Build Order Selection in StarCraft* Utilizing a Customized Bayesian Multi-Armed Bandit Algorithm

Hao Pan
QOMPLX, Inc.
Reston, VA
hao.pan@QOMPLX.com

Abstract

Solutions to the multi-armed bandit (MAB) problem have been used in various areas including Real Time Strategy (RTS) games such as StarCraft®. Within StarCraft, one challenging area to apply such solutions is build order (BO) selection. The build order is defined as the concurrent action sequences that, constrained by unit dependencies and resource availability, create a certain number of units and structures in the shortest possible time span (Churchill and Buro 2011). Due to the uncertain and potentially non-stationary game outcome (e.g., winning/losing a game) of each build order, a customized Bayesian multi-armed bandit algorithm is proposed in this paper to provide an effective way to make the optimal decision. To demonstrate the effectiveness of the proposed algorithm, a case study is done using data from test results of a few selected StarCraft: Brood War bots. In addition, the results are compared with those from alternative algorithms.

Introduction

In MAB problems, one can imagine a gambler facing a row of slot machines (each machine is a one-armed bandit as it has a single lever to pull). The gambler seeks to find the best strategy to maximize the rewards resulting from pulling a certain amount of machines sequentially. Such strategy involves choosing which machines to pull/play and in which order to play them. As early as 1952, Herbert Robbins realized the importance of the MAB problem and devised techniques to have convergent population selection for the design and analysis of sampling experiments (Robbins 1952). (Katehakis and Robbins 1985) sought the fastest rate of convergence to the population with the highest mean (in the context of MAB, this can be viewed as the arm yielding the best reward). To do so, they considered sampling sequentially from multiple populations in order to maximize the total outcome in the long run. Since then, (Auer, Cesa-Bianchi, and Fischer 2002; Katehakis and Veinot 1987) formulated the multi-armed bandit problem. The MAB problem is a classic reinforcement learning problem. The gambler starts from zero knowledge of the bandits, and as time progresses, the gambler eventually learns the reward of each machine. The gambler is bound to the exploration-exploitation dilemma. At first some exploration must be conducted. This could be testing each bandit once, or randomly sampling some bandits. Sooner or later, the gambler has to decide which machines give better rewards than the others and stick with them, or equivalently, at what time exploitation is to happen. Alternatively, one can view each arm as an independent Markov machine. As time progresses, some of the arms are pulled, and the states of the machines would advance to the next ones based on the Markov state evolution probabilities. This formulation is the so-called “restless bandit problem” (White 1988).

The multi-armed bandit problem is seen in many areas such as clinical trials (Gittins 1989; Berry and Fristedt 1985), financial portfolio design (Brochu, Hoffman, and de Freitas 2010; Shen et al. 2015), and in RTS games such as StarCraft (Ontañón 2017; Moraes et al. 2018). Upper Confidence Trees (UCT), a Monte Carlo Tree Search planning algorithm, was used by (Balla and Fern 2009) in some sequential decision problems (which are extensions of MAB problems) for the RTS game Wargus. (Justesen et al. 2014) used two extensions of UCT to tackle the challenging problem of controlling unit actions (especially those used in combat) in StarCraft. To optimize a build order, or more specifically, to search for the minimum-makespan action sequences given a static BO goal (such as producing 4 tanks as quickly as possible), (Churchill and Buro 2011) searched through the vast game space, and (Justesen and Risi 2017) utilized the Continual Online Evolution Planning (COEP) algorithm. Such approaches are thorough but often come up with a single optimal BO, which may not be practical, especially when in a tournament such as SSCAIT [1], where oftentimes a bot does not win by using one strategy. There are bots which can analyze opponent’s BOs and devise counter measures. A single BO is easily exploitable in this case. One solution to this phenomenon is to try to reach the mixed strategy Nash equilibrium (Osborne and Rubinstein 1994). However, the Nash equilibrium is not easy to compute, and might not even matter. The authors of the superhuman poker AI Pluribus (Brown and Sandholm 2019) decided to go fully empirical

*StarCraft is a trademark or registered trademark of Blizzard Entertainment, Inc., in the U. S. and/or other countries. Nothing in this paper should be interpreted as approval, endorsement, or sponsorship by Blizzard Entertainment, Inc.

[1]SSCAIT homepage: https://sscaitournament.com/

and not consider the problem of Nash equilibrium. Regardless of the approaches, it is desirable to intelligently choose the optimal strategy/BOs out of a pool of feasible ones. Each build order can be viewed as an arm in the MAB problem. The Upper Confidence Bound method (usually referred to as the UCB1 algorithm) is used by UAlbertaBot and many others (Ontañón et al. 2013) to choose the optimal BOs.

Yet there are challenges that are not fully resolved: 1) Uncertainty in game outcome. Such uncertainty can come from multiple sources such as map factors (map size, choke point width, etc) and unit behavior (decision to attack/defend, etc); 2) Non-stationary game outcome. This means the game outcome can change over time. Some opponents can adapt. Bayesian approaches can come to the rescue. They are already used to solve problems in StarCraft (Synnaeve and Bessière 2015) and to solve MAB problems (Urteaga and Wiggins 2018; Scott 2010). However, to the best of the authors' knowledge, there is little to no literature on formulating Bayesian MAB problem for BO selection. This paper aims at fertilizing this "barren land", as well as tackling the aforementioned two challenges by: 1) proposing a customized Bayesian multi-armed bandit algorithm; 2) quantifying uncertainty in reward by acquiring posterior distribution; 3) applying a moving-window technique to handle the non-stationary nature of the reward. In addition, suggestions will be made to select hyper-parameters for such a Bayesian multi-armed bandit algorithm. The effectiveness of the proposed algorithm will be compared with that of several alternatives, namely, UCB1, EXP3 (Seldin et al. 2012) and random pick (RP: one picks BOs randomly with equal weights assigned to each BO).

Methodology

In a multi-armed bandit problem, there are $K \in \mathbb{N}^+$ machines, each with a lever to pull, yielding a reward $R_i, i = 1, 2, \ldots, K$ which is unknown in nature but often assumed to follow a distribution. One is to choose one machine at a time, and repeat (choosing the same machine or a different one) for a total of T times. If the distribution is identical for all T time steps, such formulation of the MAB problem is called stochastic bandit. The UCB1 algorithm is used as the standard solution to explore the action space efficiently while exploiting the best action to achieve minimum regret (Yu and Sra 2019). If the distribution is allowed to vary over time, such formulation is called adversarial bandit. The standard algorithm for this is the EXP3 algorithm which computes and updates unbiased estimates of cumulative rewards via importance sampling. There are other solutions to find the best series of actions. One such solution is called the Follow the Perturbed Leader (FPL) algorithm (Hutter and Poland 2005) and its main steps are provided below.

FPL algorithm

```
Initialize: $R_i^{(0)} = 0.5 \forall i$
At time $t$ ($t = 1, 2, ..., T$):
for ($i$ in $1 : K$)
  Update reward at time $t$ for arm $i$ based on all of
    its outcomes in the past $t-1$ time steps: $R_i^{(t)}$
    Generate noise: $\epsilon_i^{(t)}$ by drawing a random number
    from an exponential distribution $Exp(\eta)$
    Perturb the reward (potentially the leader) by
    computing the adjusted reward: $\hat{R}_i^{(t)} = R_i^{(t)} + \epsilon_i^{(t)}$
End for loop
Find arm $j$ such that $j = \underset{i}{\operatorname{argmax}}\{\hat{R}_i^{(t)}\}$ and pull it.
Record the outcome (e.g., success or failure): to be
  used in the Update step when time progresses to $t+1$.
```

To solve the challenge of finding the reward distribution using Bayesian statistics, the reward in the context of selecting build order in an RTS game is to be defined first. There are multiple ways to define the reward, such as using StarCraft's in-game metrics which take into account unit, building, resource, and combat scores. But for the sake of simplicity, the reward is defined here as the win rate. It is assumed that $R_i \sim Bernoulli(\theta_i), i = 1, 2, \ldots, K$. The distribution of the win rate is initially unknown but can be inferred. Due to the flexibility to model a range of distributions, the two-parameter beta distribution is assumed for the prior distribution $\pi(\theta_i)$: $\theta_i \sim Beta(\alpha_0, \beta_0)$, where $\alpha_0 > 0$ and $\beta_0 > 0$. Using the Bayes theorem, the posterior distribution can be expressed as:

$$p(\theta_i|R_i) \propto \pi(\theta_i) * \mathcal{L}_{\theta_i|R_i=r_i}(\theta_i)$$

where $\mathcal{L}_{\theta_i|R_i=r_i}(\theta_i)$ is the likelihood function and it is expressed as:

$$\mathcal{L}_{\theta_i|R_i=r_i}(\theta_i) = \binom{S_i + F_i}{S_i} \theta_i^{S_i}(1-\theta_i)^{F_i}$$

where S_i and F_i are the number of successes (wins) and failures (losses) associated to arm i observed in the previous $t-1$ steps. The likelihood function can be understood as the probability of acquiring S_i successes from a total of $S_i + F_i$ trials given θ_i.

Recall that a random variable θ on the interval $(0, 1)$ following a Beta distribution with parameters α and β has the density function defined as:

$$\pi(\theta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

where $\Gamma()$ is the Gamma function.

By computing the integral $\int \pi(\theta_i)\mathcal{L}_{\theta_i|R_i=r_i}(\theta_i)d\theta_i$ as the normalizing factor (which can be understood as the weighted average of all possible posteriors), the density of the posterior distribution is expressed as:

$$\begin{aligned} p(\theta_i|R_i) &\propto \frac{\pi(\theta_i) * \mathcal{L}_{\theta_i|R_i=r_i}(\theta_i)}{\int \pi(\theta_i) * \mathcal{L}_{\theta_i|R_i=r_i}(\theta_i)d\theta_i} \\ &\propto \frac{\binom{S_i+F_i}{S_i} B(\alpha_0, \beta_0)\theta_i^{\alpha_0+S_i-1}(1-\theta_i)^{\beta_0+F_i-1}}{\int_0^1 \binom{S_i+F_i}{S_i} B(\alpha_0, \beta_0)\theta_i^{\alpha_0+S_i-1}(1-\theta_i)^{\beta_0+F_i-1}d\theta_i} \\ &\propto B(\alpha_0 + S_i, \beta_0 + F_i)\theta_i^{\alpha_0+S_i-1}(1-\theta_i)^{\beta_0+F_i-1} \end{aligned}$$

where $B(\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$ is the Beta function.

The posterior distribution is then found to also follow a beta distribution:

$$p(\theta_i|R_i) \sim Beta(\alpha_0 + S_i, \beta_0 + F_i)$$

The posterior mean is considered as the actual value of the reward at time t and is expressed as:

$$\overline{\theta} = \frac{\alpha_0 + S_i}{\alpha_0 + \beta_0 + S_i + F_i}$$

In the context of selecting build order, where the reward is essentially the win rate, the parameters of the prior distribution are assumed to be equal to each other: $\alpha = \beta$ for symmetry. Moreover, a flat prior ($\alpha = \beta = 1.0$) is preferred over a shaped one (e.g., one where $\alpha = \beta = 2.0$). Utilizing a flat prior reflects the reality better: all win rate values, from 0% to 100%, are equally possible.

The rate parameter η of the exponential distribution which the artificial noise (the perturbation term) follows must be chosen carefully. η should ideally be set in such a way to provide enough chances for the algorithm to perform exploration (rather than exploiting all the time), while at the same time minimizing its disturbance to the reward. A value between 5 and 10 is recommended here for η, as such value corresponds to a swing in win rate roughly between 10% and 20% which is a disturbance significant enough but not overpowering the underlying true value of the win rate. With this consideration and the posterior distribution of the reward, the proposed algorithm (baseline) can be expressed as the following steps:

```
Proposed baseline algorithm

At time $t$:
for ($i$ in $1 : K$)
  Update reward at time $t$ for arm $i$:
    $R_i^{(t)} = \frac{\alpha + S_i}{\alpha + \beta + S_i + F_i}$
  Generate noise: $\epsilon_i^{(t)}$ by drawing a random number
  from an exponential distribution $Exp(\eta)$
  Perturb the reward by computing
   the adjusted reward: $\hat{R}_i^{(t)} = R_i^{(t)} + \epsilon_i^{(t)}$
End for loop
find arm $j$ such that
$j = \underset{i}{\mathrm{argmax}}\{\hat{R}_i^{(t)}\}$ and pull it
Record the outcome (e.g., success or failure): to be
  used in the Update step when time progresses to $t + 1$.
```

The proposed baseline algorithm does not require any previous result of pulling arms. While for UCB1, it is required that each arm is pulled at least once before it can function properly.

As is mentioned earlier, the distribution of the rewards can vary over time. This assumption is practical because there are bots which use multiple BOs and can vary their strategy by choosing BOs intelligently (using algorithms such as UCB1). Moreover, some bots such as Steamhammer can perform opponent modeling, or opponent BO identification [2] and adapt by selecting from a pool of weighted BOs deemed advantageous for themselves. To ensure that the algorithm is relatively free to break out of the accumulated knowledge of the opponent should drastic changes occur in the distribution of the rewards, the proposed baseline algorithm is extended to a Moving Window (MW) variant where a MW is applied to the number of games to be examined in the past when calculating the reward (as compared to using all the games occured from time 0 up to time t). More specifically, the MW is applied in the **Update** step where S_i and F_i are counted using only the most recent N (the size of the MW) games. The introduction of MW here, together with how the reward is updated, are the two main differences between the MW variant and the FPL algorithm.

There is a tradeoff when one chooses the size N of the MW. If the size is too small, there wouldn't be enough games to draw a firm conclusion on the posterior distribution of the reward. If the size is too large, one risks picking the non-optimal BOs, especially when the stationarity assumption is violated. The size of the MW is set to be 40 based on empirical study which was conducted to minimize the impact of sudden change in the distribution of the rewards. A properly selected MW size can help to mitigate such impact. One can verify this by comparing the results before and after the stress test scenario which is to be described in the next section.

Case Study

The setup of the case study is as follows. A few of the top SSCAIT (Čertický et al. 2018; Čertický and Churchill 2017; Churchill et al. 2016) bots on the BASIL [3] ladder were used as opponents against the authors' own bot (renamed as bot X here). The opponent bots and their factions are: 1) Tomas Vajda: Protoss (P); 2) Marian Devecka: Zerg (Z); 3) Iron bot: Terran (T); 4) BananaBrain: (P); 5) tscmoo: (T) and 6) Steamhammer: (Z). Bots 1-3 use a single BO and they'll be referred to as non-adapting bots. Bots 4-6 use multiple BOs and can choose them intelligently. They'll be referred to as adapting bots. Among the adapting bots, "... BananaBrain uses UCB1 algorithm to choose which build to use"[4]. Both tscmoo and Steamhammer use some weighted random algorithm to choose BOs. Bot X plays as the Terran faction.

An extensive number of games were run against non-adapting bots for each BO of bot X, so that the true value of the win rate can be estimated. The results are summarized in Table 1. The highest winrates, or the best results, are highlighted in bold font. This remains true for all the subsequent tables in this paper. In Table 1, the data in each cell mean the number of games won vs a specific opponent using a particular BO (the first number) and the total number of games played vs this opponent with the BO (the second number). The win rate is presented in the parentheses. There are 4 BOs for X to choose from, and they are quite different

[2]Jay Scott's blog post: http://satirist.org/ai/starcraft/blog/archives/362-Steamhammers-opponent-model.html

[3]BASIL ranking page: https://basil.bytekeeper.org/ranking.html

[4]BananaBrain's wiki: https://liquipedia.net/starcraft/BananaBrain

Table 1: Win rate vs non-adapting opponents

	Tomas Vajda	Marian Devecka	Iron bot
BO1	86/98 (88%)	0/98 (0%)	28/98 (29%)
BO2	36/98 (37%)	83/98 (85%)	**69/98 (70%)**
BO3	92/98 (94%)	**94/98 (96%)**	0/98 (0%)
BO4	**94/98 (96%)**	1/98 (1%)	0/98 (0%)

from each other, ranging from rush builds to macro (aiming for the long game) ones. In addition, these BOs were chosen so as to discretize the vast game/BO space based on their popularity and effectiveness (especially in the human competitive scenes).

For adapting bots, the winrates can vary depending on how bot X and the opponent choose the BOs, respectively, as time progresses. One can call this a co-evolution problem. The winrates can also vary depending on which BO-selecting algorithm is used. Due to the limit on page numbers, such winrates are not presented here but can be made available per request.

In this case study, there are 5 algorithms whose performance is to be compared, namely: 1) the proposed baseline algorithm, or simply, baseline; 2) MW variant; 3) UCB1; 4) EXP3 and 5) RP. For each of the 5 algorithms, 1000 games were played between bot X and each of the opponents. The games were simulated in the sense that the outcome was determined by the estimated winrates shown in Table 1, instead of by playing each entire game out. 1000 games are considered sufficient to produce an outcome that is statistically similar to that acquired by running the actual games.

It is assumed that there is no significant change in a bot's behavior when executing a particular BO within each of these 1000-game runs. Such behavior can be unit micro, unit training/building construction priorities, etc. There are two criteria to measure how good an algorithm is: 1) average regret (AR). This value measures the averaged difference between the realized reward and the best reward. At each time step a regret value is computed and all such values are averaged over T time steps; 2) The number of times when the best BO was selected, or number of successes (NS). Criterion 1 is considered more indicative about an algorithm's performance than criterion 2 because cases where winrates of the best BO and the second best BO are similar can easily devalue some algorithms when they choose the second best BO (which isn't a bad choice) frequently. Table 2 holds the result using various algorithms when bot X was pitched against non-adapting bots.

The proposed baseline algorithm achieved the best result in terms of both AR (the smaller the value, the better, or less regret) and NS (the larger the value, the better, or more times the optimal BO is selected). The performance of the

Table 2: Comparison of results using various algorithms: vs non-adapting opponents

	Tomas Vajda	Marian Devecka	Iron bot
		Baseline	
AR	**1%**	**1%**	**1%**
NS	**713**	**955**	**973**
		MW Variant	
AR	2%	2%	6%
NS	442	910	885
		UCB1	
AR	4%	5%	5%
NS	424	709	898
		EXP3	
AR	9%	25%	23%
NS	421	596	614
		RP	
AR	16%	49%	45%
NS	256	243	255

Table 3: Comparison of results using various algorithms: vs adapting opponents

	Banana-Brain	tscmoo	Steam-hammer
		Baseline	
AR	**2%**	**5%**	**2%**
NS	**858**	**813**	688
		MW Variant	
AR	4%	8%	5%
NS	445	746	521
		UCB1	
AR	4%	**5%**	6%
NS	431	594	**841**
		EXP3	
AR	7%	13%	11%
NS	591	606	596
		RP	
AR	12%	66%	22%
NS	239	262	261

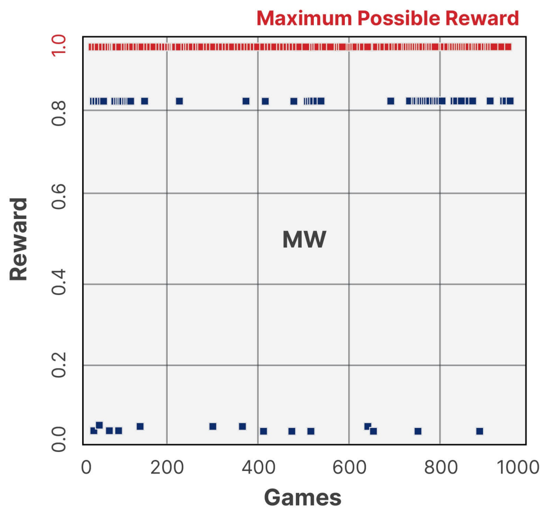

Figure 1: Time series of rewards obtained by the MW variant: bot X vs Marian Devecka

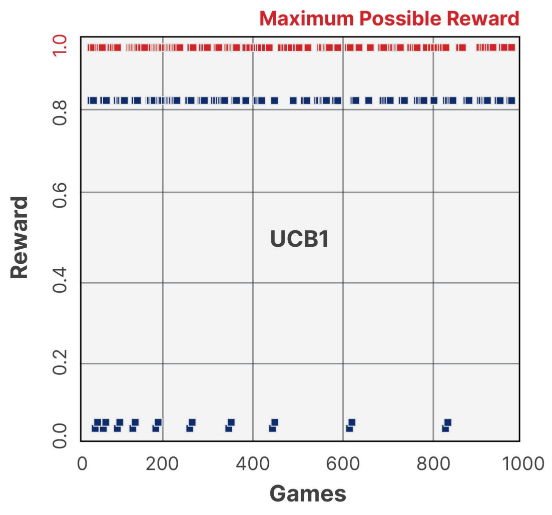

Figure 2: Time series of rewards obtained by UCB1: bot X vs Marian Devecka

MW variant is slightly worse than that of baseline but still is better than UCB1, EXP3 and RP.

Using games of bot X vs Marian Devecka, Figures 1 and 2 compare the time series of the rewards (win rate values from the column "Marian Devecka" in Table 1) obtained by the MW variant and UCB1, respectively. Ideally, one would like to get the maximum possible reward by choosing the best BO as many times as possible (meaning the more often the grey dots fall on the red line, the better). The two figures can also be viewed as the time series of the BOs chosen by the two algorithms, as each BO has a unique reward (win rate) in this case (bot X vs Marian Devecka). It is quite obvious that the MW variant identified the best BO faster, and made much fewer choices of the second best BO than UCB1. UCB1 struggled a lot between picking the second best BO and the best one. One possible explanation for this, as (Audibert, Munos, and Szepesvári 2008) pointed out, is that during the first draws the (unknown) optimal arm gives low rewards, and this makes the reward of this arm inferior to that of the other arms. As a result, the algorithm might get stuck, potentially not choosing the optimal arm any more.

Similarly, for each of the 5 algorithms and each of the adapting opponents, 1000 games were run, and Table 3 shows the result. The proposed baseline algorithm performed the best again, except for NS vs Steamhammer. This is largely due to the fact that when using the baseline algorithm vs Steamhammer, the winrates of the two best BOs only differ by 2%, while the difference was large enough (30%) when using UCB1. It should also be noted that, although the performance of EXP3 is better than that of RP, it is still considerably worse than that of either baseline, MW variant, or UCB1. EXP3 tends not to punish those BOs which lose initially. Consider this simple case where one starts fresh (no BO is ever played). EXP3 initiates by setting weights of all BOs equal. Suppose BO1 gets played once and it loses, its weight gets updated but remains essentially the same due to the reward being zero. This continues to happen so long as the chosen BO results in a loss. In other words, EXP3 then wanders among all the BOs until it becomes clearer which one is better.

Next, a stress test scenario is artificially constructed to examine how the algorithms react to sudden change in our own behavior. More specifically, our best and worst winrates are swapped after the previous 1000-game runs. Such a scenario can happen in reality. For example, a production freeze bug would cause a bot's best BO to perform the worst. The worst BO can become the best one if recent development improved it significantly. In the column of "Iron bot", the two worst winrates (associated to BO3 and BO4) are the same. BO3 was chosen for this stress test. Then another 1000 games were run using the new winrates for each opponent and each algorithm. It is again assumed that there is no significant change in a bot's behavior within these 1000-game runs. Similarly, Tables 4 and 5 show results vs non-adapting and adapting opponents, respectively.

After the stress test, it turns out the results obtained by both the proposed baseline algorithm and EXP3 are much worse than those before the stress test. This is mainly because both algorithms rely heavily on the accumulated knowledge, and when there's a sudden change, the two algorithms lack a way to adapt/would require a large number of games before they identify the new best BO. On the other hand, the MW variant performed the best overall, while UCB1 leads slightly in the cases vs Iron bot and vs Steamhammer in terms of NS. Both of these algorithms prove to be able to adapt to the sudden change in rewards. Figures 3 and 4 show time series of the rewards obtained by the two algorithms after the reward change, using games of bot X vs Marian Devecka again. By looking at games right after the reward change (those on the left side of the figures, right after game #1000), it is obvious that the MW variant took considerably fewer games to adapt (to learn to

Table 4: Comparison of results using various algorithms: after reward change, vs non-adapting opponents

	Tomas Vajda	Marian Devecka	Iron bot
	Baseline		
AR	17%	27%	22%
NS	107	351	683
	MW Variant		
AR	**2%**	**3%**	**7%**
NS	**694**	**907**	882
	UCB1		
AR	5%	5%	**7%**
NS	486	865	**885**
	EXP3		
AR	11%	36%	53%
NS	101	145	189
	RP		
AR	17%	50%	46%
NS	251	264	247

Table 5: Comparison of results using various algorithms: after reward change, vs adapting opponents

	Banana Brain	tscmoo	Steam-hammer
	Baseline		
AR	13%	30%	10%
NS	301	327	402
	MW Variant		
AR	**5%**	**8%**	**6%**
NS	**366**	**769**	435
	UCB1		
AR	7%	9%	8%
NS	240	363	**827**
	EXP3		
AR	17%	32%	27%
NS	125	139	125
	RP		
AR	12%	67%	22%
NS	237	254	251

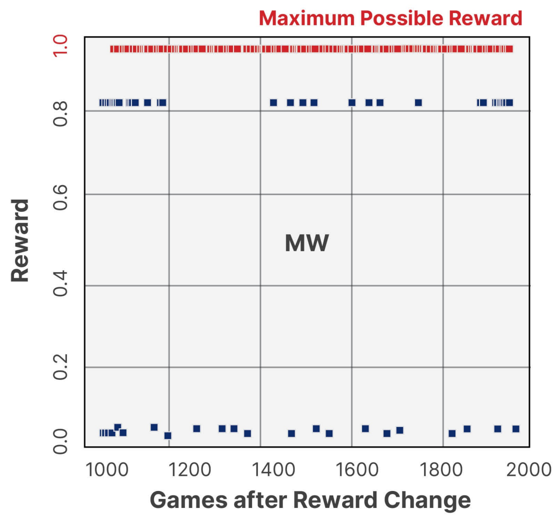

Figure 3: Time series of rewards obtained by the MW variant after reward change: bot X vs Marian Devecka

not choose the previous best BO) than UCB1.

Conclusion

In this paper, a customized Bayesian Multi-Armed Bandit algorithm was proposed to help selecting the best build order for use in the game StarCraft: Brood War. The proposed algorithm intended to solve two challenges by means of: 1) having a closed form of posterior distribution to describe the reward; 2) applying a MW to deal with potentionally non-stationary reward distribution. A case study was conducted to demonstrate the effectiveness of the proposed algorithm and compare results with those from the alternatives (UCB1, EXP3, and RP) using two measures: AR and NS. A stress test scenario was also constructed and it confirmed the superiority of the proposed algorithm over the alternatives when handling non-stationary reward.

There are things to be improved in the future: 1) One can use more accurate models to depict opponents' strategies/BOs and the way they choose them. Then there is motivation to develop X bot's counter measures on anticipating an opponent's BO. This will affect how our BOs are selected; 2) It is risky to run only 4 BOs for each matchup, as doing so could potentially oversimplify the BO selection problem. However, the builds used by pro players are branched from only a few main ones. This remains a debatable topic and the effect of the number of BOs bot X uses on the proposed algorithm would be interesting to study. Of particular interest is how much this would impact the parameter η used in the perturbation term; 4) The flexibility of the beta distribution used in the proposed algorithms would enable applications in areas other than build order selection in StarCraft, so long as the reward can be modeled as a distribution. Such generalization is possible but requires careful examination and study; 5) Thompson sampling (Russo et al. 2017; Thompson 1933) introduced an intelligent way to allocate

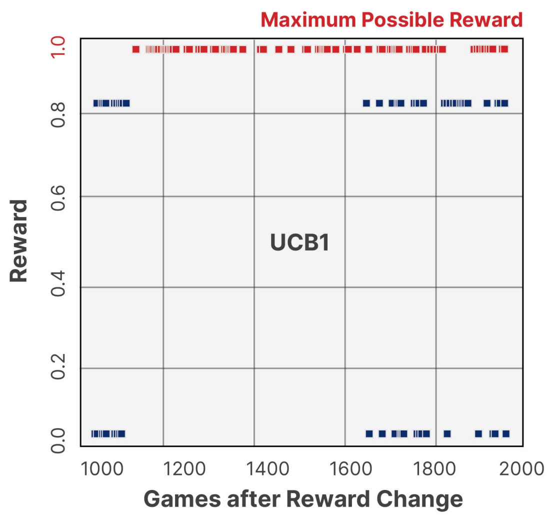

Figure 4: Time series of rewards obtained by UCB1 after reward change: bot X vs Marian Devecka

exploration effort. However during the development of the proposed algorithm, it was discovered that Thompson sampling encouraged too much exploration and hinders the timely discovery of the optimal BO. It's not included in the proposed algorithms but remains something to consider.

Acknowledgements

The authors would like to specifically thank Nathan Roth (MSc) for his dedicated effort in helping performing the replay analysis of bot X and advising strategy/BOs to use for each matchup. The authors would also like to thank the SSCAIT community for their kind support. The authors are equally thankful for Dennis Waldherr and his BASIL ladder, as it provided an invaluable test-bed like environment to see games/replays in a timely fashion.

References

Audibert, J.-Y.; Munos, R.; and Szepesvári, C. 2008. Variance estimates and exploration function in multi-armed bandit. Research report, IMAGINE (previously CERTIS), Ecole des Ponts ParisTech (ENPC).

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multi-armed bandit problem. *Machine Learning* 47(2/3):235–256.

Balla, R.-K., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. *Proceedings of the International Joint Conference on Artificial Intelligence* 40–45.

Berry, D. A., and Fristedt, B. 1985. *Bandit problems: sequential allocation of experiments, monographs on statistics and applied probability*. London: Chapman & Hall.

Brochu, E.; Hoffman, M. W.; and de Freitas, N. 2010. Portfolio allocation for Bayesian optimization. *arXiv:1009.5419*.

Brown, N., and Sandholm, T. 2019. Superhuman AI for multiplayer poker. *American Association for the Advancement of Science (AAAS)*. DOI: 10.1126/science.aay2400.

Churchill, D., and Buro, M. 2011. Build order optimization in StarCraft. *The Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)* 14–19.

Churchill, D.; Preuss, M.; Richoux, F.; Synnaeve, G.; Uriarte, A.; Ontañnón, S.; and Čertický, M. 2016. *StarCraft Bots and Competitions*. Cham: Springer International Publishing. 1–18.

Čertický, M., and Churchill, D. 2017. The current state of StarCraft AI competitions and bots. *Proceedings of the AIIDE 2017 Workshop on Artificial Intelligence for Strategy Games*.

Čertický, M.; Churchill, D.; Kim, K.-J.; Čertický, M.; and Kelly, R. 2018. StarCraft AI competitions, bots and tournament manager software. *IEEE Transactions on Games (ToG)* 1(13):1–1.

Gittins, J. C. 1989. *Multi-armed bandit allocation indices*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Ltd.

Hutter, M., and Poland, J. 2005. Adaptive online prediction by following the perturbed leader. *Journal of Machine Learning Research* 6:639–660.

Justesen, N., and Risi, S. 2017. Continual online evolutionary planning for in-game build order adaptation in starcraft. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, 187–194. ACM.

Justesen, N.; Tillman, B.; Togelius, J.; and Risi, S. 2014. Script- and cluster-based UCT for StarCraft. *IEEE Conference on Computational Intelligence and Games* 1–8.

Katehakis, M. N., and Robbins, H. 1985. Sequential choice from several populations. *Proceedings of the National Academy of Sciences of the United States of America* 92(19):8584–8585.

Katehakis, M. N., and Veinot, A. F. 1987. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research* 12(2):262–268.

Moraes, R. O.; Mariño, J. R. H.; Lelis, L. H. S.; and Nascimento, M. A. 2018. Action abstractions for combinatorial multi-armed bandit tree search. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 74–80.

Ontañón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of Real-Time Strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in games* 5(4):1–19.

Ontañón, S. 2017. Combinatorial multi-armed bandits for RTS games. *Journal of Artificial Intelligence Research (JAIR)* 58:665–702.

Osborne, M. J., and Rubinstein, A. 1994. *A course in Game Theory*. The MIT Press.

Robbins, H. 1952. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society* 58(5):527–535.

Russo, D.; Roy, B. V.; Kazerouni, A.; Osband, I.; and Wen, Z. 2017. A tutorial on Thompson sampling. *arXiv:1707.02038*.

Scott, S. 2010. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry* 26:639–658.

Seldin, Y.; Szepesvári, C.; Auer, P.; and Abbasi-Yadkori, Y. 2012. Evaluation and analysis of the performance of the EXP3 algorithm in stochastic environments. *10th European Workshop on Reinforcement Learning (EWRL)* 103–116.

Shen, W.; Wang, J.; Jiang, Y.-G.; and Zha, H. 2015. Portfolio choices with orthogonal bandit learning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 974–980. AAAI Press.

Synnaeve, G., and Bessière, P. 2015. Multiscale Bayeisian modeling for RTS games: an application to StarCraft AI. *IEEE Transactions on Computational Intelligence and AI in Games* 8:338–350.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* (3/4):285–294.

Urteaga, I., and Wiggins, C. H. 2018. Bayesian bandits: balancing the exploration-exploitation tradeoff via double sampling. *arXiv:1709.03162*.

White, P. 1988. Restless bandits: activity allocation in a changing world. *Journal of Applied Probability* 287–298.

Yu, T., and Sra, S. 2019. Efficient policy learning for non-stationary MDPs under adversarial manipulation. *arXiv:1907.09350*.