Sequential Halving Using Scores

Nicolas Fabiano[1] and Tristan Cazenave[2]

[1] DI ENS, ENS Paris
[2] LAMSADE, Université Paris-Dauphine, CNRS, Paris ; PSL, 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 for instance be provided by a neural network or a heuristic such as all-moves-as-first (AMAF) in the context of a Monte-Carlo Tree Search. We provide both theoretical justifications and experiments.

1 Introduction

Since it was introduced in [6, 11], the Monte Carlo Tree Seach (MCTS) algorithm has known a great success in AI, especially in turn-based games like Go, and some of its refinements are state of the art for most games.

The general idea of this algorithm is the following: from the root configuration, it picks a move, and generates a random playout from it. If the player to move wins, this means that the move was probably good, and if they lose, it was probably bad. Then the algorithm continues by picking more moves, deeper and deeper in the game tree, respecting a fixed amount of playout (or time) budget.

One of the key elements for MCTS to be efficient is the choice of what moves to investigate, with the usual search for the optimal exploration-exploitation trade-off. 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 mostly good ones.

However, in the context of games, the success of simulations does not matter in itself. The only goal is that the final output of the algorithm is as good a move 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 [10], a new bandit algorithm named Sequential Halving (SH) was introduced. It is proven to have a small expected simple regret 0-1 (see Fig. 1) and is also shown to have a small expected simple regret through numerical experimentation. It has successfully been used as an alternative to UCB in MCTS, in particular as a replacement in the root node with UCB used in the rest of the tree [12], in Partially Observable Games [13] or even in the whole tree with SHOT [3].

However, for most games, the unmodified UCB is not state of the art. For many games such as Go, moves typically commute, so the RAVE algorithm, which uses the all-moves-as-first (AMAF) heuristic [1], was introduced [9]. For some games, once again including Go [14], Neural Networks (NN) can provide an algorithm with reliable priors, which are incorporated in the PUCT algorithm [14].

The aim of this paper will be to incorporate exterior knowledge like AMAF or NN to the SH algorithm, and to compare the result both to the simple 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 experiments in a theoretical setup. 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 experiments on games.

Fig. 1: The various notions of regret

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

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

2 The Sequential Halving Algorithm

The SH algorithm is round-based. For every round, each arm is sampled the same amount of times, and only a set fraction of the best arms are kept. This step is repeated until there is only one arm left.

The theoretical bounds presented in [10] suggest that the same total budget should be spent for each round, and that the fraction removed should be constant for every step (denoted $1 - \lambda$). For a precise description, see Algorithm 1.

This version of the algorithm differs from the original one in two ways. First, by the introduction of the parameter λ, which allows for values other than the original 1/2. Second, the computation of the budget per round is slightly improved, to ensure that less budget is left unspent in case of multiple issues with rounding.

Note 1. Contrary to other bandit algorithms like UCB, SH assigns a lot of the budget at once to each arm, which has practical advantages like simpler parallelisation and less back-and-forth in the search tree. This is especially true when λ is small (few rounds).

2.1 Restart vs Stockpile

In [10], for the theoretical computations to be rigorous, one has to assume that rounds are independent, which means that statistics are discarded from one round to the other.

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

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

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 t_r times each arm in S_r
  $S_{r+1} \leftarrow S_r$ deprived of the fraction $1 - \lambda$ of the worst arms
end for
Output: arm in S_R

In theory, this may cause the following issue: if, for one round, a rather bad arm is sampled disproportionately, these statistics will be stockpiled for the next round, which will cause it to be kept even further; whereas restarting would decrease the probability for that bad arm to be chosen, as it would have to be wrongly selected twice. This issue is particularly important when λ is close to 1, as the stockpiled statistics contribute significantly to the overall ones in that case.

A compromise can be found between the two pure approaches, as one can keep the statistics from the previous round and give it a decaying factor d between 0 (pure stockpile) and 1 (pure restart).

The experiments of the next section clearly show that stockpiling is always better, even more so than choosing $0 < d \ll 1$.

Note 3. We successfully replicated the SH part of the experiments of [10], and it would appear that they were done using stockpiling, as restarting gives significantly worse results.

2.2 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 to win and the expected value.

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

In setting (1), the optimal arm has a value of 0.5 and the others have a value of 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 of 0.5, the worst has a value of 0.25, and the others have values such that i/δ_i^2 is constant, with δ_i the difference in value with the optimal arm. This setting is suggested by the fact that the theoretical bounds of [10] rely on these values, and thus the theoretical guaranty is the strongest.

In setting (N), the values are distributed according to the sigmoid of a normal distribution with 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 of, for each step from 1 to the budget, picking the arm that maximises the empirical value, added to a term to force exploration, of the form

$$c\sqrt{\frac{\log(\text{playouts})}{\text{playouts}_z}} \tag{1}$$

We tested various values for λ and d 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.

Rounding the number of arms left is handled as follows: always round up, except when this would cause the amount of arms to remain constant, in which case round down.

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 index of experiment e and the same arm i, if the value of 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 monotonous in terms of budget).

As announced, in every setting, the best results are obtained for $d = 0$, showing that in practice, stockpiling is more efficient than restarting.

The optimal λ depends on the setting. The experiments globally suggest that, for the interesting case $d = 0$, $\lambda \approx 0.7$ is often the best value, but the difference is small and the algorithm performs well on a wide range of λ that includes the classical value $\lambda = 0.5$. That problem is actually very complex, and some less rigorous experiments suggest that it is better not to decrease geometrically but rather to start with large decreasing factors and to end with smaller ones (eg $20 \to 8 \to 4 \to 3 \to 2 \to 1$ rather than $20 \to 10 \to 5 \to 3 \to 2 \to 1$).

The effect of uniforming is mixed, which suggests that there is room for practical improvement concerning the way the budget is distributed among the rounds.

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

3 Scores

The aim of this part will be to develop a variant of the SH algorithm that takes 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 evaluation 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 $[0, 1]$.

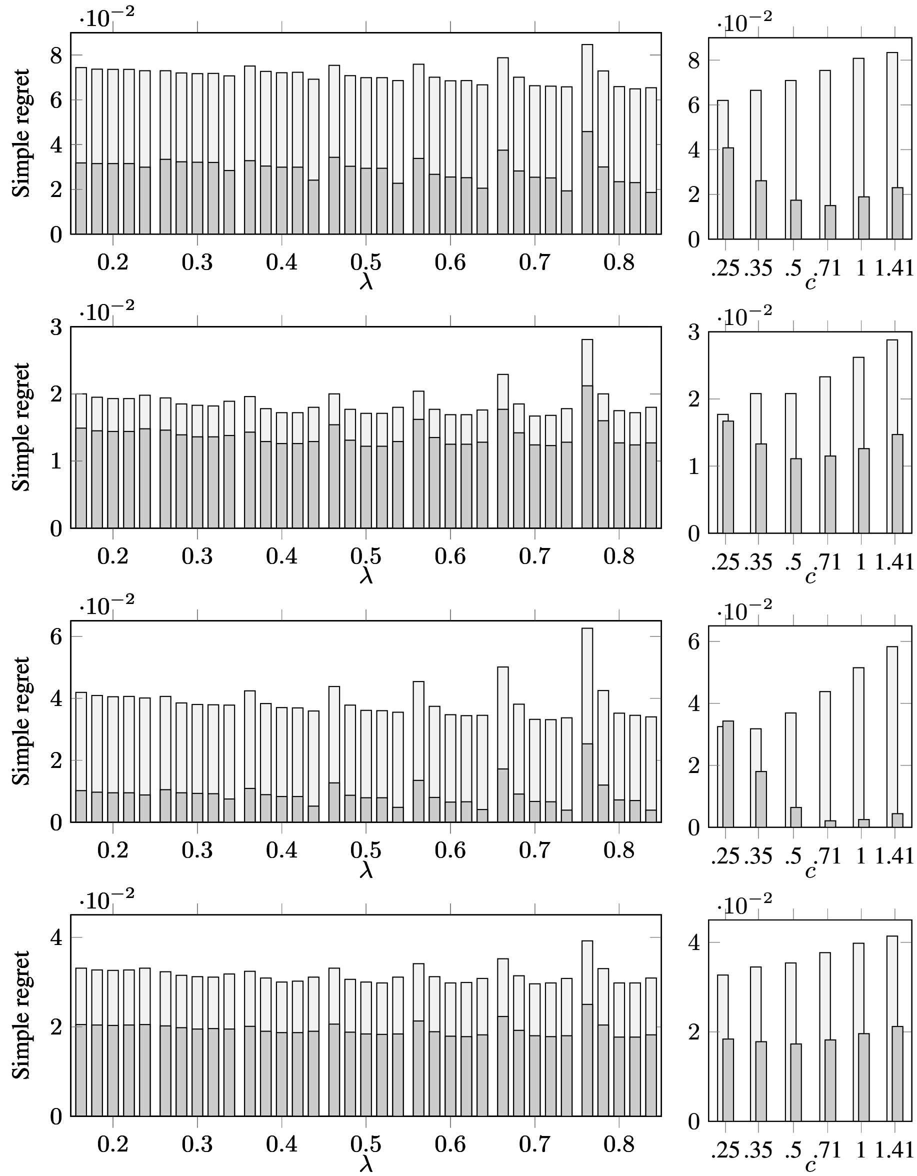

Fig. 2: Simple regret obtained with SH in various settings. In every setting, the budget is taken equal to $T = 2048$. From top to bottom, we report settings (1), (A), (S) and (N). For each setting, the left plot corresponds to SH, and the right one corresponds to UCB. For SH, for each λ, the bars correspond (from left to right) to $d = 1$, $d = 0.5$, $d = 0.1$, $d = 0$, and $d = 0$ with uniforming. The darker bars correspond to $n = 20$, and the lighter ones to $n = 80$.

3.1 Theoretical Model

We don't know precisely how to estimate the expected simple regret: the bounds provided in [10] are far from tight in practical cases and only describe the expected simple regret 0-1. Still, it will globally 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 $q_r^{(i)}$ which optimizes $P(q_r^{(i)} < q_r^{(j)})$ to use instead.

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 number of 1s obtained) are binomials with a same first parameter t and centered on respectively tx and ty. $\tilde{X}$ and $\tilde{Y}$ (the scores, eg the AMAF statistics) 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. This is not the case for NN, but we will see how to handle this in Section 3.5.

3.2 Optimal Combination

As a particular case of the central limit theorem, we know that (for a more quantified statement, see for instance [7]):

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 $\tilde{t}$ such that $\tilde{\sigma}_0^2 = \frac{\tilde{\delta}_0^2\sigma^2}{\delta^2 \tilde{t}}$. This gives

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

but with an intensity $\frac{\tilde{\delta}_0}{\delta}$ that is too large. We define

$$\tilde{t}' = \frac{\tilde{\delta}_0\sigma^2}{\delta\tilde{\sigma}_0^2} \tag{3}$$

We showed that the problem is (almost) equivalent to maximizing the probability of choosing 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)

$$q_z = Z + \tilde{t}'\tilde{Z} \tag{4}$$

Similar reasoning gives the same result for t reasonably large.

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

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

The resulting algorithm is presented as Algorithm 2. In it, t_r^+ corresponds to the total budget used in $p_r^{(i)}$: $t_r^+ = t_r$ with restart and $t_r^+ = t_0 + \cdots + t_r$ with 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 t_r times each arm in S_r, giving an empirical mean $p_r^{(i)}$ to arm i out of t_r^+ trials
  $q_r^{(i)} = p_r^{(i)} + \frac{\tilde{t}'}{t_r^+}\tilde{X}_r^{(i)}$
  $S_{r+1} \leftarrow S_r$ deprived of the fraction $1 - \lambda$ of the worst arms in terms of $q_r^{(i)}$
end for
Output: arm in S_R

3.3 Selection Bias

One issue that may occur is that, after any given round, the arms that remain have their $\tilde{Z}$ biased by the fact that they were among the best. Thus, even if during 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 theoretically should. The fact that stockpiling is so powerful suggest that this issue is not too important, so we will neglect it.

3.4 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 small toolbox consisting 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 [1] consists in evaluating a move m for a player p in a game state s, considering the win/loss ratio of every game where p plays

m, not only in s itself but in any of its descendants in the game tree (or even its cousins, in some variants of AMAF like GRAVE [2]).

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 a mostly unimportant 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 m 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 caution.

The most interesting aspect about AMAF in this context is that the score becomes more and more accurate as evaluations are performed. Thus, taking $\tilde{t}'$ as a constant throughout the algorithm is not adequate. Instead, one can model the distribution of $\tilde{\delta}$ as based on the following:

- the fact that AMAF is a heuristic causes an error distributed as 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.

Equation 3 applied with this variance gives

$$\tilde{t}'_r = \frac{\tilde{\delta}_0}{\delta} \frac{\sigma^2}{\sigma^2_{\text{heu}} + \sigma^2_{\text{stat}}/s_r} \approx \frac{\tilde{\delta}_0}{\delta} \frac{1}{\sigma^2_{\text{heu}}/\sigma^2 + 1/s_r} \tag{5}$$

This time, there are 2 hyperparameters to choose values for.

$\frac{\tilde{\delta}_0}{\delta}$ 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. Giving this hyperparameter 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 $\tilde{t}'_r$.

3.5 Case of Prior Score: Pruning

In this subsection, we assume that the $\tilde{X}_i$ are known a priori (before any evaluation is performed). This can be applied to some extent in cases where some score is known a priori but refined during the algorithm, like GRAVE. We start with a general discussion, before dealing with the specific case of neural networks (NN) applied to MCTS.

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

In addition to these trivial pruning operations, it is often worth pruning 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 though we had already spent some amount $\tilde{t}$ of budget on each arm before round 0, which we will consider to be a round labelled -1. The philosophy of SH (exploited in the performance proof in [10]) 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 subsequent rounds can be seen as stockpiling when it is purely a prior).

Then

$$\pi_{-1} = n_{-1} \cdot \tilde{t} \tag{6}$$

$$\pi = \pi_0 = \lambda n_{-1} \cdot \frac{T}{\log_{1/\lambda}(n_{-1}) \cdot n_{-1}} \tag{7}$$

$$n_{-1} \log_{1/\lambda}(n_{-1}) = \frac{\lambda T}{\tilde{t}} \tag{8}$$

For NN in MCTS, all the previous theoretical foundation has to be slightly adapted, given that bandits don't give 0 or 1 but the value of the leaf evaluated by the NN instead.

More importantly, the score is given by the policy of the root. It is meant to be monotonic with the value, but the way it uses a softmax layer makes the rest of our model about its distribution fail. Thus, the safest way to use it is for pruning, and then the remaining arms are explored using a basic SH that does not use the policy.

The previous formula for n_{-1} should hold for the same reasons. Now, the value of $\tilde{t}$ describes how much budget is needed for the exploration to be as good as the policy. As the budget is typically distributed among the rest of the tree by an algorithm like PUCT, designed to be good asymptotically but not for small values, $\tilde{t}$ is typically quite large. In addition, given that the policy is not stockpiled, it is better to overestimate the value of $\tilde{t}$ to make use of the policy as much as possible.

3.6 Experiments with AMAF

First, we test SHUSS using the score AMAF, to compare it with RAVE [8, 9].

The latter uses the AMAF score as follows: the value of the arm, to which the exploration term is added, is taken equal to

$$(1 - \beta_z) Z + \beta_z \tilde{Z} \tag{9}$$

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

$$\beta_z = \frac{s_z}{s_z + t_z + bias \times s_z \times t_z} \tag{10}$$

Table 1: Percentage of games won by Hybrid-SHUSS (using AMAF and RAVE) against RAVE, in various games.
$T = 10000$; $bias = 10^{-7}$; $\lambda = 1/2$; 500 matches.

Game $\setminus \tilde{t}'$	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
Average	42.76	48.25	48.83	49.10	**50.45**	48.60	47.40	45.85	45.23	43.68

Table 2: Percentage of games won by Hybrid-SHUSS (using a NN and PUCT) against PUCT, for the game of Go.
$c = 0.2$; $\lambda = 1/2$; 400 matches.

$T \setminus n_{-1}$	3	4	5	6	7	8	9
32	31.00	**46.00**	43.50	26.50	20.50		
64	57.75	60.00	57.00	**71.50**	38.75		
128	39.75	46.50	**54.50**	39.75	41.25		
256				55.25	**71.50**	60.75	
512				25.50	**60.00**	47.25	
1024					60.75	**67.75**	55.50

[12] 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 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, for many classical games. Both algorithms use a budget (number of playouts) per move equal to 10000. 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 with (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 regardless of whether or not the move is first.

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

The optimal value of $\tilde{t}'$ depends on the game, but using 1024 gives a reasonably good performance for every game with this budget.

3.7 Experiments with a Neural Network

We then test SHUSS with a prior given by a NN in the game of Go.

The state of the art NNs in the game of Go use two heads, one for the policy and one for the value. The MCTS algorithm used in current computer Go programs since AlphaGo is PUCT. It uses the NN score as follows : the exploration term is replaced by

$$c \times \tilde{Z} \times \frac{\sqrt{t}}{1 + t_z} \tag{11}$$

with $\tilde{Z}$ the policy, t the budget already used for this node and t_z the budget already used for this node on move z.

We use as a NN a simple MobileNet of 16 blocks, a trunk of 64 and 384 planes in the bottleneck block [4, 5]. MobileNets give better results than usual residual networks for the game of Go.

As explained in section 3.5, in SHUSS, the policy is used to prune at the root, and the remaining algorithm is SH at the root and PUCT for the remaining of the tree, in a similar Hybrid fashion.

Experiments showed that PUCT performs best against Hybrid-SHUSS for $c = 0.2$. Table 2 report the results of 400 matches of Hybrid-SHUSS against PUCT for various budgets and n_{-1}, and we see that with the Hybrid-SHUSS outperforms PUCT for large enough budgets.

Concerning the relationship between T and the optimal n_{-1}, it seems to be logarithmic, while it was theoretically expected to be closer to being linear. Regardless, the size of the game of Go forced us to stick to rather small budgets, for which this part of the theory may not apply yet as it is asymptotic.

4 Conclusion

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

We discussed two ways of using the budget, restarting and stockpiling, with the latter being much better experimentally.

We also showed that a cutting parameter $\lambda \approx 0.7$ for SH is experimentally slightly better than the classical $\lambda = 0.5$, but that globally the algorithm is very robust for a wide range of λ. Nonetheless, 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, while still leaving plenty of room for improvement depending on the precise nature of the score.

Work still remains to be done to handle scores that are very asymmetrical among the arms in terms of quality.

We associated SHUSS with AMAF statistics and RAVE under the root play in a variety of different games against RAVE with the same parameters. The results are mixed, and depend on the game.

We also made SH, pruning using the policy with PUCT under the root node, play Go against PUCT with the same parameters. SH with pruning outperforms PUCT for well chosen numbers of moves kept, but it is quite sensitive to this value and it is unclear how to choose it in general.

Acknowledgment

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

References

1. Bouzy, B., Helmstetter, B.: Monte-Carlo Go developments. In: ACG. pp. 159–174 (2003)
2. Cazenave, T.: Generalized rapid action value estimation. In: IJCAI. pp. 754–760 (2015)
3. Cazenave, T.: Sequential halving applied to trees. IEEE TCIAIG **7**(1), 102–105 (2015)
4. Cazenave, T.: Improving model and search for computer Go. In: IEEE CoG (2021)
5. Cazenave, T.: Mobile networks for computer Go. IEEE Transactions on Games (2021)
6. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo Tree Search. In: Computers and Games. pp. 72–83 (2006)
7. Feller, W.: On the normal approximation to the binomial distribution. In: Selected Papers I, pp. 655–665. Springer (2015)
8. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: ICML. pp. 273–280 (2007)
9. Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go. Artif. Intell. **175**(11), 1856–1875 (2011)
10. Karnin, Z.S., Koren, T., Somekh, O.: Almost optimal exploration in multi-armed bandits. In: ICML. pp. 1238–1246 (2013)
11. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: ECML. pp. 282–293 (2006)
12. Pepels, T., Cazenave, T., Winands, M.H.M., Lanctot, M.: Minimizing simple and cumulative regret in monte-carlo tree search. In: CGW 2014. pp. 1–15. Springer (2014)
13. Pepels, T., Cazenave, T., Winands, M.H.: Sequential halving for partially observable games. In: CGW 2015, pp. 16–29. Springer (2015)
14. 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., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (jan 2016)