Computing 2-Step Predictions for Interval-Valued Finite Stationary Markov Chains

Marcilia Andrade Campos[1], Graçaliz Pereira Dimuro[2], Antônio Carlos da Rocha Costa[2], and Vladik Kreinovich[3]

[1]Centro de Informática, Universidade Federal de Pernambuco
50670-901, Recife, Brazil, mac@cin.ufpe.br
[2]Escola de Informática, Universidade Católica de Pelotas
Rua Felix da Cunha 412, 96010-000, Pelotas, Brazil
liz@atlas.ucpel.tche.br, rocha@atlas.ucpel.tche.br
[3]Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA, vladik@cs.utep.edu

Abstract

Markov chains are a useful tool for solving practical problems. In many real-life situations, we do not know the exact values of initial and transition probabilities; instead, we only know the intervals of possible values of these probabilities. Such interval-valued Markov chains were considered and analyzed by I. O. Kozine and L. V. Utkin in their *Reliable Computing* paper. In their paper, they propose an efficient algorithm for computing interval-valued probabilities of the future states. For the general case of non-stationary Markov chains, their algorithm leads to the exact intervals for the probabilities of future states.

In the important case of stationary Markov chains, we can still apply their algorithm. For stationary chains, 1-step predictions are exact but 2-step predictions sometimes lead to intervals that are wider than desired. In this paper, we describe a modification of Kozine-Utkin algorithm that always produces exact 2-step predictions for stationary Markov chains.

1 Formulation of the Problem

Markov chains: brief reminder. In many real-life systems ranging from weather to hardware to psychological systems, transition probabilities do not depend on the history, only on the current state. Such systems are naturally described as finite *Markov chains*; see, e.g., [2, 3, 8, 14, 15, 16, 17].

In a finite Markov chain, we have a finite set of possible states $s_1, \ldots, s_n$. At any moment of time t, the state of a system is described by the probabilities $b_i(t)$ of the state s_i; these probabilities must add up to 1: $\sum\limits_{i=1}^{n} b_i(t) = 1$. We also have, for each moment t, the transition probabilities

$$p_{ij}(t) \stackrel{\text{def}}{=} P(s(t) = s_j \,|s(t-1) = s_i)$$

that satisfy the condition

$$\sum_{j=1}^{n} p_{ij} = 1. \tag{1}$$

Once we know the state probability vector $b(t) = (b_1(t), \ldots, b_n(t))$ at the moment t, we can predict the probabilities $b_i(t+1)$ at the next moment of time $t+1$ by using the formula $b_j(t+1) = \sum\limits_{i=1}^{n} b_i(t) \cdot p_{ij}(t+1)$. Once we know the initial probabilities $b_i(0)$ and the transition probabilities $p_{ij}(t)$, we can use the above formula to compute $b_i(1)$, $b_i(2)$, $\ldots$, and thus, to predict the probabilities $b_i(t)$ of different states at different future moments of time.

A practically important case of a Markov chain is a *stationary* Markov chain in which the transition probabilities do not depend on time: $p_{ij}(t) = p_{ij}$.

Interval-valued finite Markov chains: reminder. In many practical situations, we do not know the exact values of the state and transition probabilities. For example, for transition probabilities $p_{ij}(t)$, often, we only know a lower bound $\underline{p}_{ij}$ and an upper bound $\overline{p}_{ij}$. In other words, for every i and j, we only know the interval $\mathbf{p}_{ij} = [\underline{p}_{ij}, \overline{p}_{ij}]$ of possible values of p_{ij}.

The general situation in which we have partial information about the probability distribution is described, e.g., in [11, 19, 20]. In the corresponding theory, to describe uncertainty, instead of a *single* probability distribution, we must describe a *class* of probability distributions (e.g., all the distributions consistent with given measurements and/or given expert estimates). Interval-valued probabilities are an important particular case of such a class.

In their pioneer work [9], I. O. Kozine and L. V. Utkin extended the formalism of finite Markov chains to such interval-valued probabilities. In their approach, the information about the initial state is described by listing, for each i, the interval $[\underline{b}_i(0), \overline{b}_i(0)]$ of possible values of $b_i(0)$. This means that the actual probabilities $b_i(0)$ must satisfy the conditions

$$\sum_{i=1}^{n} b_i(0) = 1;\ \underline{b}_1(0) \le b_1(0) \le \overline{b}_1(0);\ \ldots\ \underline{b}_n(0) \le b_n(0) \le \overline{b}_n(0). \tag{2}$$

We assume that these probability intervals are *coherent*, i.e., that for every i and for every value $b_i(0) \in [\underline{b}_i(0), \overline{b}_i(0)]$, there exists a vector

$(b_1(0), \ldots, b_i(0), \ldots, b_n(0))$ with this value $b_i(0)$ that satisfies the condition (2). Similarly, the information about the transition probabilities $p_{ij}(t)$ is described by listing, for each i, j, and t, the interval $[\underline{p}_{ij}(t), \overline{p}_{ij}(t)]$.

Based on this information, we must predict the probability $b_i(t)$ of different states at a future moment of time t. Since we only know the values $b_i(0)$ and $p_{ij}(t)$ with interval uncertainty, we can only predict the *interval* $[\underline{b}_i(t), \overline{b}_i(t)]$ of possible values of $b_i(t)$.

Kozine-Utkin algorithm: reminder. In [9], Kozine and Utkin proved that once we know the interval-valued state probabilities $[\underline{b}_i(t-1), \overline{b}_i(t-1)]$ and the interval-valued transition probabilities $[\underline{p}_{ij}(t), \overline{p}_{ij}(t)]$, we can compute the (endpoints of the) interval $[\underline{b}_i(t), \overline{b}_i(t)]$ by solving the following two linear programming problems: $\underline{b}_j(t) = \inf\limits_{b_i(t-1)} \sum\limits_{i=1}^{n} b_i(t-1) \cdot \underline{p}_{ij}(t)$ and $\overline{b}_j(t) = \sup\limits_{b_i(t-1)} \sum\limits_{i=1}^{n} b_i(t-1) \cdot \underline{p}_{ij}(t)$ under the conditions

$$\sum_{i=1}^{n} b_i(t-1) = 1;$$

$$\underline{b}_1(t-1) \le b_1(t-1) \le \overline{b}_1(t-1); \ \ldots \ \underline{b}_n(t-1) \le b_n(t-1) \le \overline{b}_n(t-1).$$

They show that this algorithm leads to the exact interval for $[\underline{b}_i(t), \overline{b}_i(t)]$.

Thus, if we know the initial interval-valued probabilities $[\underline{b}_i(0), \overline{b}_i(0)]$ and we want to predict the state at time t, we can use the Utkin-Kozine algorithm to sequentially compute the interval-valued probabilities $[\underline{b}_i(1), \overline{b}_i(1)]$, $[\underline{b}_i(2), \overline{b}_i(2)]$, $\ldots$, until we get the desired probabilities $[\underline{b}_i(t), \overline{b}_i(t)]$.

For stationary interval-valued Markov chains, there is room for improvement. For the *general* non-stationary case, when at each moment of time t, we may have different transition probabilities $p_{ij}(t) \in [\underline{p}_{ij}(t), \overline{p}_{ij}(t)]$, this step-by-step algorithm leads to the exact interval for $[\underline{b}_i(t), \overline{b}_i(t)]$.

Let us show, on a simple example, that for *stationary* Markov chains, when the transition probabilities are the same at all moments of time, this algorithm may lead to a proper enclosure, i.e., to the interval that is wider than desired.

In our simple example, there are two states s_1 and s_2. Initially, the system is in the state s_1, i.e., $\mathbf{b}_1(0) = [1, 1]$ and $\mathbf{b}_2(0) = [0, 0]$. The information about the transition probabilities is as follows:

- we know that the state s_2 always transforms into a state s_1 at the next moment of time, and
- we have no information about the transitions from the state s_1.

In precise terms, $\mathbf{p}_{11} = [0,1]$, $\mathbf{p}_{12} = [0,1]$, $\mathbf{p}_{21} = [1,1]$, $\mathbf{p}_{22} = [0,0]$. Based on this information, we want to predict the state at moment $t = 2$. We assume that the Markov chain is stationary, i.e., that the transition probabilities are the same for the transition from moment 0 to moment 1 and for the transition from moment 1 to moment 2: $p_{ij}(2) = p_{ij}(1)$.

Let us first describe the possible values of the desired probabilities $b_i(2)$; after that, we will show that the Kozine-Utkin algorithm leads to a wider interval.

In this example, there is no uncertainty in the initial state, the only uncertainty is in the transition probabilities. Once we select p_{11}, we can use the fact that $p_{11} + p_{12} = 1$ to determine the probability p_{12} as $1 - p_{11}$. Thus, once we select p_{11}, we uniquely determine the transition probability matrix

$$P = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix} = \begin{pmatrix} p_{11} & 1 - p_{11} \\ 1 & 0 \end{pmatrix},$$

and therefore, we can determine the state $b(2) = (b_1(2), b_2(2))$ by using the known formula $b(2) = b(0)P^2$ (see, e.g., [8]). In our case, we have

$$b(1) = (p_{11}, 1 - p_{11}) \text{ and } b(2) = (p_{11}^2 + 1 - p_{11}, p_{11} \cdot (1 - p_{11})).$$

In particular, $b_2(2) = p_{11} \cdot (1 - p_{11})$.

When $p_{11} \in [0,1]$, the possible values of $p_{11} \cdot (1 - p_{11})$ form the interval $[0, 0.25]$. Thus, the desired interval of possible values $\mathbf{b}_2(2)$ is $[0, 0.25]$. Similarly, for b_1, we get the range $\mathbf{b}_1(2) = [0.75, 1]$.

Let us now describe what the algorithm from [9] will compute. For our example, according to the algorithm from [9], we must first compute $\mathbf{b}(1)$, and then use these interval-valued probabilities to compute (2). Since $b(1) = (p_{11}, 1 - p_{11})$ and $p_{11} \in [0,1]$, the set of possible values of $b_1(1)$ is exactly $[0,1]$, and the set of possible values of $b_2(1)$ is also exactly $[0,1]$. One can easily check that the above linear programming programs lead to exactly these intervals.

The interval-valued state $\mathbf{b}(1) = ([0,1], [0,1])$ means, in effect, that we have no information about the probabilities $b_i(1)$. In this case, intuitively, we cannot predict anything about $b_i(2)$ either, i.e., we should have $\mathbf{b}(2) = ([0,1], [0,1])$. Indeed, the above linear programming problem leads to exactly these intervals.

Thus, for $b_1(2)$, we get an interval $[0,1] \supset [0.75, 1]$, and for $b_2(2)$, we get an interval $[0,1] \supset [0, 0.25]$. In both cases, the resulting intervals are wider than desired.

What we are planning to do. In this paper, we describe a modification of Kozine-Utkin algorithm that always produces exact 2-step predictions for stationary Markov chains.

2 Main Idea

As we have mentioned, the Kozine-Utkin algorithm provides the exact interval for the state probabilities for 1-step predictions, i.e., for the case when we know the initial (interval-valued) state probabilities $\mathbf{b}(0)$ and the (interval-valued) transition probabilities $\mathbf{p}_{ij}$, and we need to apply these transition probabilities only once.

To help 2-step predictions, it is therefore reasonable to describe interval-valued 2-step transition probabilities $\mathbf{p}_{ij}^{(2)}$ that describe transitions from the moment $t = 0$ directly into the moment $t = 2$. Once we get these 2-step probabilities, we can apply the Kozine-Utkin algorithm to $\mathbf{b}(0)$ and $\mathbf{p}_{ij}^{(2)}$ and get the exact estimates for $\mathbf{b}(2)$.

If we knew the exact values p_{ij} of the (1-step) transition probabilities, then we would be able to compute the 2-step transition probabilities

$$p_{ij}^{(2)} \stackrel{\text{def}}{=} P(s(t) = s_j \,|\, s(t-2) = s_i)$$

by using the known formula:

$$p_{ij}^{(2)} = \sum_{k=1}^{n} p_{ik} \cdot p_{kj}. \tag{3}$$

Thus, in the interval-valued case, in order to find the bounds $\underline{p}_{ij}$ and $\overline{p}_{ij}$, we must solve the corresponding quadratic optimization problems:

$$\underline{p}_{ij}^{(2)} = \inf_{p_{kl}} \sum_{k=1}^{n} p_{ik} \cdot p_{kj} \text{ and } \overline{p}_{ij}^{(2)} = \inf_{p_{kl}} \sum_{k=1}^{n} p_{ik} \cdot p_{kj}$$

under the conditions

$$\sum_{l=1}^{n} p_{kl} = 1; \quad \underline{p}_{kl} \le p_{kl} \le \overline{p}_{kl}.$$

From the mathematical viewpoint, the problem is solved: this optimization problem will leads us to the exact interval for $p_{ij}^{(2)}$ – and thus, to the exact interval for $\mathbf{b}(2)$.

However, from the computational viewpoint, we still have a problem. Indeed, in the formulation of the Kozine-Utkin algorithm, we need to solve a *linear* programming problem. It is known that there exist efficient algorithms for solving such problems, so we can simply use one of the known algorithms.

In our case, we must solve a *quadratic* optimization problem. It is known that in general, quadratic optimization is NP-hard [10, 18]. This means, crudely speaking, that no efficient general algorithm is possible for solving such problems. As a result, we cannot rely on such a general algorithm, we must design a new efficient algorithm for solving our specific quadratic optimization problems.

This algorithm will be presented in the following section.

3 Main Result

In this section, we describe efficient ($O(n^3)$) algorithms for computing the exact lower and upper bounds for 2-step transition probabilities for interval-valued Markov chains.

Each algorithm consists of three parts. Both algorithm use an auxiliary "peeling" algorithm for solving quadratic optimization problems with few variables. This peeling algorithm is described in the following section.

The justification for our algorithms is given in the Appendix.

3.1 Auxiliary Peeling Algorithm for Solving Quadratic Optimization Problems: Reminder

As part of our new algorithms, we will need to find the maximum and the minimum of a given quadratic function under linear constraints (equalities and inequalities).

To solve these auxiliary optimization problems, we can use the idea of peeling; see, e.g, [6]. The idea of peeling is a natural extension of the known simplex techniques for solving linear programming problems. This idea can be described as follows.

From the geometric viewpoint, a region described by linear equalities and inequalities is a polytope. The maximum (or a minimum) of a function in this region is attained either in the interior of this polytope, or in one of its lower-dimensional boundary polyhedral elements: faces, faces of the faces, ..., all the way to 0-dimensional elements – vertices.

Based on the equalities and inequalities that describe a polytope, we can explicitly describe all these polyhedral elements; there are $\approx 2^m$ of them, where m is the overall number of variables and constraints. For each of these boundary elements, we can select independent variables $x_{i_1}, \ldots, x_{i_d}$ – as many as the dimension d of this boundary element – and explicitly describe other variables x_i as linear functions of these independent ones. (In the limit case, when we consider vertices – 0-dimensional boundary elements – there are no independent variables at all.) If we substitute the expressions for all the variables in terms of independent ones into the optimized quadratic function, then we get an expression $E(x_{i_1}, \ldots, x_{i_d})$ for this quadratic function in terms of d independent variables $x_{i_1}, \ldots, x_{i_d}$ only. If the minimum or maximum of this expression is in the interior of the boundary element, then all d partial derivatives w.r.t. these variables should be equal to 0:

$$\frac{\partial E}{\partial x_{i_k}} = 0. \tag{4}$$

Since the derivative of a quadratic function is a linear function, the equations (4) forms a system of d linear equations with d unknowns $x_{i_1}, \ldots, x_{i_d}$. This system

is easy to solve, so for each boundary element, we get a possible optimum point. Of course, we also need to check that this solution does belong to this boundary element (and not to its extension) by checking that all linear inequalities that define this element are satisfied. (For vertices, we simply compute the value of the quadratic function.)

Summarizing: we know that, e.g., the minimum is always attained either in the interior of the polytope, or in the interior of one of the boundary elements, and for each boundary element, we know how to compute the point where this minimum is attained (if at all). Thus, to find the minimum of the given quadratic function, we simply analyze all the boundary elements this way, and then take the smallest of the corresponding values of the minimized quadratic function.

3.2 Algorithm for Computing the Exact Upper Bound

First part. First, for each i from 1 to n, we sort the values $\overline{p}_{1i}, \ldots, \overline{p}_{ni}$ into a decreasing sequence.

Second part. Then, for each i from 1 to n, to compute the exact upper bound $\overline{p_{ii}^{(2)}}$ for $p_{ii}^{(2)}$, we do the following. First, by deleting the value $\overline{p}_{ii}$ from the sorted sequence, we get a sorting of all the values $\overline{p}_{ki}$ ($k \neq i$) into a decreasing sequence

$$\overline{p}_{(1)i} \geq \overline{p}_{(2)i} \geq \ldots \geq \overline{p}_{(n-1)i}. \tag{5}$$

Then, for each $k' = 1, 2, \ldots, n-1$, we do the following:

- first, we compute the value $c_{k'}$; for $k' = 1$, we use the formula

$$c_1 = 1 - \sum_{k:k>1} \underline{p}_{i(k)}; \tag{6}$$

  for $k' > 1$, we use the formula

$$c_{k'} = c_{k'-1} - \overline{p}_{i(k'-1)} + \underline{p}_{i(k')}; \tag{7}$$

- we use peeling to solve the problem of optimizing a quadratic function

$$p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ii}^2 \to \max \tag{8}$$

  of two variables $p_{i(k')}$ and p_{ii} under linear conditions that

$$\underline{p}_{i(k')} \leq p_{i(k')} \leq \overline{p}_{i(k')}, \tag{9}$$

$$\underline{p}_{ii} \leq p_{ii} \leq \overline{p}_{ii}, \tag{10}$$

  and

$$p_{i(k')} + p_{ii} = c_{k'}; \tag{11}$$

- we compute the value $W_{k'}$; for $k' = 1$, we use the formula

$$W_1 = \sum_{k:k>1} \underline{p}_{i(k)} \cdot \overline{p}_{(k)i}; \tag{12}$$

  for $k' > 1$, we use the formula

$$W_{k'} = W_{k'-1} + \overline{p}_{i(k'-1)} \cdot \overline{p}_{(k'-1)i} - \underline{p}_{i(k')} \cdot \overline{p}_{(k')i}; \tag{13}$$

- based on the solution of the quadratic optimization problem, we compute $V_{k'}$ by using the formula

$$V_{k'} = W_{k'} + p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ii}^2. \tag{14}$$

The actual maximum of $p_{ii}^{(2)}$ can then be determined as the largest of the corresponding values $V_{k'}$.

Third part. Finally, for each i from 1 to n and for each j from 1 to n, to compute the exact upper bound $\overline{p_{ij}^{(2)}}$ for $p_{ij}^{(2)}$, we do the following. First, by deleting the values $\overline{p}_{ii}$ and $\overline{p}_{ij}$ from the sorted sequence, we get a sorting of all the values $\overline{p}_{ki}$ $(k \neq i, j)$ into a decreasing sequence

$$\overline{p}_{(1)i} \geq \overline{p}_{(2)i} \geq \cdots \geq \overline{p}_{(n-2)i}. \tag{15}$$

Then, for each $k' = 1, 2, \ldots, n-1$, we do the following:

- first, we compute the value $c_{k'}$; for $k' = 1$, we use the formula (6); for $k' > 1$, we use the formula (7);

- we use peeling to solve the problem of optimizing a quadratic function

$$p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ij} \cdot (p_{ii} + \overline{p}_{jj}) \to \max \tag{16}$$

  of three variables $p_{i(k')}$, p_{ii}, and p_{ij} under linear conditions that

$$\underline{p}_{i(k')} \leq p_{i(k')} \leq \overline{p}_{i(k')}, \tag{17}$$

$$\underline{p}_{ii} \leq p_{ii} \leq \overline{p}_{ii}, \tag{18}$$

$$\underline{p}_{ij} \leq p_{ij} \leq \overline{p}_{ij}, \tag{19}$$

  and

$$p_{i(k')} + p_{ii} + p_{ij} = c_{k'}; \tag{20}$$

- we compute the value $W_{k'}$; for $k' = 1$, we use the formula (12); for $k' > 1$, we use the formula (13);

- based on the solution of the quadratic optimization problem, we compute $V_{k'}$ by using the formula

$$V_{k'} = W_{k'} + p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ij} \cdot (p_{ii} + \overline{p}_{jj}). \tag{21}$$

The actual maximum of $p_{ij}^{(2)}$ can then be determined as the largest of the corresponding values $V_{k'}$.

3.3 Algorithm for Computing the Exact Lower Bound

First part. First, for each i from 1 to n, we sort the values $\underline{p}_{1i}, \ldots, \underline{p}_{ni}$ into a decreasing sequence.

Second part. Then, for each i from 1 to n, to compute the exact lower bound $\underline{p_{ii}^{(2)}}$ for $p_{ii}^{(2)}$, we do the following. First, by deleting the value $\underline{p}_{ii}$ from the sorted sequence, we get a sorting of all the values $\underline{p}_{ki}$ $(k \neq i)$ into a decreasing sequence

$$\underline{p}_{(1)i} \geq \underline{p}_{(2)i} \geq \cdots \geq \underline{p}_{(n-1)i}. \tag{22}$$

Then, for each $k' = 1, 2, \ldots, n-1$, we do the following:

- first, we compute the value $c_{k'}$; for $k' = 1$, we use the formula

$$c_1 = 1 - \sum_{k:k>1} \overline{p}_{i(k)}; \tag{23}$$

for $k' > 1$, we use the formula

$$c_{k'} = c_{k'-1} - \underline{p}_{i(k'-1)} + \overline{p}_{i(k')}; \tag{24}$$

- we use peeling to solve the problem of optimizing a quadratic function

$$p_{i(k')} \cdot \underline{p}_{(k')i} + p_{ii}^2 \to \min \tag{25}$$

of two variables $p_{i(k')}$ and p_{ii} under linear conditions that

$$\underline{p}_{i(k')} \leq p_{i(k')} \leq \overline{p}_{i(k')}, \tag{26}$$

$$\underline{p}_{ii} \leq p_{ii} \leq \overline{p}_{ii}, \tag{27}$$

and

$$p_{i(k')} + p_{ii} = c_{k'}; \tag{28}$$

- we compute the value $W_{k'}$; for $k' = 1$, we use the formula

$$W_1 = \sum_{k:k>1} \overline{p}_{i(k)} \cdot \underline{p}_{(k)i}; \tag{29}$$

  for $k' > 1$, we use the formula

$$W_{k'} = W_{k'-1} + \underline{p}_{i(k'-1)} \cdot \underline{p}_{(k'-1)i} - \overline{p}_{i(k')} \cdot \underline{p}_{(k')i}; \tag{30}$$

- based on the solution of the quadratic optimization problem, we compute $V_{k'}$ by using the formula

$$V_{k'} = W_{k'} + p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ii}^2. \tag{31}$$

The actual minimum of $p_{ii}^{(2)}$ can then be determined as the smallest of the corresponding values $V_{k'}$.

Third part. Finally, for each i from 1 to n and for each j from 1 to n, to compute the exact lower bound $\underline{p_{ij}^{(2)}}$ for $p_{ij}^{(2)}$, we do the following. First, by deleting the values $\underline{p}_{ii}$ and $\underline{p}_{ij}$ from the sorted sequence, we get a sorting of all the values $\underline{p}_{ki}$ $(k \neq i, j)$ into a decreasing sequence

$$\underline{p}_{(1)i} \geq \underline{p}_{(2)i} \geq \cdots \geq \underline{p}_{(n-2)i}. \tag{32}$$

Then, for each $k' = 1, 2, \ldots, n-1$, we do the following:

- first, we compute the value $c_{k'}$; for $k' = 1$, we use the formula (23); for $k' > 1$, we use the formula (24);
- we use peeling to solve the problem of optimizing a quadratic function

$$p_{i(k')} \cdot \underline{p}_{(k')i} + p_{ij} \cdot (p_{ii} + \underline{p}_{jj}) \to \min \tag{33}$$

  of three variables $p_{i(k')}$, p_{ii}, and p_{ij} under linear conditions that

$$\underline{p}_{i(k')} \leq p_{i(k')} \leq \overline{p}_{i(k')}, \tag{34}$$

$$\underline{p}_{ii} \leq p_{ii} \leq \overline{p}_{ii}, \tag{35}$$

$$\underline{p}_{ij} \leq p_{ij} \leq \overline{p}_{ij}, \tag{36}$$

  and

$$p_{i(k')} + p_{ii} + p_{ij} = c_{k'}; \tag{37}$$

- we compute the value $W_{k'}$; for $k' = 1$, we use the formula (29); for $k' > 1$, we use the formula (30);

- based on the solution of the quadratic optimization problem, we compute $V_{k'}$ by using the formula

$$V_{k'} = W_{k'} + p_{i(k')} \cdot \underline{p}_{(k')i} + p_{ij} \cdot (p_{ii} + \underline{p}_{jj}). \tag{38}$$

The actual minimum of $p_{ij}^{(2)}$ can then be determined as the smallest of the corresponding values $V_{k'}$.

3.4 Example

For our simple example, in which the 1-step (interval-valued) transition probabilities are:

$$\mathbf{P} = \begin{pmatrix} [0,1] & [0,1] \\ [1,1] & [0,0] \end{pmatrix},$$

the above algorithm leads to the following interval-valued 2-step transition probabilities:

$$\mathbf{P}^{(2)} = \begin{pmatrix} [0.75,1] & [0,0.25] \\ [0,1] & [0,1] \end{pmatrix}.$$

When we apply the Kozine-Utkin algorithm to these interval-valued transition probabilities and to our initial state $\mathbf{b}(0) = ([1,1],[0,0])$, we get $\mathbf{b}(2) = ([0.75,1],[0,0.25])$ – exactly as desired.

3.5 How Efficient Are Our Algorithms?

We have mentioned that both our algorithms require computation time $O(n^3)$ to compute the exact lower and upper bounds for the 2-step transition probabilities. A natural question is: is it possible to design even faster algorithms?

An answer to this question comes from the fact that for the case of degenerate intervals, when we know the exact values of 1-step transition probabilities p_{ij}, the algorithms must still produce the standard 2-step transition probabilities: $p_{ij}^{(2)} = \sum_{k=1}^{n} p_{ik} \cdot p_{kj}$. According to this formula, to compute each of n^2 values $p_{ij}^{(2)}$, we need n multiplications and $n-1$ additions; hence, the total number of arithmetic operations is equal to $n^2 \cdot (n + (n-1)) = O(n^3)$.

So, the algorithm for a more general interval-valued case cannot be faster than $O(n^3)$. Since our algorithms require exactly this much time, we can conclude that, in terms of computation time, our algorithms are (asymptotically) optimal.

Comment. It is worth mentioning that for large n, we can, in principle, compute the 2-step transition probabilities faster than in $O(n^3)$ time: indeed, the formula (3) simply means that we multiply a matrix p_{ij} by itself, and it is known (see, e.g., [1] and references therein) that there are algorithms that multiply two

matrices in time $O(n^\alpha)$ for $\alpha < 3$. The first such algorithm was proposed by Strassen, and several other fast matrix multiplication algorithms have been proposed.

However, these algorithms are rarely practically used for Markov chains because they becomes more efficient than straightforward formula (3) only for very large n, and in most practical applications of Markov chains, we need chains with $n \ll 100$ states; see, e.g., [15].

4 Final Comment: What About 3-Step Probabilities?

A natural question is: we know how to compute the exact intervals for 2-step transition probabilities, what about 3-step transition probabilities? 4-step probabilities?

How to compute these probabilities and whether it is even possible to compute the exact intervals for these probabilities in reasonable time, is an open question. The reason why it may be difficult is that in general, just like the formula (3) means that the matrix formed by 2-step probabilities is a square of the matrix p_{ij}, the matrix formed by 3-step probabilities is a cube of the original matrix p_{ij}. The square of a matrix can be reduced to a single-use expression and thus, computed exactly even in the interval case. The cube of a matrix is difficult to represent in SUE form and, in general, computing the exact cube of an interval matrix is NP-hard [12].

This NP-hardness of a general problem does not necessarily mean that the sub-problem of computing the cube of a probability matrix is necessarily NP-hard, so there is hope that computing the exact intervals for 3-step transition probabilities may also turn out to be feasible.

Acknowledgments

This work was partially supported by the Brazilian funding agencies CNPq/CTPETRO, CNPq/CTINFO, and FAPERGS.

V.K. was also partly supported by NASA under cooperative agreement NCC5-209, by Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0365, by NSF grants EAR-0112968 and EAR-0225670, by the Army Research Laboratories grant DATM-05-02-C-0046, and by IEEE/ACM SC2003 Minority Serving Institutions Participation Grant.

This work was mainly done when Graçaliz Pereira Dimuro and Antônio Carlos da Rocha Costa visited El Paso, Texas; this visit was sponsored by CNPq/CTPETRO and CNPq/CTINFO.

The authors are very thankful to the anonymous referees for extremely useful suggestions.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, and Mc-Graw Hill Co., N.Y., 2001.

[2] D. Gamerman, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, CRC Press, Boca Raton, Florida, 1997.

[3] W. R. Gilks, S. Richardson, and D. L. Spiegelhalter (Eds.), *Markov Chain Monte Carlo in Practice*, Chapman & Hall, Boca Raton, Florida, 1996.

[4] E. Hansen, "Sharpness in interval computations", *Reliable Computing*, 1997, Vol. 3, pp. 7–29.

[5] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*, Springer, London, 2001.

[6] R. B. Kearfott, *Rigorous Global Search: Continuous Problems*, Kluwer, Dordrecht, 1996.

[7] R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

[8] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*, Springer Verlag, N.Y., 1976.

[9] I. O. Kozine and L. V. Utkin, "Interval-valued finite Markov chains", *Reliable Computing*, 2002, Vol. 8, No. 2, pp. 97–113.

[10] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1997.

[11] V. P. Kuznetsov, *Interval Statistical Models*, Radio and Communications publ., Moscow, 1991 (in Russian).

[12] G. Mayer and V. Kreinovich, *Computing $A \cdot A \cdot A$ is NP-hard*, working paper.

[13] R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[14] A. Papoulis, "Brownian Movement and Markoff Processes." In: *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1984, pp. 515–553.

[15] L. R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition", *Proceedings of the IEEE*, 1989, Vol. 77, No. 2, pp. 257–286.

[16] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, New Jersey, 1995.

[17] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, J. Wiley, New York, 2001.

[18] S. A. Vavasis, *Nonlinear Optimization: Complexity Issues*, Oxford University Press, New York, 1991.

[19] P. Walley, *Statistical Reasoning with Imprecise Probabilities*, Chapman & Hall, London, 1991.

[20] P. Walley, "Measures of uncertainty", *Artificial Intelligence*, 1996, Vol. 83, pp. 1–58.

Appendix: Analysis of the Problem and a Step-By-Step Design of Efficient Algorithms for Solving This Problem

Reduction to SUE expressions. We would like to use interval computations (see, e.g., [5, 6, 7, 13]) in our estimates. In interval computations, one known source of excess width is repetition of variables. It is known that if a formula is a *single-use expression* (SUE), i.e., if in this formula, each variable only occurs once, that for such formulas, straightforward interval computations lead to the exact range (see, e.g., [4]). To avoid this excess width, let us first represent the expression (3) in SUE form.

The original formulas have few repetitions of variables, so this reduction can be easily done. The resulting expressions are different for $i = j$ and for $i \neq j$. For $i = j$, we get the following SUE expression:

$$p_{ii}^{(2)} = \sum_{k \neq i} p_{ik} \cdot p_{kj} + p_{ii}^2. \tag{39}$$

For $i \neq j$, we get the following SUE expression:

$$p_{ij}^{(2)} = \sum_{k \neq i,j} p_{ik} \cdot p_{kj} + p_{ij} \cdot (p_{ii} + p_{jj}). \tag{40}$$

Peeling produces the exact range but requires lots of computations. To compute the exact range of $p_{ij}^{(2)}$, we must find the maximum and the minimum of the corresponding expressions (39) and (40) under the conditions (1) and

$$\underline{p}_{ij} \le p_{ij} \le \overline{p}_{ij}. \tag{41}$$

In other words, we want to optimize a quadratic function under linear constraints (equalities and inequalities). In principle, to optimize such a function, we can use the peeling algorithm described in the main text.

The only problem with this algorithm – as mentioned in [6] – is that since we have $\approx 2^n$ boundary elements, this algorithm requires exponential ($\approx 2^n$) time.

So, if we want to compute the range in polynomial time, we must design a new algorithm. Our new algorithm will actually use peeling – but not peeling applied to the original problem, but peeling applied to reduced problems (with fewer variables).

Reduction to a fewer-dimensional problem: Step 1. Let us describe how this reduction can be done.

We will start with the case when $i = j$ and we are looking for the maximum of the quadratic expression (39). In this case, we want to solve the following problem:

$$\sum_{k \ne i} p_{ik} \cdot p_{ki} + p_{ii}^2 \to \max \tag{42}$$

under the conditions that $p_{ab} \in \mathbf{p}_{ab}$ for all a and b and that

$$\sum_{k \ne i} p_{ik} + p_{ii} = 1. \tag{43}$$

In (42), the coefficients at p_{ki} are non-negative; therefore, the maximum is attained when each of the terms p_{ki} attains the largest possible value $\overline{p}_{ki}$. In other words, the solution to the problem (42) is also a solution to the following problem with fewer unknowns:

$$\sum_{k \ne i} p_{ik} \cdot \overline{p}_{ki} + p_{ii}^2 \to \max \tag{44}$$

under the conditions (43) and

$$\underline{p}_{ik} \le p_{ik} \le \overline{p}_{ik}. \tag{45}$$

Reduction to a fewer-dimensional problem: Step 2. Let p_{ii} be the value for which the maximum is attained. Then, if we fix the value p_{ii}, we get the following problem with one fewer variable:

$$\sum_{k \ne i} p_{ik} \cdot \overline{p}_{ki} \to \max \tag{46}$$

under the conditions (45) and

$$\sum_{k \neq i} p_{ik} = 1 - p_{ii}. \tag{47}$$

Reduction to a fewer-dimensional problem: Step 3. To perform a further reduction, let us sort that the coefficients $\overline{p}_{ki}$ $(k \neq i)$ in decreasing order, i.e., in such a way that

$$\overline{p}_{(1)i} \geq \overline{p}_{(2)i} \geq \ldots \geq \overline{p}_{(n-1)i}. \tag{48}$$

The sums in (46) and (47) do not depend on the order in which we add the terms. Thus, the above optimization problem can be reformulated as follows:

$$\sum_{k} p_{i(k)} \cdot \overline{p}_{(k)i} \to \max \tag{49}$$

(where $p_{i(k)}$ denotes the value p_{il} for which $\overline{p}_{li} = \overline{p}_{(k)i}$) under the conditions

$$\underline{p}_{i(k)} \leq p_{i(k)} \leq \overline{p}_{i(k)} \tag{50}$$

and

$$\sum_{k} p_{i(k)} = 1 - p_{ii}. \tag{51}$$

In this case, if, for some $k_1 < k_2$, we have $p_{i(k_1)} < \overline{p}_{i(k_1)}$ and $p_{i(k_2)} > \underline{p}_{i(k_2)}$, then we can subtract a small positive value $\varepsilon > 0$ from $p_{i(k_2)}$ and add this value to $p_{i(k_1)}$, i.e., replace $p_{i(k_1)}$ with $p'_{i(k_1)} = p_{i(k_1)} + \varepsilon$ and $p'_{i(k_2)} = p_{i(k_2)} - \varepsilon$ (we keep all other values $p_{i(k)}$ unchanged). If ε is small enough, we still satisfy the conditions $p_{i(k_1)} \in \mathbf{p}_{i(k_1)}$ and $p_{i(k_2)} \in \mathbf{p}_{i(k_2)}$. Since we added and subtracted the same value, the sum of the resulting probabilities remains the same hence, the condition (51) is still satisfied, so the new values $p'_{i(k)}$ satisfy all the necessary conditions.

If we replace $p_{i(k)}$ by $p'_{i(k)}$, then the value of the optimized function (49) is increased by $\varepsilon \cdot (\overline{p}_{(k_1)i} - \overline{p}_{(k_2)i})$. Since the values $\overline{p}_{(k)i}$ are sorted in decreasing order, and $k_1 < k_2$, we conclude that the increase is non-negative. Thus, if $p_{i(k_1)} < \overline{p}_{i(k_1)}$ and $p_{i(k_2)} > \underline{p}_{i(k_2)}$, we can change the values of $p_{i(k)}$ in such a way that one of these conditions is no longer true, and increase (or at least not decrease) the value of the optimized function.

Hence, a maximum is attained at a vector $(p_{i(1)}, p_{i(2)}, \ldots)$ for which the above condition is never satisfied, i.e., for which:

- once $p_{i(k')} < \overline{p}_{i(k')}$, we have $p_{i(k)} = \underline{p}_{i(k)}$ for all $k > k'$, and
- once $p_{i(k')} > \underline{p}_{i(k)'}$, we have $p_{i(k)} = \overline{p}_{i(k)}$ for all $k < k'$.

Thus, if there is a k' for which $\underline{p}_{i(k')} < p_{i(k')} < \overline{p}_{i(k')}$, we have $p_{i(k)} = \overline{p}_{i(k)}$ for all $k < k'$ and $p_{i(k)} = \underline{p}_{i(k)}$ for all $k > k'$.

If there is no such k', i.e., if for every k, $p_{i(k)} = \underline{p}_{i(k)}$ or $p_{i(k)} = \overline{p}_{i(k)}$, then once $p_{i(k)} = \overline{p}_{i(k)}$ and hence $p_{i(k)} > \underline{p}_{i(k)}$, we have $p_{i(k')} = \overline{p}_{i(k')}$ for all $k' < k$. Similarly, once $p_{i(k)} = \underline{p}_{i(k)}$ and hence $p_{i(k)} < \overline{p}_{i(k)}$, we have $p_{i(k')} = \underline{p}_{i(k')}$ for all $k' > k$.

In all these cases, there is a borderline value k' such that $p_{i(k)} = \overline{p}_{i(k)}$ for all $k < k'$ and $p_{i(k)} = \underline{p}_{i(k)}$ for all $k > k'$. Thus, once we fixed k', the optimal values of all the variables $p_{i(k)}$ are fixed except for one variable: $p_{i(k')}$. Hence, once k' is fixed, the original optimization problem takes the following form:

$$p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ii}^2 \to \max \tag{52}$$

under the conditions that

$$\underline{p}_{i(k')} \le p_{i(k')} \le \overline{p}_{i(k')}, \tag{53}$$

$$\underline{p}_{ii} \le p_{ii} \le \overline{p}_{ii}, \tag{54}$$

and

$$p_{i(k')} + p_{ii} = c_{k'}, \tag{55}$$

where we denoted

$$c_{k'} \stackrel{\text{def}}{=} 1 - \sum_{k:k<k'} \overline{p}_{i(k)} - \sum_{k:k>k'} \underline{p}_{i(k)}. \tag{56}$$

This is a quadratic optimization problem with 2 variables under linear constraints, so, for this problem, the peeling method leads to a solution in $2^2 = \text{const}$ number of computational steps.

Once we compute the optimal values of $p_{i(k')}$ and p_{ii}, we can compute the corresponding value $V_{k'}$ of the original objective function as

$$V_{k'} = \sum_{k:k<k'} \overline{p}_{i(k)} \cdot \overline{p}_{(k)i} + p_{i(k')} \cdot \overline{p}_{(k')i} + \sum_{k:k>k'} \underline{p}_{i(k)} \cdot \overline{p}_{(k)i} + p_{ii}^2. \tag{57}$$

The actual maximum of $p_{ii}^{(2)}$ can then be determined as the largest of the corresponding values $V_{k'}$.

Resulting algorithm for computing the upper bound for $p_{ii}^{(2)}$: first draft. The above analysis leads to the following algorithm for computing, for a given i, the upper endpoint $\overline{p_{ii}^{(2)}}$ of the interval of possible values of $p_{ii}^{(2)}$:

- First, we sort the values $\overline{p}_{ki}$ $(k \neq i)$ in decreasing order: $\overline{p}_{(1)i} \ge \overline{p}_{(2)i} \ge \cdots \ge \overline{p}_{(n-1)i}$.

- Then, for each $k' = 1, \ldots, n-1$, we do the following:
  - we compute the value $c_{k'}$ by using formula (56);
  - we solve the problem (52)–(55) of optimizing a quadratic function of two variables $p_{i(k')}$ and p_{ii} with linear constraints;
  - based on the solution, we compute $V_{k'}$ by using the formula (57).
- Finally, we return the largest of the values $V_1, \ldots, V_{n-1}$ as the solution to the original optimization problem.

What is the computational complexity of this algorithm? Sorting requires $O(n \cdot \log(n))$ steps (see, e.g., [1]). After sorting, for each k', we need:

- $O(n)$ steps to compute the sums in $c_{k'}$,
- then a constant number of steps to solve the optimization problem with 2 unknowns, and
- then, again $O(n)$ steps to compute $O(n)$ steps –

the total of $O(n)$ steps. Since we need $O(n)$ steps of each of $n-1$ values k', we thus need a total of $O(n^2)$ steps. The final computation of the largest of $n-1$ values $V_{k'}$ requires $O(n)$ steps, so the overall computational complexity of the after-sorting part of this algorithm is $O(n^2) + O(n \cdot \log(n)) = O(n^2)$.

This is much larger than $O(n)$ steps that is necessary to compute the value of $p_{ii}^{(2)}$ in the non-interval case, by using the formula (3). It is therefore desirable to decrease the computation time of our algorithm. How can we do that?

Decreasing the computation time of the resulting algorithm. It is indeed possible to reduce the above computation time because we do not really need to compute $c_{k'}$ and $V_{k'}$ "from scratch" every time: for each $k' > 1$, we can compute the values of these variables by modifying the previous values. More specifically, we can compute the auxiliary values

$$W_{k'} \stackrel{\text{def}}{=} \sum_{k:k<k'} \overline{p}_{i(k)} \cdot \overline{p}_{(k)i} + \sum_{k:k>k'} \underline{p}_{i(k)} \cdot \overline{p}_{(k)i}, \tag{58}$$

for which

$$V_{k'} = W_{k'} + p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ii}^2. \tag{59}$$

To be more precise, we do need to compute the original values c_1 and W_1; they will be computed as

$$c_1 = 1 - \sum_{k:k>1} \underline{p}_{i(k)} \tag{60}$$

and

$$W_1 = \sum_{k:k>1} \underline{p}_{i(k)} \cdot \overline{p}_{(k)i}. \tag{61}$$

After that, once we know the values c_{k-1} and W_{k-1}, we can compute the next values of c_k and W_k by using the following easy-to-deduce formulas:

$$c_k = c_{k-1} - \overline{p}_{i(k-1)} + \underline{p}_{i(k)} \tag{62}$$

and

$$W_k = W_{k-1} + \overline{p}_{i(k-1)} \cdot \overline{p}_{(k-1)i} - \underline{p}_{i(k)} \cdot \overline{p}_{(k)i}. \tag{63}$$

After this modification, the after-sorting part of the algorithm requires that for each $k' = 1, \ldots, n-1$, we do the following:

- first, we compute the value $c_{k'}$; for $k' = 1$, we use the formula (60); for $k' > 1$, we use the formula (62);
- we solve the problem (52)–(55) of optimizing a quadratic function of two variables $p_{i(k')}$ and p_{ii} with linear constraints;
- we compute the value $W_{k'}$; for $k' = 1$, we use the formula (61); for $k' > 1$, we use the formula (63);
- based on the solution of the quadratic optimization problem, we compute $V_{k'}$ by using the formula (59).

Here, for $k' = 1$, we need $O(n)$ steps, but for every other k', we only need finitely many steps. Thus, the overall after-sorting complexity of this algorithm is $O(n)$ – exactly the same as in the non-interval case.

Similar algorithm for computing the upper bound for $p_{ij}^{(2)}$ ($j \neq i$). For the case $j \neq i$, similar reductions, when applied to the formula (40), lead to the conclusion that the desired upper endpoints is a solution to the following simplified optimization problem:

$$\sum_{k \neq i,j} p_{ik} \cdot \overline{p}_{kj} + p_{ij} \cdot (p_{ii} + \overline{p}_{jj}) \to \max \tag{64}$$

under the conditions that $p_{ab} \in \mathbf{p}_{ab}$ for all a and b and that

$$\sum_{k \neq i,j} p_{ik} + p_{ii} + p_{ij} = 1. \tag{65}$$

After sorting the values $\overline{p}_{kj}$ ($k \neq i, j$) in decreasing order, we can prove that there exists a borderline value k' for which $p_{i(k)} = \overline{p}_{i(k)}$ for all $k < k'$, $p_{i(k)} = \underline{p}_{i(k)}$ for all $k > k'$, and the values $p_{i(k')}$, p_{ii}, and p_{ij} can be obtained by solving the following quadratic optimization problem with linear constraints:

$$p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ij} \cdot (p_{ii} + \overline{p}_{jj}) \to \max \tag{66}$$

under the conditions that

$$\underline{p}_{i(k')} \le p_{i(k')} \le \overline{p}_{i(k')}, \tag{67}$$

$$\underline{p}_{ii} \le p_{ii} \le \overline{p}_{ii}, \tag{68}$$

$$\underline{p}_{ij} \le p_{ij} \le \overline{p}_{ij}, \tag{69}$$

and

$$p_{i(k')} + p_{ii} + p_{ij} = c_{k'}, \tag{70}$$

where we denoted

$$c_{k'} \stackrel{\text{def}}{=} 1 - \sum_{k:k<k'} \overline{p}_{i(k)} - \sum_{k:k>k'} \underline{p}_{i(k)}. \tag{71}$$

This is a quadratic optimization problem with 3 variables under linear constraints, so, for this problem, the peeling method leads to a solution in $2^3 = \text{const}$ number of computational steps.

The above expression for the objective function can be reformulated as

$$V_{k'} = W_{k'} + p_{i(k')} \cdot \overline{p}_{(k')i} + p_{ij} \cdot (p_{ii} + \overline{p}_{jj}). \tag{72}$$

Thus, similarly to the case $i = j$, we can set up the after-sorting part of our algorithm as doing the following for each $k' = 1, \ldots, n-2$:

- first, we compute the value $c_{k'}$; for $k' = 1$, we use the formula (60); for $k' > 1$, we use the formula (62);
- we solve the problem (66)–(70) of optimizing a quadratic function of three variables $p_{i(k')}$, p_{ii}, and p_{ij} with linear constraints;
- we compute the auxiliary value $W_{k'}$; for $k' = 1$, we use the formula (61); for $k' > 1$, we use the formula (63);
- based on the solution of the quadratic optimization problem, we compute $V_{k'}$ by using the formula (72).

Here, for $k' = 1$, we need $O(n)$ steps, but for every other k', we only need finitely many steps. Thus, the overall after-sorting complexity of this algorithm is $O(n)$ – exactly the same as in the non-interval case.

Overall computational complexity. Overall, we need to sort n sequences corresponding to n different values of i. Thus, all the sorting requires

$$n \cdot O(n \cdot \log(n)) = O(n^2 \cdot \log(n)) \ll O(n^3) \tag{73}$$

steps. After sorting, we need $O(n)$ steps to compute each of n^2 upper bounds; therefore, we need $O(n^3)$ after-sorting steps. Overall, the above algorithm requires $O(n^2 \cdot \log(n)) + O(n^3) = O(n^3)$ steps – asymptotically the same number of steps as in the non-interval case.

Similar algorithm for computing the lower bound for $p_{ij}^{(2)}$. To compute the lower endpoint for $p_{ij}^{(2)}$, we must use $\underline{p}_{ki}$ instead of $\overline{p}_{ki}$; therefore, we must sort the values $\underline{p}_{ki}$ instead of $\overline{p}_{ki}$, and we must solve the corresponding minimization problems instead of the maximization ones.

As a result, we arrive at the $O(n^3)$ algorithms for computing 2-step transition probabilities for interval Markov chains that are described in the main text.