A Refined Binomial Lattice for Pricing American Asian Options

PRASAD CHALASANI chal@asu.edu
Computer Science Dept, Arizona State University, Tempe, AZ 85287

SOMESH JHA sjha@cs.cmu.edu
Computer Science Dept, Carnegie Mellon University, Pittsburgh, PA 15213

FEYZULLAH EGRIBOYUN feyz@fir.fbc.com
CS First Boston, NY

ASHOK VARIKOOTY avarikoo@fir.fbc.com
CS First Boston, NY

Abstract. We present simple and fast algorithms for computing very tight upper and lower bounds on the prices of American Asian options in the binomial model. We introduce a new refined version of the Cox-Ross-Rubinstein [4] binomial lattice of stock prices. Each node in the lattice is partitioned into "nodelets", each of which represents all paths arriving at the node with a specific geometric stock price average. The upper bound uses an interpolation idea similar to the Hull-White [6] method. From the backward-recursive upper-bound computation, we estimate a good exercise rule that is consistent with the refined lattice. This exercise rule is used to obtain a lower bound on the option price using a modification of a conditional-expectation based idea from Rogers-Shi [12] and Chalasani-Jha-Varikooty [3]. Our algorithms run in time proportional to the number of nodelets in the refined lattice, which is smaller than $n^4/20$ for n periods.

Keywords: American Options, Asian Options, Path-dependent Options, Binomial Model, Stopping times

1. Introduction

The payoff of an Asian option on an asset is based on the arithmetic average price of the asset. Whether or not early exercise is allowed, the valuation of such path-dependent options has been a hard and interesting problem that has attracted the attention of several researchers [2, 3, 6, 7, 8, 9, 10, 11, 13, 14, 15]. In the Cox-Ross-Rubinstein [4] binomial model, the primary difficulty in valuing these options stems from the fact that every stock price path has a different arithmetic stock-price average, and the distribution of the arithmetic average is hard to characterize. Nevertheless, some accurate numerical approximations for European Asian options (i.e., those that can only be exercised at expiration) have recently been proposed [3, 6, 12, 14]. However for Asian options that have an early-exercise provision, or American Asian options, very little progress has been made. While Monte Carlo simulation produces reasonable approximations to the price of an European Asian option, the technique cannot handle American Asian options since it is difficult to estimate an exercise rule that is "close" to the optimal one [6]. Only recently, Hull and White [6] have reported an interpolation method to compute an upper bound on the American Asian option price.

In this paper we present simple and fast algorithms to compute very accurate *upper and lower bounds* on the price of an American Asian option in the binomial tree model of Cox-Ross-Rubinstein [4]. More precisely, we consider American Asian options where the underlying stock price follows a binomial process (see Section 2). We assume that the average is taken over the stock prices at all the discrete times (up to the time of exercise) considered in the binomial tree process, and that the only exercise times allowed are these discrete time points [1] For a given size Δt of the time-intervals in the binomial process, the American Asian option has a certain arbitrage-free value $B_{\Delta t}$. It is known that as $\Delta t \to 0$, $B_{\Delta t}$ converges to C, the value of the American Asian option where the underlying is driven by a geometric Brownian motion, with *continuously-sampled averaging* and exercise permitted at any continuous time (see, e.g.,[5]). Our algorithms produce an upper bound $U_{\Delta t}$ and a lower bound $L_{\Delta t}$ of the value $B_{\Delta t}$ under the binomial model. It therefore follows that in the limit as $\Delta t \to 0$, our upper and lower bounds converge to upper and lower bounds of the "true" option value C.

For the Asian options considered by Hull-White [6], for a 60-step binomial tree, our upper and lower bounds compute in a total of less than 20 seconds, and the gap between them is often less than 0.002 percent of the initial stock price. Thus we are able to quickly produce a tight bracket around the option value. By contrast, Hull and White [6] show only how to compute an upper bound on the option price, but they are not able to estimate the error of their method since there is no satisfactory Monte Carlo algorithm for estimating the price of an American Asian option.

Our algorithms are based on two new ideas. The first idea is to use a refinement of the standard Cox-Ross-Rubinstein binomial lattice for stock prices. To compute the upper bound we use this refined lattice and an interpolation idea similar to the one used by Hull and White [6]. The advantage of our method over the Hull-White algorithm is that our interpolation grid corresponds to actual paths in the binomial tree (in a sense which will be made clearer later), and this enables us to estimate a "good" exercise rule from which a lower bound on the American Asian option price can be computed.

Here we give an informal overview of these ideas; the rest of the paper gives the details. The ideas are best explained by first examining a naive way to upper- and lower-bound the option value. We follow the standard notation: p is the up-tick probability, u and $1/u$ are the factors by which the stock price moves up and down respectively in each period, n is the number of steps in the binomial tree or lattice, and R is the one-period discount factor. Also the random variable S_k is the stock price at time k, H_k is the number of up-ticks by time k, and A_k is the arithmetic average of the $k+1$ stock prices from time 0 to time k. Note that $S_k = S_0 u^{2H_k - k}$.

Let us first recall what the nodes of a standard Binomial lattice represent. At level k in the standard lattice, each node represents the paths of the binomial tree reaching the same value of the stock price S_k, or equivalently, the same number of up-ticks H_k. A node can therefore be identified by the coordinates (k, h) where k represents the number of steps from the starting node, and h is the number of up-ticks.

Here is one way to get a crude upper bound on the price of an American Asian option. Compute, at each lattice node (k, h), the average value $\widetilde{A}(k, h)$ of the arithmetic stock-price average A_k over all paths reaching this node. The quantity $\widetilde{A}(k, h)$ can be thought of as a "representative" arithmetic average for the node. These quantities can be computed

easily at each node by a simple forward induction on the lattice. Then compute the upper bound by backward-recursion and linear interpolation as in the Hull-White [6] method. Essentially, this method treats the option value at time k as a function of (H_k, A_k) and estimates this value at pairs $(h, \widetilde{A}(k,h))$ that correspond to the nodes (k,h) at level k. For $k = n$ the option value is set equal to the immediate payoff, $[\widetilde{A}(n,h) - L]^+$. For $k < n$, the option value is computed backward-recursively as the maximum of the immediate payoff and the expected discounted option value at time $k+1$. This will in general require knowledge of the option value at time $k+1$ at arithmetic averages that are different from, but close to, the representative averages $\widetilde{A}(k+1,.)$. Interpolation is used to estimate these missing values (details are in Section 4). The backward recursion finally computes the value estimate at time 0 at arithmetic average S_0 (the initial stock price), which can be shown to be an upper bound on the option value.

Based on this crude upper bound, we can obtain a crude lower bound as follows. From the upper-bound computation, we first estimate a "good" exercise rule as follows. For each node (k,h), if either $k = n$ or the option value estimate at this node equals the immediate payoff, namely $(\widetilde{A}(k,h) - L)^+$, then mark node (k,h) as a "stopped node". This corresponds to the exercise rule: "If $H_k = h$ and (k,h) was marked as a stopped node, then exercise the option." In other words, on any path through the lattice nodes, the exercise point is the *earliest* node that was marked as a stopped node. Note that of all the paths reaching a stopped node (k,h), the only ones that have exercise points at time k are those that do not pass through earlier stopped nodes.

Now the expected discounted payoff under this exercise rule is a lower bound on the option price. This is simply the sum, over all stopped lattice nodes (k,h), of $W(k,h)$ where $W(k,h)$ is defined as the average of $R^{-k}(A_k - L)^+$ over all paths that are *exercised* at (k,h). Each $W(k,h)$ is of course just as difficult to compute as the original option price itself. Fortunately Jensen's inequality allows us to lower bound $W(k,h)$ by $R^{-k}(\widehat{A}(k,h) - L)^+$ where $\widehat{A}(k,h)$ is the average value of A_k over all paths that are exercised at (k,h). This is an extension of an idea based on conditional expectations used to lower bound the value of a European Asian option [3, 12]. We can compute the various $\widehat{A}(k,h)$ values by a forward induction similar to the one used to compute $\widetilde{A}(k,h)$ (except that we now take stopped lattice nodes into account).

The upper and lower bounds obtained above are crude because the binomial lattice is too coarse: The arithmetic averages A_k on paths reaching a node (k,h) will in general vary over a wide range around the representative average $\widetilde{A}(k,h)$. Our idea is to partition each node (k,h) of the lattice into *nodelets*, each of which represents binomial tree paths that reach (k,h), *and* have the same *geometric* stock price average from time 0 to time k. The upper and lower bound computations are then essentially the same as described above, except that we work on the refined lattice. As we will show, the algorithms take time proportional to the total number of nodelets, which is less than $n^4/20$ for an n-period tree with $n \geq 14$. For instance, for $n = 60$, our upper and lower bound computations take a total of less than 20 seconds on a Sun Sparc Ultra Workstation. By contrast the "brute force" algorithm that examines all 2^n paths would take about $2^{60}(20/60^4)$ times longer, or about 8×10^{11} seconds, which is more than 10 million days! Moreover, the upper and lower bounds turn out to be extremely close to each other in most cases. The intuition for

why this approach works so well is the following. Our algorithms are in effect treating each path that reaches a given nodelet as if it had an arithmetic average equal to the *average* of the arithmetic averages over all paths reaching the nodelet. The error incurred in doing this is very small since paths with the same geometric stock-price average that end at the same stock price will have arithmetic stock-price averages that do not vary too much (see Fig. 3 for an illustration of this).

Section 2 establishes some basic notation and defines the standard binomial lattice of [4]. In Section 3 we introduce the refined binomial lattice and show how to construct it efficiently. Sections 4 and 5 describe the upper bound and lower bound algorithms respectively. Section 6 shows experimental comparisons between our method and previous ones, including in particular the Hull-White method. Section 7 concludes with a discussion of future research directions.

2. The CRR Binomial Tree and Basic Notation

We first establish the notation that we will use throughout this paper. We refer to the asset underlying the American Asian option as the "stock", and make the standard assumption [1] that in a risk-neutral world, the price of the stock $S(t)$ at time t satisfies the stochastic differential equation for geometric Brownian motion:

$$dS(t) = rS(t)\ dt + \sigma S(t)\ dB(t),$$

where the annual continuously-compounded risk-free interest-rate r and annual volatility σ are constant, and $B(t)$ is a Brownian motion process. The derivative security under consideration expires at time T, expressed in years in this paper.

The continuous-time function $S(t)$ can be approximated in the form of a Cox-Ross-Rubinstein [4] binomial tree, as follows. The life of the option is divided into n time steps of length $\Delta t = T/n$. In time Δt the stock price moves up by a factor u with probability p, or down by a factor $d = 1/u$ with probability $q = 1 - p$, where

$$u = e^{\sigma\sqrt{\Delta t}},$$

$$p = \frac{e^{r\Delta t} - d}{u - d}.$$

Time k in this discrete model corresponds to continuous-time $k\Delta t$. Since r is the risk-free rate of interest, a unit of currency lent or borrowed at time k will be worth $R = e^{r\Delta t}$ units at time $k + 1$. This binomial tree model will be assumed throughout the paper.

More formally, we have a sample space Ω of all possible sequences of n independent coin-tosses (i.e. H's and T's), where an H corresponds to an up-tick of the stock and a T corresponds to a down-tick. A typical sequence (or "path") in Ω is written $\omega = \omega_1\omega_2 \ldots \omega_n$, where $\omega_i \in \{H, T\}$ represents the ith coin-toss. We will use the term "path" loosely to refer to any prefix of a path in Ω. The **length** of a path is the number of branches it contains. A k-**prefix** of a path $\omega \in \Omega$ is the length-k prefix of ω. For $k = 1, 2, \ldots, n$, we let X_k be the random variable representing the k'th **tick** and define X_k to be 1 if $\omega_k =$

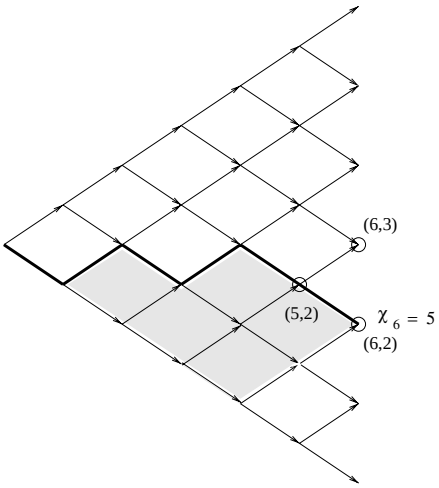

Figure 1. An example of a binomial lattice. For instance the node $(5,2)$ represents the paths that have 2 up-ticks by time 5. The nodelets inside the circled nodes are shown in Fig. 2. A path reaching node $(6,2)$ is shown highlighted. The area $\mathcal{X}_6$ of this path at time 6 is the number of boxes (shown shaded) contained between this path and the lowest path reaching node $(6,2)$.

H and 0 otherwise. The random variables $X_1, X_2, \ldots$, thus represent an asymmetric **random walk**. The random variable H_k is defined as the number of **heads**, or **up-ticks** by time k:

$$H_k = \sum_{i=1}^{k} X_k, \quad k = 1, 2, \ldots, n,$$

and H_0 is defined as 0. Similarly, the random variable T_k is the number of **tails**, or **down-ticks** by time k:

$$T_k = k - H_k.$$

For a specific path A and a random variable Z, we write $Z(A)$ to refer to the value of Z on A.

The **binomial lattice** is a recombining binomial tree whose **nodes** represent possible values of (k, H_k) on the paths of the tree. We say that a path A of the tree **passes through,** or **reaches**, a node (k,h), if $H_k(A) = h$ on this path. Fig. 1 shows an example of a lattice.

The stock price at time k is denoted S_k, $k = 0, 1, \ldots, n$, and is given by

$$S_k = S_0 u^{H_k - T_k} = S_0 u^{2H_k - k}.$$

Thus the stock price at lattice node (k,h) is $S_0 u^{2h-k}$. The average stock price at time k is defined as

$$A_k = (S_0 + S_1 + \ldots + S_k)/(k+1), \quad k \geq 0.$$

The payoff of an **Asian call** with strike L at time n is $(A_n - L)^+$. Let $\mathcal{F}$ denote the σ-algebra consisting of all subsets of Ω, and $\mathcal{F}_k$ denote the σ-algebra generated by the first k coin-tosses, i.e., by $X_1, \ldots, X_k$. We will always assume the **risk-neutral probability measure P** on $(\Omega, \mathcal{F})$ given by:

$$\mathbf{P}[X_k = 1] = p, \quad k = 1, 2, \ldots, n.$$

The **value** of this option at time 0 is defined as the maximum expected discounted payoff achievable over all possible exercise strategies τ:

$$V_0 = \max_{\tau} \mathbf{E}[(A_\tau - L)^+ / R^\tau]. \tag{1}$$

This is the quantity we want to estimate in this paper.

3. The Refined Binomial Lattice

In order to describe the lattice refinement we need to introduce a new random variable $\mathcal{X}_k$, which we call the **area** at time k:

$$\mathcal{X}_k = \sum_{i=1}^{k} (1 - X_i) H_i, \quad k = 1, 2, \ldots, n, \tag{2}$$

and define $\mathcal{X}_0 = 0$. In other words, on any path, $\mathcal{X}_k$ is the sum, over all down-ticks on the path, of the number of number of up-ticks so far. It is easy to visualize $\mathcal{X}_k$ for a specific path geometrically in the lattice diagram of Fig. 1: For any node (k, h) in the lattice, we define the **lowest path** reaching (k, h) as the path with $k - h$ down-ticks followed by h up-ticks. Similarly, the **highest path** reaching (k, h) is the one with h up-ticks followed by $k - h$ down-ticks. Then the area $\mathcal{X}_k(\omega)$ of any path ω reaching (k, h) is the number of diamond-shaped boxes that are enclosed between the k-prefix of ω and the lowest path reaching (k, h). The maximum possible area of any path reaching (k, h) is clearly the number of boxes between the highest and lowest paths reaching (k, h), which is $h(k - h)$. Also the minimum possible area of any path reaching (k, h) is 0.

As mentioned in the introduction we want to partition each node (k, h) of the binomial lattice into nodelets where each nodelet represents the paths arriving at node (k, h) with the same geometric stock price average. From the Lemma below, this is equivalent to partitioning the paths reaching a node according to their areas. We say that a path ω **reaches**, or **passes through** a **nodelet** (k, h, a) if

$$H_k(\omega) = h, \;\; \mathcal{X}_k(\omega) = a.$$

This means that for any path through a given nodelet (k, h, a), at time k it has h up-ticks and area a. Fig. 2 shows the nodelets in the nodes $(5, 2)$, $(6, 3)$ and $(6, 2)$.

We collect here some useful properties of the area $\mathcal{X}_k$.

LEMMA 1 *1. For a given node (k, h) in the lattice, there is a one-one correspondence between the possible geometric stock-price averages and the possible areas of the paths reaching (k, h).*

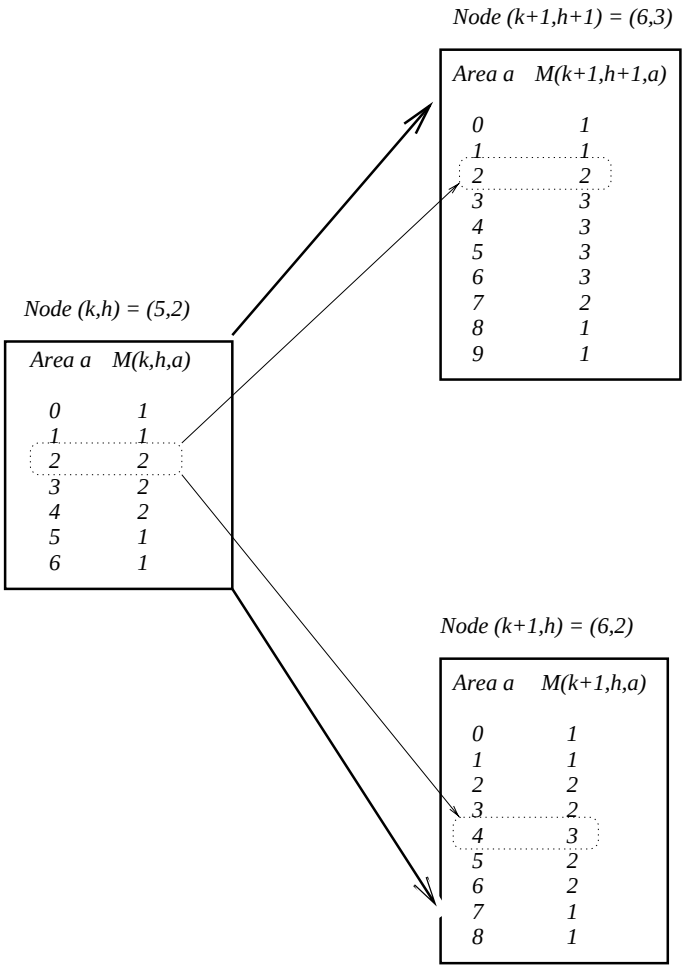

Figure 2. Nodelets inside the circled nodes $(5,2),(6,3),(6,2)$ of Fig. 1. Each box represents a node (k,h), and each row a in the box represents nodelet (k,h,a), i.e., the collection of paths reaching the node with area a. We have shown the number of paths $M(k,h,a)$ reaching each nodelet (k,h,a). An example of the forward induction update is shown: nodelet $(5,2,2)$ updates the values at nodelet $(6,3,2)$ and $(6,2,4)$.

2. *The set of possible areas of paths reaching (k, h) is*

$$\{0, 1, \ldots, h(k-h)\}. \tag{3}$$

3. *For a path A reaching a nodelet (k, h, a), if A has an up-tick at time $k+1$, it will reach nodelet $(k+1, h+1, a)$, and otherwise it will reach nodelet $(k+1, h, a+h)$.*

Proof: The geometric average G_k of the stock prices from time 0 to time k is:

$$\begin{aligned} G_k &= (S_0 S_1 \ldots S_k)^{1/(k+1)} \\ &= \left(\Pi_{i=0}^{k} S_0\, u^{2H_i - i}\right)^{1/(k+1)} \\ &= S_0\, u^{\frac{2\sum_{i=0}^{k} H_i - \sum_{i=0}^{k} i}{k+1}} \\ &= S_0\, u^{\frac{2\sum_{i=0}^{k} H_i - k(k+1)/2}{k+1}} \end{aligned}$$

The head-sum $\sum_{i=0}^{k} H_i$ in this expression can be written in terms of H_k and $\mathcal{X}_k$:

$$\begin{aligned} \sum_{i=0}^{k} H_i &= \sum_{i=0}^{k} X_i H_i + \sum_{i=0}^{k} (1 - X_i) H_i \\ &= \sum_{j=0}^{H_k} j + \mathcal{X}_k \\ &= H_k(H_k + 1)/2 + \mathcal{X}_k. \end{aligned} \tag{4}$$

Thus we can write G_k as

$$G_k = S_0\, u^{H_k(H_k+1)/(k+1) + 2\mathcal{X}_k/(k+1) - k/2}. \tag{5}$$

For a given H_k, G_k is a strictly increasing monotone function of $\mathcal{X}_k$, so for paths arriving at a given lattice node, there is a one-to-one correspondence between possible values of G_k and possible values of $\mathcal{X}_k$, which establishes the first part of the lemma.

Part 2 is easy to show geometrically. As already noted in the above discussion, the minimum and maximum possible areas of any path reaching node (k, h) are 0 and $h(k-h)$ respectively. We only need to establish that *every* integer between 0 and $h(k-h)$ is the area of some path reaching (k, h). To see this, consider any path reaching (k, h) that has area less than the maximum possible area $(k-h)h$. Such a path must contain somewhere a down-tick followed by an up-tick. Now if we replace this by an (up-tick,down-tick) pair, the new path has area one unit greater than before, and still reaches (k, h). Therefore every integer between 0 and $(k-h)h$ is the area of some path reaching (k, h).

Part 3 is also easily seen geometrically. Consider a path A reaching node (k, h) in the lattice. Suppose the area of A at time k is $\mathcal{X}_k(A) = a$. If A has an up-tick after this point, it reaches node $(k+1, h+1)$ at the next level of the lattice. In this case path A and the *lowest* path (call it B) reaching $(k+1, h+1)$ share the edge connecting (k, h) and $(k+1, h+1)$ in the lattice. Therefore the number of boxes enclosed between the $k+1$-prefixes of A and

n	H_n	$\mathcal{X}_n$	Paths	G_n	Min. A_n	$\mathbf{E}(A_n\|\mathcal{X}_n, H_n)$	Max. A_n	$\mathrm{var}(A_n\|\mathcal{X}_n, H_n)^{\frac{1}{2}}$
8	6	1	4	115.19	115.57	115.83	116.35	0.31
10	8	8	5	120.89	121.55	121.91	122.61	0.40
16	14	14	8	134.99	136.82	137.38	138.50	0.58
20	18	18	10	143.01	145.83	146.48	147.81	0.66
30	28	28	15	160.75	166.49	167.34	169.06	0.82
30	20	126	279260	127.61	128.05	128.56	130.64	0.33

Figure 3. Illustrating the small variance of the arithmetic stock-price average A_n over the set of paths that have the same geometric stock-price average G_n and up-ticks H_n. The parameters assumed are $S_0 = 100,\ L = 90,\ \sigma = .2$, and $r = .05$. "Paths" indicates the number of paths with the indicated value of H_n (the number of up-ticks) and $\mathcal{X}_n$ (the area). "Min. A_n," "Max. A_n," and $\mathbf{E}(A_n|\mathcal{X}_n, H_n)$ respectively indicate the smallest, largest and average value of A_n over this set of paths; all these paths have the same value of G_n. Finally, $\mathrm{var}(A_n|\mathcal{X}_n, H_n)$ is the variance of A_n over this set of paths. Note the significant difference between the arithmetic and geometric stock-price averages.

B is the same as the number of boxes between the k-prefixes of these paths, which is a. Therefore the path A reaches nodelet $(k+1, h+1, a)$. On the other hand suppose A has a down-tick at time $k+1$, which means A reaches node $(k+1, h)$. It is easy to see from the lattice diagram of Fig. 1 that the number of boxes between the $k+1$-prefixes of A and the lowest path reaching $(k+1, h)$ is now $a+h$. This means the path A reaches nodelet $(k+1, h, a+h)$. ■

In our algorithms, we will need to compute the *average of the arithmetic stock-price average* over all paths reaching a given nodelet (k, h, a), which we denote by $\widetilde{A}(k, h, a)$:

$$\widetilde{A}(k, h, a) = \mathbf{E}[A_k | H_k = h, \mathcal{X}_k = a], \quad k = 0, 1, \ldots, n,\ h \le k. \tag{6}$$

Since all paths through nodelet (k, h, a) have the same probability, this is simply the (unweighted) average of A_k over these paths.

To see how to compute $\widetilde{A}(k, h, a)$ we introduce the random variable $\mathbb{S}_k$ defined as the sum of the stock prices from time 0 to k:

$$\mathbb{S}_k = \sum_{i=0}^{k} S_i, \quad k = 0, 1, \ldots, n. \tag{7}$$

Note that $A_k = \mathbb{S}_k/(k+1)$ for $k = 0, 1, \ldots, n$. We also define $\mathcal{S}(k, h, a)$ as the sum of $\mathbb{S}_k$ over all paths passing through the nodelet (k, h, a). Since $\widetilde{A}(k, h, a)$ is simply the unweighted average of A_k over all paths reaching the nodelet (k, h, a), it follows that

$$\widetilde{A}(k, h, a) = \frac{\mathcal{S}(k, h, a)}{(k+1)M(k, h, a)}, \tag{8}$$

where $M(k, h, a)$ is the number of paths reaching nodelet (k, h, a). Our algorithm computes the quantities $M(k, h, i)$ and $\mathcal{S}(k, h, i)$ by forward induction on k as follows. First,

for $k = 1, 2, \ldots, n$, $h = 0, 1, \ldots, k$ and $a = 0, 1, \ldots, h(k-h)$, we initialize $M(k,h,i)$ and $\mathcal{S}(k,h,i)$ to 0. To start the induction we set, in accordance with their definitions, $M(0,0,0) = 1$ and $\mathcal{S}(0,0,0) = S_0$. Let us assume inductively that we have computed all the $M(m,h,a)$ and $\mathcal{S}(m,h,a)$ values for $m = 0, 1, \ldots, k$, and we want to compute these quantities at level $m = k+1$. To do this we consider each nodelet (k,h,a) at level k and increment M and $\mathcal{S}$ at the appropriate nodelets at level $k+1$ by an amount equal to the contribution made by paths reaching nodelet (k,h,a).

Specifically, we proceed as follows. Suppose we are considering the contribution of nodelet (k,h,a) at level k to nodelets at level $k+1$. By Lemma 1, any path through nodelet (k,h,a) that has an up-tick at time $k+1$ must pass through nodelet $(k+1, h+1, a)$ at level $k+1$. Thus the paths reaching nodelet $(k+1, h+1, a)$ via (k,h,a) contribute an amount $M(k,h,a)$ to $M(k+1,h+1,a)$, so in the forward induction we increment $M(k+1,h+1,a)$ by $M(k,h,a)$. Also, the paths reaching $(k+1,h+1,a)$ via (k,h,a) contribute

$$\mathcal{S}(k,h,a) + M(k,h,a)S_k = \mathcal{S}(k,h,a) + M(k,h,a)S_0 u^{2(h+1)-(k+1)}$$

to $\mathcal{S}(k+1,h+1,a)$, so we increment the latter by this amount. Similarly, by Lemma 1, all paths through nodelet (k,h,a) that have a down-tick at time $k+1$ must pass through nodelet $(k+1, h, a+h)$ at level $k+1$. By a similar reasoning as above, in the forward induction, we increment $M(k+1,h,a+h)$ by $M(k,h,a)$, and we increment $\mathcal{S}(k+1,h,a+h)$ by

$$\mathcal{S}(k,h,a) + M(k,h,a)S_0 u^{2h-(k+1)}.$$

Example. We illustrate the computations involved in the forward induction by means of the nodelets shown in Fig. 2. Suppose we have computed all the $M(k,h,a)$ and $\mathcal{S}(k,h,a)$ at level $k = 5$, and we are considering the contribution of nodelet $(k,h,a) = (5,2,2)$ to the nodelets at level 6. By Lemma 1, any path through this nodelet that has an up-tick at time 6 will go through the nodelet $(5,3,2)$ at level 6. The contribution of nodelet $(5,2,2)$ to the M-value at nodelet $(5,3,2)$ is $M(5,2,2)$, so we add $M(5,2,2)$ to the current value of $M(6,3,2)$. Similarly, by Lemma 1 any path through this nodelet that has an up-tick at time 6 will go through the nodelet $(6,2,4)$ at level 6. We therefore add $M(5,2,2)$ to $M(6,2,4)$. Note that some other nodelet in node $(3,2)$ will contribute 1 to the M value at nodelet $(6,2,4)$, so eventually $M(6,2,4)$ will reach the correct value 3 as shown in the figure. The updates of the $\mathcal{S}$ values are done in a similar manner. ■

In summary, the M and $\mathcal{S}$ values are computed as follows. To start the induction we set $M(0,0,0)$ to 1 and $\mathcal{S}(0,0,0)$ to S_0, and initialize $M(k,h,a)$ and $\mathcal{S}(k,h,a)$ to 0 for $k = 1, \ldots, n$, $h = 0, \ldots, k$ and $a = 0, \ldots, h(k-h)$. Then the forward induction in pseudocode is shown in Fig. 4.

3.1. Time and space complexity

We now analyze the time and space required by our algorithm. The quantities $M(k,h,a)$ and $\mathcal{S}(k,h,a)$ are stored in a 3-dimensional array, where the index k ranges from 0 to n, the index h ranges from 0 to k, and the index i ranges from 0 to $h(k-h)$ (see expression

```
for k := 0 to n − 1 do
    for h := 0 to k do
        for a := 0 to h(k − h) do
            for v := 0 to 1 do
                j := a + (1 − v)h;
                M(k + 1, h + v, j) += M(k, h, a);
                S(k + 1, h + v, j) += S(k, h, a) + M(k, h, a)S_0 u^{2(h+v)−(k+1)};
            end
        end
    end
end
```

Figure 4. Pseudo-code for the forward induction to compute the M and $\mathcal{S}$ values at the nodelets of the refined lattice. We use the standard notation $x += y$ to stand for the assignment $x := x + y$.

(3) of Lemma 1). Any element of these arrays can be accessed in essentially a constant amount of time. It is clear that the innermost initialization loop above is executed once for each nodelet (k, h, a) in the lattice, whereas the innermost loop in the forward induction is executed twice for each nodelet up to level $n-1$ in the lattice. For a given nodelet, the body of the induction loop consists only of simple additions, multiplication, exponentiation and array-access operations. For our asymptotic time analysis we will consider these operations as taking a "constant" amount of time. Thus the total time of the algorithm is proportional to the number of nodelets in the lattice from level 0 to level n, and we show below that this number is smaller than $n^4/20$ for $n \geq 14$. Thus the total time required by the algorithm is proportional to n^4. Since there is a constant amount of storage corresponding to each nodelet, the total space requirement of the algorithm is also proportional to n^4.

LEMMA 2 *The total number of nodelets in an n-step refined binomial lattice is proportional to n^4, and for $n \geq 14$, this number is smaller than $n^4/20$.*

Proof: The number of nodelets at a given node (k, h) is $1 + h(k - h)$, so the total number of nodelets at level k in the lattice is

$$\begin{aligned}\sum_{h=0}^{k}(1 + kh - h^2) &= 1 + k.k(k+1)/2 - k(k+1)(2k+1)/6 \\ &= 1 + (k^3 - k)/6,\end{aligned}$$

and the total number of nodelets from level 0 to level n is

$$\begin{aligned}\sum_{k=0}^{n}\left[1 + (k^3 - k)/6\right] &= (n+1) + \frac{1}{6}\sum_{k=1}^{n}k^3 - \frac{1}{6}\sum_{k=1}^{n}k \\ &= n + n^2(n+1)^2/24 - n(n+1)/12 \\ &= (n^4 + 2n^3 - n^2)/24 + 11n/12 + 1 \\ &\leq n^4/20 \quad \text{for } n \geq 14.\end{aligned}$$

■

4. Upper bound: interpolation

Once the quantities $\widetilde{A}(k,h,a)$ have been computed at all the nodelets, we are ready to compute the upper bound on the option price. The algorithm works along the same lines as the crude algorithm described in the introduction, except that we work on the refined lattice rather than on the original binomial lattice. The algorithm uses an interpolation idea similar to the one in Hull-White [6].

We begin by defining the **value function** $V(k,h,x)$ as "value of the American Asian option at time k, *given* that the number of up-ticks H_k so far is h, *and* the arithmetic stock-price average A_k equals x ", or more precisely,

$$V(n,h,x) = (x-L)^+, \quad h \le n$$

$$V(k,h,x) = \max\left\{(x-L)^+, \right. \tag{9}$$
$$\left. (1/R)\left(pV(k+1,h+1,x^u(k,h)) + (1-p)V(k+1,h,x^d(k,h))\right)\right\},$$
$$k < n,\ h \le k, \tag{10}$$

where

$$x^u(k,h) = [x(k+1) + S_0\, u^{2h+2-k-1}]/(k+2)$$

is the arithmetic stock-price average A_{k+1} given that $H_k = h, A_k = x$ and there is an up-tick at time $k+1$, and

$$x^d(k,h) = [x(k+1) + S_0\, u^{2h-k-1}]/(k+2)$$

is the arithmetic stock-price average A_{k+1} given that $H_k = h, A_k = x$ and there is a down-tick at time $k+1$. Note that there may not necessarily exist a path on which $H_k = h$ and $A_k = x$, so the arguments of the value function $V(k,h,x)$ are not always "legal". However, for any path ω such that $H_k(\omega) = h$ and $A_k(\omega) = x$, the value of the option at time k on ω is $V(k,h,x)$. In particular, $V(0,0,S_0)$ is the option value at time 0.

Our goal is to estimate the value $V(0,0,S_0)$, and in this section we show how we can get a tight upper-bound. The algorithm estimates the function at combinations (k,h,x) of the form $(k,h,\widetilde{A}(k,h,a))$ for $a = 0,1,\ldots,h(k-h)$. In other words, at each lattice node (k,h), the value function V is estimated only at the "representative" arithmetic averages $\widetilde{A}(k,h,a)$ corresponding to the nodelets in the node. We denote our estimate of $V(k,h,x)$ by $W'(k,h,x)$. We compute the estimates $W'(k,h,\widetilde{A}(k,h,a))$ backward-recursively in a manner analogous to the definition of V (equations (10)). For brevity we denote $W'(k,h,\widetilde{A}(k,h,a))$ by $W(k,h,a)$. Thus we begin by setting

$$W(n,h,a) = [\widetilde{A}(n,h,a) - L]^+$$

for all $h \le n$, and $a = 0,1,\ldots,h(n-h)$. However for $k < n$, we cannot directly use the backward recursion (10) with V replaced by W'. This is because the quantity $x^u(k,h)$

will in general *not* equal a representative average $\widetilde{A}(k+1,h+1,a')$ at node $(k+1,h+1)$ and so $W'(k+1,h+1,x^u(k,h))$ would not have been computed backward recursively. Similarly, $x^d(k,h)$ will not necessarily be a representative average at node $(k+1,h)$. We therefore use linear interpolation for these "missing" W values. A similar idea is used by Hull and White [6] to compute an upper bound on the Asian option price (see Section 6 for a more detailed comparison with their method). That is, for a given $x = \widetilde{A}(k,h,a)$, we find (the lemma below shows that this can always be done) a b such that for some $0 \le \lambda \le 1$,

$$x^u(k,h) = \lambda \widetilde{A}(k+1,h+1,b) + (1-\lambda)\widetilde{A}(k+1,h+1,b+1),$$

and on the right hand side of (10), instead of $V(k+1,h+1,x^u(k,h))$ we use

$$W^u(k,h,a) = \lambda w_1 + (1-\lambda)w_2,$$

where $w_1 = W(k+1,h+1,b)$ and $w_2 = W(k+1,h+1,b+1)$. Similarly instead of $V(k+1,h,x^d(k,h)$ we use an interpolated value $W^d(k,h,x)$. Our actual backward recursion is therefore, for $x = \widetilde{A}(k,h,a)$, $a = 0,1,\ldots,h(k-h)$,

$$W(k,h,a) = \max\Big\{[\widetilde{A}(k,h,a) - L]^+, \\ (1/R)\left(pW^u(k,h,a) + (1-p)W^d(k,h,a)\right)\Big\}, \qquad (11)$$

where W^u, W^d are computed as above. From the fact that $V(k,h,x)$ is a convex function of x, as the following lemma shows, it follows that the estimate $W(0,0,0)$ is an upper bound on the option value at time 0.

LEMMA 3 *1. For fixed k and h, $\widetilde{A}(k,h,a)$ is a strictly monotone increasing function of a.*

2. For any $a = 0,1,\ldots,h(k-h)$ if $x = \widetilde{A}(k,h,a)$, then there exists a b such that

$$\widetilde{A}(k+1,h+1,b) \le x^u(k,h) \le \widetilde{A}(k+1,h+1,b+1),$$

and similarly there exists a b such that

$$\widetilde{A}(k+1,h,b) \le x^d(k,h) \le \widetilde{A}(k+1,h,b+1).$$

3. The estimate $W(k,h,a)$ is an upper bound on the value function $V(k,h,\widetilde{A}(k,h,a))$, and in particular $W(0,0,0)$ upper bounds the option value $V(0,0,S_0)$.

Proof: The first part can be seen from the geometric interpretation of the area of a path. For every path ω reaching a node (k,h) with area a, there is a path ω' reaching the same node with area $a+1$ that *dominates* ω, i.e., for each $i \le k$, the stock price $S_i(\omega') \ge S_i(\omega)$, with strict inequality holding for at least one i. Thus $A_k(\omega') > A_k(\omega)$, and therefore the average of $A_k(\omega')$ over all ω' that reach nodelet $(k,h,a+1)$ is strictly greater than the average of $A_k(\omega)$ over all ω that reach (k,h,a).

Next we claim that if $x = \widetilde{A}(k,h,a)$ then $x^u(k,h)$ always lies between the minimum representative average $\widetilde{A}(k+1,h+1,0)$ and the maximum $\widetilde{A}(k+1,h+1,(h+1)(k-h))$ at node $(k+1,h+1)$. To see this, first note that at any node (k,h) there is only one path reaching each of the extreme nodelets $(k,h,0)$ and $(k,h,h(k-h))$ at that node, and so the quantities $\widetilde{A}(k,h,0)$ and $\widetilde{A}(k,h,h(k-h))$ are respectively the minimum and maximum possible values of A_k at node (k,h). Now we know that $x = \widetilde{A}(k,h,a)$ for some nodelet (k,h,a), so x lies between the minimum and maximum possible A_k values (call them w and y respectively) at node (k,h). Therefore, $x^u(k,h)$ lies between $w^u(k,h)$ and $y^u(k,h)$, and the latter quantities are actual values of A_{k+1} for some path reaching node $(k+1,h+1)$. Therefore $x^u(k,h)$ must lie between the minimum and maximum possible values of A_{k+1} at node $(k+1,h+1)$, namely $\widetilde{A}(k+1,h+1,0)$ and $\widetilde{A}(k+1,h+1,(h+1)(k-h))$. The analogous claim for $x^d(k,h)$ is shown similarly. This claim combined with part (1) of the lemma implies part (2).

For part (3), observe that since the payoff function $(x-L)^+$ is a convex function of x, so is the value function $V(k,h,x)$. By definition, $W(n,h,a) = V(n,h,\widetilde{A}(k,h,a))$. Now assume inductively that $W(m,h,a) \geq V(m,h,\widetilde{A}(m,h,a))$ for $m = k+1,\ldots,n$ and all $h \leq m$ and $a = 0,1,\ldots,h(m-h)$. Consider the backward recursion (11), with $x = \widetilde{A}(k,h,a)$, and w_1, w_2, λ defined as in that paragraph. We know that $x^u(k,h)$ is a convex combination $\lambda x_1 + (1-\lambda)x_2$ where $x_1 = \widetilde{A}(k+1,h+1,b)$, $x_2 = \widetilde{A}(k+1,h+1,b+1)$. By induction assumption, $w_1 \geq v_1 = V(k+1,h+1,x_1)$ and $w_2 \geq v_2 = V(k+1,h+1,x_2)$. Therefore

$$W^u(k,h,x) \geq \lambda v_1 + (1-\lambda)v_2,$$

and since $V(k,h,x)$ is a convex function of x, the right hand side is at least $V(k+1,h+1,x^u(k,h))$. Similarly we can show that the quantity $W^d(k,h,x)$ is at least $V(k+1,h,x^d(k,h))$. Therefore the right hand side of the backward recursion (11) is at least as large as the right hand side of the backward recursion (10), which implies the second part of the lemma. ■

4.1. Time complexity

We analyze the time required by our backward-recursion-interpolation procedure. When the algorithm is computing the estimate W at a nodelet (k,h,a), if we denote $\widetilde{A}(k,h,a)$ by x, it needs to find a b such that $x^u(k,h)$ lies between $\widetilde{A}(k+1,h+1,b)$ and $\widetilde{A}(k+1,h+1,b+1)$, and it needs to find a c such that $x^d(k,h)$ lies between $\widetilde{A}(k+1,h,c)$ and $\widetilde{A}(k+1,h,c+1)$. The existence of such a b and c is guaranteed by Lemma 3, part (2). Since part (1) of that lemma says that the $\widetilde{A}(k,h,a)$ is monotone increasing in a, the areas b and c can be found by a simple binary search on the nodelets at node $(k+1,h+1)$ and at node $(k+1,h)$ respectively. In general the binary search would require probing a number of nodelets which is proportional to the logarithm (base 2) of the total number of nodelets in the node, which is of the order of kh (Lemma 1). In our implementation we employ some heuristics that considerably speed up discovery of the right nodelet – it is usually found after 3 or 4 probes even for $n = 60$. One such heuristic is based on the

monotonicity of $\widetilde{A}(k,h,a)$ w.r.t. a: Suppose that for nodelet (k,h,a) we have already found a b in node $(k+1,h+1)$ with the required properties. Then for the next nodelet $(k,h,a+1)$ at node (k,h), the monotonicity property guarantees us that the corresponding b will be no smaller than the b for nodelet (k,h,a), so we have a smaller set of nodelets to search among. In practice our upper-bound computation takes time proportional to about 3 or 4 times the total number of nodelets in the refined lattice, which is less than $n^4/20$ for an n-step lattice.

5. Lower bound: estimating a good exercise rule

As mentioned in the introduction, we compute a lower bound on the American Asian option price by first estimating a "good" exercise rule for the option from the above upper-bound computation. Any exercise rule can be described by a *stopping time* τ. Once we fix an exercise rule τ, for any adapted process Z_k (i.e. Z_k is $\mathcal{F}_k$-measurable for each k), the value of the option (given by (1)) is lower-bounded by

$$\begin{aligned}\mathbf{E}\left[\frac{(A_\tau - L)^+}{R^\tau}\right] &= \mathbf{E}\left[\mathbf{E}\left(\frac{(A_\tau - L)^+}{R^\tau}\middle| Z_\tau\right)\right]. \\ &\geq \mathbf{E}\left[\mathbf{E}\left(\frac{A_\tau - L}{R^\tau}\middle| Z_\tau\right)^+\right],\end{aligned}$$

where the inequality follows from Jensen's inequality since $f(x) = x^+$ is a convex function. This type of lower bound has been used for European Asian options by Rogers and Shi [12] and Chalasani, Jha and Varikooty [3]. If $Z_i \neq Z_j$ whenever $i \neq j$, then R^τ is measurable with respect to the σ-algebra generated by Z_τ, so we can write the above lower bound as

$$\mathbf{E}\left[\frac{\mathbf{E}\,(A_\tau - L|Z_\tau)^+}{R^\tau}\right] = \mathbf{E}\left[\frac{(\mathbf{E}(A_\tau|Z_\tau) - L)^+}{R^\tau}\right]. \tag{12}$$

For our lower bound we use the quantity on the right in (12) with the random variable $Z_k = (k, H_k, \mathcal{X}_k)$. Note that each possible value of the random variable Z_k corresponds to a nodelet in the refined lattice. For the exercise rule τ, we use a rule that is based on the upper-bound computation described above. During the upper-bound computation, whenever $k = n$, or $k < n$ and

$$W(k,h,a) \geq [pW^u(k,h,a) + (1-p)W^d(k,h,a)]/R,$$

in the backward recursion (11), we set stop$(k,h,a) = 1$, and set stop$(k,h,a) = 0$ otherwise. If stop$(k,h,a) = 1$ we say (k,h,a) is a **stopped** nodelet. Imagining we are moving forward along a path on the binomial tree (or equivalently, through the refined lattice), the rule we want to use is: *Exercise the option at the current nodelet* (k,h,a) *if and only if it is a stopped nodelet.* In other words, on any path ω, the exercise point is the *earliest* time k such that stop$(k, H_k(\omega), \mathcal{X}_k(\omega)) = 1$. Formally, the stopping time τ corresponding to this rule is the random variable

$$\tau = \min\{k : \text{stop}(k, H_k, \mathcal{X}_k) = 1\}.$$

This choice of exercise rule turns out to give very tight lower bounds, as our numerical experiments show in the next section.

To complete the description of our algorithm, we describe how the lower bound (12) is computed. For any path ω, $Z_\tau(\omega)$ specifies not only an exercise time $k = \tau(\omega)$, but also a nodelet $(k, H_k(\omega), \mathcal{X}_k(\omega))$. Therefore for any given nodelet (k, h, a), the inner conditional expectation

$$\mathbf{E}\left(A_\tau | Z_\tau = (k, h, a)\right)$$

can be written as

$$A'(k, h, a) = \mathbf{E}\left(A_k | \tau = k, H_k = h, \mathcal{X}_k = a\right).$$

Thus $A'(k, h, a)$ is simply the (unweighted) average of the arithmetic stock-price average A_k over all paths that are *exercised* at nodelet (k, h, a). Note that A' is similar to $\widetilde{A}$ (see (6)). The only difference is that to compute $A'(k, h, a)$ we average only over those paths that are exercised at (k, h, a) and not over all paths reaching this nodelet. We can therefore compute $A'(k, h, a)$ at each nodelet (k, h, a) using essentially the same forward induction (see Fig. 4) for computing the $\widetilde{A}$ values, except that we execute the innermost loop only if $\text{stop}(k, h, a) = 0$. That is, we forward-propagate the M and $\mathcal{S}$ values only if the current nodelet is *not* a stopped node. Once all the A' values are computed, the expectation (12) is computed backward-recursively in the obvious way: At each nodelet (k, h, a), we compute a quantity $Y(k, h, a)$ as follows, starting from $k = n$ and moving backward on the lattice. If $\text{stop}(k, h, a) = 1$ we set $Y(k, h, a) = (A'(k, h, a) - L)^+$, and otherwise we set

$$Y(k, h, a) = [pY(k+1, h+1, a) + (1-p)Y(k+1, h+1, a+h)]/R. \qquad (13)$$

It is easy to see that $Y(0, 0, 0)$ equals the expectation in (12), and is therefore a lower bound on the option value.

5.1. Time complexity

As mentioned above, to compute the lower bound, we first need to do a forward induction similar to the one described in Section 3 in order to compute the A' values at each nodelet. Next, the Y values are computed at the nodelets using backward recursion. Each of these takes time proportional to the total number of nodelets in the refined lattice, which is less than $n^4/20$ for $n \geq 14$.

6. Comparison with the Hull-White Method

In this section we compare the accuracy of our algorithms with those of Hull-White (HW) [6], which is the previously best-known approximation method for American Asian options. One important advantage of our method is that we produce both an *upper and lower bound* on the option value, whereas the HW method only produces an upper bound. Since

there are no satisfactory Monte Carlo algorithms to estimate American Asian option values, in the HW method there is no way of knowing how far the results are from the true value.

Our upper bound procedure is in fact inspired by the HW interpolation method. To make a detailed comparison with their method, it is important to understand how it works. Recall that in our upper-bound computation we compute the option value estimate W only at certain "representative values" of the arithmetic average, namely the $\widetilde{A}$ values at the nodelets in each node. We use linear interpolation to obtain the missing values needed for backward recursion. The HW method works the same way, except that at each lattice level $k = 1, 2, \ldots, n$, the value estimate is computed at certain *special* arithmetic average values of the form $S_0 e^{mh}$ where m is a positive or negative integer, and h is a chosen *grid size* (such as 0.01). At each level k, all integer values of m in a certain range are considered, and this range is chosen so that the minimum and maximum values of $S_0 e^{mh}$ span the set of all possible values of the arithmetic average A_k.

A reasonable question is whether (as we do for our lower-bound computation) a good exercise rule can be extracted from the upper-bound computation in the Hull-White method. We see no way of doing this since the grid points (i.e., the points with A_k values of the form e^{mh}) do not correspond to actual paths in the binomial tree. For instance we cannot say "exercise at time k if the arithmetic stock price average A_k is e^{mh}." In our algorithm, on the other hand, the "grid" points are associated with the nodelets, which do correspond to actual paths in the binomial tree. We can therefore formulate an exercise rule that says whether or not to exercise depending on the current nodelet.

Figure 5 shows a detailed comparison of the option value estimates computed by the HW method (with $h = 0.005$ as reported in [6]) and our method. In can be seen that in all cases, the HW upper-bound is higher than our upper-bound, indicating that our bound is tighter than theirs. Moreover, the gap between our lower and upper bounds is in most cases on the order of 0.001, or 0.002 % of the initial stock price. In Figure 6 we compare the running times of our implementation of the HW algorithm with those of our method. Notice that for each n (except $n = 20$) in the table, to (roughly) match our upper-bound value, the HW algorithm must be run with a grid size $h = 0.002$; for this grid size, the HW algorithm is somewhat faster than ours for larger n. In contrast to the HW algorithm, however, we are able to extract a lower bound from our upper-bound computation, and the additional time required for this is small compared to the upper-bound computation time. One limitation of our method is that for a given n we are restricted to a fixed set of nodelets, which defines a fixed grid for the interpolation. We suspect that it should be possible to vary the number of nodelets according to some parameter (and in particular use fewer nodelets), and reduce the upper-bound computation time without loosing accuracy.

7. Conclusion and Future Work

In this paper we introduced simple, elegant and fast algorithms for computing very tight lower and upper bounds on the value of an American Asian option. We introduced a way of refining the traditional binomial lattice by conditioning on the "area" of a path, which is closely related to the geometric stock-price average. In the refined lattice, each node is partitioned into "nodelets" that represent the cluster of paths that reach the node with a given

Option Life T	Strike L	HW	LB	UB
0.5	40	12.115	12.111	12.112
	45	7.261	7.255	7.255
	50	3.275	3.269	3.269
	55	1.152	1.148	1.148
	60	0.322	0.320	0.320
1.0	40	13.153	13.150	13.151
	45	8.551	8.546	8.547
	50	4.892	4.888	4.889
	55	2.536	2.532	2.534
	60	1.208	1.204	1.206
1.5	40	13.988	13.984	13.985
	45	9.652	9.648	9.650
	50	6.199	6.195	6.197
	55	3.771	3.767	3.770
	60	2.194	2.190	2.193
2.0	40	14.713	14.709	14.712
	45	10.623	10.620	10.623
	50	7.326	7.322	7.325
	55	4.886	4.882	4.885
	60	3.171	3.167	3.170

Figure 5. Comparison with Hull-White (HW) [6] (grid size $h = 0.005$) with $n = 40$ steps, volatility $\sigma = 0.3$ per year, continuously-compounded risk-free interest-rate $r = 0.1$ per year, initial stock price $S_0 = 50$ dollars, and strike price $K = 50$ dollars. The option life T is in years. Our lower and upper bounds are indicated by LB and UB respectively.

Steps n	HW (h=0.05)	HW (h=0.005)	HW (h=0.003)	HW (h=0.002)	UB	LB
20	4.971 (.1)	4.815 (.4)	4.814 (.5)	4.813 (.8)	4.814 (.2)	4.812 (.1)
40	5.080 (.6)	4.892 (2)	4.890 (3)	4.889 (5)	4.889 (4)	4.888 (1)
60	5.124 (2)	4.924 (6)	4.920 (10)	4.918 (13)	4.918 (18)	4.917 (4)
80	5.145 (3)	4.942 (13)	4.936 (20)	4.935 (29)	4.934 (59)	4.933 (15)

Figure 6. Comparison with Hull-White (HW) [6], using different time steps n, and different grid-sizes h (for the Hull-White method), with option life $T = 1.0$ year, volatility $\sigma = 0.3$ per year, continuously-compounded risk-free interest-rate $r = 0.1$ per year, initial stock price $S_0 = 50$ dollars, and strike $L = 50$ dollars. Our lower and upper bounds are indicated by LB and UB respectively. Computation times on a Sun Ultra SPARC 1 machine are shown in parentheses in seconds. In column LB, the *additional* time to compute our lower bound is shown. Note that in all cases the HW upper-bound with $h = 0.002$ matches our upper-bound.

area. The upper-bound is obtained by backward-recursively computing a value estimate at each nodelet, and using linear interpolation to compute missing values. We presented a novel way to guess a good exercise rule for the American option, based on the upper-bound computation: exercise the option at the current nodelet as soon as the immediate-exercise value equals the value-estimate computed during the upper-bound calculation. Such path-clustering/conditioning ideas can have applications to other path-dependent options besides Asians. We are currently studying such an approach for the valuation of mortgage-backed securities (MBS), which often have path-dependent pre-payment functions.

Notes

1. In particular, we do not consider Asian options where the average is taken over certain specified sampling intervals, such as weekly closing prices, etc. Thus, as the number of time-divisions in the binomial model goes to infinity, so does the number of points over which the stock-price average is taken.

References

1. F. Black and M.S. Scholes. The pricing of options and corporate liabilities. *J. Political Economy*, 81:637–654, 1973.
2. A.P. Carverhill and L.J. Clewlow. Flexible convolution. *RISK*, pages 25–29, April 1990.
3. P. Chalasani, S. Jha, and A. Varikooty. Accurate approximations for european asian options. *Journal of Computational Finance*, 1(4), 1998.
4. J. Cox, S. Ross, and M. Rubinstein. Option pricing: A simplified approach. *J. Financial Economics*, 7:229–264, 1979.
5. D. Duffie. *Dynamic Asset Pricing Theory*. Princeton University Press, 2 edition, 1996.
6. J. Hull and A. White. Efficient procedures for valuing european and american path-dependent options. *Journal of Derivatives*, 1:21–31, 1993.
7. A.G.Z. Kemna and A.C.F. Vorst. A pricing method for options based upon average asset values. *J. Banking and Finance*, pages 113–129, March 1990.
8. E. Levy. Asian arithmetic. *RISK*, pages 7–8, May 1990.
9. E. Levy. Pricing european average rate currency options. *J. International Money and Finance*, 11:474–491, 1992.
10. E. Levy and S. Turnbull. Average intelligence. *Risk Magazine*, Feburary 1992.
11. P. Ritchken, L. Sankarasubramanian, and A.M. Vijh. The valuation of path-dependent contracts on the average. *Management Science*, 39(10):1202–1213, 1993.
12. L.C.G. Rogers and Z. Shi. The value of an asian option. *J. Appl. Prob.*, 32:1077–1088, 1995.
13. A. Ruttiens. Classical replica. *RISK*, pages 5–9, Feb 1992.
14. S. Turnbull and L.M. Wakeman. A quick algorithm for pricing european average options. *J. Financial and Quantitative Analysis*, 26(3):377–390, 1991.
15. M. Yor. On some exponential functionals of brownian motion. *Adv. Appl. Prob.*, 1992.