A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft

Gabriel Synnaeve (gabriel.synnaeve@gmail.com) Pierre Bessière (pierre.bessiere@imag.fr)

***Abstract*—This paper presents a Bayesian model to predict the *opening* (first strategy) of opponents in real-time strategy (RTS) games. Our model is general enough to be applied to any RTS game with the canonical gameplay of gathering resources to extend a technology tree and produce military units and we applied it to StarCraft[1]. This model can also predict the possible technology trees of the opponent, but we will focus on *openings* here. The parameters of this model are learned from *replays* (game logs), labeled with openings. We present a semi-supervised method of labeling replays with the expectation-maximization algorithm and key features, then we use these labels to learn our parameters and benchmark our method with cross-validation. Uses of such a model range from a commentary assistant (for competitive games) to a core component of a dynamic RTS bot/AI, as it will be part of our StarCraft AI competition entry bot.**

I. Introduction

A. RTS gameplay and AI

We first introduce the basic components of a real-time strategy (RTS) game. In a RTS, players need to gather resources to build military units and defeat their opponents. To that end, they often have *worker units* (or extraction structures) than can gather resources needed to build *workers*, *buildings*, *military units* and *research upgrades*. Workers are often also builders (as in StarCraft) and are weak in fights compared to military units. Resources may have different uses, for instance in StarCraft: minerals are used for everything, whereas gas is only required for advanced buildings or military units, and technology upgrades. Buildings and research upgrades define technology trees (directed acyclic graphs) and each state of a "tech tree" allow for different unit type production abilities and unit spells/abilities. The military can be of different types, any combinations of ranged, casters, contact attack, zone attacks, big, small, slow, fast, invisible, flying... Units can have attacks and defenses that counter each others as in rock-paper-scissors.

Each unit and building has a *sight range* that provides the player with a view of the map. Parts of the map not in the sight range of the player's units are under *fog of war* and the player ignores what is and happens there. In RTS games jargon, an *opening* denotes the same thing than in Chess: an early game plan for which the player has to make choices. In Chess because one can not move many pieces at once (each turn), in RTS games because during the development phase, one is economically limited and has to choose between economic and military priorities and can only open so many tech paths at once. The *opening* corresponds to the first military (tactical) moves that will be performed and, in StarCraft, it corresponds to the 5 (early rushes) to 15 minutes (advanced technology / late push) timespan. Players have to find out what opening their opponents are doing to be able to effectively deal with the strategy (army composition) and tactics (military moves: where and when) thrown at them. For that, players scout each other and reason about the incomplete information they can bring together about army and buildings composition. This paper presents a probabilistic model able to predict the *opening* of the enemy that is used in a StarCraft AI competition entry bot (see Figure 1).

Most real-time strategy (RTS) games AI are either not challenging or not fun to play against. They are not challenging because they do not adapt well dynamically to different strategies (long term goals and army composition) and tactics (army moves) that a human can perform. They are not fun to play againt because they cheat economically, gathering resources faster, and/or in the intelligence war, bypassing the fog of war. We believe that creating AI that adapt to the strategies of the human player would make RTS games AI much more interesting to play against and increase greatly the "re-playability" of RTS games.

B. StarCraft

We worked on StarCraft: Brood War, which is a canonical RTS game, as Chess is to board games. It had been around since 1998, it has sold 10 millions licenses and was the best competitive RTS for more than a decade. There are numerous international competitions (World Cyber Games, Electronic Sports World Cup, BlizzCon, OSL, MSL). In *South Korea*, 4.5 millions of licenses have been sold and the average salary of a pro-gamer was up to 4 times the average salary. StarCraft helped define a particular genre of RTS gameplay, based as much on the strategy than the tactics. Nowadays, StarCraft 2 seems to overtake the original StarCraft, but the gameplay is exactly the same and many buildings and units are shared between the two games. There are 3 factions (Protoss, Terran and Zerg) that are totally different in terms of units, tech trees and thus gameplay styles.

StarCraft and most RTS provide a tool to record game logs into *replays* that can be re-simulated by the game engine and watched to improve strategies and tactics. All high level players use this feature heavily either to improve their play or study opponents style. Observing replays allows players to see what happened under the fog of war, so that they can understand timing of technologies and attacks and find clues/evidences leading to infer the strategy as well as

[1]StarCraft and its expansion StarCraft: Brood War are trademarks of Blizzard Entertainment™

978-1-4577-0011-8/11/$26.00 ©2011 IEEE

weak points (either strategic or tactical). We used this re
feature to extract players actions and learn the probabil
of tech trees to happen at a given time and, in this pa
also given a labeled opening.

C. Our Approach

The main idea of this paper comes from expert pla
StarCraft: human players can have a mental model of
probabilities of their opponents current openings and/or
trees. They try to update this mental model by scouting
opponent base and looking at the time at which oppon
build their (tech or producing) structures, number of u
at given times and so on. For instance in StarCraft,
players need to have buildings to gather resources (comm
center, refinery...), the time at which players build thei
a first indication. A player wanting to do advanced u
very fast will need gas and this resource type neec
refinery/extractor/assimilator to be gathered. The tim
which it is built is typical of tech opening versus econom
or rush openings.

We made the buildings part of tech trees the central pa
our model because buildings can be more easily scouted
units and our main focus was our bot implementation
Figure 1), but nothing hinders us to use units and upgrade
well in a setting without fog of war (commentary assis
game AI that cheat). There is not a direct mapping betw
the build time of structures and openings or strateg
different timings of buildings can lead to the same tech
but different openings or strategies, whereas the same tim
of buildings can later lead to different tech trees. Finally,
that if one does not want to predict specific openings bu
probabilities of tech trees, one does not need to have lab
game logs but only game logs.

In the next section, we discuss related works on stra
prediction and game logs exploitation. We also introc
the probabilistic framework used to describe our mode
section III, we describe our methodology to put openings
labels on replays and the Bayesian model for the recognition.
We then evaluate our recognition model, proving it leads to
significant information for the bot to adapt dynamically to
its opponent and that it is possible to perform the predictions
in real time.

II. Background

A. Related Works

This work was encouraged by the reading of Weber and
Mateas' Data Mining Approach to Strategy Prediction [1]
and the fact that they provided their dataset, that we used.
They tried and evaluated several machine learning algorithms
on replays that were labeled with strategies (openings) with
rules.

There are related works in the domains of opponent mod-
eling [2], [3], [4]. The main methods used to these ends are
case-based reasoning (CBR) and planning or plan recognition
[5], [6], [7], [8], [9]. There are precedent works of Bayesian
plan recognition [10], even in games with Albrecht *et al.*

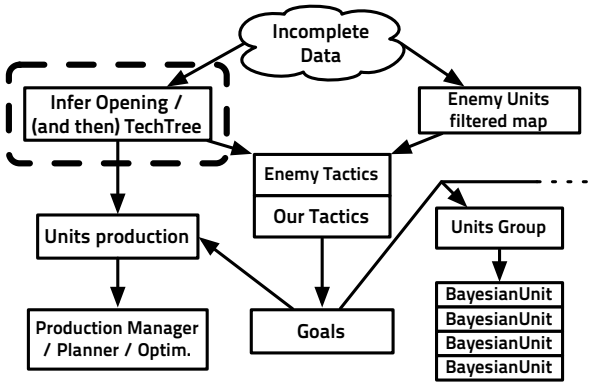

Fig. 1. Data flow of the full StarCraft robotic player BROODWARBOTQ. In this paper, we only deal with the upper left part (in a dotted line).

[11] using dynamic Bayesian networks to recognize a user's plan in a multi-player dungeon adventure.

Aha *et al.* [5] used CBR to perform dynamic plan retrieval extracted from domain knowledge in Wargus (Warcraft II clone). Ontañón *et al.* [6] base their real-time case-based planning (CBP) system on a plan dependency graph which is learned from human demonstration. In [7], [12], they use CBR and expert demonstrations on Wargus. They improve the speed of CPB by using a decision tree to select relevant features. Hsieh and Sun [2] based their work on Aha *et al.*'s CBR [5] and used StarCraft replays to construct states and building sequences. Strategies are choices of building construction order in their model.

Schadd *et al.* [3] describe opponent modeling through hierarchically structured models of the opponent behaviour and they applied their work to the Spring RTS (Total Annihilation clone). Hoang *et al.* [8] use hierarchical task networks (HTN) to model strategies in a first person shooter with the goal to use HTN planners. Kabanza *et al.* [4] improve the probabilistic hostile agent task tracker (PHATT [13], a simulated HMM for plan recognition) by encoding strategies as HTN.

The work described in this paper can be classified as probabilistic plan recognition. Strictly speaking, we present model-based machine learning used for prediction of plans, while our model is not limited to prediction. It performs two levels of plan recognition, both are learned from the replays: tech tree prediction (unsupervised) and opening prediction (semi-supervised or supervised depending on the labeling method).

B. Bayesian Programming

Probability is used as an alternative to classical logic and we transform incompleteness (in the experiences, the perceptions or the model) into uncertainty [14]. We introduce 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 [15]. 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 explaining the relevant *variables* $\{X^1, \ldots, X^n\}$ and explain their dependencies by *decomposing* the joint distribution $P(X^1 \ldots X^n|\delta, \pi)$ 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 $P(Searched|Known)$, with $Searched$ and $Known$ two disjoint subsets of the variables. $P(Searched|Known)$

$$= \frac{\sum_{Free} P(Searched,\ Free,\ Known)}{P(Known)}$$

$$= \frac{1}{Z} \times \sum_{Free} P(Searched,\ Free,\ Known)$$

General Bayesian inference is practically intractable, but conditional independence hypotheses and constraints (stated in the description) often simplify the model. Also, there are different well-known approximation techniques, for instance Monte Carlo methods [**?**] and variational Bayes [16]. In this paper, we will use only simple enough models that allow complete inference to be computed in real-time.

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

For the use of Bayesian programming in sensory-motor systems, see [17]. For its use in cognitive modeling, see [18]. For its first use in video games (first person shooter gameplay, Unreal Tournament), see [19].

III. Methodology

A. Replays Labeling

We used Weber and Mateas [1] dataset of labeled replays. It is composed of 9316 StarCraft: Broodwar game logs, between $\approx$ 500 and 1300 per *match-up*. A match-up is a set of the two opponents races, Protoss versus Terran (PvT) is a match-up, PvZ is another one. They are distinguished because strategies distribution are very different across match-ups (see Table II). Weber and Mateas used logic rules on building sequences to put their labels, concerning only tier 2 strategies (no tier 1 rushes).

Openings are closely related to *build orders* (BO) but different BO can lead to the same opening and some BO are shared by different openings. Particularly, if we do not take the time at which the buildings are constructed, we may be wrong too often. For that reason, we tried to label replays with the statistical appearance of key features with a semi-supervised method (see Figure 2). Indeed, the purpose of our opening prediction model is to help our StarCraft playing bot to deal with rushes and special tactics. This was not the main focus of Weber and Mateas' labels, which follow more the development of the tech tree. We used the key components of openings that we want to be aware of as features for our labeling algorithm as show in Table I.

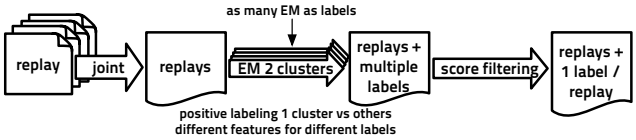

Fig. 2. Data centric view of our semi-supervised labeling of replays

The selection of the features along with the opening labels is the supervised part of our labeling method. The knowledge of the features and openings comes from expert play and the StarCraft liquipedia[2]. They are all presented in Table I. For instance, if we want to find out which replays correspond to the "fast Dark Templar" (DT, Protoss invisible unit) opening, we put the time at which the first Dark Templar is constructed as a feature and perform clustering on replays with it. This is what is needed for our playing bot: to be able to know when he has to fear "fast DT" opening and build a detector unit quickly to be able to deal with invisibility.

For the clustering part, we tried k-means, expectation-maximization (EM) with equal shape (bivariate normal distribution with proportional covariances matrices) and EM with the normal distribution shapes and volumes chosen with a Bayesian information criterion (BIC). Best BIC models were almost always the most agreeing with expert knowledge (15/17 labels). We used the R package Mclust [20], [21] to perform full EM clustering. We produce "2 bins clustering" for each set of features (corresponding to each opening), and label the replays belonging to the cluster with the lower norm of features' appearances (that is exactly the purpose of our features). Figures 4 and 5 show the clusters out of EM with the features of the corresponding openings. We thought of clustering because there are two cases in which you build a specific military unit of research a specific upgrade: either it is part of your opening, or it is part of your longer term game plan or even in reaction to the opponent. So the distribution over the time at which a feature appears is bimodal, with one (sharp) mode corresponding to "opening with it" and the other for the rest of the games, as can be seen in Figure 3.

TABLE II
Openings distributions for Terran in all the match-ups

	vs Protoss		vs Terran		vs Zerg	
Opening	Nb	Percentage	Nb	Percentage	Nb	Percentage
bio	62	6.2	25	4.4	197	22.6
fast_exp	438	43.5	377	65.4	392	44.9
two_facto	240	23.8	127	22.0	116	13.3
vultures	122	12.1	3	0.6	3	0.3
drop	52	5.2	10	1.7	121	13.9
unknown	93	9.3	34	5.9	43	5.0

[2]http://wiki.teamliquid.net/starcraft/

TABLE I
OPENING/STRATEGIES LABELS OF THE REPLAYS (WEBER'S AND OURS ARE NOT ALWAYS CORRESPONDING)

Race	Weber and Mateas' labels	Our labels	Features	Note (what we fear)
Protoss	FastLegs	speedzeal	Legs, GroundWeapons+1	quick speed+upgrade attack
	FastDT	fast_dt	DarkTemplar	invisible units
	FastObs	nony	Goon, Range	quick long ranged attack
	ReaverDrop	reaver_drop	Reaver, Shuttle	tactical attack zone damages
	Carrier	corsair	Corsair	flying units
	FastExpand	templar	Storm, Templar	powerful zone attack
		two_gates	SecondGateway, Gateway, Zealot	aggressive rush
	Unknown	unknown	(no label or >2 labels with $\approx$ probabilities)	
Terran	Bio	bio	ThirdBarracks, SecondBarracks, Barracks	aggressive rush
	TwoFactory	two_facto	SecondFactory	strong push (long range)
	VultureHarass	vultures	Mines, Vulture	aggressive harass, invisible
	SiegeExpand	fast_exp	Expansion, Barracks	economical advantage
	Standard			
	FastDropship	drop	DropShip	tactical attack
	Unknown	unknown	(no label or >2 labels with $\approx$ probabilities)	
Zerg	TwoHatchMuta	fast_mutas	Mutalisk, Gas	early air raid
	ThreeHatchMuta	mutas	ThirdHatch, Mutalisk	massive air raid
	HydraRush	hydras	Hydra, HydraSpeed, HydraRange	quick ranged attack
	Standard	(speedlings)	(ZerglingSpeed, Zergling)	(removed, quick attacks/mobility)
	HydraMass			
	Lurker	lurkers	Lurker	invisible and zone damages
	Unknown	unknown	(no label or >2 labels with $\approx$ probabilities)	

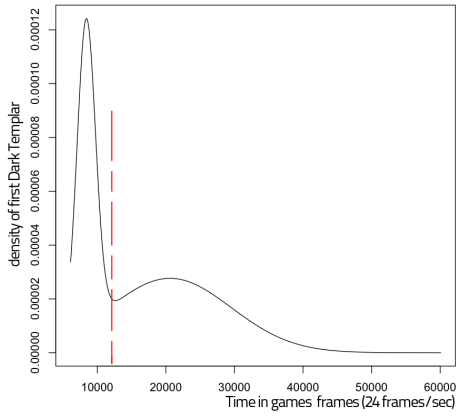

Fig. 3. Protoss vs Terran distribution of first appearance of Dark Templars (Protoss invisible unit).

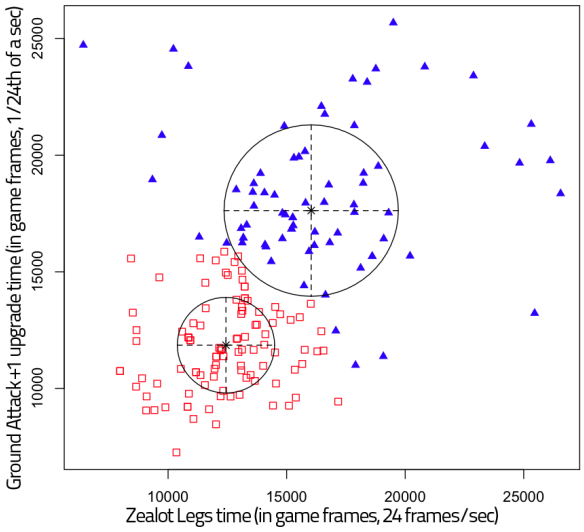

Fig. 4. Protoss vs Protoss Ground Attack +1 and Zealot Legs upgrades timings. The bottom left cluster (squares) is the one labeled as *speedzeal*.

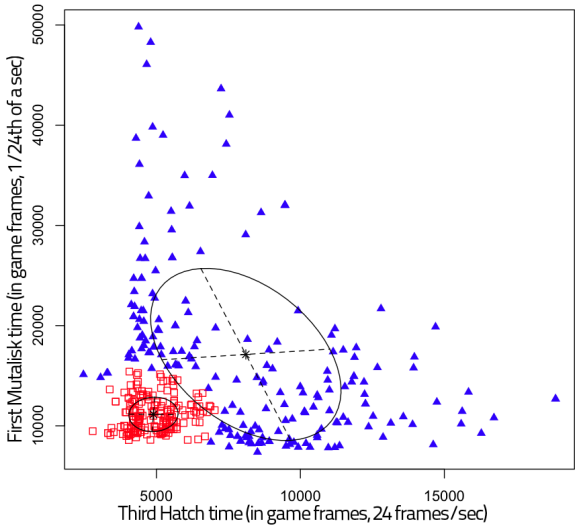

Fig. 5. Zerg vs Protoss time of the third Hatch and first appearance of Mutalisks. The bottom left cluster (squares) is the one labeled as *mutas*.

Finally, some replays are labeled two or three times with different labels (due to the different time of effect of different openings), so we apply a filtering to transform multiple label replays into unique label ones (see Figure 2). For that we choose the openings labels that were happening the earliest (as they are a closer threat to the bot in a game setup) if and only if they were also the most probable or at 10% of probability of the most probable label (to exclude transition boundaries of clusters) for this replay. We find the earliest by comparing the norms of the clusters means in competition. All replays without a label or with multiple labels (*i.e.* which did not had a unique solution in filtering) after the filtering were labeled as *unknown*. We then used this labeled dataset as well as Weber and Mateas' labels in the testing of our Bayesian model for opening prediction.

B. Opening Prediction Model

Our predictive model is a Bayesian program, it can be seen as the "Bayesian network" represented in Figure 6. It is a generative model and this is of great help to deal with the parts of the observations' space where we do not have too much data (RTS games tend to diverge from one another as the number of possible actions grow exponentially). Indeed, we can model our uncertainty by putting a large standard deviation on too rare observations and generative models tend to converge with fewer observations than discriminative ones [22]. Here is the description of our Bayesian program:

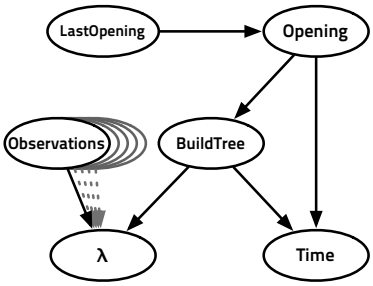

Fig. 6. Graph representation of the opening (and tech tree) prediction model

1) Variables:

- $BuildTree \in [\emptyset, building_1, building_2, building_1 \wedge building_2, techtrees, \dots]$: all the possible building trees for the given race. For instance $\{pylon, gate\}$ and $\{pylon, gate, core\}$ are two different $BuildTrees$.
- N $Observations$: $O_{i \in [\![1 \dots N]\!]} \in \{0, 1\}$, O_k is 1 $(true)$ if we have seen (observed) the kth building (it can have been destroyed, it will stay "seen").
- $Opening$: $Op^t \in [opening_1 \dots opening_M]$ take the various opening values (depending on the race).
- $LastOpening$: $Op^{t-1} \in [opening_1 \dots opening_M]$, Opening value of the previous time step (allows filtering, taking previous inference into account).
- $\lambda \in \{0, 1\}$: coherence variable (restraining $BuildTree$ to possible values with regard to $O_{[\![}1 \dots N]\!]$)
- $Time$: $T \in [\![1 \dots P]\!]$, time in the game (1 second resolution).

At first, we generated all the possible (according to the game rules) $BuildTree$ values (between ≈ 500 and 1600 depending on the race). We observed that a lot of possible $BuildTree$ values are too absurd to be performed in a competitive match and were never seen during the learning. So, we restricted $BuildTree$ to have its value in all the build trees encountered in our replays dataset. There are 810 build trees for Terran, 346 for Protoss and 261 for Zerg (≈ 3000 replays/race), all learned from the (unlabeled) replays.

2) Decomposition: The joint distribution of our model is the following:

$$\begin{aligned} & P(T, BuildTree, O_1 \dots O_N, Op^t, Op^{t-1}, \lambda) \\ = \; & P(Op^t | Op^{t-1}) \\ & P(Op^{t-1}) \\ & P(BuildTree | Op^t) \\ & P(O_{[\![1 \dots N]\!]}) \\ & P(\lambda | BuildTree, O_{[\![1 \dots N]\!]}) \\ & P(T | BuildTree, Op^t) \end{aligned}$$

This can also be see as Figure 6.

3) Forms:

- $P(Op^t | Op^{t-1})$ is optional, we use it as a filter so that the previous inference impacts the current one. We use a functional Dirac:

$$\begin{aligned} & P(Op^t | Op^{t-1}) \quad (Dirac) \\ = \; & 1 \text{ if } Op^t = Op^{t-1} \\ = \; & 0 \text{ else} \end{aligned}$$

  This does not prevent our model to switch predictions, it just uses previous inference posterior $P(Op^{t-1}$ to average $P(Op^t)$.
- $P(Op^{t-1})$ copied from one inference to another (mutated from $P(Op^t)$). The first $P(Op^{t-1})$ is bootstrapped with the uniform distribution, we could also use a prior on openings in the given match-up.
- $P(BuildTree | Op^t)$ is learned from the labeled replays. $P(BuildTree | Op^t)$ are $\text{card}(\{openings\})$ different histogram over the values of $BuildTree$.
- $P(O_{[\![1 \dots N]\!]})$ is unspecified, we put the uniform distribution (we could use a prior over the most frequent observations).
- $P(\lambda | BuildTree, O_{[\![1 \dots N]\!]})$ is a functional Dirac that restricts $BuildTree$ values to the ones than can co-exist with the observations.

$$\begin{aligned} & P(\lambda = 1 | buildTree, o_{[\![1 \dots N]\!]}) \\ = \; & 1 \text{ if } buildTree \text{ can exist with } o_{[\![1 \dots N]\!]} \\ = \; & 0 \text{ else} \end{aligned}$$

  A $BuildTree$ value ($buildTree$) is compatible with the observations if it covers them fully. For instance, $BuildTree = \{pylon, gate, core\}$ is compatible with $o_{\#core} = 1$ but it is not compatible with $o_{\#forge} = 1$. In other words, $buildTree$ is incompatible with $o_{[\![1 \dots N]\!]}$ *iff* $\{o_{[\![1 \dots N]\!]} \backslash \{o_{[\![1 \dots N]\!]} \wedge buildTree\}\} \neq \emptyset$.
- $P(T | BuildTree, Op^t)$ are "bell shape" distributions (discretized normal distributions). There is one bell shape per couple $(opening, buildTree)$. The parameters of these discrete Gaussian distributions are learned from the labeled replays.

4) Identification (learning): The learning of the $P(BuildTree | Op^t)$ histogram is straight forward counting of occurrences from the labeled replays. The learning of

the $P(T|BuildTree, Op^t)$ bell shapes parameters takes into account the uncertainty of the couples $(buildTree, opening)$ for which we have few observations. Indeed, the normal distribution $P(T|buildTree, opening)$ begins with a high σ^2, and **not** a Dirac with μ on the seen T value and $sigma = 0$. This accounts for the fact that the first observation may have been an outlier. This learning process is independent on the order of the stream of examples, seeing point A and then B or B and then A in the learning phase produces the same result.

5) Questions: The question that we will ask in all the benchmarks is:

$$\begin{aligned} &P(Op|T = t, O_{[\![1...N]\!]} = o_{[\![1...N]\!]}, \lambda = 1) \\ &\propto P(Op).P(o_{[\![1...N]\!]}) \\ &\times \textstyle\sum_{BuildTree} P(\lambda|BuildTree, o_{[\![1...N]\!]}) \\ &.P(BuildTree|Op).P(t|BuildTree, Op) \end{aligned}$$

Note that if we see $P(BuildTree, Time)$ as a plan, asking $P(BuildTree|Opening, Time)$ boils down to use our "plan recognition" mode as a planning algorithm, which could provide good approximations of the optimal goal set [9]. This gives us a distribution on the build trees to follow (build orders) to achieve a given opening.

IV. Results

A. Prediction

For each match-up, we ran cross-validation testing with 9/10th of the dataset used for learning and the remaining 1/10th of the dataset used for testing. We ran tests finishing at 5, 10 and 15 minutes to capture all kinds of openings (early to late ones). To measure the predictive capability of our model, we used 3 metrics:

- the *final* prediction, which is the opening that is predicted at the end of the test,
- the *online twice* (OT), which counts the openings that have emerged as most probable twice a test (so that their predominance is not due to noise),
- the *online once* > 3 (OO3), which counts the openings that have emerged as most probable openings after 3 minutes (so that these predictions are based on really meaningful information).

After 3 minutes, a Terran player will have or be building his first supply depot, barracks, refinery (gas), and at least factory or expansion. A Zerg player would have his first overlord, zergling pool, extractor (gas) and most of the time his expansion and lair tech. A Protoss player would have his first pylon, gateway, assimilator (gas), cybernectics core, and sometimes his robotics center, or forge and expansion.

Table III sums up all the prediction probabilities (scores) of our model in all the match-ups with both labeling of the game logs. Please note that when an opening is mispredicted, the distribution on openings is often not $P(badopening) = 1, P(others) = 0$ and that we can extract some value out of these distributions. Also, we observed that $P(Opening = unknown) > P(others)$ is often a case of misprediction:

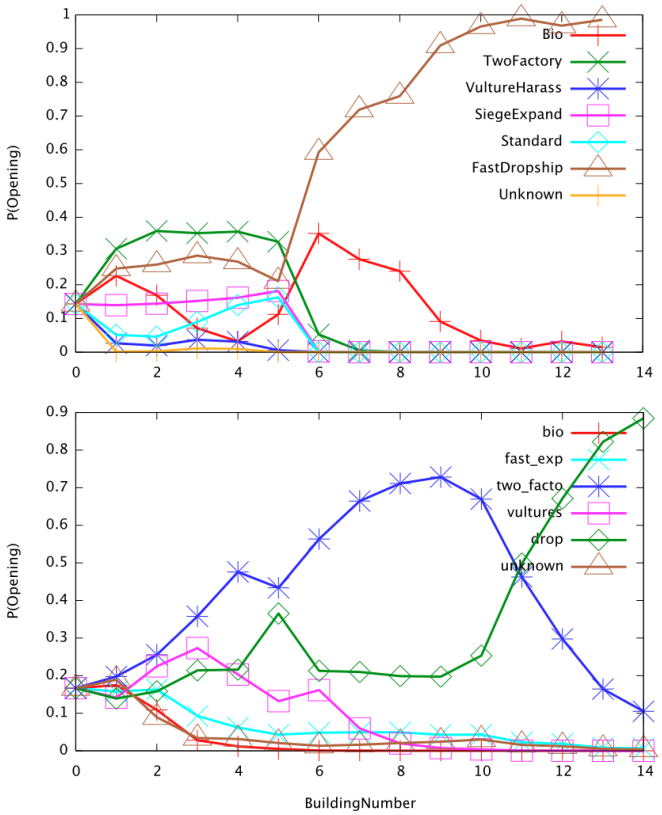

Fig. 7. Evolution of $P(Opening)$ with increasing observations in a TvP match-up, with Weber's labeling on top, our labeling on the bottom. The x-axis corresponds to the construction of buildings.

our bot would use the next prediction in this case. Figure 7 shows the evolution of the distribution $P(Opening)$ during a replay for Weber's and our labelings. Figure 8 shows the resistance of our model to noise. We randomly removed some observations (buildings, attributes), from 1 to 15, knowing that for Protoss and Terran we use 16 buildings observations and 17 for Zerg. We think that our model copes well with noise because it backtracks unseen observations: for instance if we have only the $core$ observation, it will work with build trees containing $core$ that will passively infer unseen $pylon$ and $gate$. Also, uncertainty is handled natively.

B. Performances

The first iteration of this model was not making use of the structure imposed by the game in the form of "possible build trees" and was at best very slow, at worst intractable without sampling. With the model presented here, the performances are ready for production as shown in Table IV. The memory footprint is around 3.5Mb on a 64bits machine. Learning computation time is linear in the number of games logs events ($O(N)$ with N observations), which are bounded, so it is linear in the number of game logs. It can be serialized and done only once when the dataset changes. The prediction computation corresponds to the sum in the question (III.B.5) and so its computational complexity is in $O(N \cdot M)$ with N build trees and M possible observations, as $M << N$, we can consider it linear in the number of build trees (values of $BuildTree$).

TABLE III
PREDICTION PROBABILITIES FOR ALL THE MATCH-UPS

	Weber and Mateas' labels									Our labels								
	5 minutes			10 minutes			15 minutes			5 minutes			10 minutes			15 minutes		
match-up	final	OT	OO3	final	OT	OO3	final	OT	OO3	final	OT	OO3	final	OT	OO3	final	OT	OO3
PvP	0.65	0.53	0.59	0.69	0.69	0.71	0.65	0.67	0.73	0.78	0.74	0.68	0.83	0.83	0.83	0.85	0.83	0.83
PvT	0.75	0.64	0.71	0.78	0.86	0.83	0.81	0.88	0.84	0.62	0.69	0.69	0.62	0.73	0.72	0.6	0.79	0.76
PvZ	0.73	0.71	0.66	0.8	0.86	0.8	0.82	0.87	0.8	0.61	0.6	0.62	0.66	0.66	0.69	0.61	0.62	0.62
TvP	0.69	0.63	0.76	0.6	0.75	0.77	0.55	0.73	0.75	0.50	0.47	0.54	0.5	0.6	0.69	0.42	0.62	0.65
TvT	0.57	0.55	0.65	0.5	0.55	0.62	0.4	0.52	0.58	0.72	0.75	0.77	0.68	0.89	0.84	0.7	0.88	0.8
TvZ	0.84	0.82	0.81	0.88	0.91	0.93	0.89	0.91	0.93	0.71	0.78	0.77	0.72	0.88	0.86	0.68	0.82	0.81
ZvP	0.63	0.59	0.64	0.87	0.82	0.89	0.85	0.83	0.87	0.39	0.56	0.52	0.35	0.6	0.57	0.41	0.61	0.62
ZvT	0.59	0.51	0.59	0.68	0.69	0.72	0.57	0.68	0.7	0.54	0.63	0.61	0.52	0.67	0.62	0.55	0.73	0.66
ZvZ	0.69	0.64	0.67	0.73	0.74	0.77	0.7	0.73	0.73	0.83	0.85	0.85	0.81	0.89	0.94	0.81	0.88	0.94
overall	0.68	0.62	0.68	0.73	0.76	0.78	0.69	0.76	0.77	0.63	0.67	0.67	0.63	0.75	0.75	0.63	0.75	0.74

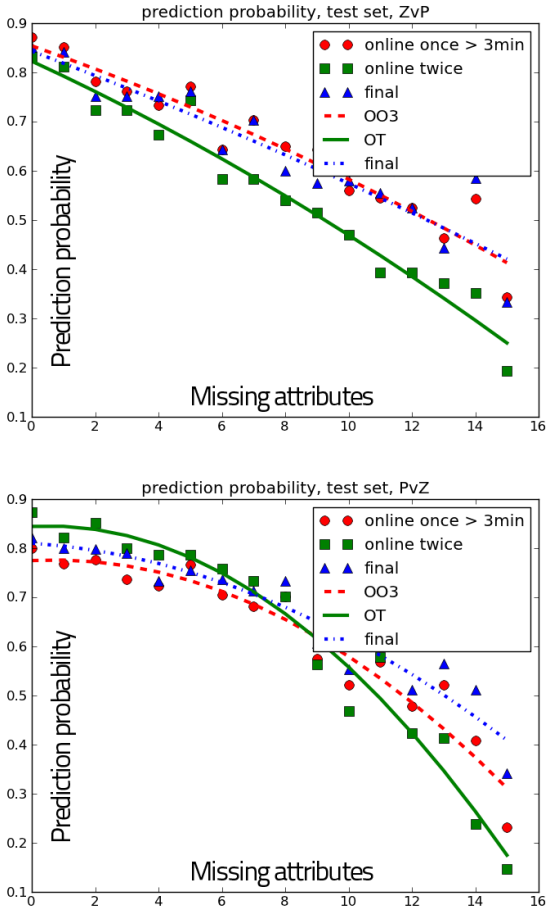

Fig. 8. Two extreme evolutions of the 3 probabilities of opening recognition with increasing noise (15 missing attributes/observations/buildings correspond to 93.75% missing information for Protoss and Terran openings prediction and 88.23% of missing attributes for Zerg openings prediction). Zerg opening prediction probabilitly on top, Protoss bottom.

TABLE IV
EXTREMES OF COMPUTATION TIME VALUES (IN SECONDS, C2D 2.8GHZ)

Race	Nb Games	Learning time	Inference μ	Inference σ^2
T (max)	1036	0.197844	0.0360234	0.00892601
T (Terran)	567	0.110019	0.030129	0.00738386
P (Protoss)	1021	0.13513	0.0164457	0.00370478
P (Protoss)	542	0.056275	0.00940027	0.00188217
Z (Zerg)	1028	0.143851	0.0150968	0.00334057
Z (Zerg)	896	0.089014	0.00796715	0.00123551

V. CONCLUSIONS

A. Possible Uses

Developing beforehand a RTS game AI that specifically deals with whatever strategies the players will come up is very hard. And even if game developers were willing to patch their AI afterwards, it would require a really modular design and a lot of work to treat each strategy. With our model, the AI can adapt to the evolutions in play by learning its parameters from the replay, and it can dynamically adapt during the games by using the reverse question $P(BuildTree|Opening, Time)$, or even $P(TechTree|Opening, Time)$ if we use a $TechTree$ variable encoding buildings and technology upgrades. This question would give the distribution over technology trees knowing the opening we want to perform at which time. This would allow for the bot to dynamically choose/change build orders.

We will also investigate the use of our model in a commentary assistant AI. In the StarCraft and StarCraft 2 communities, there are a lot of progamers tournaments that are commented and we could provide a tool for commentators to estimate the probabilities or different openings or technology paths. As in commented poker matches, where the probabilities of different hands are drawn on screen for the spectators, we could display the probabilities of openings. In such a setup we could use more features as the observers and commentators can see everything that happens (upgrades, units) and we limited ourselves to "key" buildings in the work presented in this paper.

B. Improvements

First, our prediction model can be upgraded to have a higher recognition rate: we could reason about $t+1$ explicitly before computing the distribution over possible openings at t and thus compute the distribution over technology trees at $t+1$. Perhaps it would increase the results of $P(Opening|Observations)$, but it almost surely would increase $P(BuildTree^{t+1}|Observations)$ which is important for late game predictions. We could also make use of more features as we currently only use at most 20 features (only buildings), and never all at once. Perhaps also that incorporating priors per match-up would lead to better results.

Then, we could feed it with *more* replays during the learning by scrapping more progamers level replays websites. Also, we could learn from replays of bot vs bot matches. For the learning part, the labeling of replays is very important, and our labeling methods can be improved. We could explore auto-supervised learning [23]. Clearly, some match-ups are handled better, either in the replays labeling part and/or in the prediction part. Replays could be labeled by humans and we would do supervised learning then. Or they could be labeled by a combination of rules (as in [1]) and statistical analysis (as the method presented here). Finally, the replays could be labeled by match-up dependent openings (instead of race dependent openings currently) and could contain either the two parts of the opening or the game time at which the label is the most relevant, as openings are often bimodal ("fast expand into mutas", "corsairs into reaver", etc.).

Finally, a hard problem is detecting the "fake" builds of very highly skilled players. Indeed, some progamers have build orders which purpose are to fool the opponent into thinking that they are performing opening A while they are doing B. For instance, they could leadthe opponent to think they are going to *tech* and perform an early rush instead. We think that this can be handled by our model by changing $P(Opening|LastOpening)$ by $P(Opening|LastOpening, LastObservations)$ and adapting the influence of the last prediction with regard to the last observations (i.e., we think we can learn some "fake" label on replays).

C. Conclusion

We contributed a probabilistic model to be able to compute the distribution over openings (strategies) of the opponent in a RTS game from partial and noisy observations. The bot can adapt to the opponent's strategy as it predicts the opening with $63-68\%$ of recognition rate at 5 minutes and $>70\%$ of recognition rate at 10 minutes (up to 94%), while having strong robustness to noise ($>50\%$ recognition rate with 50% missing observations). It can be used in production due to its low CPU (and memory) footprint. We also contributed a semi-supervised method to label RTS game logs (replays) with openings (strategies). Both our implementations are free software and can be found online[3]. We will use this model (or and upgraded version of it) in our StarCraft AI competition entry bot as it enables it to deal with the incomplete knowledge gathered from scouting.

[3]https://github.com/SnippyHolloW/OpeningTech/

REFERENCES

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

[2] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *IJCNN*, 2008, pp. 3106–3111.

[3] F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games," in *GAMEON*, 2007, pp. 61–70.

[4] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust, "Opponent behaviour recognition for real-time strategy games," in *AAAI Workshops*, 2010.

[5] D. W. Aha, M. Molineaux, and M. J. V. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *ICCBR*, 2005, pp. 5–20.

[6] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Learning from demonstration and case-based planning for real-time strategy games," in *Soft Computing Applications in Industry*, ser. Studies in Fuzziness and Soft Computing, B. Prasad, Ed. Springer Berlin / Heidelberg, 2008, vol. 226, pp. 293–310.

[7] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Proceedings of the 7th International conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ser. ICCBR '07. Springer-Verlag, 2007, pp. 164–178.

[8] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical plan representations for encoding strategic game ai," in *AIIDE*, 2005, pp. 63–68.

[9] M. Ramírez and H. Geffner, "Plan recognition as planning," in *Proceedings of the 21st international joint conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 2009, pp. 1778–1783.

[10] E. Charniak and R. P. Goldman, "A bayesian model of plan recognition," *Artificial Intelligence*, vol. 64, no. 1, pp. 53–79, 1993.

[11] D. W. Albrecht, I. Zukerman, and A. E. Nicholson, "Bayesian models for keyhole plan recognition in an adventure game," *User Modeling and User-Adapted Interaction*, vol. 8, pp. 5–47, January 1998.

[12] K. Mishra, S. Ontañón, and A. Ram, "Situation assessment for plan retrieval in real-time strategy games," in *ECCBR*, 2008, pp. 355–369.

[13] C. W. Geib and R. P. Goldman, "A probabilistic plan recognition algorithm based on plan tree grammars," *Artificial Intelligence*, vol. 173, pp. 1101–1132, July 2009.

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

[15] J. Diard, P. Bessière, and E. Mazer, "A survey of probabilistic models using the bayesian programming methodology as a unifying framework," in *Conference on Computational Intelligence, Robotics and Autonomous Systems, CIRAS*, 2003.

[16] M. J. Beal, "Variational algorithms for approximate bayesian inference," *PhD. Thesis*, 2003.

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

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

[19] R. Le Hy, A. Arrigoni, P. Bessiere, and O. Lebeltel, "Teaching bayesian behaviours to video game characters," *Robotics and Autonomous Systems*, vol. 47, pp. 177–185, 2004.

[20] C. Fraley and A. E. Raftery, "Model-based clustering, discriminant analysis and density estimation," *Journal of the American Statistical Association*, vol. 97, pp. 611–631, 2002.

[21] ——, "MCLUST version 3 for R: Normal mixture modeling and model-based clustering," University of Washington, Department of Statistics, Technical Report 504, 2006, (revised 2009).

[22] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *NIPS*, 2001, pp. 841–848.

[23] P. Dangauthier, P. Bessiere, and A. Spalanzani, "Auto-supervised learning in the Bayesian Programming Framework," in *ICRA*. IEEE, 2005, pp. 1–6.