The Beta Distribution in the UCB Algorithm Applied to Monte-Carlo Go

John Stogin
Undergraduate '11,
Princeton University,
Princeton, New Jersey. USA
jstogin@princeton.edu

Yung-Pin Chen
Associate Professor of Statistics,
Lewis and Clark College,
Portland, Oregon. USA
ychen@lclark.edu

Peter Drake
Associate Professor of Computer Science,
Lewis and Clark College,
Portland, Oregon. USA
drake@lclark.edu

Seth Pellegrino
Undergraduate '09,
Lewis and Clark College,
Portland, Oregon. USA
sethp@lclark.edu

January 5, 2009

Abstract

Problems in Artificial Intellegence sometimes require a computer to make decisions based on information from past experience. In such a case, computational effort must be balanced between exploring to gain more information and taking advantage of information gained so far. This paper addresses a formal description of such a problem, known as the K-Armed Bandit problem, as well as a modern policy for dealing with it. This paper also introduces an improvement to this modern policy and an application of the improvement to the Asian game of Go.

Keywords: UCT, UCB, Monte-Carlo, Computer Go, K-Armed Bandit Problem, Exploration vs. Exploitation

1 Introduction

1.1 Go in brief

The game of Go is a grand challenge of artificial intellegence [3]. It is a deterministic, zero-sum game of perfect information. Compared to other popular strategy games, such as Chess and Checkers, there are far more possible board configurations in Go. For an explanation of the rules of Go, see Baker [2]. Expert humans play better than any programs developed so far.

Today's best algorithms consider various moves and evaluate the resulting board configurations using a Monte-Carlo approach. That is, the computer plays random moves to the end of the game and counts the score. After many such random playouts, the computer chooses the move that has the highest probability of resulting in a victory.

With limited time for each turn, a computer would not want to waste its time exploring moves that are not likely to be very helpful. On the other hand, restricting explorations to the seemingly best move would risk missing an even better move that hasn't been explored enough to reveal its potential. Thus, we are faced with a problem of exploration (sampling the less explored moves) versus exploitation (sampling the moves that are more likely to be optimal).

1.2 K-Armed Bandit Problem

This paper is mainly concerned with the well-known and simpler K-Armed Bandit problem, which has similarities to the problem mentioned above. A one-armed bandit is a slot machine with some (constant) probability of yielding a win each time the arm is pulled. A K-armed bandit is a set of machines each with a probability of yielding a win when its arm is pulled. The goal of the K-Armed Bandit problem is to play machines so as to maximize total reward after a period of time.

From now on, we will use the following terms:

K is the number of machines available from which to play.

M_i is the ith machine.

m_i is the probability that playing M_i will result in a win. (This quantity is unknown to the player.)

P_i is the number of times, at a given moment, M_i has been played.

R_i is the total reward, at a given moment, from M_i, or the number of wins observed so far.

$P_T = \sum_{i=1}^{K} P_i$

1.3 UCB1-Tuned

One of the most popular strategies is the UCB1-Tuned strategy defined by Auer et al [1].

Initially, play each machine once.
Play machine M_i that maximizes the following expression:

$$\frac{R_i}{P_i} + \sqrt{\frac{\ln(P_T)}{P_i} \min\left(\frac{1}{4}, V_i\right)} \tag{1.1}$$

and

$$V_i = \frac{R_i}{P_i} - \left(\frac{R_i}{P_i}\right)^2 + \sqrt{\frac{2\ln(P_T)}{P_i}} \tag{1.2}$$

We can express (1.1) and (1.2) in the following form:

$$\frac{R_i}{P_i} + \sqrt{\ln(P_T) \min\left(\frac{1}{4P_i}, V_i\right)} \text{ and } V_i = \frac{R_i\,(P_i - R_i)}{P_i^3} + \frac{\sqrt{\frac{2\ln P_T}{P_i}}}{P_i} \tag{1.3}$$

The first term of (1.3) is the expected value of M_i. The second term is a confidence interval that grows slowly as M_i is ignored (resulting from $\ln(P_T)$). In the latter expression, V_i is an estimate of the variance of data for M_i that likewise grows slowly as M_i is ignored.

2 Theory

2.1 Introduction of Beta Distribution

For this specific problem, we believe that the use of a different model of variance would lead to more optimal results. Unlike UCB1-Tuned, we use a new variance based on a non-Gaussian distribution. When P_i is low, a Gaussian distribution is a poor approximation, and its variance is zero until $0 < R_i < P_i$. Appendix A has a couple plots of the Gaussian distribution and the Beta distribution (defined below) for a machine where $m_i = 2/3$. Notice that unlike the Gaussian distribution, the Beta distribution never expects m_i to possibly lie outside of the interval $[0, 1]$ and is always defined for any non-negative P_i and R_i.

Definition: The Euler Beta function is defined to be
$\beta(a, b) = \int_0^1 x^{a-1}(1-x)^{b-1}dx$.

Definition: A Beta Distribution with parameters a and b is defined on [0,1] to be $\frac{1}{\beta(a,b)}x^a(1-x)^b$. On $\mathbb{R} \setminus [0,1]$ the Beta Distribution has value 0.

Theorem 1: If m_i is equally likely to be any probability before any pulls of M_i, the probability density of m_i follows the Beta Probability Distribution where $a = R_i + 1$ and $b = P_i - R_i + 1$.

Proof: Given an event $\mathcal{E}$ and a hypothesis $\mathcal{H}$, the probability that $\mathcal{H}$ is true provided $\mathcal{E}$ has taken place is given by

$$\Pr[\mathcal{H}|\mathcal{E}] = \frac{\Pr[\mathcal{E}|\mathcal{H}]\Pr[\mathcal{H}]}{\Pr[\mathcal{E}]}$$

Let $p \in [0,1]$ be a possible value of the probability that M_i yields a win and let $\delta/2 > 0$ be a small radius about p. We seek the probability of the event $\mathcal{H}$ that m_i is within $\delta/2$ of p.

$$\Pr[\mathcal{H}] = \Pr[|m_i - p| < \delta/2] = \delta$$
$$\Pr[\mathcal{E}|\mathcal{H}] = p^{R_i}(1-p)^{P_i - R_i}$$
$$\Pr[\mathcal{E}] = \int_0^1 \Pr[m_i = x]x^{R_i}(1-x)^{P_i-R_i} = \int_0^1 dx \cdot x^{R_i}(1-x)^{P_i-R_i} = \beta(R_i{+}1, P_i{-}R_i{+}1)$$
$$\Pr[\mathcal{H}|\mathcal{E}] = \frac{\delta \cdot p^{R_i}(1-p)^{P_i-R_i}}{\beta(R_i+1, P_i-R_i+1)}$$

Dividing by δ yields the appropriate Beta Distribution and completes the proof. □

2.2 New Player Formula

At this point, we introduce our suggestion for a K-armed bandit player, which is based primarily on a replacement of the original variance with the variance of a beta distribution, which is $V = \frac{ab}{(a+b)^2(a+b+1)}$.

Initially, play each machine once.
Play machine M_i that maximizes the following expression:

$$\frac{R_i}{P_i} + \sqrt{\ln(P_T)V_i} \tag{2.1}$$

and

$$V_i = \frac{(R_i+1)(P_i-R_i+1)}{(P_i+2)^2(P_i+3)} + \frac{\sqrt{\frac{2\ln P_T}{P_i}}}{P_i} \tag{2.2}$$

One other difference must be addressed. We do not take a minimum inside the radical of (2.1) because the new variance is usually smaller than the original variance. Auer et al. implemented 1/4 as "an upper bound on the variance of a Bernoulli random variable." [1] However, the variance of a beta distribution never exceeds 1/12 and is always smaller than the Gaussian variance used in (1.3). For practical purposes, an additional change should be considered, which we will discuss along with our experimental data.

3 Results

3.1 K-Armed Bandit Simulations

In practice, expressions (1.1) and (2.1) are tweaked to alter the weight put on exploration. This is done by including a coefficient γ (called the exploration coefficient) as shown in (3.1).

$$\frac{R_i}{P_i} + \gamma\sqrt{\ln(P_T)V_i} \tag{3.1}$$

We can then run simulations of the K-armed bandit problem and plot the total reward recieved by each strategy as a function of the exploration coefficient. For comparative purposes, an additional factor of 1/2 has been included along with γ in (3.1). Otherwise, its maximal performance lies on a different interval from that of UCB1-Tuned.

In this case, we run three different experiments, which are designed to be similar to the situation a Go player would encounter. Ten machines are set up randomly and the player is allowed 10,000 pulls. At a given board position in the game of Go, the possible moves usually do not range from 0 to 1 in probability. Instead, depending on how well a player is doing, the moves lie on a smaller interval $[a, b] \subset [0, 1]$. Therefore, the machines in Figure 7 have values $m_i \in [0.2, 0.4]$, the machines in Figure 8 have values $m_i \in [0.4, 0.6]$, and the machines in Figure 9 have values $m_i \in [0.6, 0.8]$. They are evenly distributed.

It appears from the results (see Appendix B) of these experiments that the new player is a better strategy ($p < .1$) for *any* non-zero exploration coefficient and board position. In fact, all but two exploration coefficients are significantly ($p < .05$) better. The two excluded coefficients are $\gamma = .5$ and $\gamma = .6$ for $m_i \in [0.2, 0.4]$. While there are some coefficients where both players achieve similar reward totals, it is difficult in practice to determine an optimal coefficient for an actual Go player. Therefore, it is important that a new strategy perform better for any $\gamma > 0$.

3.2 Games Against GnuGo

We also tested the UCB1-Tuned and new player against a different algorithm called GnuGo (v. 3.6). [4] GnuGo uses heuristics that do not depend as much on time, which allows many quick games to be played for testing purposes. Both UCB1-Tuned and our new algorithm were allowed 8000 random playouts to form a game tree. This is equivalent to about 1/4 of a second of thinking time for each game. After playing 1000 games each, the UCB1-Tuned algorithm managed a peak (along γ) win percentage of 32.0% while our new algorithm had a win percentage of 34.8%.

4 Conclusion

Based on experimental results, we see that the new player performs better than the UCB1-Tuned algorithm in K-arm bandit tests as well as in actual Go games against a standard player. It is able to do so by using Bayesian Analyis to make more accurate estimations about expected machine values. Since it is similar to the UCB1-Tuned algorithm, there should be little difficulty in changing any Go program that uses the UCB1-Tuned algorithm to the new algorithm.

Acknowledgments: We thank the National Science Foundation for funding the research that led to this result as well as Lewis and Clark College for providing computers for testing.

References

[1] Peter Auer and Jyrki Kivinen. Finite-time analysis of the multiarmed bandit problem. In *Machine Learning*, pages 235–256, 2002.

[2] K. Baker. The way to go. New York, NY: American Go Association, 1986.

[3] B. Bouzy and T. Cazenave. Computer go: An ai oriented survey. In *Artificial Intellegence*, pages 39–103, 2001.

[4] Man Lung Li et. al. http://www.gnu.org/software/gnugo/.

A Comparing The Gaussian and Beta Distributions

A machine with expected value $m_i = 2/3$ is explored during a test run. The blue distributions show the likelihood of m_i taking a certain value based on the Beta Distribution. The red distributions show the likelihood of m_i taking a certain value based on the Gaussian (i.e. Normal) Distribution. Notice that the Gaussian distribution does not appear until both a win and a loss have occurred. When there is no data, the Gaussian distribution is essentially flat, and when only either wins or losses are recorded, it resembles the Dirac Delta function.

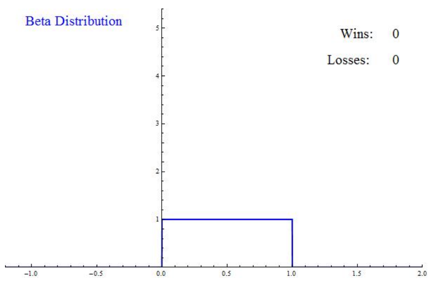

Figure 1: Probability of M_i without any prior knowledge

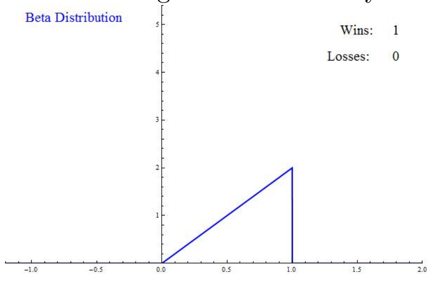

Figure 2: Probability of M_i after receiving a win

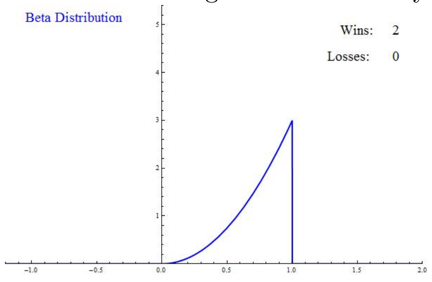

Figure 3: Probability of M_i after receiving a second win

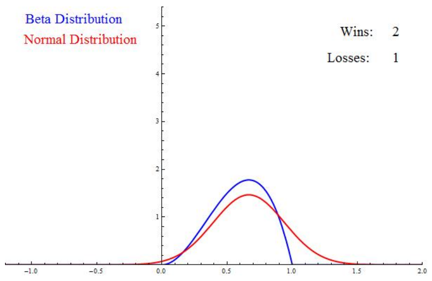

Figure 4: Probability of M_i after receiving two wins and a loss

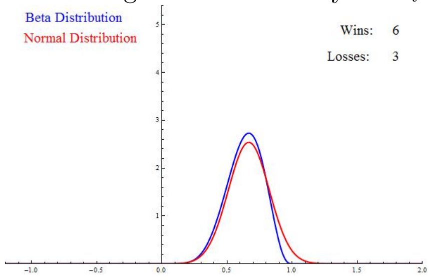

Figure 5: Probability of M_i after receiving 6 wins and 3 losses

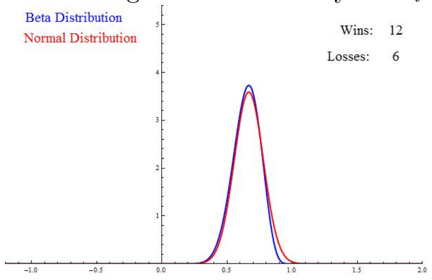

Figure 6: Probability of M_i after receiving 12 wins and 6 losses

B Results from K-armed Bandit experiments

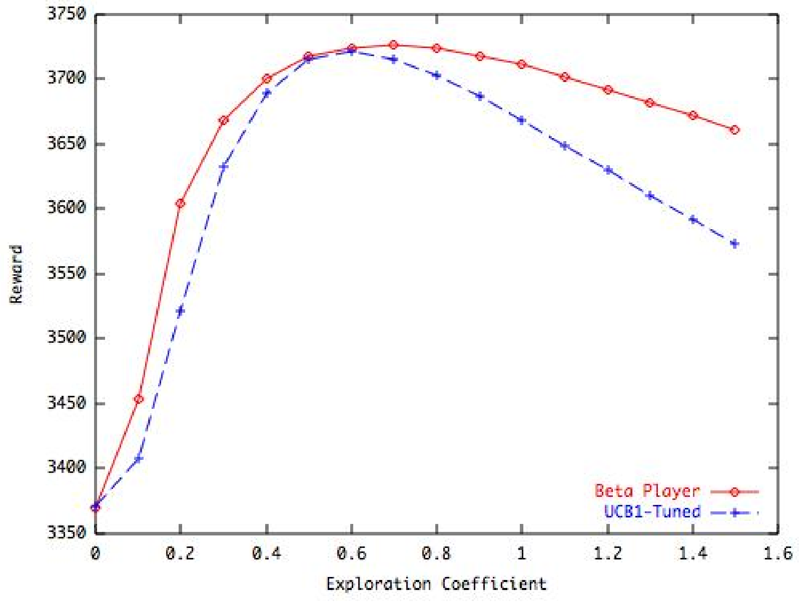

Figure 7: $m_i \in [.2, .4]$

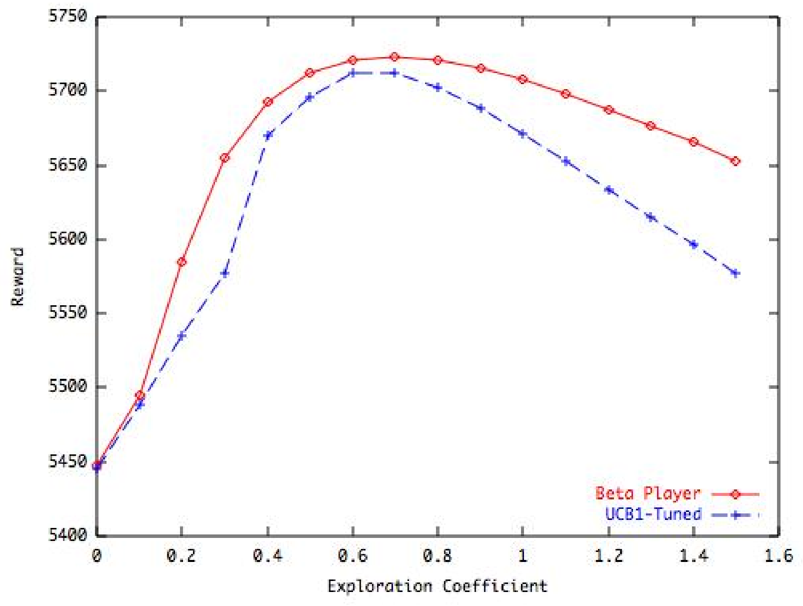

Figure 8: $m_i \in [.4, .6]$

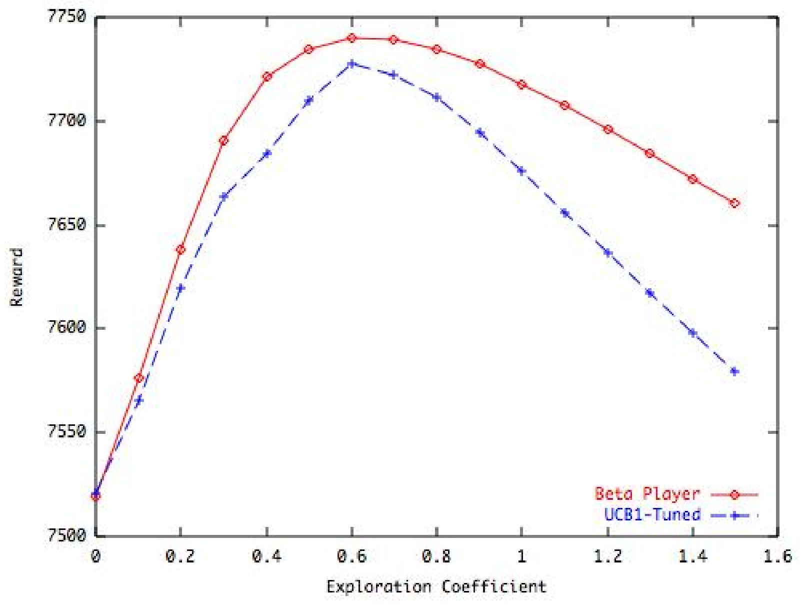

Figure 9: $m_i \in [.6, .8]$