Sequential Halving Using Scores

Nicolas Fabiano[1,2] **and Tristan Cazenave**[2]
[1] DI, ENS Paris, PSL, France
[2] LAMSADE, Université Paris-Dauphine, PSL, CNRS, Paris, France
nicolabiano22@yahoo.fr
Tristan.Cazenave@dauphine.psl.eu

Abstract

We study the multi-armed bandit problem, where the aim is to minimize the simple regret with a fixed budget. The Sequential Halving algorithm is known to tackle it efficiently. We present a more elaborate version of this algorithm to integrate some exterior knowledge or "scores", that can be provided for instance by a neural network of a heuristic like all-moves-as-first (AMAF). We provide both theoretical justifications and experiments.

Figure 1: The various notions of regret

With i^* the optimal arm and $\hat{i}$ the chosen one,

- Cumulative regret : $\mathcal{R}_{\text{cum}} = \sum_{r \text{ round}} (p_{i^*} - p_r)$
- Simple regret : $\mathcal{R} = p_{i^*} - p_{\hat{i}}$
- Simple regret 0-1 : $\mathcal{R}_{0-1} = 1$ if $i^* \neq \hat{i}$ else 0

Introduction

Since it was introduced in (Coulom 2006; Kocsis and Szepesvári 2006), the Monte Carlo Tree Seach (MCTS) algorithm has known a great success in AI, especially in turn-based games like Go or Chess, and some of its refinements are the state of the art for most games.

The general idea of this algorithm is the following: on the root configuration, pick a move, and generate a random playout from it. If the player to move wins, this means that the move was probably good, and if she loses it was probably bad. Then loop by picking more moves, deeper and deeper in the game tree, with a fixed amount of playout (or time) budget.

One of the keys for MCTS to be efficient is to choose what moves to investigate, with the usual exploration vs exploitation balance to find. To perform this, one typically uses the Upper Confidence Bound (UCB) bandit algorithm, which has good properties in terms of cumulative regret. This means that, for every investigated configuration, the moves tested were overall not bad.

However, in the context of games, the success of simulations does not really matter. The only aim is that, at the end, the algorithm outputs a move that is as good as possible. This means that, instead of cumulative regret, a more relevant quantification is the expected simple regret (see Fig. 1 for a precise definition).

In (Karnin, Koren, and Somekh 2013), a new bandit algorithm named Sequential Halving (SH) was introduced. It is proved to have a small expected simple regret 0-1, and it experimentally shows to also have a small expected simple regret. It has successfully been used as an alternative to UCB in MCTS, especially as a replacement in the root with UCB used in the rest of the tree (Pepels et al. 2014), in Partially Observable Games (Pepels, Cazenave, and Winands 2015) or even is the whole tree with SHOT (Cazenave 2015b).

However, for most games, the plain UCB is not state of the art. For many games like Go, moves typically commute, so the RAVE algorithm was introduced (Gelly and Silver 2011), which uses the all-moves-as-first (AMAF) heuristic (Bouzy and Helmstetter 2003). For some games also including Go (Silver et al. 2016), Neural Networks (NN) can provide the algorithm with reliable priors, which are incorporated in the PUCT algorithm (Silver et al. 2016).

The aim of this paper will be to incorporated exterior knowledge like AMAF or NN prior to the SH algorithm, and to compare the result both to plain SH and to the state of the art MCTS algorithms RAVE and PUCT.

The first part will discuss the SH algorithm in general, and report a few experiments. The second part will present a theoretical foundation for a new algorithm named SHUSS, Sequential Halving USing Scores. It will also discuss some variations around it, and report a few experiments.

The Sequential Halving algorithm

The SH algorithm is round-based. On every round, each arm is sampled the same amount of times, and then some fraction of the worst arms is removed, until there is only one arm left.

The theoretical bounds presented in (Karnin, Koren, and Somekh 2013) suggest that the same total budget should be spent for each round, and that the fraction removed should be constant on every step (denoted $1 - \lambda$). For a precise description, see Algorithm 1.

This version of the algorithm is slightly different from the original in two ways. First, a parameter λ is introduced, while it was fixed to 1/2. Second, the computation of the budget per round is improved, to ensure that less budget is left unspent in case of multiple roundings.

Note 1 *Contrary to other bandit algorithms like UCB, the fact that SH gives a lot of budget at once to one single arm has practical advantages like an easier parallelism and less back-and-forth in the search tree. This is especially true when there are few rounds (λ small).*

Algorithm 1 Sequential Halving

Parameter: cutting ratio λ
Input: total budget T, set of arms S
$S_0 \leftarrow S, T_0 \leftarrow T$
$R \leftarrow$ number of rounds before $|S_R| = 1$
for $r = 0$ **to** $R - 1$ **do**
  $t_r \leftarrow \lfloor \frac{T_r}{|S_r| \cdot (R-r)} \rfloor$
  $T_{r+1} \leftarrow T_r - t_r |S_r|$
  sample each arm in S_r t_r times
  $S_{r+1} \leftarrow S_r$ without the fraction $1-\lambda$ of the worst arms
end for
Output: arm in S_R

Restart vs stockpile

In (Karnin, Koren, and Somekh 2013), for the theoretical computations to be rigorous, one has to assume that rounds are independents, which means that statistics are discarded from one round to the other.

However, to gather more accurate statistics, it may be worth, instead of *restarting* at every round, to *stockpile* the statistics from the previous round. In terms of budget, this adds a factor almost $1/(1 - \lambda)$.

Note 2 *Getting the factor almost* $1/(1 - \lambda)$ *from the first rounds implies to twist the distribution of weight to give more at the beginning, but less at the end. Doing this will be referred to as* uniforming.

In theory, this may cause the following issue: if, on one round, a rather bad arm is lucky and has good stats, it will be stockpiled for the next round and cause it to be kept even further; while if we restarted it would be rare that an arm is lucky twice. This issue is especially important when λ is close to 1, as the stockpiled statistics form a huge part of the overall ones.

In addition to the two extremes of restart and stockpile, one can keep the statistics from the previous round, and give it a weight w between 1 (pure stockpile) and 0 (pure restart).

Experiments of the next section clearly show that stockpiling is always the best, even better than $w = 1 - \epsilon$.

Note 3 *We successfully replicated the SH part of the experiments of (Karnin, Koren, and Somekh 2013), and it appears that they must have been done using stockpiling, as restarting gives significantly worse results.*

Experiments

Even if we could be more general, we focus on the case where the only possible outcomes are 0 (loss) and 1 (win). Thus, every arms is described by its *value*, which is both the probability of win and the expected value.

The performance of bandit algorithms highly depend on the distribution of the arm's values. We consider 4 settings for the n arms.

In setting (1), the optimal arm has a value 0.5 and the $n-1$ others have a value 0.4.

In setting (A), the values form an arithmetic sequence from 0.5 to 0.25.

In setting (S), the optimal arm has a value 0.5, the worst has a value 0.25, and the others have values such that i/δ_i^2 is constant, with δ_i the difference with the optimal arm. This setting is suggested by the fact that the theoretical bounds of (Karnin, Koren, and Somekh 2013) rely on these values.

In setting (N), the values are distributed according to the sigmoid of a normal of parameters 0.5 and $\sigma^2 = 0.01$. This setting induces richer behaviours, and we believe it to be a more realistic model of the actual distributions in games.

The results are compared to UCB, the standard MCTS bandit. It consists in, for each step from 1 to the budget, picking the arm that maximises the empirical value, plus a term to force exploration of the form

$$c\sqrt{\frac{\log(\text{playouts})}{\text{playouts}_z}}$$

We tested various values for λ and w for SH, and compared it to various values for the exploration constant c in UCB. We also tested the uniforming variant discussed in Note 2. The results are shown in Fig. 2.

Roundings of the number of arms left are handled as follows: always round up, except when this would cause the amount of arms to remain constant (then round down instead).

Each result is averaged over 10000 tests. To reduce the covariance from one setting to another, the bandits are seeded using *numpy.random.binomial*. For the same number of experiment e and the same arm i, if the value of the arm i is the same in two settings, then on the same round r their results are drawn out of the same sequence of win/loss (the number of successes is monotonic in terms of budget).

As announced, in every setting, the best results are obtained for $w = 1$, so that in practice stockpiling is really stronger than restarting.

The optimal λ depends on the setting, but the experiments globally suggest that, for the interesting case $w = 1$, $\lambda \approx 0.7$ is often the best value. Actually, this is a very complex issue, and some less rigorous experiments suggest that it is better not to decrease like a geometric sequence but rather to start with large decreasing factors and to end with smaller ones.

Uniforming is significantly better in settings (1) and (S) for $n = 20$, but globally slightly worse in the others. We don't know how to explain this precise behaviour, but at least this suggests that there is room for practical improvement about how the budget is distributed among the rounds.

Surprisingly, the results are globally worse than UCB for $n = 20$, especially in the setting (S) while this is the one for which the SH algorithm is theoretically designed. Still, UCB relies more heavily on a fine-tuning of its parameter c, with no universally excellent value, and for $n = 80$ SH is globally better.

Scores

The aim of this part will be to develop a variant of the SH algorithm that makes advantage of some exterior knowledge, like a NN or AMAF statistics.

We will consider the general case where we have access to what we will call a *score*, which is a numerical evaluations of every move, independent from the bandit evaluation.

The bandits are still assumed to give either 0 or 1, giving an empirical mean $p_r^{(i)} \in [0, 1]$ for arm i on round r, but the scores do not necessarily belong to $[0, 1]$.

Theoretical model

We don't know precisely how to estimate the expected simple regret: the bounds provided in (Karnin, Koren, and Somekh 2013) are far from tight in practical cases and only describe the expected simple regret 0-1.

Still, it is clear that it will essentially depend on $P(p_r^{(i)} < p_r^{(j)})$: if any two arms are often properly ordered, then the best arms have a low probability to be among the worst $1-\lambda$ fraction.

Thus, our aim will be to find an optimal formula for some replacing $q_r^{(i)}$ which optimizes $P(q_r^{(i)} < q_r^{(j)})$.

Formally, let x and y (the value of the arms) be two hidden values that we want to compare, with $x - y = \delta$.

We have access to 4 independent variables. X and Y (the empirical means) are binomial with a same first parameter t and centered on respectively x and y. $\tilde{X}$ and $\tilde{Y}$ (the scores) are such that $\tilde{X} - \tilde{Y} = \tilde{\delta}$ is hopefully globally the same sign as δ.

In the following, z can stand for x, y, or any arm.

We make the assumption that $\tilde{\delta}$ is distributed following a normal law with parameters $\tilde{\delta}_0$ and $\tilde{\sigma}_0^2$. $\tilde{\delta}_0$ has the same sign as δ, and we even have $\tilde{\delta}_0 = \delta$ when the score is unbiased.

Optimal combination

As a particular case of the central limit theorem, we know that (for a more quantified statement, see for instance (Feller 2015)):

Lemma 1 *A binomial law of parameters t and p and a normal law of parameters tp and $tp(1-p)$ have almost the same distribution, provided that t is large.*

This means that $X - Y$ is (almost) distributed as a normal law of parameters $t\delta$ and $t\sigma^2 = t(x(1-x) + y(1-y))$, which up to normalisation can be seen as having parameters $\tilde{\delta}_0$ and $\frac{\tilde{\delta}_0^2 \sigma^2}{\delta^2 t}$.

Conversely, this shows that $\tilde{X} - \tilde{Y}$ gives (almost) the same information as two binomials, with the crucial first parameter

$$\tilde{t} = \frac{\tilde{\delta}_0^2 \sigma^2}{\delta^2 \tilde{\sigma}_0^2}$$

but with an intensity $\frac{\tilde{\delta}_0}{\delta}$ too large.

We define

$$\tilde{t}' = \frac{\tilde{\delta}_0 \sigma^2}{\delta \tilde{\sigma}_0^2}$$

We showed that the problem is (almost) equivalent to maximizing the probability to choose the best arm among two, knowing that one has succeeded $X + \tilde{t}'\tilde{X}$ times out of $t + \tilde{t}$ trials, and the second $Y + \tilde{t}'\tilde{Y}$ times.

Thus, it is optimal to use (for $\frac{\tilde{\delta}_0^2 \sigma^2}{\delta^2 \tilde{\sigma}_0^2}$ reasonably large)

$$q_z = Z + \tilde{t}'\tilde{Z}$$

A similar reasoning give the same result for t reasonably large.

One could be tempted to use the $\tilde{Z}$ to approximate σ. However, given the final goal is to sort all the arms on one single scale, $\tilde{t}'$ has to be the same for every pair of arms.

The simplest solution is to choose an hyperparameter $\tilde{t}'$ that corresponds to an overall reasonable guess. We will see how to improve it in some particular cases.

The resulting algorithm is presented as Algorithm 2.

In this algorithm, t_r^+ corresponds to the total budget used in $p_r^{(i)}$: $t_r^+ = t_r$ if we restart and $t_r^+ = t_0 + \cdots + t_r$ if we stockpile.

Algorithm 2 Sequential Halving USing Scores (SHUSS)

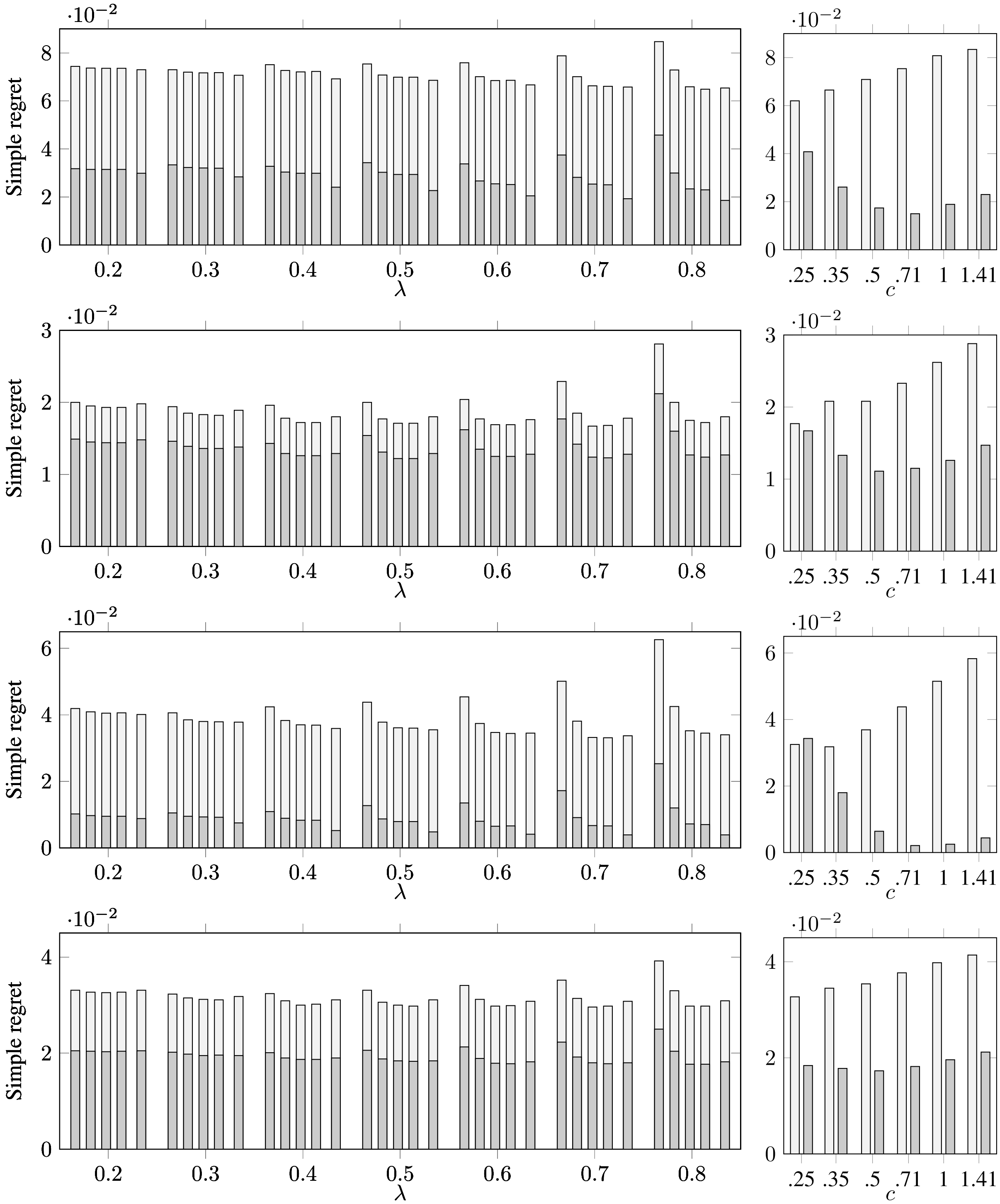
Parameter: cutting ratio λ, $\tilde{t}'$
Input: total budget T, set of arms S, online scores $\tilde{X}_r^{(i)}$
$S_0 \leftarrow S, T_0 \leftarrow T$
$R \leftarrow$ number of rounds before $|S_R| = 1$
for $r = 0$ **to** $R - 1$ **do**
  $t_r \leftarrow \lfloor \frac{T_r}{|S_r| \cdot (R-r)} \rfloor$
  $T_{r+1} \leftarrow T_r - t_r |S_r|$
  sample each arm in S_r t_r times, giving an empirical mean $p_r^{(i)}$ to arm i out of t_r^+ trials
  $q_r^{(i)} = p_r^{(i)} + \frac{\tilde{t}'}{t_r^+} \tilde{X}_r^{(i)}$
  $S_{r+1} \leftarrow S_r$ without the fraction $1-\lambda$ of the worst arms in terms of $q_r^{(i)}$
end for
Output: arm in S_R

Selection bias

One issue that may occur with this algorithm is that, after some round, the arms that remains have their $\tilde{Z}$ biased by the fact that they were among the best. Thus, even if at the first round they are indeed normal laws, it is unclear how they look like after a few rounds.

However, this issue is very similar to the issue of stockpiling, as all arms tend to have better stats than they should. The fact that stockpiling is so powerful suggest that this issue is not too important, so we will neglect it.

Figure 2: Simple regret obtained with SH in various settings. In every setting, the budget is taken equal to $T = 2048$. From top to bottom, we report settings (1), (A), (S) and (N). For each setting, the left plot corresponds to SH, and the right one corresponds to UCB. For SH, for each λ, the bars correspond (from left to right) to $w = 0$, $w = 0.5$, $w = 0.9$, $w = 1$, and $w = 1$ with uniforming. The darker bars correspond to $n = 20$, and the lighter ones to $n = 80$.



Case of AMAF: a better formula for $\tilde{t}'$

This subsection discusses the special case where the scores are given by AMAF statistics. It should be seen as a little toolbox of a few ideas that can be used to do better than taking $\tilde{t}'$ as a constant, based on a case study.

The AMAF (all-moves-as-first) score (Bouzy and Helmstetter 2003) consists to evaluate a move m for a player p in a configuration c, considering the win/loss ratio of every game where p plays m, not only in c itself but in any of its descendants in the game tree (or even its cousins, in some variants of AMAF like GRAVE (Cazenave 2015a)).

First of all, this score is not independent from the value of the bandits. In the first rounds of the algorithm, there are many bandits, so the AMAF scores are almost independent from each of them, which makes it not a big issue.

In the last rounds, however, it is highly correlated with the stats of some, if not all, bandits. In some games, one could imagine that some properties of the moves generate important biases, for instance if the move c can only appear after few of the remaining moves considered. We will see a general way to address this problem, but this could be more tricky for some particular games and we recommend cautiousness.

The main interesting thing about AMAF in this context is that the score is more and more accurate as simulations are performed. Thus, taking $\tilde{t}'$ as a constant throughout the algorithm may not be appropriate.

Instead, one can model the distribution of $\tilde{\delta}$ as follows:

- the fact that AMAF is a heuristic causes an error distributed a a normal law of variance σ^2_{heu}, centered somewhere between δ and the local average value;
- the fact that the AMAF stats are only gathered on a finite number s_r of moves on round r causes an error distributed as a binomial law, which is almost (see Lemma 1) and after normalization a centered normal law of variance $\sigma^2_{\text{stat}}/s_r$.

Provided that σ^2_{heu} is small (i.e. the heuristic makes sense in the application context), and the values of the arms are not too extreme, σ and σ_{stat} are almost equal.

We get

$$\tilde{t}'_r = \frac{\tilde{\delta}_0}{\delta}\frac{\sigma^2}{\sigma^2_{\text{heu}} + \sigma^2_{\text{stat}}/s_r} \approx \frac{\tilde{\delta}_0}{\delta}\frac{1}{\sigma^2_{\text{heu}}/\sigma^2 + 1/s_r}$$

This time, we have 2 hyperparameters to choose.

$\frac{\tilde{\delta}_0}{\delta}$ describes how much AMAF flattens the stats, and can easily be measured experimentally. It may be relevant to make it depend on the number of arms left and on the variant of AMAF used.

$\sigma_{\text{heu}}/\sigma$ describes how accurate the heuristic is, compared to the accuracy provided by binomial stats. Taking this hyperparameter as a relatively high value also ensures that, in the last rounds where s_r is large, the value of $\tilde{t}'_r$ stops increasing, which addresses the previously mentioned issue of correlation.

Note that this reasoning works only if, on each round, s_r is globally the same for every arm (or if, for every arm, $1/s_r \ll \sigma^2_{\text{heu}}/\sigma^2$), as we need a common value of $\tilde{t}'_r$.

Case of prior score: pruning

In this subsection, we assume that the $\tilde{Z}$ are known a priori (before any budget is attributed), for instance using a NN. This can be applied to some extent in cases where some score is known a priori but is refined during the algorithm, like GRAVE.

In the following, the arms are sorted by the value of $\tilde{Z}$, with $\tilde{X}_0$ the largest and $\tilde{X}_{n-1}$ the smallest.

Even before the algorithm begins, some arms have no chance of being chosen at the end, if for instance $\tilde{Z}$ is smaller than the median (for $\lambda = 1/2$) minus $1/\tilde{t}'_0$.

In addition to these trivial pruning, it is often worth to prune some more arms, as the budget saved will compensate for the risk taken.

As we saw in a previous section, the prior can be interpreted as if we have already spent some amount $\tilde{t}$ of budget on each arm before round 0, which we will consider as a round number -1.

The philosophy of SH (exploited in the performance proof in (Karnin, Koren, and Somekh 2013)) is that, when bandits are pruned up to number n_r with a budget t_r, the product $\pi_r := n_r \cdot t_r$ is equal to some π that does not depend on r.

Thus, it is natural to prune up to arm n_{-1}, where n_{-1} is chosen so that $\pi_{-1} = \pi$.

For a precise computation, we neglect the rounding issues when dividing by λ. We also make the computations as if we were not stockpiling (note that using the score on the further rounds can be seen as stockpiling when it is purely a prior).

Then

$$\pi_{-1} = n_{-1} \cdot \tilde{t}$$

$$\pi = \pi_0 = \lambda n_{-1} \cdot \frac{T}{\log_{1/\lambda}(n_{-1}) \cdot n_{-1}}$$

$$n_{-1} \log_{1/\lambda}(n_{-1}) = \frac{\lambda T}{\tilde{t}}$$

Experiments

We chose to test SHUSS using the score AMAF, to compare it with RAVE (Gelly and Silver 2007, 2011). The latter uses the AMAF score as follows: the value of the arm, which the exploration term is added to, is taken equal to

$$(1 - \beta_z)Z + \beta_z\tilde{Z}$$

with t_z the number of playouts starting with z, s_z the number of playouts containing z and

$$\beta_z = \frac{s_z}{s_z + t_z + bias \times s_z \times t_z}$$

(Pepels et al. 2014) demonstrates how to combine the SH algorithm with UCT in the Hybrid-MCTS algorithm: SH is used only at the root, and the rest of the tree expansion uses UCB. We followed this idea, by combining SHUSS at the

Table 1: Comparison of Hybrid-SHUSS with AMAF score against RAVE.

Game	0	128	256	512	1024	2048	4096	8192	16384	∞
Atarigo 7x7	44.2	47.2	49.6	**50.2**	50.0	49.6	45.2	47.8	46.4	45.2
Atarigo 9x9	35.6	41.4	40.0	38.2	41.0	41.2	**43.4**	41.4	36.4	40.2
Ataxx 8x8	30.2	33.6	35.2	34.2	42.0	46.2	55.0	62.4	62.0	**71.8**
Breakthrough 8x8	54.0	**57.8**	56.8	56.0	56.6	55.2	53.8	51.0	55.0	52.4
Domineering 8x8	41.4	47.8	44.8	**49.0**	46.2	47.2	46.2	45.6	43.0	42.4
Go 7x7	45.2	49.2	46.2	53.8	**58.6**	50.2	42.6	33.2	31.0	15.8
Go 9x9	43.4	53.2	**58.2**	52.2	50.8	43.8	35.6	26.4	19.0	12.2
Hex 11x11	15.8	43.0	43.4	**51.4**	48.4	50.2	46.4	46.6	43.4	42.6
Knightthrough 8x8	61.0	61.6	**65.0**	63.8	62.2	60.2	54.2	54.4	56.2	52.8
NoAtaxx 8x8	**91.0**	87.4	76.8	72.0	62.8	55.2	53.8	44.6	45.8	43.2
NoBreakthrough 8x8	37.8	40.8	44.0	46.2	**51.4**	44.2	46.4	44.0	50.0	46.6
NoDomineering 8x8	40.4	45.6	49.4	46.0	48.4	**50.0**	47.6	47.4	45.0	47.6
NoGo 7x7	38.8	40.8	45.6	44.0	50.8	47.6	50.8	49.4	47.6	**51.8**
NoGo 9x9	30.0	37.8	38.8	40.0	41.0	42.0	42.8	45.0	**45.8**	37.4
NoHex 11x11	46.4	48.0	48.6	49.0	**49.2**	48.6	48.6	49.2	48.8	49.2
NoKnightthrough 8x8	29.0	36.8	38.8	39.6	47.8	46.2	46.0	45.2	**48.2**	47.6

root with RAVE for the rest of the tree, in an algorithm naturally named Hybrid-SHUSS.

Table 1 reports the results of 500 matches (250 as White and 250 as Black) between Hybrid-SHUSS and RAVE, both with 10000 playouts per move, for many classical games.

RAVE uses the classical parameter $bias = 10^{-7}$, both in the inner parts of Hybrid-SHUSS and its opponent. SHUSS uses the classical parameter $\lambda = 1/2$. Different values of $\tilde{t}'$ are experimented (to keep things simple, $\tilde{t}'$ is a constant).

The extreme case $\tilde{t}' = 0$ is the usual SH algorithm without AMAF (it is only used to break ties), and $\tilde{t}' = \infty$ is relying purely on AMAF, with the same weight whether or not the move is first.

In most games, SHUSS performs better than both pure SH and pure AMAF.

For some games, even with a tuned $\tilde{t}'$, Hybrid-SHUSS is a bit worse than RAVE. For some others, Hybrid-SHUSS outperforms RAVE, sometimes even for a wide scale of $\tilde{t}'$. We didn't find any general property to heuristically guess in which category a given game is.

Conclusion

In the first section, we have discussed the SH algorithm in general.

We discussed the difference between stockpile and restart, and stockpiling is experimentally way better.

We also showed that a parameter $\lambda \approx 0.7$ is apparently better than the classical $\lambda = 0.5$. Still, it appears that some more flexible budget attribution or cuts may be better.

In the second section, we presented our new algorithm Sequential Halving USing Scores (SHUSS).

A theoretical model suggests a very simple way to combine the score with the bandit statistics, but there is plenty of room for improvement depending on the exact nature of the score.

Work still has to be done to handle scores that are very asymmetrical among the arms (for instance, if we have plenty of AMAF data for one move but very few for another).

In addition, SHUSS requires the scores to be linked to the statistics given by MCTS, while currently most neural networks predicts the probability that each move is chosen. This implies to either post-treat the output of the neural network, to do a one ply search to get the scores associated to moves or ideally to use a whole other training pipeline.

Acknowledgment

This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

References

Bouzy, B.; and Helmstetter, B. 2003. Monte-Carlo Go Developments. In *ACG*, volume 263 of *IFIP*, 159–174. Kluwer.

Cazenave, T. 2015a. Generalized rapid action value estimation. In *24th International Conference on Artificial Intelligence*, 754–760.

Cazenave, T. 2015b. Sequential Halving Applied to Trees. *IEEE Trans. Comput. Intell. AI Games* 7(1): 102–105.

Coulom, R. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, 72–83.

Feller, W. 2015. On the normal approximation to the binomial distribution. In *Selected Papers I*, 655–665. Springer.

Gelly, S.; and Silver, D. 2007. Combining online and offline knowledge in UCT. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007),* *Corvallis, Oregon, USA, June 20-24, 2007*, 273–280. doi: 10.1145/1273496.1273531.

Gelly, S.; and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.* 175(11): 1856–1875.

Karnin, Z. S.; Koren, T.; and Somekh, O. 2013. Almost Optimal Exploration in Multi-Armed Bandits. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, 1238–1246.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, 282–293.

Pepels, T.; Cazenave, T.; and Winands, M. H. 2015. Sequential halving for partially observable games. In *CGW 2015*, 16–29. Springer.

Pepels, T.; Cazenave, T.; Winands, M. H. M.; and Lanctot, M. 2014. Minimizing Simple and Cumulative Regret in Monte-Carlo Tree Search. In *CGW 2014*, 1–15. Springer.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529(7587): 484–489. ISSN 0028-0836. doi:10.1038/nature16961.