A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft

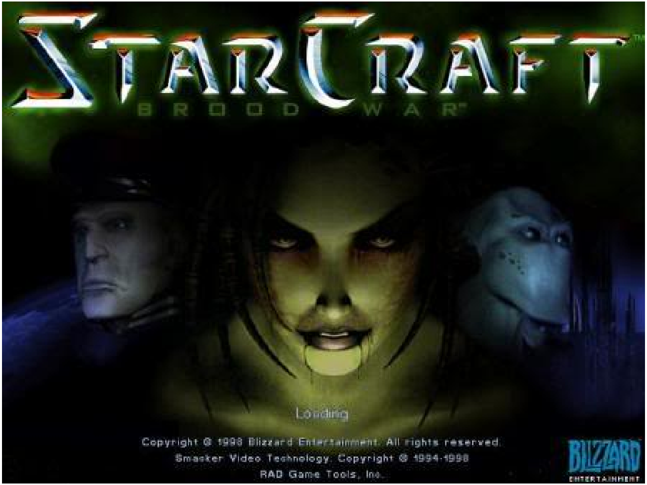
Gabriel Synnaeve and Pierre Bessière

LPPA @ Collège de France (Paris)
University of Grenoble
E-Motion team @ INRIA (Grenoble)

October 14, 2011

Starcraft: Broodwar

Starcraft (January 1998) + Broodwar (exp., November 1998)



Pro gaming and competitions

eSports, sponsorship, tournaments' dotations

WCG
WORLD CYBER GAMES

WORLD CYBER GAMES

The World Cyber Games is the world's first "Cyber Game Festival", designed to build a healthy cyber culture.
The best gamers around the world gather into different cities to share the excitement and fun of the game tournaments.

ESWC
ELECTRONIC SPORTS WORLD CUP

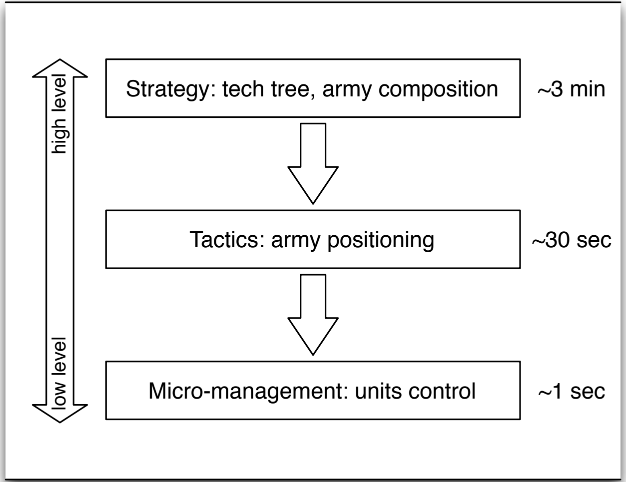
GOSU GAMERS

teamliquid
STARCRAFT PROGAMING NEWS • COMMUNITY • TEAM

Starcraft in numbers

- **12** years of competitive play
- **200** to **300** actions per minute amongst pro gamers
- **10** millions licenses sold (4.5 in South Korea)
- **160** BPM: rates of pro gamers hearts
- **4.5+** millions licenses sold for Starcraft II
- **1/24th** of a second per micro-turn

Granularity of Problems to tackle



Transmute incompleteness into uncertainty

Incompleteness	$\Longrightarrow$	Uncertainty
Many low level moves achieving the same high level goal		Considering the units as individual Bayesian robots
Fog of war (limited sight)		Seen units (viewed units filter)
Partial knowledge of opponent's units/buildings/tech		Probabilistic inference, machine learning from *replays*

low level

high level

A Bayesian program structure

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

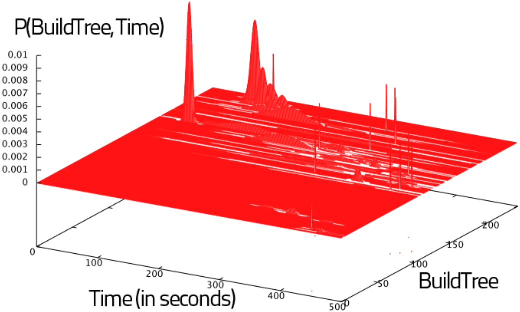
$$
P(Searched|Known)
$$

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

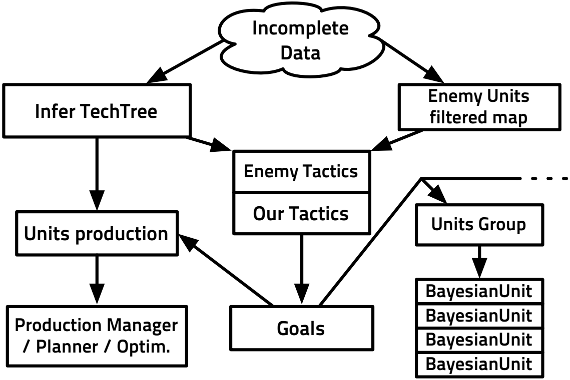
$$
= \frac{1}{Z} \times \sum_{Free} P(Searched, Free, Known)
$$

Machine learning

- reinforcement (exploration of parameters space for the Bayesian robots)
- online (adapt to particular opponent)
- from replays (parameters of predictive models)



BroodwarBotQ model overview



Not a perfect (nor what-we-want-in-the-end) model, but the actual, implemented, bot model.

Examples of cheeses

All-in fast dark templars:
Produce dark templars as fast as possible, *attempt to finish the game with a very specific unit deep in the tech path.* $\rightarrow$ Need to have detection!

All-in 2 gates zealots rush:
Produce only zealots, *attempt to finish the game before the opponent's economy or technological ROI kicked in.* $\rightarrow$ Need to play defensively!

What problem are we trying to solve?

Problem statement

Predict what the enemy **build tree**[a] is from partial observations (because of the *fog of war*) to be able to adapt our own.

[a] We will reserve the term strategy for *army composition* + *long term tactical goals*, which can be infered from the build tree and other variables

Infering the tech tree is exactly the same task as infering the build tree.

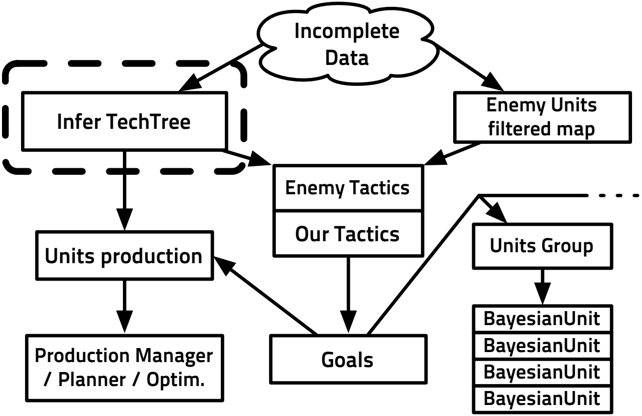
(Another problem is then to dynamically adapt our own techtree/strategy. And it can be done with the same model and extensions, see conclusion.)

Previous works

Supervised (annotated/labeled replays) and semi-supervised (clusterised into labels) learning:

- A Data Mining Approach to Strategy Prediction (2009) [Weber B. & Mateas M.]
- A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft (2011) [Synnaeve G. & Bessière P.]

Where are we?

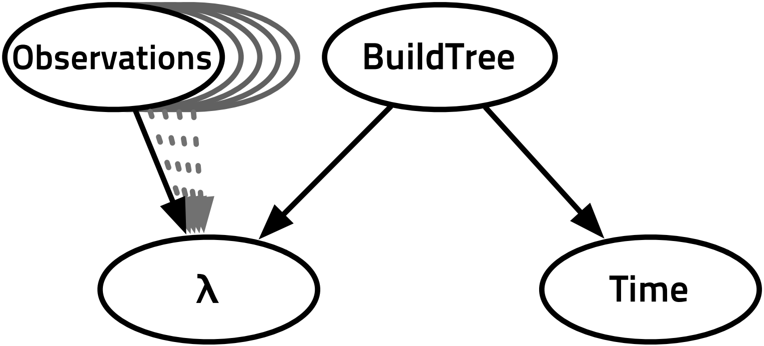

Incomplete Data
Infer TechTree
Enemy Units filtered map
Enemy Tactics
Our Tactics
Units production
Units Group
BayesianUnit
BayesianUnit
BayesianUnit
BayesianUnit
Production Manager / Planner / Optim.
Goals

Replays

Record all the actions of the player so that the game can be deterministically re-simulated (random generators seeds are serialized).

Unsupervised learning model: we just need the replays to be able to learn.

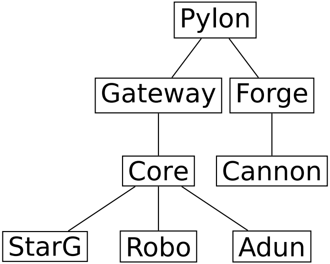
Bayesian Model



Variables

- $BuildTree \in \{\emptyset, building_1, building_2, building_1 \wedge building_2, buildtrees, \dots\}$
- *N Observations*: $O_{i \in [\![1 \dots N]\!]} \in \{0, 1\}$, O_k is 1 (*true*) $\Leftrightarrow$ we saw the unit type k.
- $\lambda \in \{0, 1\}$: coherence variable (restraining *BuildTree* to possible values with regard to $O_{[\![1 \dots N]\!]}$)
- *Time*: $T \in [\![1 \dots P]\!]$

BuildTree variable by example



$$BuildTree \in \{\emptyset, \{Pylon\}, \{Pylon, Gateway\}, \{Pylon, Forge\},$$
$$\{Pylon, Gateway, Forge\}, \{Pylon, Gateway, Core\}, \dots\}$$

Decomposition + forms

$$P(T, BuildTree, O_1 \dots O_N, \lambda) =$$
$$P(T|BuildTree).P(BuildTree)$$
$$P(\lambda|BuildTree, O_{1:N}).P(O_{1:N})$$

- $P(\lambda|BuildTree, O_{[\![1 \dots N]\!]})$ restricts *BuildTree* values to the ones that can co-exist with the observations
- $P(T|BuildTree)$ are discretized normal distributions. There is one bell shape over *Time* per *buildTree*.

A note on identification/learning

- Learning of the $P(T|BuildTree)$ bell shapes parameters takes into account the uncertainty of the couples *buildTrees* for which we have few observations by starting with a high σ^2.
- Learning on human replays for bots opening recognition does not work well. We had to impose a large minimal σ^2 (more robustness at the detriment of precision). (Next year we will use bots replays!)

Question

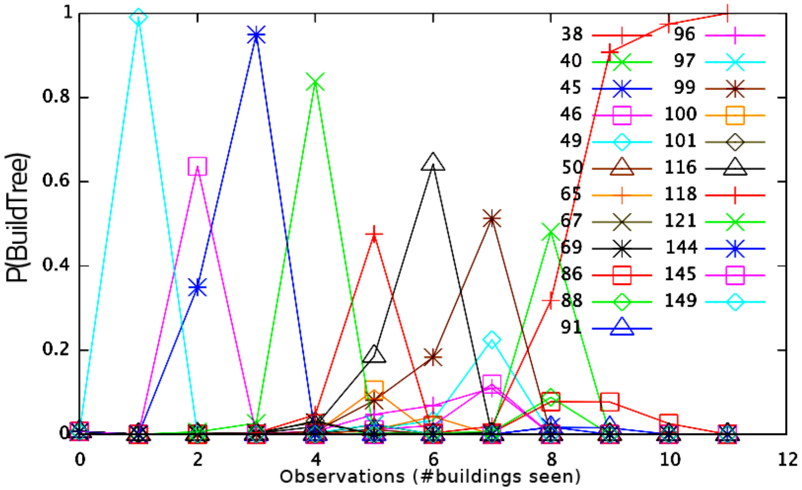
$$P(BuildTree|T = t, O_{1:N} = o_{1:N}, \lambda = 1)$$
$$\propto P(t|BuildTree).P(BuildTree)$$
$$P(\lambda|BuildTree, o_{1:N}).P(o_{1:N})$$

Dataset

- From high level StarCraft players (mainly pros),
- 8806 replays ($\approx$ 1000 / match-up),
- 10-fold cross-validation (learn on 9/10th, test on the rest).

$\Rightarrow$ a bias towards high level style of play ($\neq$ bot meta-game).

Inference


1
0.8
0.6
0.4
0.2
0
P(BuildTree)
0
2
4
6
8
10
12
Observations (#buildings seen)
38
40
45
46
49
50
65
67
69
86
88
91
96
97
99
100
101
116
118
121
144
145
149

Error metric: distance

BuildTrees distance

$$d(bt_1, bt_2) = \text{card}(bt_1 \Delta bt_2) = \text{card}((bt_1 \bigcup bt_2) \backslash (bt_1 \bigcap bt_2))$$

The error **distance** *d* between:

is 2 (it would be 1 with a tree edit distance).

$d(best, real) =$ **"best"** distance

$d(bt, real) * P(bt)$=**"mean"**: marginalized distance

Predictive power

k buildings ahead

k (> 0) next buildings for which we have a "good enough" (limit on d) prediction in future build trees in:

$$P(BuildTree^{t+k}|T = t, O_{1:N} = o_{1:N}, \lambda = 1)$$

(In the tests/results, we sometimes used $d = 1$, $d = 2$, and $d = 3$ as hard limits.)

Low CPU and memory footprint

On a 2.8 Ghz Core 2 Duo:

- Learning with 1000 replays takes $\approx$ 0.1 second,
- Inference takes $\approx$ 0.01 second,
- $\approx$ 3Mb of memory.

Recap. performance table

noise	measure	$\boldsymbol{d}$ for $k = 0$		$\boldsymbol{k}$ for $d = 1$		$\boldsymbol{k}$ for $d = 3$	
		best	"mean"	best	"mean"	best	"mean"
0%	avg	0.535	0.870	1.193	3.991	3.642	6.122
	min	0.313	0.574	0.861	2.8	3.13	4.88
	max	1.051	1.296	2.176	5.334	4.496	7.334
20%	avg	0.610	0.949	0.900	3.263	2.866	4.873
	min	0.381	0.683	0.686	2.3	2.44	3.91
	max	1.062	1.330	1.697	4.394	3.697	5.899
40%	avg	0.740	1.068	0.611	2.529	2.20	3.827
	min	0.488	0.820	0.44	1.65	1.94	3.09
	max	1.257	1.497	1.201	3.5	2.773	4.672
60%	avg	0.925	1.232	0.400	1.738	1.724	2.732
	min	0.586	0.918	0.22	1.08	1.448	2.22
	max	1.414	1.707	0.840	2.483	2.083	3.327
80%	avg	1.134	1.367	0.156	0.890	1.283	1.831
	min	0.665	1.027	0.06	0.56	1.106	1.66
	max	1.876	1.999	0.333	1.216	1.5	2.176

Recap. performance table

noise	measure	d for $k = 0$ best	d for $k = 0$ "mean"	k for $d = 1$ best	k for $d = 1$ "mean"	k for $d = 3$ best	k for $d = 3$ "mean"
0%	avg	0.535	0.870	1.193	3.991	3.642	6.122
	min	0.313	0.574	0.861	2.8	3.13	4.88
	max	1.051	1.296	2.176	5.334	4.496	7.334
20%	avg	0.610	0.949	0.900	3.263	2.866	4.873
	min	0.381	0.683	0.686	2.3	2.44	3.91
	max	1.062	1.330	1.697	4.394	3.697	5.899
40%	avg	**0.740**	**1.068**	0.611	2.529	2.20	3.827
	min	0.488	0.820	0.44	1.65	1.94	3.09
	max	1.257	1.497	1.201	3.5	2.773	4.672
60%	avg	0.925	1.232	0.400	1.738	1.724	2.732
	min	0.586	0.918	0.22	1.08	1.448	2.22
	max	1.414	1.707	0.840	2.483	2.083	3.327
80%	avg	1.134	1.367	0.156	0.890	1.283	1.831
	min	0.665	1.027	0.06	0.56	1.106	1.66
	max	1.876	1.999	0.333	1.216	1.5	2.176

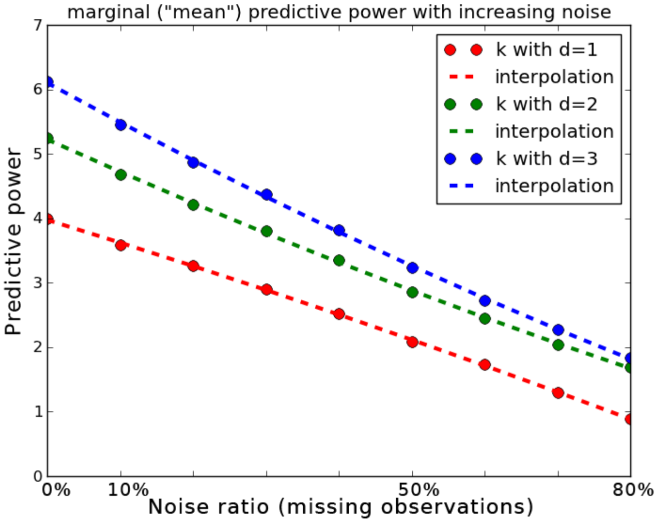
Recap. performance table

noise	measure	d for $k=0$ best	d for $k=0$ "mean"	k for $d=1$ best	k for $d=1$ "mean"	k for $d=3$ best	k for $d=3$ "mean"
0%	avg	0.535	0.870	1.193	3.991	3.642	6.122
	min	0.313	0.574	0.861	2.8	3.13	4.88
	max	1.051	1.296	2.176	5.334	4.496	7.334
20%	avg	0.610	0.949	0.900	3.263	2.866	4.873
	min	0.381	0.683	0.686	2.3	2.44	3.91
	max	1.062	1.330	1.697	4.394	3.697	5.899
40%	avg	0.740	1.068	**0.611**	**2.529**	2.20	3.827
	min	0.488	0.820	0.44	1.65	1.94	3.09
	max	1.257	1.497	1.201	3.5	2.773	4.672
60%	avg	0.925	1.232	0.400	1.738	1.724	2.732
	min	0.586	0.918	0.22	1.08	1.448	2.22
	max	1.414	1.707	0.840	2.483	2.083	3.327
80%	avg	1.134	1.367	0.156	0.890	1.283	1.831
	min	0.665	1.027	0.06	0.56	1.106	1.66
	max	1.876	1.999	0.333	1.216	1.5	2.176

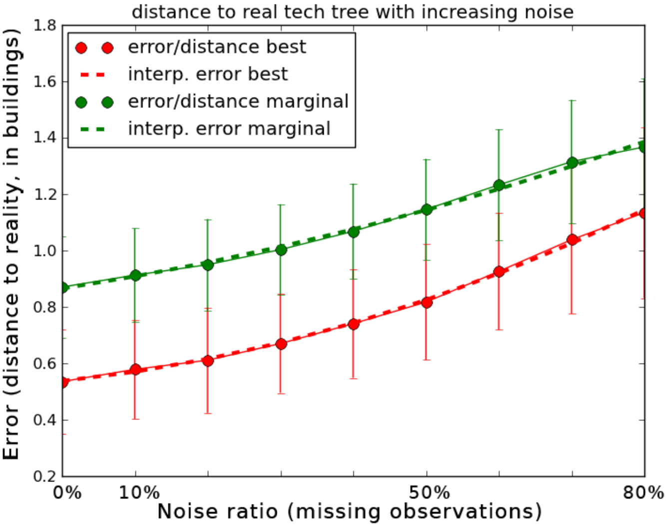
Recap. performance table

noise	measure	***d*** for $k = 0$		***k*** for $d = 1$		***k*** for $d = 3$	
		best	"mean"	best	"mean"	best	"mean"
0%	avg	0.535	0.870	1.193	3.991	3.642	6.122
	min	0.313	0.574	0.861	2.8	3.13	4.88
	max	1.051	1.296	2.176	5.334	4.496	7.334
20%	avg	0.610	0.949	0.900	3.263	2.866	4.873
	min	0.381	0.683	0.686	2.3	2.44	3.91
	max	1.062	1.330	1.697	4.394	3.697	5.899
40%	avg	0.740	1.068	0.611	2.529	**2.20**	**3.827**
	min	0.488	0.820	0.44	1.65	1.94	3.09
	max	1.257	1.497	1.201	3.5	2.773	4.672
60%	avg	0.925	1.232	0.400	1.738	1.724	2.732
	min	0.586	0.918	0.22	1.08	1.448	2.22
	max	1.414	1.707	0.840	2.483	2.083	3.327
80%	avg	1.134	1.367	0.156	0.890	1.283	1.831
	min	0.665	1.027	0.06	0.56	1.106	1.66
	max	1.876	1.999	0.333	1.216	1.5	2.176

Predictive power under noise



Error distance evolution w/ noise



Comparing results with existing works

Compared to previous work by Ben Weber (CIG 2009):

- Works with partial information (fog of war),
- Resists quite well to noise,
- Gives a distribution, not just a decision (that's how high level human player think, I think ☺).

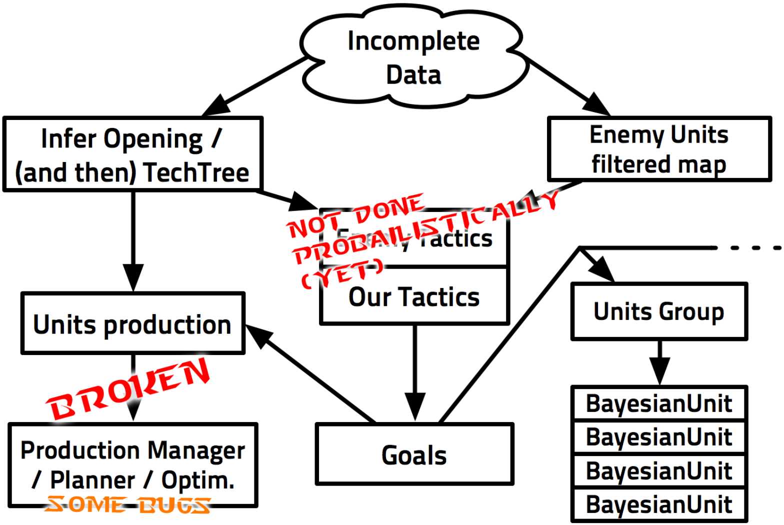
Compared to both previous works ([Weber09] and [Synnaeve11]):

- Unsupervised,
- Usable during the "end game".

Possible uses

- Adaptive RTS AI:
  - Direct rules triggers (“DT tech $\Rightarrow$ detection”),
  - Integrated in a Bayesian decision model (leveraging the distribution on *BuildTree* more easily).
- Commentary assistant (null noise, prediction of tech trees), as Poker commentary software do.

"Why does your bot suck?"



Possible Improvements

- Direct possible improvements:
  - Learning the parameters of the model from a **bigger dataset**,
  - Learning the parameters of the model from **bot vs bot replays**,
- Additional model/extension:
  - Learn which $BuildTree_1$ wins against $BuildTree_2$ so that we can ask: $P(BuildTree_{bot}|obs_{op,1:N}, time, \lambda = 1)$ by the intermediate $P(BuildTree_{op}|obs_{op,1:N}), time, \lambda = 1)$ for dynamic adaptation of our own $Build/TechTree$.
  - A filter on $P(BuildTree_{bot}^{t}|BuildTree_{bot}t - 1)$ which will balance radical changes.

Bibliography

- Bayesian Robot Programming (2004) [Lebeltel O. *et al.*]
- A Data Mining Approach to Strategy Prediction (2009) [Weber B. & Mateas M.]
- Case-Based Planning and Execution for RTS Games (2007) [Ontañón S. *et al.*]
- Opponent Behaviour Recognition for Real-Time Strategy Games (2010) [Kabanza F. *et al.*]
- Building A Player Strategy Model by Analyzing Replays of Real-Time Strategy Games [Hsieh J-L. & Sun C-T.]
- Probability Theory: The Logic of Science (2003) [Jaynes E.T.]

Thanks

Thank you for your attention,

Questions ?