Methodol Comput Appl Probab (2016) 18:487–498
DOI 10.1007/s11009-014-9436-1

Birth and Death Chains on Finite Trees: Computing their Stationary Distribution and Hitting Times

José Luis Palacios · Daniel Quiroz

Received: 24 April 2013 / Revised: 21 November 2014 /
Accepted: 29 December 2014 /Published online: 15 January 2015

Abstract Every birth and death chain on a finite tree can be represented as a random walk on the underlying tree endowed with appropriate conductances. We provide an algorithm that finds these conductances in linear time. Then, using the electric network approach, we find the values for the stationary distribution and for the expected hitting times between any two vertices in the tree. We show that our algorithms improve classical procedures: they do not exhibit ill-posedness and the orders of their complexities are smaller than those of traditional algorithms found in the literature.

Keywords Effective resistance · Conductance · Star graph · Hitting times

Mathematics Subject Classification (2010) 60J15 · 60C05

1 Introduction

Birth and death (B. D.) chains on trees are natural generalizations of ordinary B. D. chains, where the transitions occur from any given vertex of a tree to either itself or to any other neighboring vertex in the tree. The ordinary birth-and-death processes occur on the linear graph. The research involving B. D. chains on trees (Bertoncini 2011; Fayolle et al. 2004; Ma 2010) appears to be directed to infinite (random or deterministic) trees and is related to

J. L. Palacios (✉)
Department of Electrical and Computer Engineering, The University of New Mexico, Albuquerque, NM 87131, USA
e-mail: jpalacios@unm.edu

D. Quiroz
Department of Mathematics, London School of Economics, London WC2A 2AE, UK
e-mail: D.Quiroz@lse.ac.uk

the questions of whether the process is transient or recurrent and, in the latter case, whether closed form formulas can be found for the stationary distribution. In this article we will be concerned with B.D. chains which occur on finite trees, and we will find the values for the stationary distribution and for the hitting times between any two arbitrary vertices. The idea is to represent a B.D. chain on a finite tree as a random walk on the underlying tree, by means of an algorithm that assigns suitable conductances to the edges of the tree, and then use known formulas for the stationary distribution and for the hitting times given in terms of the conductances.

Since the appearance of the book of Doyle and Snell (1984), a great deal of attention has been devoted to the relation between electric networks and random walks on graphs. In particular, the computation of stationary distributions and expected hitting times sometimes is greatly simplified by this electric network approach, which consists of thinking of the edge between vertices v and u as a resistor with resistance r_{vu} (or conductance $C_{vu} = 1/r_{vu}$); then we can define the random walk on the connected undirected graph $G = (V, E)$, as the first order Markov chain $X_n, n \geq 0$, that from its current vertex v jumps to the neighboring vertex u with probability $p_{vu} = C_{vu}/C(v)$, where $C(v) = \sum_{w:w\sim v} C_{vw}$, and $w \sim v$ means that w is a neighbor of v. Note that we can assign a (fictitious) conductance C_{zz} from a vertex z to itself, giving rise to a transition probability from z to itself. We denote by $E_a T_b$ the expected value, starting from the vertex a, of the hitting time T_b of the vertex b, defined by

$$T_b = \inf\{n \geq 0 : X_n = b\}.$$

The stationary distribution $\pi = \{\pi_z\}_{z\in V}$ is the unique row probability vector that satisfies

$$\pi \mathbf{P} = \pi, \tag{1}$$

where $\mathbf{P} = (p_{vu})_{v,u\in V}$ is the transition probability matrix of the process.

In this context we have:

Theorem 1 *For a random walk on a finite tree G we have*

$$\pi_z = \frac{C(z)}{\sum_z C(z)} \tag{2}$$

and for any $a, b \in G$ and P the unique path of vertices between a and b we have

$$E_a T_b = \sum_{x\in P} R_{x,b} C^x, \tag{3}$$

where $R_{x,b}$ is the effective resistance between x and b, $C^x = \sum_{w\in G_x} C(w)$, G_x is the connected component of $G - E(P)$ that contains x and $E(P)$ is the set of edges in the path P.

Derivations of Eqs. 2 and 3 can be read in Doyle and Snell (1984) and Palacios (2009), respectively.

Doyle and Snell also noted that a finite ergodic Markov chain can be represented as a random walk on a finite graph with conductances if and only if the Markov chain is reversible. A stochastic process is said to be reversible if the future of the process at any given time has the same distribution as the process seen in reversed time. In particular, reversible Markov chains are characterized by Kolmogorov's criteria in the following way (see Kelly (1979)).

Lemma 1 *A finite ergodic Markov chain on states* $\{1, ..., N\}$*, is reversible if and only if its transition probabilities satisfy*

$$p(j_1, j_2)p(j_2, j_3)\dots p(j_{k-1}, j_k)p(j_k, j_1)$$
$$= p(j_1, j_k)p(j_k, j_{k-1})\dots p(j_3, j_2)p(j_2, j_1)$$

for any finite sequence of states $j_1, j_2, \dots, j_k \in \{1, 2, \dots, N\}$.

Hence, as trees are acyclic, B.D. chains on trees are reversible and therefore, they can be represented as random walks on the underlying tree endowed with conductances.

It was shown in Palacios and Tetali (1996) that every ordinary birth-and-death Markov chain can be represented as a random walk on the linear graph with vertices $0, 1, \dots, N$ and conductances C_k, $1 \le k \le N$, between vertices $k-1$ and k given by

$$C_k = \frac{p_1 \cdots p_{k-1}}{q_1 \cdots q_{k-1}} C_1, \quad 2 \le k \le N, \tag{4}$$

where C_1 is arbitrary. $C_{00} = \frac{s_0}{p_0} C_1$ and conductances from any vertex to itself given by

$$C_{kk} = \frac{s_k}{q_k} C_k, \quad 1 \le k \le N. \tag{5}$$

This pair of equations, which allows to explicitly find conductances on the linear graph in terms of the transition probabilities, is one of the main inspirations of our algorithm. The next lemma, which shows how to assign conductances starting from a vertex that branches out in more than two directions, is the other source of inspiration.

Lemma 2 *Any B.D. chain on the star graph with center* N *and leaves* $1, 2, \dots, N-1$, $N \ge 2$*, and transition probabilities*

$$p(N, i) = p_i, \; p(i, N) = q_i, \; 1 \le i \le N-1, \; p(i, i) = s_i, \; 1 \le i \le N$$

can be represented as a random walk on the star graph with conductances

$$C_{Ni} = \frac{C_1 p_i}{p_1}, 1 \le i \le N-1, C_{NN} = \frac{C_1 s_N}{p_1}, \tag{6}$$

$$C_{ii} = \frac{C_1 s_i p_i}{q_i p_1}, 1 \le i \le N-1, \tag{7}$$

where $C_1 > 0$ *is arbitrary.*

Proof Left to the reader. □

2 The Algorithm

To avoid trivialities, all B.D. chains considered are ergodic Markov chains, that is, there are non-zero probabilities to go from any given vertex to any neighboring vertex and back to the original vertex. We will denote by $p(v, u)$ the transition probability from v to u and the underlying tree will be $G = (V, E)$, and recall that we write $v \sim u$ if v and u are neighbors.

Then we can describe the algorithm that expresses the chain as a random walk on the tree with appropriate conductances on the edges as follows:

1. Take any vertex $v \in V$ of the tree as the root, and consider any $u \sim v$. Assign an arbitrary (positive) value to C_{vu}.
2. Letting C_{vu} play the role of C_1 in formulas (6), obtain the conductance C_{vv} and all conductances C_{vw} where w is a neighbor of v, i.e.:

$$C_{vw} = \frac{C_{vu}\,p(v,w)}{p(v,u)},\ w \sim v; \quad C_{vv} = \frac{C_{vu}\,p(v,v)}{p(v,u)}.$$

3. Taking v as the root, traverse the vertices of the tree using Breadth First Search (BFS). Every time a vertex not previously visited is reached, only one of its adjacent conductances has being assigned. Take this conductance as the C_1 used to obtain all other adjacent ones.

The fact that the procedure works, that is, the fact that we can recover the transition probabilities from the conductances can be checked easily since

$$C(v) = \sum_{w:w\sim v} C_{vw} = \frac{C_{vu}}{p(v,u)},$$

and then the motion of the random walk from v to w is dictated by

$$\frac{C_{vw}}{C(v)} = \frac{C_{vu}\,p(v,w)}{p(v,u)}\frac{p(v,u)}{C_{vu}} = p(v,w),$$

when $v \sim w$ and

$$\frac{C_{vv}}{C(v)} = \frac{C_{uv}\,p(v,v)}{p(v,u)}\frac{p(v,u)}{C_{vu}} = p(v,v),$$

as desired.

This procedure stops when all leaves, and thus all vertices, have been visited. Since trees are acyclic no vertex is visited more than once. The *number of operations* is a linear function of the number of vertices N: in this BFS algorithm, for each vertex v only one iteration is made; the number of operations per iteration is, at most, $2d(v)$, where $d(v)$ is the degree of v. This number of operations is achieved when there is a positive transition probability from vertex v to itself. In total, the number of operations is at most $\sum_{v\in V} 2d(v) = 4|E| = 4N - 4$.

Figure 1 shows an example of a B.D. chain on a tree with certain transition probabilities and the same tree with the conductances assigned when the algorithm starts at vertex 1 and $C_{12} = C_1 = 1$. The next calculation is then $C_{11} = \frac{C_{12}p(1,1)}{p(1,2)} = \frac{1/3}{2/3} = \frac{1}{2}$. The next is $C_{23} = \frac{C_{12}p(2,3)}{p(2,1)} = \frac{3/5}{1/5} = 3$. The next is $C_{22} = \frac{C_{23}p(2,2)}{p(2,3)} = \frac{3/5}{3/5} = 1$, etc.

Once the transition probabilities have been turned into conductances, the stationary distribution of the process on the tree is found with formula (2), a computation which is obviously linear in N. Also, for any pair of vertices a and b, the hitting time E_aT_b is found by computing Eq. 3, a procedure whose linearity in N is a bit more involved to justify: the summation in Eq. 3 runs over the edges of the unique path between a and b, and the computation of C^x involves adding conductances in G_x, the connected component of $G - E(P)$ that contains x; at the end, every edge of the tree is taken into account at most once during the calculation of Eq. 3.

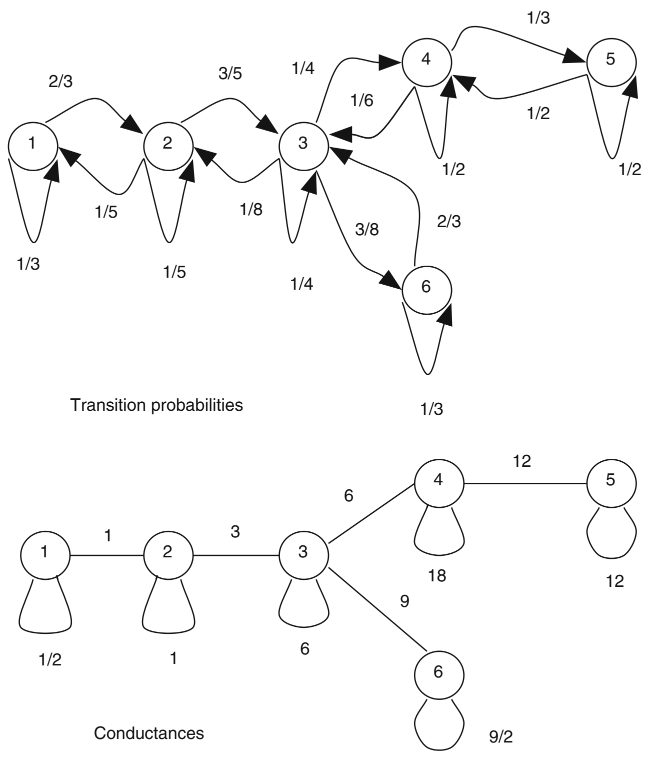

Fig. 1 From transition probabilities to conductances

One should also take into account the *storage complexity*: how much computer memory is used to store the data of the tree (first the transition probabilities, then the conductances) expressed in terms of the size N of the tree. It is natural to store transition probabilities in a matrix, and conductances in the form of an adjacency matrix with weights on the edges. But since both matrices are sparse, having m non-zero elements with $m \leq 3N - 2$, they can be stored in a smaller data structure. Indeed, each matrix can be represented with the help of three vectors $\mathbf{a} = (a_i)_{1 \leq i \leq m}$, $\mathbf{b} = (b_i)_{1 \leq i \leq m}$, and $\mathbf{c} = (c_i)_{1 \leq i \leq N+1}$ as follows: $\mathbf{a}$ contains all non-zero elements ordered by row, and b_i is the column to which the a_i belongs. The vector $\mathbf{c}$ satisfies that $c_1 = 1$ and, for $2 \leq i \leq N + 1$, c_i equals c_{i-1} plus the number of non-zero elements in the $(i - 1)$-th row of the matrix. So, in order to access the (i, j) element of a matrix compressed in this way, one should check whether $b_k = j$ for any $c_i \leq k < c_{i+1}$. If it is so, then the (i, j) element of the matrix is a_k. A different explanation of the same structure, which we believe to be folklore, and an example of its use, can be found in Dongarra (2000).

By avoiding the use of matrices in an explicit way, the memory used by this data structure consists of a fixed number of scalars and a fixed number of vectors of length at most $3N - 2$. Therefore, the storage requirement for the tree data is a linear function of N. The process of accessing elements in these data structures does not affect the linearity of the number of operations. Numerical examples are provided in Section 3 to exemplify this.

These linear procedures are substantially more efficient that the classical ones. Indeed, finding the stationary distribution of a finite Markov chain on N states entails solving the (redundant) $N \times N$ system given by Eq. 1 with the additional equation

$$\sum_z \pi_z = 1.$$

The brute force procedure to solve this system is a costly algorithm of order roughly N^3, though there are known methods for solving this type of linear systems of equations which have smaller order of complexity as they take advantage of the sparsity of the matrix $\mathbf{P}$. We will show in Section 3 that our linear procedures behave better than these methods.

Additionally, the classical procedures to obtain the hitting times involve matrix inversions, therefore having complexity roughly N^3 and sometimes exhibiting ill-posedness. That is the case, for instance, when we take $\mathbf{W}$ to be the matrix with all rows are identical to π, and then from the fundamental matrix $\mathbf{Z}$ given by

$$\mathbf{Z} = \{Z_{ij}\}_{i,j \in V} = (\mathbf{I} - \mathbf{P} + \mathbf{W})^{-1},$$

we obtain (see Grinstead and Snell, 1997)

$$E_a T_b = \frac{Z_{bb} - Z_{ab}}{\pi_b}.$$

One final note: our algorithm obtains a single hitting time in linear time, and therefore if we wanted to obtain all hitting times then the complexity of our procedure would seem to become N^3. We will show in Section 4, however, that we may reduce the complexity of computing all hitting times down to N^2.

3 Numerical Examples

In order to test the speed and precision of our algorithm we created a procedure that randomly generates birth-and-death chains on trees. This procedure is based on algorithms found in Quiroz (1989) which randomly generate trees, either with a fixed number k of descendants per vertex ($k - ary$ trees, which we call type k trees) or trees with no restriction on the number of descendants per vertex (which we call *free* trees). Our procedure takes the resulting tree and randomly assigns non-zero transition probabilities between neighboring vertices. All the calculations were implemented in Fortran using Silverfrost FTN95.

3.1 About the Computation of the Stationary Distribution

We generated trees of several thousand vertices using this procedure. For each of them, the stationary distribution π^* was computed and

$$\max_{u \in V} \frac{|(\pi^* - \pi^* \mathbf{P})_u|}{\pi^*_u},$$

that is, the maximum relative error, was recorded.

The results follow:

N	Type	Execution time (sec.)	Max. rel. error
1,000	free	0.0156	2.305 E-07
5,000	free	0.1716	2.296 E-07
20,000	free	0.4524	2.508 E-07
1,001	1	0.0468	2.227 E-07
3,165	2	0.1248	2.373 E-07
20,001	2	0.4524	2.655 E-07
15,685	3	0.3432	2.339 E-07
30,202	3	0.7020	2.694 E-07
24,893	7	0.7956	3.095 E-07
75,529	24	1.0764	4.460 E-07
116,071	73	1.5912	6.917 E-07
60,001	$600 \approx N/100$	1.3260	1.542 E-06
200,001	$2000 \approx N/100$	4.4460	2.698 E-06
40,001	$4,000 \approx N/10$	0.8736	2.421 E-06
6,523	$2,174 \approx N/3$	0.1560	2.292 E-06
15,151	$5,050 \approx N/3$	0.3276	2.400 E-06
7,003	$3,501 \approx N/2$	0.1560	2.631 E-06
5,684	$5,683 \approx N$	0.1248	1.001 E-06
12,031	$12,030 \approx N$	0.2808	4.567 E-06

Though our method is recursive, using previously obtained conductances in order to compute new ones, the result for the stationary distribution appears to be very precise even in graphs of tens of thousands of vertices. As the fourth column shows, the maximum relative error seems to grow as the type grows to N, that is, as the number of descendants per vertex grows to N; but it stays below 10^{-5}. This shows how reliable this method is for calculating the stationary distribution.

The execution time stays below 1 second for graphs of less than 50,000 vertices. It stays below 2 seconds for graphs of size up to 120,000 and only reaches 4.4 seconds for the 200,001 vertex 2000-ary tree. Therefore, even in these large graphs, the computation of the stationary distribution is made in a reasonably short time.

3.2 Linearity of the Computation of a Single Hitting Time and the Stationary Distribution

The algorithm provided in Section 2 has been shown to be linear under the assumption that there is no relevant computational cost of extracting data from the matrix of transition probabilities **P** and the matrix which stores the conductances. However, we mentioned that the storage complexity could also be made linear by representing each matrix in the form of 3 vectors of length no greater than $3N - 2$. Extracting data from this vector structure involves more computations. Here we will exemplify that the method presented is still linear when the data is stored in this way.

In order to visualize the complexity, 10 trees of size $N = 3^6$, 10 of size 3^7, and 10 of size 3^8 were generated randomly with no restriction on the degree of their vertices. The transition probability matrix **P** was stored in the form of three vectors, as specified in

Section 2. We measured the execution time of the following three procedures: the computation of corresponding conductances and their storage (also in the form of three vectors), the computation of the stationary distribution and the computation of one hitting time. Then the logarithm of the size of the trees, N, was plotted against the logarithm of the execution time, t. The plot, shown in Fig. 2, was made using MATLAB and shows the linearity of the relationship between N and t for the proposed method, for the slope of the line made by the corresponding dots is close to one. This implies the individual experimental linearity of obtaining the conductances of the random walk, the computation of a single hitting time and the computation of the stationary distribution.

In the case of the computation of the stationary distribution we mentioned that there are methods for solving sparse linear systems of equations that could bring down the cost of solving $\pi \mathbf{P} = \pi$ with the additional equation $\sum_i \pi_i = 1$. In Davis (2006) the recommended method for this type of systems is the QR decomposition with Givens' rotation. This book also mentions that MATLAB's backslash ($mldivide$) executes this method automatically when the input matrix is sparse and has more rows than columns. For the same trees mentioned before the time taken by this method to solve the corresponding system was recorded for each of them. The log-log plot of the size of the trees against the execution time is also shown in Fig. 2.

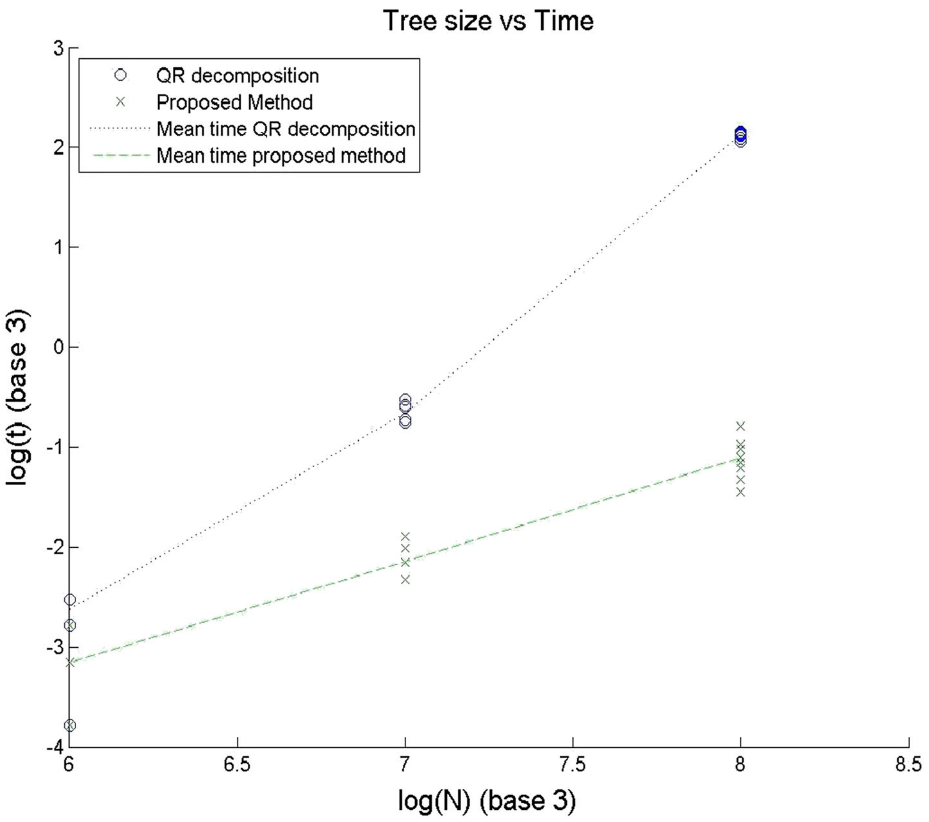

Fig. 2 Experimental complexity of the algorithms

From Fig. 2 we can see that the experimental order of the complexity of obtaining the stationary distribution through QR decomposition is clearly greater than the complexity of the method presented in this paper, and that it is approximately N^2.

3.3 About the Precision on the Computation of the Hitting Times

If the assignment of conductances is started from different vertices, with the same initial arbitrary conductance, the resulting conductances in the graph will be different. Even if the computation of the hitting times should not be affected by this, the results obtained numerically for the hitting times could differ when starting the algorithm from different vertices. In order to look for this type of error, the algorithm of assigning conductances was carried out starting from 4 different vertices. Then 4 different hitting times where obtained (the choice, from left to right in the tables, was the following: (i) both the start and the finish vertices are leaves (ii) only the start is a leaf (iii) only the end is a leaf (iv) neither the start nor the end are leaves). The maximum relative difference between the obtained hitting times was computed and set on the last row of the tables. This was performed for one graph of size N=5000 and one of size N=15000, both with no restriction on the degree of their vertices.

Initial vertex	$E_{1330}T_{1636}$	$E_{3289}T_{1904}$	$E_{2107}T_{4588}$	$E_{2187}T_{1803}$
1708	1.217774 E+21	5.975456 E+12	1.304592 E+16	4.890942 E+15
4986	1.217775 E+21	5.975454 E+12	1.304592 E+16	4.890941 E+15
293	1.217776 E+21	5.975459 E+12	1.304593 E+16	4.890942 E+15
2469	1.217776 E+21	5.975464 E+12	1.304593 E+16	4.890944 E+15
Max. Rel. Dif.	1.642340 E−06	1,673513 E−06	7.665231 E−07	6.133788 E−07

Hitting times obtained through proposed method, N=5000

Initial vertex	$E_{9879}T_{4701}$	$E_{3115}T_{11857}$	$E_{13263}T_{10736}$	$E_{472}T_{6352}$
3002	1.683652 E+27	4.278469 E+19	2.559966 E+22	3.250613 E+22
7656	1.683653 E+27	4.278470 E+19	2.559969 E+22	3.250616 E+22
14318	1.683652 E+27	4.278470 E+19	2.559969 E+22	3.250616 E+22
317	1.683652 E+27	4.278470 E+19	2.559967 E+22	3.250613 E+22
Max. Rel. Dif.	5.939469 E−07	2.337284 E−07	1.171890 E−06	9.229028 E-07

Hitting times obtained through proposed method, N=15000

For both graphs the proposed method shows no ill-posedness, presenting no relative difference larger than 10^{-5} as the initial vertex changes, for any of the hitting times computed.

We checked the results obtained by the proposed method against those obtained by the classical method which uses the fundamental matrix $\mathbf{Z}$. For this purpose a graph of size N=500 was created. Six hitting times of this graph were computed, both by the proposed method and by the classic method. The relative difference between the results obtained was also computed (normalizing by the result of smallest absolute value). Note that the classical procedure for obtaining hitting times needs the matrix $\mathbf{W}$, with all rows identical to the stationary distribution π, and it was obtained through our algorithm. Also, the matrix inversion needed to obtain $\mathbf{Z}$ was done using inv function of the free software package GNU Octave.

Hitting time	Proposed method	Classic method	Relative difference
$E_{106}T_{17}$	34.3233	−2115.01	95.943540
$E_{214}T_{158}$	2.18007	2435.48	1116.1567
$E_{341}T_{351}$	1.490487 E+12	1.492161 E+12	0.001123
$E_{461}T_{114}$	567123	566713.5	7.225873 E−4
$E_{206}T_{447}$	6.895986 E+09	−8.561326 E+09	2.241494
$E_{154}T_{413}$	8.680532 E+11	8.676188 E+11	6.506707 E−4

Hitting times obtained through both methods, N=500

For $E_{341}T_{351}$, $E_{461}T_{114}$, and $E_{154}T_{413}$ both methods obtained similar results. In these three cases the relative difference did not exceed 10^{-3}. However, for the other three hitting times the relative difference exceeded 1. Moreover, the classic method produced negative hitting times, something which shows clearly how unreliable a method which involves the inversion of a matrix can be, even for this relatively small graph.

We carried out a series of simulations to validate the results for $E_{106}T_{17}$ and $E_{214}T_{158}$. Specifically, 10,000 random walks were performed, starting from vertex 106 and ending in vertex 17. The average number of steps taken turned out to be 34.2614. Similarly, 10,000 random walks were performed from vertex 214 to vertex 158. The average number of steps was 2.1893. These results support those obtained by the proposed method. Given the magnitude of the third hitting time with big differences in the methods, $E_{206}T_{447}$, it seemed unreasonable to try to confirm this result using simulations that would take an inordinate amount of time.

A final remark on the computation of hitting times, it should be noted that sometimes, when we exceed a size of 50,000 and even sometimes for smaller graphs, the proposed algorithm returned a floating point error. This error is associated with the equations in Eq. 6. As the conductances used in this formulas can get very small, if the term they have to be multiplied by is also very small we might get a numerical 0. Another possibility is that these conductances are very large (we have noticed in our experiments that the distribution of these conductances, far from the root, has a very heavy tail) and if the term they have to be multiplied by is also very large we can get an overflow error.

4 Computing All Hitting Times in a Tree

In this section we shall assume that the B.D. chain on a tree has been turned into a random walk on the same tree through the linear algorithm discussed in the previous sections. Restricted to the case where there are no loops (no transition probability from a state to itself), we want to show that the computation of all hitting times can be brought down to an N^2 complexity. This is achieved by first computing all hitting times between adjacent vertices, and then computing the hitting times between more distant vertices.

Given any tree, it is well known that we can find a traversal walk such that all its edges are traversed exactly once in each direction (see Tarry (1895)). In the first part of this procedure we obtain the hitting times between neighbors, and to do so we use this traversal walk. When the edge (i, j) is first visited (and assuming the walk visits vertex i before vertex j), $E_i T_j$ is computed by the linear implementation of formula (3) introduced in Section 2. Eventually, this edge will be visited in the opposite direction and then $E_j T_i$ will be obtained. Since computing each hitting time is linear and $2N - 2$ hitting times are to be computed, this first step of the procedure is of order N^2 regarding the number of operations. The hitting times obtained are to be stored in a matrix, say M, such that $M_{i,j} = E_i T_j$, and so this part is also of order N^2 as far as the storage complexity is concerned.

Since we are restricted to the case were there are no loops, we can apply the following formula to obtain the remaining hitting times:

$$E_x T_y = E_x T_{v_1} + E_{v_1} T_{v_2} + \cdots + E_{v_n} T_y \tag{8}$$

where $v_1, v_2, ..., v_n$ is the (unique) path of length $n + 1, n \geq 1$ between x and y.

Given a fixed vertex x we compute $E_x T_y$ for every other vertex y through the following procedure:

1. $M_{x,x} = E_x T_x$ is assigned the value 0.
2. A BFS search starts with x as the root.
3. Given that the root is in generation 0 and its neighbors in generation 1, for every vertex i in generation $k \geq 1$, $E_x T_i$ is obtained as $E_x T_p + E_p T_i$, where p is the "parent" of i, belonging to generation $k - 1$. $M_{x,i} := E_x T_i$.

The BFS search visits every vertex once and, therefore, it is linear in N. But since we must do that search for every vertex x, this part of the procedure is quadratic in the number of operations. Now since the first part was also quadratic and both parts are performed in a sequence, then the whole procedure is of order N^2 regarding the number of operations. Finally, since the only relevant element of storage is the matrix M of hitting times, the procedure is also of order N^2 regarding the storage complexity.

References

Bertoncini O (2011) Cut-off and escape behavior for birth and death chains on trees. Lat A J Probab Math Stat Phys 8:149–162

Davis TA (2006) Direct methods for sparse linear systems, Siam book series on the fundamentals of algorithms, SIAM, Philadelphia

Dongarra J (2000) Sparse matrix storage formats. In: Bai Z, Demmel J, Dongarra J, Ruhe A, van der Vorst H (eds) Templates for the solution of algebraic eigenvalue problems: a practical guide. SIAM, Philadelphia

Doyle PG, Snell JL (1984) Random walks and electrical networks. The Mathematical Association of America, Washington DC

Fayolle G, Krikun M, Lasgouttes JM (2004) Birth and death processes on certain random trees: classification and stationary laws. Probab Theory Relat Fields 128:386–418

Grinstead CM, Snell JL (1997) Introduction to probability, second edition, American Mathematical Society. Providence RI

Kelly FP (1979) Reversibility and stochastic networks. Wiley, Chichester

Ma Y (2010) Birth-death processes on trees. Sci China Math 53:1–10

Palacios JL, Tetali P (1996) A note on expected hitting times for birth and death chains. Stat Probab Lett 30:119–125

Palacios JL (2009) On hitting times of random walks on trees. Stat Probab Lett 79:234–236

Quiroz AJ (1989) Fast random generation of binary, t-ary and other types of trees. J Class 6:223–231

Tarry G (1895) Le problème des labyrinthes. Nouvelles Annales de Mathématiques, ser 3(14):187–190