

ヨーロッパアジアンオプションの効率的な価格付け手法  
— AMO アルゴリズムの実装と解析の改良 —

塩浦昭義\* 徳山豪\*

\*東北大学大学院情報科学研究科

本稿では、二項木モデル上でのヨーロッパアジアンオプションの価格付けに対する効率的かつ正確な乱択近似アルゴリズムを提案する。 $n$  ステップの二項木上での行使価格  $X$  のオプションと任意の正整数  $k$  が与えられたとき、我々のアルゴリズムは  $n$  に依存しない誤差  $O(X/k)$  の範囲の近似値を  $O(kn^2)$  時間で求める。我々のアルゴリズムは Aingworth, Motwani, および Oldham (2000) による近似アルゴリズムを乱択アルゴリズムへ修正したものであり、近似精度を理論的にも実用的にも改善している。

Efficiently Pricing European-Asian Options  
— Ultimate Implementation and Analysis of the AMO Algorithm —

Akiyoshi Shioura\* Takeshi Tokuyama\*

\*Graduate School of Information Sciences, Tohoku University

We propose an efficient and accurate randomized approximation algorithm for computing the price of a European-Asian option on the binomial tree model. For an option with the strike price  $X$  on an  $n$ -step binomial tree and any positive integer  $k$ , we give an  $O(kn^2)$  time algorithm with the error bound  $O(X/k)$  which is independent of  $n$ . Our algorithm can be seen as a modification of the approximation algorithm developed by Aingworth, Motwani, and Oldham (2000) into a randomized algorithm, which improves the accuracy theoretically as well as practically.

## 1 Introduction

Options are popular and important financial instruments in world financial markets. One of the simplest options is *European call* option, which is a contract giving its holder the right, but not the obligation, to buy a stock or other financial asset at some point in the future (called the *expiration date*) for a specified price  $X$  (called the *strike price*). The *payoff* of an option is the amount of money its holder makes on the contract. Suppose that we have a European option on a stock, and the stock price  $S$  is more than the strike price  $X$  on the expiration date. Then, we can make some money by *exercising* the option to buy the stock and selling the stock immediately at the market price. Hence, the payoff of a European option is given by  $(S - X)^+ = \max\{S - X, 0\}$ . The price of the option is usually much less than the actual price of the underlying stock. Therefore, options hedge risk more cheaply than stocks only, and provide a chance to get large profit with a small amount of money if one's speculation is good.

The price of an option is given by the discounted expected value of the payoff. Because of the popularity of options, techniques for computing the option price have extensively been discussed in the literature [1, 2, 5, 6, 7, 8, 10]. A standard method of pricing an option is to model the movement of the underlying financial asset as geometric Brownian motion with drift and then to construct an arbitrage portfolio [4, 8]. This yields a stochastic differential equation, and its solution gives the option price. However, it is often difficult to solve this differential

equation for many complex options such as European-Asian option dealt with in this paper, and indeed no simple closed-form solution is known. Therefore, it is widely practiced to simulate geometric Brownian motion by using a discrete model, and use this model to approximate the option price. One such discrete model is the *binomial tree model* [6, 8], where the time period is decomposed into  $n$  time steps, and Brownian motion is modeled by using a biased random walk on a directed acyclic graph called a *binomial tree* of depth  $n$ . The option price obtained from the binomial tree model converges to the option price given by the differential equation if  $n$  goes to infinity. In the binomial tree model, the process of the movement of a stock price is represented by a path in a binomial tree. An option is said to be *path-dependent* [5, 8] if the option's payoff depends on the path representing the process as well as the current stock price. Although path-dependency is often useful in designing a secure option against risk caused by sudden change of the market, it makes the analysis of the price of options quite difficult.

In this paper, we consider the pricing of *European-Asian* option. European-Asian option is a kind of path-dependent options and its payoff is given as  $(A - X)^+$ , where  $A$  is the average stock price during the time from the purchase date to the expiration date of the option and  $X$  is the strike price. It is known to be #P-hard in general to compute the exact price of path-dependent options on the binomial tree model [5]. Therefore, it is desired to design an efficient approximation algorithm with provable high accuracy, and various pricing techniques have been developed so far [1, 2, 5, 7, 10].

A most naive method for computing the exact price of European-Asian options, called the *full-path method*, is to enumerate all paths in the binomial tree model. Unfortunately, the full-path method requires exponential time since there are exponential number of paths in the binomial tree. Hence, the Monte Carlo method that samples paths in the binomial tree is popularly used to compute an approximate value of the exact price. The error bound of the Monte Carlo method, however, depends on the volatility of the stock price when a polynomial number of samples are taken by naive sampling.

Aingworth–Motwani–Oldham (AMO) [1] proposed the first polynomial-time approximation algorithm with guaranteed worst-case error bound, which enables us to avoid the influence of volatility to the theoretical error bound. The idea is to prune exponential number of high-payoff paths by using mathematical formulae during the run of an aggregation algorithm based on dynamic programming and bucketing. In each of  $n$  aggregation steps the algorithm produces the error bounded by  $X/k$ , where  $k$  denotes the number of buckets used at each node of the binomial tree. Hence, the error bound of the AMO algorithm is  $nX/k$ , and the algorithm runs in  $O(kn^2)$  time. While algorithms on the “uniform” model has been mainly considered in the literature [1, 2, 5, 7], the AMO algorithm and its analysis work on the “non-uniform” model where the transition probabilities of the stock price may differ at each node [10].

Following the work by Aingworth et al. [1], many variants of the AMO algorithm were proposed to achieve a better error bound than  $nX/k$ . Akcoglu–Kao–Raghavan [2] presented various pricing techniques applicable to approximation algorithms such as the Monte Carlo method and the AMO algorithm. In particular, they use a recursive version of the AMO algorithm and reduce the error bound to  $O(n^{\frac{1+\epsilon}{2}} X/k)$  by spending almost the same time complexity under the condition that the volatility of the stock is small.

The error bound is further improved by Dai–Huang–Lyu (DHL) [7] and by Ohta–Sadakane–Shioura–Tokuyama (OSST) [10]. While the AMO algorithm uses the same number of buckets at each node of the binomial tree, the DHL algorithm [7] uses different number of buckets at each

node. By adjusting the number of buckets at each node appropriately while keeping the time complexity  $O(kn^2)$ , they achieved the error bound  $O(\sqrt{n}X/k)$ , where  $k$  is the average number of buckets used at each node. Their analysis, however, applies only to the uniform model and does not extend to the non-uniform model. On the other hand, the OSST algorithm [10] uses the idea of randomized rounding in the aggregation steps of the algorithm, and achieves the error bound  $O(\sqrt{n}X/k)$  for the non-uniform model. Moreover, it is shown in [10] that for the uniform model the error bound of the OSST algorithm can be reduced to  $O(n^{1/4}X/k)$ .

In this paper, we further reduce the error bound by giving a randomized approximation algorithm with an  $O(kn^2)$  time complexity and an  $O(X/k)$  error bound. The error bound of our algorithm is independent of the depth  $n$  of the binomial tree, although those of the AMO algorithm and its previous variants [2, 7, 10] are dependent on  $n$ . Our algorithm uses the ideas in Dai et al. [7] and Ohta et al. [10]. As in [10], we regard the aggregation steps of the algorithm as a Martingale process with  $O(n^2)$  random steps by using novel random variables. It can be shown that the expected value of the output by our algorithm equals the exact price, and that the error in each single step is bounded by a function of the number of buckets at a node of the binomial tree. Thus, we can apply Azuma's inequality [3] to the Martingale process to obtain the error bound. If we choose  $k$  as the number of buckets at each node, the algorithm coincides with the one in [10]. To reduce the error bound as much as possible, we adjust the number of buckets at each node and obtain the error bound  $O(X/k)$ , where  $k$  is the average number of buckets used at each node. Since the value  $X/k$  can be seen as the "average" of the absolute error produced at each node of the binomial tree, the error bound of our algorithm is the best possible within the framework of the AMO algorithm. We also show the practical quality of the approximate value computed by our algorithm by some numerical experiments; indeed, its accuracy is better by a factor of nearly 100 than that of the AMO algorithm when  $n = 35$ .

## 2 Preliminaries

A *binomial tree*, also called a *recombinant binary tree*, of depth  $n$  is a leveled directed acyclic graph defined as follows. A binomial tree of depth  $n$  has  $n + 1$  levels. There are  $i + 1$  nodes in the  $i$ -th level ( $0 \leq i \leq n$ ) and each node is labeled as  $(i, j)$ , where  $j$  ( $0 \leq j \leq i$ ) denotes the numbering of the nodes. The node  $(0, 0)$  in the 0-th level is called the *root*, and each node  $(n, j)$  in the  $n$ -th level is called a *leaf*. Each non-leaf node  $(i, j)$  has two children  $(i + 1, j)$  and  $(i + 1, j + 1)$ . Therefore, each non-root node  $(i, j)$  has two parents  $(i - 1, j - 1)$  and  $(i - 1, j)$  if  $1 \leq j \leq i - 1$ , and each of  $(i, 0)$  and  $(i, i)$  has only one parent.

Let us consider a discrete random process simulating the movement of a stock price. We divide the time from the purchase date to the expiration date of an option into  $n$  time periods, and the  $i$ -th time step means the end of the  $i$ -th time period. In particular, 0-th (resp.,  $n$ -th) time step is the purchase (resp., expiration) date of the option. For  $i = 0, 1, \dots, n$ , let  $S_i$  be a random variable representing the stock price at the  $i$ -th time step, where  $S_0$  is the initial stock price known in advance. The fundamental assumption in the binomial tree model is that in each time step the stock price  $S$  either rises to  $uS$  or falls to  $dS$ , where  $u$  and  $d$  are predetermined constants satisfying  $u > d$  and  $u = 1/d$ . Thus, we can model the stock price movement by using a binomial tree.

Suppose that we are at a non-leaf node  $(i, j)$  in the binomial tree model and the current

stock price is  $S$ . With probability  $p_{ij}$ , we move to the node  $(i + 1, j)$  and the stock price rises to  $uS$ ; with probability  $1 - p_{ij}$ , we move to the node  $(i + 1, j + 1)$  and the stock price falls to  $dS$ . Thus, the stock price at the node  $(i, j)$  is  $S_i(j) = u^{i-j}d^j S_0$ .

The binomial tree model is said to be *uniform* if  $p_{ij} = p$  for each node  $(i, j)$ ; otherwise it is *non-uniform*. The uniform model has been widely considered [1, 2, 5, 7] since  $p$  is uniquely determined under the non-arbitrage condition of the underlying stock. The non-uniform model, however, is often useful to deal with various stochastic models. For each node  $(i, j)$ , we denote by  $\omega(i, j)$  the probability that the random walk reaches to  $(i, j)$ . In the uniform model, we have  $\omega(i, j) = \binom{i}{j} p^{i-j} (1-p)^j$ , where  $\binom{i}{j} = i! / (i-j)! j!$ .

Let  $X$  be the strike price of an option. The *payoff* of an option is the amount of money its holder makes on the contract. We adopt a convention to write  $F^+$  for the value  $\max\{F, 0\}$ .

*European option* is one of basic options, and its payoff is given by  $(S_n - X)^+$  which is determined by the stock price  $S_n$  on the expiration date. It is quite easy to compute the expected value of the payoff of European options under the binomial tree model. A drawback of European options is that the payoff may be affected drastically by the movement of the stock price just before the expiration date; even if the stock price goes very high during most of time periods, it may happen that the option does not make money at the end.

*European-Asian option* is more reliable for the holder than European option, and its payoff is given by  $(A_n - X)^+$ , where  $A_n = (\sum_{i=0}^n S_i) / (n + 1)$  is the average of the stock prices during  $n$  time periods. Let  $T_j = \sum_{i=0}^j S_i$  be the *running total* of the stock price up to the  $j$ -th time step. Once  $T_j$  exceeds the threshold  $(n + 1)X$ , the option holder will surely exercise it on the expiration date and obtain the payoff of at least  $T_j / (n + 1) - X$ .

Our aim is to compute the price of European-Asian options. Since the price of an option is given by the discounted expected value of the payoff, it suffices to compute the expected payoff. A simple method is to compute the running total  $T_n(\mathcal{P})$  of the stock price for each path  $\mathcal{P}$  in the binomial tree together with the probability  $\Pr(\mathcal{P})$  that the path occurs, and exactly compute the value

$$E((A_n - X)^+) = \sum \{\Pr(\mathcal{P}) \cdot (T_n(\mathcal{P}) / (n + 1) - X)^+ \mid \mathcal{P} : \text{a path from the root to a leaf}\}.$$

We call the expected value of the payoff computed as above the *exact value* of the expected payoff, and denote  $U = E((A_n - X)^+)$ . This simple method, however, needs exponential time since there are  $2^n$  paths in a binomial tree. The Monte Carlo method is a popular method to reduce computation time, although we need a huge number of paths to assure a small provable error bound if we use naive random sampling of paths.

## 3 A New Algorithm for Pricing European-Asian Options

### 3.1 A Basic Algorithm

We describe a basic approximation algorithm for the option's expected payoff. This algorithm is a slight generalization of the AMO algorithm, and the previous approximation algorithms in [1, 7, 10] can be seen as specialized versions of this basic algorithm.

As in [1], the basic algorithm uses dynamic programming to compute an approximate value of the option's expected payoff. For a path  $\mathcal{P}$  from the root to a node  $(i, j)$  in the  $i$ -th level, we define the *state* of  $\mathcal{P}$  as a pair  $(S_i(j), T_i)$  of the stock price  $S_i(j) = u^{i-j}d^j S_0$  and the running

total  $T_i$ . Note that the states of two different paths reaching a node  $(i, j)$  can be the same. We define the *weight* of the state  $(S_i(j), T_i)$  as the probability that a path  $\mathcal{P}$  with the state  $(S_i(j), T_i)$  occurs. The basic algorithm is based on a simple observation that if the running total of a current state is above the threshold  $(n+1)X$ , then the conditional expectation of the payoff at this state can be analytically computed, and such a state can be pruned away.

Suppose that we are at a node  $(i, j)$  in the  $i$ -th level and the current state is  $(S, T)$ , where  $T \geq (n+1)X$ . Then, the payoff's conditional expectation is given as  $\{T + h(i, j)\}(n+1) - X$ , where  $h(i, j)$  is the value defined by the following recursive formula [1, 10]:

$$h(i, j) = \begin{cases} 0 & \text{if } i = n, \\ p_{ij}\{h(i+1, j) + S_{i+1}(j)\} + (1 - p_{ij})\{h(i+1, j+1) + S_{i+1}(j+1)\} & \text{if } i < n. \end{cases}$$

Hence, we need to consider only the states with running total less than  $(n+1)X$ , which may be exponential many. Rather than dealing with each unpruned state individually, we instead aggregate the states by using buckets that divide the interval  $[0, (n+1)X)$ . At each node  $(i, j)$  in the  $i$ -th level, the algorithm creates  $k_{ij}$  buckets  $B_i(j, h)$  ( $h = 0, 1, \dots, k_{ij} - 1$ ), each of which corresponds to the interval  $[b_h, b_{h+1}) = [\frac{(n+1)X}{k_{ij}}h, \frac{(n+1)X}{k_{ij}}(h+1))$ . Each unpruned state of a path terminating at the node  $(i, j)$  is stored in one of  $k_{ij}$  buckets according to its running total. The algorithm chooses a value  $R_i(j, h)$  in the interval  $[\frac{(n+1)X}{k_{ij}}h, \frac{(n+1)X}{k_{ij}}(h+1))$  appropriately, and approximates all states in the bucket  $B_i(j, h)$  by a single state  $(S_i(j), R_i(j, h))$ , where its weight  $w_i(j, h)$  is given by the sum of the weights of all states in  $B_i(j, h)$ . Then, the algorithm produces two new states  $(S_{i+1}(j), R_i(j, h) + S_{i+1}(j))$  and  $(S_{i+1}(j+1), R_i(j, h) + S_{i+1}(j+1))$  in the  $(i+1)$ -st level, and inserts these state in appropriate buckets at the nodes  $(i+1, j)$  and  $(i+1, j+1)$ , respectively, or computes the conditional expectation of the payoff at these states.

**Theorem 1.** *The basic algorithm computes a value  $\Psi$  satisfying  $|\Psi - U| \leq X \sum_{i=0}^n \sum_{j=0}^i \omega(i, j)/k_{ij}$ . The time and space complexity are  $O(\sum_{i=0}^n \sum_{j=0}^i k_{i,j})$ .*

### 3.2 Previous Algorithms

We can obtain the algorithms in [1, 7, 10] by customizing the number of buckets  $k_{ij}$  and the choice of the value  $R_i(j, h)$ .

The AMO algorithm [1], which has the time complexity  $O(kn^2)$  and the deterministic error bound  $nX/k$  for the non-uniform model, can be obtained by setting  $k_{ij} = k$  with a positive integer  $k$  for all nodes  $(i, j)$  and  $R_i(j, h) = \frac{(n+1)X}{k}h$ . Note that the AMO algorithm computes a lower bound of the exact value  $U$  of the expected payoff; we can also compute an upper bound by setting  $R_i(j, h) = \frac{(n+1)X}{k}(h+1)$  instead. We denote by AMO-LB (resp., AMO-UB) the AMO algorithm for computing the lower bound (resp., the upper bound) of the exact value.

Dai et al. [7] proposed four approximation algorithms nUnifDown, nUnifCvg, nUnifUp, and nUnifSpl, where the first two compute lower bounds of the exact value  $U$  and the last two compute upper bounds. All algorithms use  $k_{ij}$  values defined as follows:

$$k_{ij} = \left\lceil \frac{k(n+1)(n+2)}{2} \times \frac{\sqrt{\omega(i, j)}}{\sum_{i'=0}^n \sum_{j'=0}^{i'} \sqrt{\omega(i', j')}} \right\rceil \quad \text{for all nodes } (i, j),$$

where  $k$  is a positive integer corresponding to the average number of buckets at each node. In the algorithm nUnifDown (resp., nUnifUp) we set  $R_i(j, h) = \frac{(n+1)X}{k_{ij}}h$  (resp.,  $R_i(j, h) = \frac{(n+1)X}{k_{ij}}(h+1)$ ).

The algorithms `nUnifCvg` and `nUnifSpl` are modified versions of `nUnifDown` and `nUnifUp` by using heuristics. Although the deterministic error bound  $\sqrt{n}X/k$  of `nUnifDown` and `nUnifUp` is better than that of the AMO algorithms, the analysis in [7] applies only to the uniform model. While the error bounds of `nUnifCvg` and `nUnifSpl` are the same as those of `nUnifDown` and `nUnifUp` theoretically, they are much better practically.

The OSST algorithm [10] is a randomized algorithm, and can be obtained as follows. To set the value  $R_i(j, h)$ , we choose a “representative” state  $(S_i(j), T)$  in the bucket  $B_i(j, h)$  randomly, where a state with weight  $w$  is chosen with probability  $w/w_i(j, h)$ , and set  $R_i(j, h) = T$ . We also set  $k_{ij} = k$  with a positive integer  $k$  for all nodes  $(i, j)$ , as in the AMO algorithm. The OSST algorithm has the probabilistic error bound  $O(\sqrt{n}X/k)$  for the non-uniform model and  $O(n^{1/4}X/k)$  for the uniform model.

### 3.3 Our Algorithm and Analysis

Our algorithm is based on the ideas used in Dai et al. [7] and Ohta et al. [10], and can be obtained from the basic algorithm by setting  $k_{ij}$  and  $R_i(j, h)$  as follows.

First, we set  $R_i(j, h)$  in the same way as in [10], i.e., we choose a representative state  $(S_i(j), T)$  in the bucket  $B_i(j, h)$  randomly, where a state with weight  $w$  is chosen with probability  $w/w_i(j, h)$ , and set  $R_i(j, h) = T$ . We explain later how to choose  $k_{ij}$ .

Let  $\Psi$  be the payoff value computed by our algorithm. Since our algorithm is randomized,  $\Psi$  is a random variable depending on the coin-flips to choose representatives of states in the buckets. Let  $Y_{i,j}$  be the random variable giving the future value of the payoff just after the algorithm processes the node  $(i, j)$  in the  $i$ -th level, i.e., after the choice of representatives in all buckets has been determined up to the  $j$ -th node in the  $i$ -th level. By definition,  $Y_{0,0} = U$  and  $Y_{n,n} = \Psi$ . Thus, we have a random process with  $\sum_{i=0}^n (i+1) = (n+1)(n+2)/2$  steps.

The following lemma shows that random variables  $Y_{0,0}, Y_{1,0}, Y_{1,1}, \dots, Y_{n,n-1}, Y_{n,n}$  constitute a *Martingale sequence*.

**Lemma 2.**  $E(Y_{i,j} \mid Y_{0,0}, Y_{1,0}, Y_{1,1}, \dots, Y_{i,j-1}) = Y_{i,j-1}$  for  $i = 0, 1, \dots, n, j = 0, 1, \dots, i$ .

Lemma 2 also shows that the expected value of the payoff  $\Psi$  equals the exact value  $U$  of the expected payoff, i.e.,  $E(Y_{n,n}) = E(\Psi) = U$ .

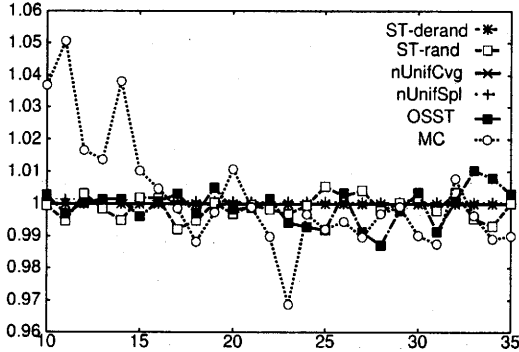
When the algorithm processes a node  $(i, j)$ , the running totals of the paths terminating at  $(i, j)$  are approximated with the error less than  $X/k_{ij}$ , and the running totals of other paths remain the same. Hence, from the argument in Section 3.1 we have  $|Y_{i,j+1} - Y_{i,j}| < X\omega(i, j+1)/k_{i,j+1}$  ( $0 \leq j < i \leq n$ ) and  $|Y_{i+1,0} - Y_{i,i}| < X\omega(i+1, 0)/k_{i+1,0}$  ( $0 \leq i < n$ ). Thus, Azuma’s inequality [3] applies (see [9, Theorem 4.16] for the present form).

**Theorem 3 (Azuma’s inequality).** *Let  $Z_0, Z_1, \dots$  be a Martingale sequence such that  $|Z_k - Z_{k-1}| < c_k$  for each  $k$ , where  $c_k$  is a constant depending on  $k$ . Then, we have*

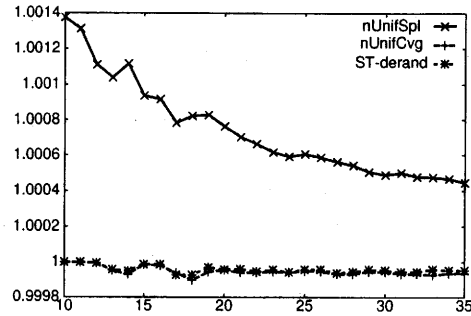
$$\Pr[|Z_t - Z_0| \geq \lambda] \leq 2 \exp(-\lambda^2/2 \sum_{k=1}^t c_k^2) \quad (\forall t = 1, 2, \dots, \forall \lambda > 0).$$

Theorem 3 yields  $\Pr[|Y_{n,n} - U| \geq \lambda] \leq 2 \exp(-\lambda^2/2X^2\Gamma)$ , where  $\Gamma = \sum_{i=1}^n \sum_{j=0}^i (\omega(i, j)/k_{ij})^2$ . Hence, we have the following lemma.

**Lemma 4.** *Let  $c$  be any positive real number. Then, our algorithm computes in  $O(\sum_{i=0}^n \sum_{j=0}^i k_{i,j})$  time a value  $\Psi$  satisfying  $|\Psi - U| \leq cX\sqrt{\Gamma}$  with probability at least  $1 - 2e^{-c^2/2}$ .*



(a) Comparison of our algorithms with nUnifCvg, nUnifSpl, OSST, and MC



(b) Comparison of nUnifCvg, nUnifSpl, and ST-derand

Figure 1: Relative errors of approximate option prices computed by several algorithms

To minimize the error bound  $cX\sqrt{\Gamma}$  while keeping the time complexity  $O(kn^2)$ , we define the number of buckets at node  $(i, j)$  by  $k_{ij} = \lceil k(n+2)\omega(i, j)/2 \rceil$ . Since  $\Gamma \leq 2/k^2$  and  $\sum_{i=0}^n \sum_{j=0}^i k_{ij} = O(kn^2)$ , we have the following theorem:

**Theorem 5.** *For any positive integer  $k$  and a positive real number  $c$ , our algorithm computes in  $O(kn^2)$  time a value  $\Psi$  satisfying  $|\Psi - U| \leq \sqrt{2}cX/k$  with probability at least  $1 - 2e^{-c^2/2}$ .*

Although the error bound  $O(X/k)$  of our algorithm shown in the last section is better than the previous approximation algorithms, our algorithm is randomized and therefore the error bound only holds with “high” probability. Hence, it is desired to derandomize our algorithm without losing its accuracy. One idea for derandomization is to take the weighted mean of running totals of the states in each bucket  $B_i(j, h)$  as the value  $R_i(j, h)$ , as in the algorithm nUnifCvg by Dai et al. [7]. Although the theoretical analysis does not certify the  $O(X/k)$  error bound, it is experimentally shown that the error bound of this derandomized algorithm is better than the original randomized algorithm.

### 3.4 Experimental Results

We show some experimental results to illustrate the performance of our randomized approximation algorithm and its derandomized version. In particular, we compare the quality of the option price computed by our algorithms with those by other approximation algorithms. We implemented the full-path method to compute the exact price, and approximation algorithms such as the naive Monte Carlo method (MC), the AMO algorithms (AMO-LB, AMO-UB), the DHL algorithms [7] (nUnifDown, nUnifCvg, nUnifUp, nUnifSpl), and the OSST algorithm [10] (OSST). We denote our randomized and derandomized algorithms by ST-rand and ST-derand, respectively. The experiment is done by a Pentium IV 2.60CGHz PC and all programs are implemented in C++.

In the experiment, we consider a uniform model with  $S_0 = X = 100$ ,  $u = 1.1$ ,  $d = 1/u$ ,  $pu + (1-p)d = (1.06)^{1/n}$ . The parameter  $k$  is set to 100 in the approximation algorithms except for MC. Recall that the positive integer  $k$  denotes the number of buckets used at each node for

AMO-LB/UB and OSST while  $k$  is the average number of buckets used at each node for the DHL algorithms and ours. The Monte Carlo method MC takes  $400n$  sample paths so that it runs in almost the same time as other approximation algorithms. In the experiment, only one trial is made for each algorithm.

Figure 1 gives the result of the experiment in the range  $n \in [10, 35]$ , showing the ratio of the approximate prices computed by approximation algorithms to the exact price. The running time of the approximation algorithms are almost the same and less than 0.05 seconds, and the full-path method takes more than 9 hours when  $n = 35$ . The results of AMO-LB/UB and  $n\text{UnifDown/Up}$  are not shown in the graphs since the relative errors of these are always more than 0.2 and much worse than the relative errors of the other algorithms.

The graph (a) shows that the relative errors of the algorithms  $n\text{UnifCvg}$ ,  $n\text{UnifSpl}$ , and  $\text{ST-derand}$  are better than those of the other algorithms. In particular, our derandomized algorithm  $\text{ST-derand}$  performs much better than the original randomized algorithm  $\text{ST-rand}$ . In the graph (b) we compare the three algorithms  $n\text{UnifCvg}$ ,  $n\text{UnifSpl}$ , and  $\text{ST-derand}$ . We see from the graph (b) that the relative error of our derandomized algorithm  $\text{ST-derand}$  is quite accurate and as good as  $n\text{UnifCvg}$ . This result shows that the error bound of our derandomized algorithm  $\text{ST-derand}$  is much better than the error bound  $O(X/k)$  of the randomized algorithm  $\text{ST-rand}$ . It is an interesting open question whether  $\text{ST-derand}$  also has the theoretical error bound  $O(X/k)$ , which is left for further research.

## References

- [1] D. Aingworth, R. Motwani, and J. D. Oldham, Accurate approximations for Asian options, *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms* (2000), 891–900.
- [2] K. Akcoglu, M.-Y. Kao, and S. V. Raghavan, Fast pricing of European Asian options with provable accuracy: single-stock and basket options, *Proc. Annual European Symposium on Algorithms 2001, Lecture Notes in Computer Science* **2161** (2001), 404–415.
- [3] K. Azuma, Weighted sum of certain dependent random variables, *Tohoku Mathematical Journal* **19** (1967) 357–367.
- [4] F. Black and M. Scholes, The pricing of options and corporate liabilities, *Journal of Political Economy* **81** (1973), 637–654.
- [5] P. Chalasani, S. Jha, and I. Saias, Approximate option pricing, *Algorithmica* **25** (1999), 2–21.
- [6] J. C. Cox, S. A. Ross, and M. Rubenstein, Option pricing: a simplified approach, *Journal of Financial Economics* **7** (1979), 229–263.
- [7] T.-S. Dai, G.-S. Huang, and Y.-D. Lyuu, Extremely accurate and efficient tree algorithms for Asian options with range bounds, 2002 NTU International Conference on Finance, National Taiwan University, Taiwan, May 2002.
- [8] J. C. Hull, *Options, Futures, and Other Derivatives, Fifth Edition*, Prentice Hall, Upper Saddle River, NJ, 2002.
- [9] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge Univ. Press, London, 1995.
- [10] K. Ohta, K. Sadakane, A. Shioura, and T. Tokuyama, A fast, accurate and simple method for pricing European-Asian and Saving-Asian options, *Proc. Annual European Symposium on Algorithms 2002, Lecture Notes in Computer Science* **2461** (2002), 772–784.