On Stationary Distributions for Auto-ATC

Zouhair Mahboubi

March 19, 2015

1 Introduction

The goal of this note is to outline how to compute the stationary distribution for a continuous-time MDP.

We will discuss stationary distributions for a Markov chain and how they relate to a CTBN, then give a methodology to compute them for CTMDPs.

1.1 Definitions and Notation

We use similar notation as used in the note on Kronecker sums.

1.2 Markov Chain

A Markov chain described by a transition matrix P has a (steady-state) stationary distribution $\theta^\intercal$ which satisfies:

$$\theta^\intercal P = 0 \tag{1}$$

There are different ways to find $\pi^\intercal$, but the most straight forward is the power iteration method where we start with any valid distribution θ_0:

$$\theta^\intercal = \lim_{k\to\infty} \theta_0^\intercal P^k$$

1.3 CTBN

A CTBN described by a rate matrix Q has a (steady-state) stationary distribution $\alpha^\intercal$ [1] which satisfies:

$$\alpha^\intercal Q = 0 \tag{2}$$

An interesting fact is that If Q is factored into its underlying Markov chain matrix P and sojourn matrix M using $Q = M(P - I)$, the stationary distributions for Q and P are related as follow:

[1] conditions of existence are beyond the scope of this note

$$\alpha = M^{-1}\theta$$

This can be used to verify the methodology we present later for simple problems where P is readily available. The proof is as follow (Note that M is diagonal and therefore symmetric)

$$\begin{aligned}
\alpha^\intercal Q &= \theta^\intercal M^{-\intercal} M(P - I) \\
&= \theta^\intercal (P - I) \\
&= \theta^\intercal - \theta^\intercal \\
&= 0
\end{aligned}$$

Unfortunately, power iteration does not work to find the null vector. Luckily, Dianne [2] describes a neat trick for finding the null-space vector. Although the method is for a Markov chain, in the process it is (unknowingly) transformed into a CTBN. The idea is that to solve the rank deficient linear system described in 2, the problem is transformed by replacing any column j of Q by a random non-zero vector w to obtain $\tilde{Q}$, and enforcing $\alpha^\intercal w = 1$. This yields a linear system of the form $\alpha^T \tilde{Q} = e_j^\intercal$. This linear system can then be solved using iterative methods such as the Jacobi method:

$$\alpha_{k+1}^T \leftarrow (e_j^\intercal - \alpha_k^T (L + U))D^{-1}$$

where D, L, U are the diagonal, lower triangle, and upper triangle portions of $\tilde{Q}$. Note that this yields a null vector up to a constant. If we choose $w = \vec{1}$, the result should be a valid probability distribution since $\alpha^\intercal w = \sum \alpha_i = 1$ (assuming α indeed exists, we expect that $\alpha_i \geq 0$ will follow).

As for guarantees of convergence, there are none... Seriously though, according to the Wiki entry on Jacobi iteration, convergence is guaranteed if the matrix is irreducibly diagonally dominant. The original Q is weakly diagonally dominant by construction. However, this is not necessarily the case for $\tilde{Q}$, so we might have to be more careful about how we choose w.

1.4 CTMDP

A CTMDP is described by a set of rate matrices Q_a, one for each action $a \in \mathcal{A}$. Finding the stationary distribution α_π under a policy $a = \pi(s)$ requires computing the left null vector of the matrix Q_π, whose ith row is given by the ith row of $Q_{\pi(i)}$[3]. Once Q_π is obtained, we can use the Jacobi method described previously to solve for α_π.

[2] O'leary, Dianne P. Iterative methods for finding the stationary vector for Markov chains. Springer New York, 1993

[3] Overloading π to operate on the state numbering of s

Although the problem is well defined, the difficulty lies in the fact that we need to compute Q_π in order to carry-out the Jacobi iterations. For large problems, this is of course not feasible.

Previously, we were able to get around this problem for value iteration by only computing the relevant row of Q. This was possible because each iteration of Gauss-Seidel computes a new component of the value function and olnly involves one row of Q_π at a time. In contrast, computing just one component of α_{k+1}requires a column from Q_π, which involves knowing all of the actions of the policy. Unfortunately, the methodology we described previously for value iteration only works for efficiently computing the rows.

But we can overcome this problem by computing the vector-Matrix product involved in the Jacobi iteration in batch mode. The algorithm is as follow

Algorithm 1 Jacobi Iteration for CTMDP

```
Require: Policy $\pi$, rate matrices $\{Q_a\}$
 1: $n \leftarrow$ state-space size
 2: $\alpha_0 \leftarrow [\frac{1}{n}]_n$
 3: $w \leftarrow [1]_n$
 4: $d \leftarrow [0]_n$
 5: while not converged do
 6:     $\alpha_1 \leftarrow [0]_n$
 7:     $\alpha_1[j] \leftarrow 1$
 8:     for $i \in 1 : n$ do
 9:         $q \leftarrow Q_{\pi(i)}[i, :]$
10:         $q[j] \leftarrow w[j]$
11:         $d[i] \leftarrow 1/q[i]$
12:         $q[i] \leftarrow 0$
13:         $\alpha_1 \leftarrow \alpha_1 - \alpha_0[i]q$
14:     end for
15:     $\alpha_0 \leftarrow \alpha_1 \odot d$
16: end while
17: return $\frac{\alpha_0}{||\alpha_0||}$
```

A few comments:

- Line 11 only needs to be done on the first outer loop iteration
- Unlike Gauss-Seidel, this requires double-storage since we can't do this in place.
- The choice of the w and j is arbitrary and is only there in order to modify the Jacobi method to work for finding a null vector. Although we might have to be more judicious in our choice to guarantee convergence.

Otherwise, as long as we are able to compute q, the ith row of $Q_{\pi(i)}$, this should be computationally tractable.

The one hickup is that for factored CTMDPs, we cannot swap the order of the actions (unlike in Value Iteration). This creates a slight complexity since our methodology for finding rows of Kronecker sums assumed a certain structure. Luckily, it turns out that the result we had can be generalized as follow:

$$A_1 \oplus A_2 \ldots \oplus A_K = \sum_{u=1}^{K} P_{(n^u, n^{K-u})} \left(I_{n^{K-1}} \otimes A_i \right) P^{\intercal}_{(n^u, n^{K-u})} \tag{3}$$

Computing a given row of the above series of Kronecker sum turns out to only require a small change to the algorithm that we used previously.