Polynomial Algorithms for pricing path dependent contracts

H. Moritsch, G.Ch. Pflug

AURORA TR2002-32
December 2002

Special Research Program SFB F011 "AURORA" funded by the Austrian Science Fund
Subproject 6

Polynomial Algorithms for pricing path dependent contracts

H. Moritsch and Georg Ch. Pflug [1]

Abstract

In this paper we study algorithms for pricing of financial contracts using a lattice process of interest rates. If the cash-flows generated by the contract depend on the history of the interest rates (path dependent contracts), then the pricing algorithms are typically of exponential complexity. We demonstrate that for some models, there exist polynomial algorithms for path dependent contracts. These models include product form cash-flows, additive cash-flows, delayed cash-flows and limited path dependent cash-flows.

1 Introduction

Pricing of financial contracts on the basis of a calibrated scenario lattice of interest rates and a given cash-flow function using the general formula *"present value equals the expected, discounted cash-flow"* is one basic problem of computational finance.

It is well known that the problem is of polynomial order if the lattice has polynomial growth and the cash-flows may be represented on the lattice. However, there are many examples of "exotic" contracts, which define the payable cash-flows as a function of the whole history of the interest rate process. In general, the history tree grows exponentially and therefore the complexity of pricing such contracts is non-polynomial. For certain subclasses of cash-flow functions however, a polynomial pricing algorithm exists.

As an example, Figure 1 shows the *lookback cap* instrument [PS97] with its coupon payment specified as

$$c(t) \quad = \quad d \max\{\xi_(\tau) - \rho : t - d \leq \tau \leq t\}$$

where ξ_t denotes the LIBOR (London Interbank Offered Rate). The cash flow depends on the path of interest rates from time $t - d$ to t.

[1]Department of Statistics and Decision Support Systems, University of Vienna, 1090 Wien-Vienna, Universitaetsstrasse 5

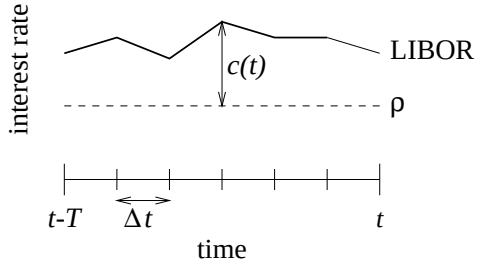

Figure 1: Lookback cap instrument

The paper is organized as follows. In section 2, we describe the setup and the brute force exponential algorithm. Section 3 presents models with polynomial algorithms. In section 4, the algorithm for limited path dependent cash-flows is discussed in detail and computational results are presented.

2 The setup of the problem

2.1 Lattice and tree processes

Let $(\xi_t), t = 0, \ldots, T$ be a (possibly imhomogeneous) Markov process on $\mathbb{Z}$ with $\xi_0 = 0$. Denote by $p_{t,i,j}$ the transition probabilities $p_{t,i,j} = p^{(1)}_{t,i,j} = \mathbb{P}\{\xi_{t+1} = j | \xi_t = i\}$ and by $p^{(k)}_{t,i,j}$ the k-fold transition probabilities $p^{(k)}_{t,i,j} = \mathbb{P}\{\xi_{t+k} = j | \xi_t = i\}$.

Let S_t be the set of states which are reachable at time t from the starting point $\xi_0 = 0$. In particular, $S_0 = \{0\}$. Let $\Pi_\ell(s, j)$ be the set of all possible paths of length ℓ starting in state j at time t.

The time-state lattice $\mathcal{L}$ consists of nodes (t, i); $i \in S_t, t = 0, \ldots, T$. Nodes (t, i) and $(t + 1, j)$ are linked by an arc, if $p_{t,i,j} > 0$. The arc carries the probability $p_{t,i,j}$.

In contrast, the history tree $\mathcal{T}$ consists of nodes $(0, i_1, \ldots, i_t)$; $i_s \in S_s, t = 0, \ldots T$. A sequence $(0, i_1, \ldots, i_t)$ is a valid node, if $\mathbb{P}\{\xi_1 = i_1, \ldots \xi_t = i_t\} > 0$. Nodes $(0, i_1, \ldots i_t)$ and $(0, i_1, \ldots i_t, i_{t+1})$ are linked by an arc, if $p_{t,i_t,i_{t+1}} > 0$. The root of the tree is the node (0). The leaves $(0, i_1, \ldots, i_T)$ carry the probabilities $p_{0,0,i_1} \cdot p_{1,i_1,i_2} \cdots p_{T-1,i_{T-1},i_T}$.

Example 1. The symmetric binomial random walk

The symmetric binomial random walk is characterized by

$$p_{t,i,i+1} = p_{t,i,i-1} = 1/2.$$

The reachability set S_t is

$$S_t = \begin{cases} -t, -t+2, \ldots, -1, 1, \ldots, t-2, t & \text{if } t \text{ is odd} \\ -t, -t+2, \ldots, -2, 0, 2, \ldots, t-2, t & \text{if } t \text{ is even} \end{cases}$$

The time-state lattice is

$$\mathcal{L} = \{(t, j) : j = -t, -(t-2), \ldots, t-2, t; t = 1, \ldots T\}.$$

The number of nodes is $1 + 2 + \ldots + (T+1) = (T+1)(T+2)/2$. The pertaining history tree $\mathcal{T}$ has $1 + 2 + \ldots + 2^T = 2^{T+1} - 1$ nodes.

Example 2. The trinomial random walk

The symmetric trinomial random walk is characterized by

$$p_{t,i,i+1} = p_{t,i,i-1} = q; p_{t,i,i} = 1 - 2q.$$

The reachability set S_t is

$$S_t = \{-t, -t+1, \ldots, -1, 0, 1, \ldots, t-1, t\}.$$

The state-time lattice is

$$\mathcal{L} = \{(t, j) : j = -t, -(t-1), \ldots, t-1, t; t = 1, \ldots T\}.$$

The number of nodes is $1 + 3 + 5 \ldots + (2T+1) = (T+1)^2$. The pertaining history tree $\mathcal{T}$ has $1 + 3 + \ldots + 3^T = (3^{T+1} - 1)/2$ nodes.

In the following, we will take the binomial random walk as the standard model. The results carry over to the trinomial model or more general models by slight changes only.

The number of time periods T is the complexity parameter. The number of nodes of the state-time lattice is quadratic in T, while the number of nodes of the history tree is exponential in T. Algorithms, which require to expand the lattice to the history tree cannot be polynomial in T.

2.2 Interest rates and cash-flows

Let ξ_t be a Markov process, which can be represented on a lattice. The interest rates $r_t(\xi_t)$ in time period $[t, t+1]$ are functions of the state ξ_t. Let $f_t(\xi_t)$ be the discount factor $f_t(\xi_t) = (1 + r_t(\xi_t))^{-1}$. In practical terms, the interest rates $r_t(j)$ and the factors $f_t(j)$ "sit" on the nodes (t, j).

A cash flow stream is a sequence of functions $c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1)$, which assign to each path of the history tree a value that is paid at time t by the contract under consideration.

Figures 2 and 3 illustrate the situation: Whereas the lattice carries the interest rate process and therefore the discount factors, the cash-flows are carried by the pertaining history tree in general.

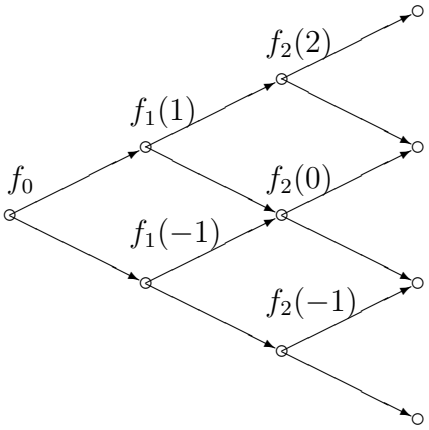

Figure 2. A binary lattice carrying the discount factors

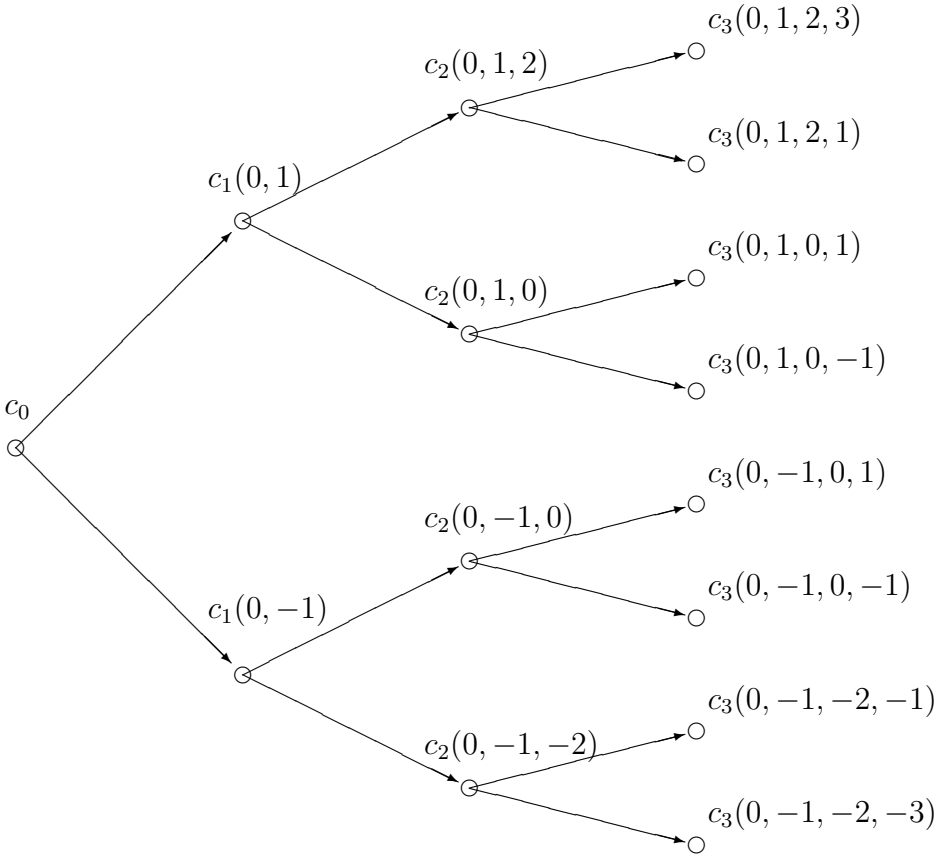

Figure 3. The pertaining binary tree carrying the cash-flows

We distinguish between

- Path-independent cash-flows.
  In this situation, the cash-flows at time t do only depend on the state of the system ξ_t at time t, i.e.
  $$c_t(\xi_t, \ldots, \xi_1) = c_t(\xi_t) \qquad t \geq 0.$$

- Delayed cash-flows.
  The cash-flows at time t do only depend on the state of the system at time $t - \ell$, i.e.
  $$c_t(\xi_t, \ldots, \xi_1) = c_t(\xi_{t-\ell}) \qquad t \geq \ell.$$

- Limited path-dependent cash-flows.
  There is an integer d such that the cash-flows at time t do only depend on the state of the system at times $t, t-1, \ldots, t-d$, i.e.
  $$c_t(\xi_t, \xi_{t-1} \ldots, \xi_1) = c_t(\xi_t, \ldots, \xi_{t-d}) \qquad t \geq d+1.$$
  The case $d = 0$ is the path-independent case.

- Full path-dependent cash-flows.
  This is the general case
  $$c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1).$$

Notice again that the discount factors f are defined on the lattice

$$f_t(j);\ (t, j) \in \mathcal{L},$$

whereas the cash-flows are defined on the tree in general

$$c_t(i_1, \ldots, i_t);\ (0, i_1, \ldots, i_t) \in \mathcal{T}.$$

For a cash-flow structure $c = (c_0, c_1, \ldots, c_T)$, the expected present value PV_0 (the today's price of the contract c) is

$$PV_0(c) = c_0 + \mathbb{E}[\sum_{t=1}^{T} c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1)\, f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0].$$

The present value operator PV is linear: If $c^{(1)}$ and $c^{(2)}$ are two cash-flow structures, then

$$PV_0(\gamma_1 c^{(1)} + \gamma_2 c^{(2)}) = \gamma_1 PV_0(c^{(1)}) + \gamma_2 PV_0(c^{(2)}).$$

2.3 The basic exponential algorithm

It is always possible to calculate the present value on the basis of the history tree: Let $\Pi_T = \Pi_T(0,0)$ be the set of all possible path $(0, i_1, \ldots, i_T)$ of length T, i.e. the set of the leaves of the history tree. Then

$$PV_0(c) = c_0 + \sum_{(0,i_1,\ldots,i_T)\in\Pi_T} [\sum_{s=1}^{T} c_s(i_s,\ldots,i_1) f_{s-1}(i_{s-1}) \cdots f_1(i_1) \cdot f_0]\; p_{0,0,i_1} \cdot p_{1,i_1,i_2} \cdots p_{T-1,i_{T-1},i_T}$$

The following algorithm requires that the functions f_t are defined on S_t, whereas the functions c_t are defined on $\Pi_t(0,0)$.

Algorithm FULL TREE EXPANSION

function `full-present-value` $(c_0, c_1, \ldots, c_T, f_0, f_1, \ldots, f_T)$ **returns** PV_0
$PV_0 := c_0$
forall $(0, i_1, \ldots, i_T) \in \Pi$
  $Z := 0$
  for $s := T$ **downto** $s = 1$
    $Z := [Z + c_s(i_s, \ldots, i_1)] f_{s-1}(i_{s-1})$
  endfor
  $PV_0 := PV_0 + Z \cdot p_{0,0,i_1} \cdot p_{1,i_1,i_2} \cdots p_{T-1,i_{T-1},i_T}$
endforall

Since the outer loop is over all leaves of the history tree, the complexity of this algorithm is $T \cdot 2^T$.

2.4 Path-independent cash-flows

For path-independent cash flows, $c_t(\xi_t, \ldots, \xi_1) = c_t(\xi_t)$ the above calculation can be performed in a recursive way. There are two approaches to this recursion.

- **The backward recursion.**

  Introduce the conditional present values for a future state (t, j) as the conditional expectation

$$PV_{t,j}(c) = \mathbb{E}[\sum_{s=t}^{T} c_s(\xi_s)\, f_{s-1}(\xi_{s-1}) \cdots f_t(\xi_t) | \xi_t = j].$$

The conditional present values $PV_{t,j}$ satisfy a recursion

$$\begin{aligned}
PV_{T,j}(c) &= c_T(j) \\
PV_{t,j}(c) &= \mathbb{E}[\sum_{s=t}^{T} c_s(\xi_s)\, f_{s-1}(\xi_{s-1}) \cdots f_t(\xi_t) | \xi_t = j] \\
&= c_t(j) + \mathbb{E}[\sum_{s=t+1}^{T} \sum_{k \in S_{t+1}} c_s(\xi_s)\, f_{s-1}(\xi_{s-1}) \cdots f_t(j) \mathbb{1}_{\xi_{t+1}=k} | \xi_t = j] \\
&= c_t(j) + f_t(j) \sum_{k \in S_{t+1}} \mathbb{E}[\sum_{s=t+1}^{T} c_s(\xi_s)\, f_{s-1}(\xi_{s-1}) \cdots f_{t+1}(\xi_{t+1}) \mathbb{1}_{\xi_{t+1}=k} | \xi_t = j] \\
&= c_t(j) + f_t(j) \sum_{k \in S_{t+1}} \mathbb{E}[\sum_{s=t+1}^{T} c_s(\xi_s)\, f_{s-1}(\xi_{s-1}) \cdots f_{t+1}(\xi_{t+1}) | \xi_{t+1} = k] p_{t,j,k} \\
&= c_t(j) + f_t(j) \sum_{k \in S_{t+1}} PV_{t+1,k}(c)\, p_{t,j,k} \\
PV_0(c) &= PV_{0,0}(c)
\end{aligned}$$

For the following two algorithms both f_t and c_t are defined on S_t.

Algorithm BACKWARD RECURSION

function `simple-present-value1` $(c_0, c_1, \ldots, c_T, f_0, f_1, \ldots, f_T)$ **returns** PV_0
forall $j \in S_T$: $PV_{T,j} := c_T(j)$ **endforall**
for $t := T-1$ **downto** $t = 0$
  forall $j \in S_t$
    $PV_{t,j} := c_t(j) + f_t(j) \sum_k PV_{t+1,k}\, p_{t,j,k}$
  endforall
endfor

- **The forward recursion.**
  It is based on the notion of state prices. The state $SP_{t,j}$ is the present value of a contract which pays one unit at time t, if the state at this time is j and nothing else:

$$SP_{t,j} = PV_0(\mathbb{1}_{(t,j)}) = \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\{\xi_t = j\}}]. \tag{1}$$

It is easy to see that the state prices fulfil the following forward recursion

$$\begin{aligned}
SP_{0,0} &= 1 \\
SP_{t,j} &= \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\{\xi_t = j\}}] \\
&= \sum_{k \in S_{t-1}} \mathbb{E}[f_{t-1}(k) \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\{\xi_{t-1}=k\}} \mathbb{1}_{\{\xi_t = j\}}]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{k \in S_{t-1}} f_{t-1}(k) \mathbb{E}[f_{t-2}(\xi_{t-2}) \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\{\xi_{t-1}=k\}}] \mathbb{P}\{\xi_t = j | \xi_{t-1} = k\} \\
&= \sum_{k \in S_{t-1}} SP_{t-1,k}\ f_{t-1}(k)\ p_{t,k,j}
\end{aligned}
$$

The price of $c = (c_1, \ldots, c_T)$ is

$$
\begin{aligned}
PV_0(c) &= \mathbb{E}[\sum_{t=1}^{T} c_t(\xi_t)\, f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0] \\
&= \sum_{t=1}^{T} \sum_{j \in S_t} c_t(j)\, \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\xi_t = j}] \\
&= \sum_{t=1}^{T} \sum_{j \in S_t} c_t(j)\ SP_{t,j}.
\end{aligned}
$$

Algorithm FORWARD RECURSION

function `simple-present-value2` $(c_0, c_1, \ldots, c_T, f_0, f_1, \ldots, f_T)$ **returns** PV_0
$SP_{0,0} := 1$, $PV_0 := 0$
for $t := 1$ **to** $t = T$
  forall $j \in S_t$
    $SP_{t,j} := \sum_k SP_{t-1,k}\ f_{t-1}(k)\ p_{t,k,j}$
  endforall
endfor
for $t := 1$ **to** $t = T$
  $PV_0 := PV_0 + \sum_j c_t(j) \cdot SP_{t,j}$
endfor

The complexity of this algorithm is of the order of the number of the nodes of the lattice, which is $O(T^2)$.

The usual way of calculating the price for a path-dependent cash flow is to expand the lattice to the full tree and do he calculations on the tree. This algorithms is non-polynomial, in contrast to the polynomial algorithm for the non-path dependent case.

However, at least for some path dependent situations, polynomial algorithms are available.

3 Special cash-flow structures

In this section, we study the pricing algorithms for special classes of cash-flow functions. For simplicity, assume that $c_0 = 0$, i.e. no payment takes place at time 0.

3.1 Product-form cash flows

Suppose that the cash-flows are of product form

$$c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) = c_{t,t}(\xi_t) \cdot c_{t,t-1}(\xi_{t-1}) \cdots c_{t,1}(\xi_1)$$

where the functions $c_{t,s}$ are defined on S_s. Then

$$\begin{aligned} PV_0(c_1, \ldots, c_T) &= \mathbb{E}[\sum_{t=1}^{T} c_{t,t}(\xi_t) c_{t,t-1}(\xi_{t-1}) \ldots c_{t,1}(\xi_1) \cdot f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) f_0] \\ &= \mathbb{E}[\sum_{t=1}^{T} c_{t,t}(\xi_t) \cdot g_{t,t-1}(\xi_{t-1}) \cdots g_{t,1}(\xi_1) g_{t,0}] \end{aligned}$$

Here

$$\begin{aligned} g_{t,0} &= f_0 \\ g_{t,1}(\xi_1) &= c_{t,1}(\xi_1) \cdot f_1(\xi_1) \\ &\vdots \\ g_{t,t-1}(\xi_{t-1}) &= c_{t,t-1}(\xi_{t-1}) \cdot f_{t-1}(\xi_{t-1}) \end{aligned}$$

The structure of the formula reveals that a recursive calculation is possible. The constants $g_{t,s}$ are the new discounting factors. In contrast to the path-independent situation they carry two indices. Therefore a recursive calculation can be done for the contracts which pay only at a fixed time. The total present value is obtained by summing over all times in an additional loop.

As in the history-independent case we may do a backward recursion and a forward recursion.

- **The backward recursion.** For $t \leq s$ let $PV_{t,j}^{(s)}$ be the conditional value of the contract which pays $c_s(\xi_s, \ldots, \xi_1)$ at time s and nothing at other times conditional that the state at time t is j:

$$\begin{aligned} PV_{t,j}^{(s)} &= \mathbb{E}[c_{s,s}(\xi_s) c_{s,s-1}(\xi_{s-1}) \cdots c_{s,t}(\xi_t) f_{s-1}(\xi_{s-1}) \ldots f_t(\xi_t) | \xi_t = j] \\ &= \mathbb{E}[c_{s,s}(\xi_s) g_{s,s-1}(\xi_s) \cdots g_{s,t}(\xi_t) | \xi_t = j] \end{aligned}$$

  The backward recursion for the calculation of $PV_{t,j}^{(s)}$ is

$$\begin{aligned} PV_{s,j}^{(s)} &= c_{s,s}(j) \\ PV_{t,j}^{(s)} &= \sum_k PV_{t+1,k}^{(s)} g_{s,t}(j) p_{t,j,k} \\ &= c_{s,t}(j) \cdot f_t(j) \sum_k PV_{t+1,k}^{(s)} p_{t,j,k} \end{aligned}$$

The final price is

$$PV_0(c_1, \ldots, c_T) = \sum_{s=1}^{T} PV_{t,j}^{(s)}.$$

Algorithm PRODUCT FORM BACKWARD RECURSION

function `product-form1` $(c_0, c_{1,1}, c_{2,2}, c_{2,1}, \ldots, c_{T,T}, \ldots c_{T,1}, f_0, f_1, \ldots, f_T)$ **returns** PV_0
$PV_{0,0} := 0$
for $s := 1$ `to` $s = T$
  $PV_{s,j}^{(s)} := c_{s,s}(j)$.
  for $t := s$ **downto** $t := 1$
    forall $j \in S_t$
      $PV_{t,j}^{(s)} = c_{s,t}(j) \cdot f_t(j) \sum_k PV_{t+1,k}^{(s)} p_{t,k,j}$
    endforall
  endfor $PV_{0,0} := PV_{0,0} + PV_{0,0}^{(s)}$ **endfor**

Because of the additional loop, the complexity is $\sum_{t=1}^{T} O(t^2) = O(T^3)$.

- **The forward recursion.**
  Define for $t \leq s$ the values $V_{t,j}^{(s)}$ as

$$V_{t,j}^{(s)} = \mathbb{E}[g_{t-1}(\xi_{t-1}) \cdots g_1(\xi_1) \cdot g_0 \cdot \mathbb{1}_{\{\xi_t = j\}}].$$

  These values are the present values of a contract, which pays $c_{s,t-1}(\xi_s) \cdots c_{s,1}(\xi_1)$ at time t, but only if the state at time t is j. These values fulfil the recursion

$$\begin{aligned} V_{0,0}^{(s)} &= 1 \\ V_{t,j} &= \sum_{k \in S_{t-1}} V_{t-1,k}^{(s)} \; g_{t-1,s}(k) \; p_{t,k,j} \\ &= \sum_{k \in S_{t-1}} V_{t-1,k}^{(s)} \; c_{t-1,s}(k) \; f_{t-1}(k) p_{t,k,j} \end{aligned}$$

  The final present value is

$$PV_{0,0} = \sum_{t=1}^{T} \sum_{j \in S_t} c_{t,t}(j) V_{t,j}^{(t)}.$$

Algorithm PRODUCT FORM FORWARD RECURSION

```
function product-form2 (c_0, c_1, ..., c_T, f_0, f_1, ..., f_T) returns PV_0
PV_0 := 0
for s := 1 to s := T
    V^(s)_{0,0} := 1
    for t := 1 to t = s
      forall j ∈ S_t
        V^(s)_{t,j} := Σ_k V^(s)_{t-1,k} c_{t-1,s}(k) f_{t-1}(k) p_{t,k,j}
      endforall
    endfor
  endfor for s := 1 to s = T
   PV_0 := PV_0 + Σ_j c_{t,t}(j) · V^(t)_{t,j}
endfor
```

The complexity is $\sum_{t=1}^{T} O(t^2) = O(T^3)$, the same as for the backward algorithm.

3.1.1 Simple product form

If the cash-flow function is a simple product:

$$c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) = c_t(\xi_t) \ldots c_{t-1}(\xi_{t-1}) \ldots c_1(\xi_1)$$

then the situation is even simpler.

Using the formula

$$PV_0(c) = \mathbb{E}[\sum_{t=1}^{T} c_t(\xi_t) \cdot c_{t-1}(\xi_{t-1}) \cdots c_1(\xi_1)\, f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0] = \mathbb{E}[\sum_{t=1}^{T} c_t(\xi_t)\, \tilde{f}_{t-1}(\xi_{t-1}) \cdots \tilde{f}_1(\xi_1) \cdot f_0]$$

where $\tilde{f}_s(j) = f_s(j) \cdot c_s(j)$ one sees that this case may be reduced to the history-independent case by introducing the new discounting factors

$$\tilde{f}_s(j) := f_s(j) \cdot c_s(j).$$

The present value is calculated with the cash-flows c_t and the discounting factors $\tilde{f}$. The complexity is $O(T^2)$.

3.2 Delayed cash flow

Suppose that the cash flow at time t depends only on on $\xi_{t-\ell}$

$$\begin{aligned} c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) &= \tilde{c}_{t-\ell}(\xi_{t-\ell}) \qquad \text{for } t \geq \ell \\ c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) &= 0 \qquad \text{for } t < \ell \end{aligned}$$

where ℓ is a fixed number. Let for $s < t$

$$F_{s,j}^{(t)} = \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdot f_{t-2}(\xi_{t-2}) \cdots f_s(\xi_s) | \xi_s = j].$$

These values fulfill the following recursion

$$\begin{aligned}
F_{t,j}^{(t)} &= f_t(j) \\
F_{s,j}^{(t)} &= \sum_k \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_{s+1}(k) \cdot f_s(j) \mathbb{1}_{\{\xi_{s+1}=k\}} | \xi_s = j] \\
&= \sum_k f_s(j) \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_{s+1}(k) | \xi_{t-1} = k] \mathbb{P}\{\xi_{s+1} = k | \xi_s = j\} \\
&= f_s(j) \sum_k F_{s+1,k}^{(t)} p_{s,j,k}
\end{aligned}$$

Then

$$\begin{aligned}
PV_0(c_1, \ldots, c_T)) &= \mathbb{E}[\sum_{t=\ell}^{T} \tilde{c}_{t-\ell}(\xi_{t-\ell}) f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0] \\
&= \sum_{t=\ell}^{T} \sum_{j \in S_{t-\ell}} \tilde{c}_{t-\ell}(j) \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0 \mathbb{1}_{\xi_{t-\ell}=j}] \\
&= \sum_{t=\ell}^{T} \sum_{j \in S_{t-\ell}} \tilde{c}_{t-\ell}(j) \mathbb{E}[f_{t-1}(\xi_{t-1}) \cdots f_{t-\ell}(\xi_{t-\ell}) \mathbb{1}_{\xi_{t-\ell}=j}] \times \\
&\qquad \mathbb{E}[f_{t-\ell-1}(\xi_{t-\ell-1} \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\xi_{t-\ell}=j}] \\
&= \sum_{t=\ell}^{T} \sum_{j \in S_t} \tilde{c}_{t-\ell}(j) \cdot F_{t-\ell,j}^{(t)} \mathbb{E}[f_{t-\ell-1}(\xi_{t-\ell} - 1) \cdots f_1(\xi_1) \cdot f_0 \cdot \mathbb{1}_{\xi_{t-\ell}=j}] \\
&= \sum_{t=\ell}^{T} \sum_{j \in S_t} \bar{c}_{t-\ell}(j) SP_{t-\ell,j}
\end{aligned}$$

where $\bar{c}_{t-\ell}(j) = \tilde{c}_{t-\ell}(j) \cdot F_{t-\ell,j}^{(t)}$ and the state prices $SP_{t,j}$ are defined as in (1). The following algorithm assumes that $\tilde{c}_t$ are defined on S_t.

Algorithm DELAYED CASH-FLOWS

function `delayed-cash-flows` $(0, \tilde{c}_1, \ldots, \tilde{c}_{T-\ell}, f_0, f_1, \ldots, f_T)$ **returns** PV_0
for $t := \ell$ **to** $t := T$
  forall $j \in S_t$
    $F^{(t)}_{t,j} = f_t(j)$
  endforall
  for $s := t-1$ **downto** $s := t-\ell$
    $F^{(t)}_{s,j} = f_s(j) \sum_k F^{(t)}_{s+1,k} p_{s,j,k}$
  endfor
  forall $j \in S_t$
    $\bar{c}_{t-\ell}(j) = F^{(t)}_{t-\ell,j} \cdot \tilde{c}_{t-\ell}(j)$
  endforall
endfor
$PV_{0,0}$`:= simple-present-value` $(\bar{c}_0, \bar{c}_1, \ldots, \bar{c}_{T-\ell}, f_0, f_1, \ldots, f_T)$

Here the function `simple-present-value` may either be `simple-present-value1` `simple-present-value2`. The complexity of `delayed-cash-flows`is $O(T^3)$.

3.2.1 Delay 1

If the delay equals 1, the algorithm is as simple as for delay 0. Suppose that $c_t(\xi_t, \ldots, \xi_1) = \tilde{c}_{t-1}(\xi_{t-1})$. Then the formula to be calculated is

$$PV_0 = \mathbb{E}[\sum_{t=1}^{T} \tilde{c}_{t-1}(\xi_{t-1}\, f_{t-1}(\xi_{t-1}) \cdots f_1(\xi_1) \cdot f_0].$$

Setting $\bar{c}_{t-1}(j) = \tilde{c}_{t-1}(j) \cdot f_{t-1}(j)$, the present value of $PV_0(0, \tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{T-1}, 0)$ equals $PV_0(\bar{c}_1, \ldots, \bar{c}_T, 0)$.

3.3 Additive cash flow

Suppose that the cash-flows are of additive form:

$$c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) = c_{t,t}(\xi_t) + c_{t,t-1}(\xi_{t-1}) + \ldots + c_{t,1}(\xi_1)$$

Then the cash-flow function can be represented as a sum of delayed cash flows. Let

$$\begin{aligned} c^{(0)} &= (c_{1,1}, \ldots, c_{T,T}), \\ c^{(1)} &= (c_{2,1}, \ldots, c_{T,T-1}, 0), \\ &\vdots \\ c^{(T-1)} &= (c_{T,1}, 0, \ldots, 0) \end{aligned}$$

The cash-flow $c^{(i)}$ has delay i. The total present value is

$$PV_{0,0}(c) = \sum_{i=0}^{T-1} PV_{0,0}(c^{(i)}).$$

The following algorithm assumes that $c_{t,s}$ are defined on S_s.

Algorithm ADDITIVE CASH-FLOWS

function `additive-cash-flows` $(c_{1,1}, c_{2,2}, c_{2,1}, \ldots, c_{T,T}, \ldots, c_{T,1}, f_0, f_1, \ldots, f_T)$
returns PV_0
$PV_0 = 0$
for $t := 1$ **to** $t := T$
$PV_0 = PV_0 +$ `delayed-cash-flows`$(0, \ldots, 0, c_{t,0}, \ldots, c_{T,T-s}, f_0, f_1, \ldots, f_T)$
endfor

3.4 Limited path dependence

Suppose that the cash-flow functions are of the form

$$\begin{aligned} c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) &= c_t(\xi_t, \xi_{t-1}, \ldots, \xi_{t-d}) \qquad t \geq d+1 \\ c_t(\xi_t, \xi_{t-1}, \ldots, \xi_1) &= 0 \qquad\qquad\qquad\qquad t < d+1 \end{aligned}$$

Set, for $t \leq T - d$,

$$IPV_{t,j}^{(d)} = \mathbb{E}[\sum_{s=t+d}^{T} c_s(\xi_s, \ldots, \xi_{s-d}) f_{s-1}(\xi_{s-1}) \cdots f_{t+1}(\xi_{t+1}) f_t(j) | \xi_t = j].$$

These values are conditional present values of an incomplete cash-flow stream, i.e. a cash-flow stream which does not pay for the first $d-1$ periods after t. Notice that $PV_0 = IPV_{0,0}^{(d)}$ and that $IPV_{t,j}^{(0)} = PV_{t,j}$.

The values $IPV_{t,j}^{(d)}$ can be computed recursively.

$$\begin{aligned} IPV_{T,j}^{(d)} &= 0 \\ IPV_{t,j}^{(d)} &= \mathbb{E}[\sum_{s=t+d}^{T} c_s(\xi_s, \ldots, \xi_{s-d}) f_{s-1}(\xi_{s-1}) \cdots f_t(j) | \xi_t = j] \\ &= \mathbb{E}[c_{t+d}(\xi_{t+d}, \ldots, \xi_t)\, f_{t+d-1}(\xi_{t+d-1}) \cdots f_t(j) | \xi_t = j] \\ &\quad + \sum_{s=t+d+1}^{T} \mathbb{E}[c_s(\xi_s, \ldots, \xi_{s-d})\, f_{s-1}(\xi_{s-1}) \cdots f_t(j) | \xi_t = j] \end{aligned}$$

$$
\begin{aligned}
&= C_{t,j} + \sum_{s=t+d+1}^{T} \sum_{k} \mathbb{E}[c_s(\xi_s, \ldots, \xi_{s-d})\, f_{s-1}(\xi_{s-1}) \cdots f_t(j) \mathbb{1}_{\{\xi_{t+1}=k\}} | \xi_t = j] \\
&= C_{t,j} + \sum_{k} \sum_{s=t+d+1}^{T} \mathbb{E}[c_s(\xi_s, \ldots, \xi_{s-d}) f_{s-1}(\xi_{s-1}) \cdots f_{t+1}(k) \cdot f_t(j) | \xi_{t+1} = k]\, p_{t,j,k} \\
&= C_{t,j} + f_t(j) \sum_{k} IPV^{(d)}_{t+1,k}\; f_{t+1}(k)\; p_{t,j,k}
\end{aligned}
$$

Here

$$C_{t,j} = \mathbb{E}[c_{t+d}(\xi_{t+d}, \ldots, \xi_t)\, f_{t+d-1}(\xi_{t+d-1}) \cdots f_t(j) | \xi_t = j].$$

Algorithm LIMITED PATH DEPENDENCE

function `limited-path-dependence`$(0, \ldots, 0, c_{d+1} \ldots, c_T, f_0, f_1, \ldots, f_T)$ **returns** PV_0
forall t
  forall $j \in S_t$ $IPV_{t,j} := 0$
endforall
for $t := T - d$ **step** -1 **downto** $t := 0$
  forall $j \in S_t$
    forall k
      $C_{t,j,k} := \sum_{(i_{t+d}, \ldots, i_{t+2})} c_{t+d}(i_{t+d}, \ldots, i_{t+2}, k, j) f_{t+d-1}(i_{t+d-1}) \ldots f_t(j) \times$
        $p_{t,j,k} \cdot p_{t+1,k,i_{t+2}} \cdots p_{t+d-1,i_{t+d-1},i_{t+d}}$
      $IPV^{(d)}_{t,j} := C_{t,j,k} + IPV^{(d)}_{t,j} + f_t(j) IPV^{(d)}_{t+1,k}\; f_{t+d}(i_{t+d}) \cdots f_{t+1}(k)\; p_{t,j,k}$
    endforall
  endforall
$PV_0 := IPV^{(d)}_{0,0}$

3.4.1 The structure of the algorithm

Figure 2 shows an example with depth $d = 4$. The cash flow at time t depends on interest rates at timesteps not before $t - 4$.

The core step in the standard backward induction is the computation of the present value at a node through the discounted cash flows and present values at the direct successor nodes. The propagation of information backwards in time reflects the present value's property of incorporating future information (i.e. future cash-flows).

Through the iterative application of the induction step, the information at all successors of a node, representing its complete future, has an impact on the present value at that node. This works well, if the future information is fully determined by the state represented by the node. In the case of path dependent products, the future cash-flows, and thus the present value, are specific to the history of the state. A valid induction step

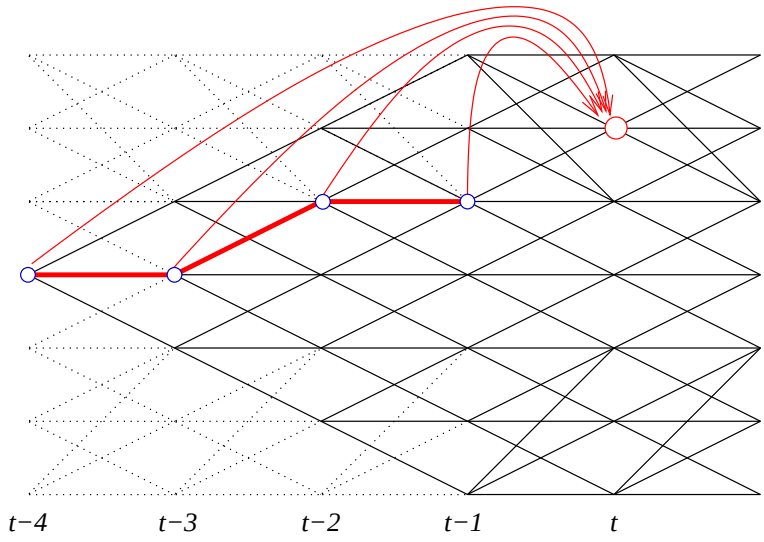

Figure 2: Limited path dependence with depth $d = 4$

must have the following property: it uses only that future information which is compatible with the history of the nodes involved. In principle, this can be achieved by extending the state variable with the necessary history information, represented by the path which determines the future of the state. This ends up with computing, and storing, for each node, the whole set of path dependent present values – equal in size to the number of different paths of length d ending in this node [2].

By introducing the "incomplete present value" (IPV) a recursion becomes available: There is no need to deal with path dependent IPV's, because they don't rely on history information. Therefore we can apply an backward induction step to it, which computes a IPV from all IPV's at $d-1$ steps later. Together with a procedure to complete a IPV this allows us to price a path dependent product.

The algorithm consists of two phases. The first phase is comparable to the standard Backward Induction which computes present values in a backward sweep through the lattice, but it computes *"incomplete"* present values (which later on are used to compute the complete present values). An incomplete value at (t, j), denoted with $IPV_{t,j}^{(d)}$, does not include the information (i.e. the cash flows) within a time d after t (i.e. it does not take into account the cash flows from time $t+1$ to $t+d-1$).

Each incomplete present value, together with a history path of length $d-1$ is related to a path dependent *complete* present value at this node: for a specific history path π with final node (t, j), we can compute the complete present value $PV_\pi(t, j)$ from $IPV_{t,j}^{(d)}$. Using π, we can compute the cash flows, which are not regarded in IPV, discount them

[2] which is an exponential function of d

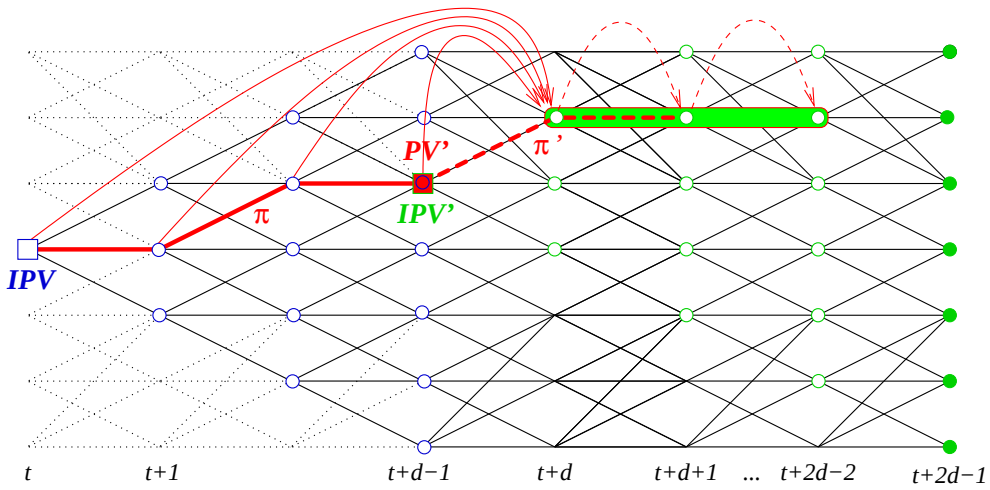

Figure 3: Recursion step

to (t, j) and thus "complete" the present value for this specific history.

In the following, we describe one recursion step. It computes the incomplete $IPV^{(d)}_{t,j}$ from the set of uncomplete IPV' [3] at the nodes $d-1$ time steps after t, which can be reached from (t, j)

Basically, we first compute the complete present values at $t+d-1$ (from the uncomplete values) and discount them to (t, j).

An enumeration starting at (t, j) generates all paths π' of length $d-1$ with some final node $(t+d-1, j')$, in order to compute the values $IPV'_{\pi'}$ at these nodes. The complete present value $PV'_{\pi'}(t+d-1, j')$ is dependent on π'. We need all the cash flows from $t+d$ and $t+2d-2$, because they are not regarded in $IPV^{(d)}_{t+d-1,j'}$. Additional paths π'' of length $d-2$ are chosen starting at $(t+d-1, j')$. The concatenation of π' and π'' now determines all cash flows which are relevant to the completion, which is performed by discounting them to $t+d-1$ and adding them to the incomplete value.

The result of the described step, namely the incomplete $IPV^{(d)}_{t,j}$, is computed as the average of the discounted (along π') complete $IPV'_{\pi'}(t+d-1, j')$.

In the second phase of the algorithm, the price at node $(0, 0)$ is computed from $IPV^{(d)}_{0,0}$. The complete $PV_{0,0}$ is not path dependent, hence we can compute it through a complete enumeration or a Monte Carlo Simulation, which samples paths starting at $(0, 0)$ of length $d-2$ to compute the remaining cash flows, and discount them to $(0, 0)$.

[3] which are not dependent on the history of $IPV^{(d)}_{t,j}$

3.4.2 A Monte Carlo version of the limited path dependence algorithm

The presented algorithm for limited path dependent cash-flows may be modified in such a way that the time consuming calculation of $\Sigma_{t,j}$ is done by Monte Carlo Simulation. This modification is advisable, if d is very large. As opposed to the standard Monte Carlo Simulation, which samples paths of the length of the whole lattice, for our combined approach only paths within sublattices of a maximum length of $2d-1$ have to be sampled.

3.4.3 Experiments

We computed the price of a variable coupon bond with a maturity of 18 periods, which pays at time t the average of the interest rates from time $t-4$ to time $t-1$. The number of nodes of the interest rate tree is 89, the total number of paths is 387,420,489. Table 1 shows the prices computed and the execution time on a Sun Ultra 4 workstation, using a standard Monte Carlo Simulation with varying sample sizes, and the algorithm for limited path dependence.

Method	number of paths	price	execution time
Monte Carlo	3874 paths	8573.70	0.65 sec
Monte Carlo	38 K paths	8590.85	6.55 sec
Monte Carlo	387 K paths	8589.27	64.17 sec
Limited Path Dependence	387 M (all) paths	8586.58	1.810 sec

Table 1: Results for different methods

References

[Hul02] J.C. Hull. *Options, Futures, and Other Derivatives*. Prentice Hall, July 2002.

[HW93] J.C. Hull and A. White. Efficient procedures for valuing european and american path dependent options. *Journal of Derivatives*, pages 21 –31, 1993.

[PS97] K. Pang and C. Strickland. Pricing interest rate exotics using term structure consistent short rate trees. In C. Strickland L. Clewlow, editor, *Exotic Options: The State Of The Art*. International Thomson Business Press, London, 1997.