	heuristic	**63**	63	58	58	58	58	...	61.7
	just prob.	54	58	68	*72*	60	61	...	63.2
	prob×heur.	61	**63**	**69**	**72**	**59**	**61**	...	67.0
1.3	heuristic	**73**	73	66	66	**69**	**69**	...	70.3
	just prob.	56	57	65	*66*	54	55	...	59.5
	prob×heuristic	72	**73**	**70**	**70**	66	66	...	71.0
1.5	heuristic	75	75	73	73	**75**	**75**	...	75.7
	just prob.	52	55	61	61	54	54	...	58.2
	prob×heur.	**75**	**76**	**74**	**75**	72	72	...	76.2

TABLE V: Winner prediction scores (in %) for the three main metrics. For the left columns ("m"), we considered only military units. For the right columns ("ws") we also considered static defense and workers. The "heuristic" metric is a baseline heuristic for battle winner prediction for comparison using army values, while "just prob." only considers $\mathrm{P}(C|EC)$ to predict the winner, and "prob×heuristic" weights the heuristic's predictions with $\sum_{C,EC} \mathrm{P}(C|EC)\mathrm{P}(EC)$.

level (abstract) view of a battle, we do not consider tactical positions, nor players' attention, actions, etc. Also, it is better (in average) to consider the heuristic with the composition of the army ("prob×heuristic") than to consider the heuristic alone, even for high forces disparity. Our heuristic augmented with the clustering seem to be the best indicator for battle situation assessment. These prediction results with "just prob.", or the fact that heuristic with $\mathrm{P}(C|EC)$ tops the heuristic alone, are a proof that the assimilation of armies compositions as Gaussian mixtures of cluster works.

Secondly, and perhaps more importantly, we can view the difference between "just prob." results and random guessing (50%) as the military **efficiency improvement** that we can (at least) expect from having the right army composition. Indeed, for small forces disparities (up to 1.1 for instance), the prediction based only on army composition ("just prob.": 63.2%) is better than the prediction with the baseline heuristic (61.7%). It means that *we can expect to win 63.2% of the time (instead of 50%) with an (almost) equal investment if we have the right composition*. Also, when we predict 58.5% of the time the accurate result of a battle with disparity up to 1.5 from "just prob.", this success in prediction is independent of the sizes of the armies. What we predicted is that the player with the better army composition won (and not necessarily the one with more or more expensive units).

More details can be found in pp.148-158 of [2]. In particular, there are explanations about how we can use these army clusters to drive our production for army clusters that fit best our strategy (our tech tree), the tactics we want to realize, while countering the opponent's army's composition.

VI. Discussion

A. About RTS AI

There are two hard problems when writing an AI of any kind: estimating the state we are in (perception), and taking decisions (action). For perception, most RTS AIs have to deal with uncertainty, coming either from partial information (fog-of-war), or stochasticity in the game rules (random action effects). Additionally, any kind of abstraction is going to have some incompleteness and thus introduce uncertainty. Probabilistic models deal directly with uncertainty, but their strength is in being able to easily build models that allow for sharing statistical power through hierarchy ("vertical continuity" in Figure 1) and sequentiality ("horizontal continuity" in Figure 1), by only dealing with probabilities distributions. Moreover, Bayesian models allow for taking decisions with access to the whole distribution instead of just point estimates, this is useful in estimating risks (dealing with the incompleteness of our own models). We can notice this in the micro-management model, where "sampling" is a better policy than "pick the best point estimate" whenever there more than half a dozen units.

There are different levels of abstraction used to reason about a game. Abstracting higher level cognitive functions (strategy and tactics for instance) is an efficient way to break the complexity barrier of writing game AI. Exploiting the vertical continuity, i.e. the conditioning of higher level decisions on

lower level decisions or actions, is possible in a hierarchical Bayesian model. For instance, that happens when we plug the distribution on the technology trees TT in the tactical model (through $\mathrm{P}(H|HP)$ and $\mathrm{P}(HP|TT)$), it conditions H only on values that are possible with the given TT values.

Real-time games may use discrete time-steps (24Hz for instance for StarCraft), but it does not prevent temporal continuity in strategies, tactics, and actions. Once a decision has been made at a given level, it may condition subsequent decisions at same level. With states S and observations O, filter models under the Markov assumption represent the joint $\mathrm{P}(S^0).\mathrm{P}(O^0|S^0).\prod_{t=1}^{T}[\mathrm{P}(S^t|S^{t-1}).\mathrm{P}(O^t|S^t)]$. Thus, from partial informations, one can use more than just the probability of observations knowing states to reconstruct the state, but also the probability of states transitions (the sequences). This way we can only consider transitions that are probable according to the current state. For instance, that happens when we infer $\mathrm{P}(Op^t|Op^{t-1})$.

Let us explain the limits of our models for "RTS games in general". First, we must note that simpler RTS games are subsumed by StarCraft, e.g. if a game does not have partial information, it only makes our models faster to compute.

- Our micro-management model (section III) is very general: in all RTS games there are units, damages, and very often terrain obstacles (except for RTS games happening in space). Moreover, it scales quite well (linearly) with the number of units, and allows for tuning it to specificities of one's game easily, by adding other sensory inputs and learning efficient policies.
- Our tactical model (section IV) is general in the model decomposition, but some of the abstractions (the H discrete variable) are specific to StarCraft tactics. These tactics may differ for other RTS games. Still, the whole approach and even model structure could still be used in any RTS game.
- Our tech tree prediction model (sections V.A/B), at the root of several of our models, are applicable to all games which have a technology tree, which is almost all RTS games. It will be more efficient (in predictive performance) the more convoluted a tech tree is (e.g. more efficient in the Age of Empire series than in the Total Annihilation series).
- Our openings prediction model (subsections V.C/D) and army composition model (subsections V.E/F) are specific to StarCraft, only in the sense that the values of the Op (openings) and of the C/EC variables are StarCraft-specific. Both these sets of values were clustered from replay data, so the exact same process and model could be applied to other RTS games.

B. Conclusion

There are mainly two points we want to make in this conclusion: good abstractions are crucial to RTS AI, and dealing with incomplete information is compulsory. Both of which are directly dealt with by Bayesian models.

As for abstractions, some of them are relatively easy to come up with and to rely on. For instance with the tech tree or the regions, because they are part (respectively) of the rules of the game or of the game design (ramps close regions). Other abstractions can come from players' expertise (tactical heuristics, openings), or from the data's statistical regularities (e.g. openings and armies clusters). In any case, their use is not limited to probabilistic models, and they can be building blocks of high level tree searchers. For instance, μRTS [32] can be seen as an abstraction over a richer RTS game. This gives an example about combinatorial multi-armed bandits variants of MCTS [32] can be applied over abstractions. For micro-management, tree search techniques, and in particular MCTS [14], will become better and better, but only by using better evaluation functions, and maybe learned policies with good situation assessment, that is, better "abstractions".

We would even drive the point of "incomplete information" further by noting that the players cannot mind-read each others. While that is fine in the case of Chess for instance, where all the strategy can be inferred (even if it's sometimes difficult) from the state of the game, such a problem is so much harder for RTS games. Consider a classic struggling case for all competitive StarCraft bots of a small squad of units running around the bot's base. No current bot is able to understand that this is not a committed attack, but a way to delay the bot's attack, while still scouting information. This kind of situation involves a tactical move that comes from the strategy ("delay the opponent while evolving tech or growing economy") and has a specific effect at the micro-management level (annoy, but do not fight).

In a given match, and/or against a given opponent, players tend to learn from their immediate mistakes, and they adapt their strategies to each other. This can be seen as a continuous learning problem. Human players call this the *meta-game*, as they enter the "I think that he thinks that I think…" game until arriving at fixed points. In this case, for all strategic models, a simple improvement would be to learn specific sets of parameters against the opponent's strategies (and consider the more "global" learning as a prior). For instance, a naive approach would be to learn a Laplace's law of succession directly on the enemy's tech tree: $P(ETechTrees = ett|Player = p) = \frac{1+nbgames(ett,p)}{\#ETT+nbgames(p)}$, to skew our inference towards what is specific to player p. The same approach can be done for their armies clusters.

We presented our approach for building Bayesian models at the levels of units control, tactics, and strategy. We showed how communication between the levels was done each time by passing the distribution of a random variable. Each of the models were evaluated separately. Several (micro-management, tactical prediction, strategy prediction) were successfully implemented in our StarCraft bot (free software[5]). While there are multiple possible improvements and further work possible, we think this showcases what it is possible to do with a probabilistic mode, to directly deal with all the sources of uncertainty and incompleteness inherent to RTS AI.

[5]https://github.com/SnippyHolloW/BroodwarBotQ

ACKNOWLEDGMENT

The work is funded in part by the European Research Council (ERC-2011-AdG-295810 BOOTPHON), the Agence Nationale pour la Recherche (ANR-10-LABX-0087 IEC, ANR-10-IDEX-0001-02 PSL*), the Fondation de France, the Ecole de Neurosciences de Paris, and the Region Ile de France (DIM cerveau et pensée).

REFERENCES

[1] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," *Computational Intelligence and AI in Games, IEEE Transactions on*, 2013.

[2] G. Synnaeve, "Bayesian Programming and Learning for Multi-Player Video Games: Application to RTS AI," PhD thesis, Université de Grenoble, Oct. 2012.

[3] G. Tesauro, "Machines that learn to play games," J. Fürnkranz and M. Kubat, Eds. Commack, NY, USA: Nova Science Publishers, Inc., 2001, ch. Comparison Training of Chess Evaluation Functions, pp. 117–130.

[4] E. T. Jaynes, *Probability Theory: The Logic of Science*. Cambridge University Press, June 2003.

[5] P. Bessière, E. Mazer, J. M. Ahuactzin, and K. Mekhnacha, *Bayesian Programming*. CRC Press, 2013.

[6] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer, "Bayesian robot programming," *Autonomous Robots*, vol. 16, no. 1, pp. 49–79, 2004.

[7] P. Bessière, C. Laugier, and R. Siegwart, *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[8] F. Colas, J. Diard, and P. Bessière, "Common bayesian models for common cognitive issues," *Acta Biotheoretica*, vol. 58, pp. 191–216, 2010.

[9] R. Le Hy, A. Arrigoni, P. Bessière, and O. Lebeltel, "Teaching Bayesian Behaviours to Video Game Characters," *Robotics and Autonomous Systems*, vol. 47, pp. 177–185, 2004.

[10] G. Synnaeve and P. Bessière, "Bayesian Modeling of a Human MMORPG Player," in *30th international workshop on Bayesian Inference and Maximum Entropy*, Chamonix, France, Jul. 2010.

[11] J. Hagelbäck, "Potential-field based navigation in starcraft," in *CIG (IEEE)*, 2012.

[12] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Stüer, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 82–98, June 2010.

[13] U. Jaidee and H. Munoz-Avila, "Modeling unit classes as agents in real-time strategy games," in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[14] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in starcraft," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[15] G. Synnaeve and P. Bessière, "A Bayesian Model for RTS Units Control applied to StarCraft," in *Proceedings of IEEE CIG*, Seoul, South Korea, Sep. 2011.

[16] B. Marthi, S. Russell, D. Latham, and C. Guestrin, "Concurrent hierarchical reinforcement learning," in *IJCAI*, 2005, pp. 779–785.

[17] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar," in *CIG (IEEE)*, 2012.

[18] A. Uriarte and S. Ontañón, "Kiting in rts games using influence maps," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[19] L. Perkins, "Terrain analysis in real-time strategy games: An integrated approach to choke point detection and region decomposition," in *AIIDE*, G. M. Youngblood and V. Bulitko, Eds. The AAAI Press, 2010.

[20] C. Bererton, "State estimation for game ai using particle filters." in *AAAI Workshop on Challenges in Game AI*, 2004.

[21] B. G. Weber, M. Mateas, and A. Jhala, "A particle model for state estimation in real-time strategy games," in *Proceedings of AIIDE*, AAAI Press. Stanford, Palo Alto, California: AAAI Press, 2011, p. 103–108.

[22] G. Synnaeve and P. Bessière, "Special Tactics: a Bayesian Approach to Tactical Decision-making," in *Proceedings of IEEE CIG*, Grenada, Spain, Sep. 2012.

[23] ——, "A Bayesian Tactician," in *Computer Games Workshop at ECAI*, Grenada, Spain, Aug. 2012.

[24] ——, "A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft," in *Proceedings of IEEE CIG*, Seoul, South Korea, Sep. 2011.

[25] G. Synnaeve and P. Bessière, "A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft," in *Proceedings of AIIDE*, AAAI, Ed., Palo Alto CA, USA, Oct. 2011, pp. 79–84, 7 pages.

[26] G. Synnaeve and P. Bessiere, "A Dataset for StarCraft AI & an Example of Armies Clustering," in *Artificial Intelligence in Adversarial Real-Time Games: Papers from the 2012 AIIDE Workshop AAAI Technical Report WS-12-15*, Palo Alto, États-Unis, Oct. 2012, pp. pp 25–30.

[27] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *CIG (IEEE)*, 2009.

[28] E. Dereszynski, J. Hostetler, A. Fern, T. D. T.-T. Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, AAAI, Ed., 2011.

[29] H.-C. Cho, K.-J. Kim, and S.-B. Cho, "Replay-based strategy prediction and build order adaptation for starcraft ai bots," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–7.

[30] B. Jónsson, "Representing uncertainty in rts games," Master's thesis, Reykjavík University, 2012.

[31] G. Schwarz, "Estimating the Dimension of a Model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[32] S. Ontanon, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.