Desarrollo de un módulo para modelar y resolver procesos de cuasi nacimiento y muerte como adición a la librería JMarkov

Trabajo de Tesis
presentado al
Departamento de Ingeniería Industrial

por

Julio César Góez

Asesor: Germán Riaño

Para optar al título de
Maestría en Ingeniería

Departamento de Ingeniería Industrial
Universidad de Los Andes
Julio 2004

Desarrollo de un módulo para modelar y resolver procesos de cuasi nacimiento y muerte como adición a la librería JMarkov

Aprobado por:

_____

Germán Riaño, Asesor

_____

Gonzalo Mejía

_____

Alvaro Torres

Fecha de Aprobación _____

A mi hermana Elvia Inés Góez, por su apoyo incondicional desde el inicio de mi maestría.

Reconocimientos

Este proyecto no habría sido posible sin el apoyo y la paciencia del profesor Germán Riaño Mendoza, Ph.D., como mi asesor de tesis de maestría, su aporte y conocimiento fueron definitivos en el proceso. Igualmente es necesario expresar mis agradecimientos a los profesores Roberto Zarama Urdaneta, Ph.D., y Andrés Medaglia, Ph.D., por el apoyo recibido en el comienzo de mis estudios de Maestría. A los profesores Gonzalo Mejía, Ph.D., y Alvaro Torres, Ph.D., agradezco el tiempo que cedieron para la lectura y evaluación de este documento. A mis padres Roger Góez y Magdalena Gutiérrez y mi hermano Roger Góez por acompañarme siempre en este proyecto. Por último quiero agradecer a mis amigos David Rojas, Natalia Santamaria, Elkin Castro, Rolando Avendaño, Juan Pablo Olarte, Javier Lozano, Gilma Bello, Catalina Colmenares y Carlos Valencia porque me acogieron como un compañero más y de quienes nunca deje de recibir un consejo o un aporte para mejorar este trabajo.

Índice

Índice de cuadros

Índice de figuras

Lista de Algoritmos

Capítulo I

Introducción

Es común encontrar la utilización de modelos de procesos de Markov como una herramienta que apoya el diseño y la toma de decisiones en diferentes áreas de investigación y desarrollo industrial, los ejemplos más conocidos, tal vez, pueden ser el área de modelos de control de producción, que en principio es el incentivo principal de este trabajo, el área de redes de telecomunicaciones y de redes de computadores.

Observando la transcendencia que han tenido los procesos de Markov en diferentes áreas de investigación se puede entender entonces el interés que inspira el trabajo con los procesos de cuasi Nacimiento y Muerte como un caso particular de procesos de Markov. Estos procesos son de gran interés dado que su estructura matricial con un comportamiento repetido ha permitido el desarrollo de algoritmos de solución eficientes para procesos infinitos, además su aplicación es extensa en variadas situaciones en las que la utilización de este tipo de modelos aporta información importante para el diseño y desarrollo de sistemas prácticos.

Un problema pequeño con un conjunto reducido de estados puede ser resuelto fácilmente utilizando recursos de computación simples y de uso común como una hoja de cálculo, sin embargo los problemas que realmente pueden inspirar un interés concreto para el diseño de un sistema o el desarrollo de un modelo social, por

ejemplo, desbordan con facilidad la capacidad de estos programas hecho que exige el uso de lenguajes de programación como C, FORTRAN, JAVA, entre otros, para el desarrollo de rutinas que utilicen los algoritmos que hasta la fecha se han desarrollado para el tratamiento de este tipo de situaciones. En particular para el caso de los procesos de cuasi-nacimietno y muerte, los desarrollos que se han realizado en el orden de herramientas computacionales han estado enmarcados en un ambiente académico, hecho que ha implicado que su diseño este orientado a proporcionar programas con la implementación puntual de algoritmos desarrollados para el calculo de las soluciones de la cadena y que pueden facilitar el trabajo con este tipo de procesos. Sin embargo a la fecha no se encuentra en la oferta un desarrollo que proporcione al usuario una utilidad integrada al sistema que le facilite, previamente al desarrollo de los cálculos, el proceso de generación de modelo de acuerdo con las reglas de comportamiento que ha observado en el sistema, es decir que facilite la conexión del modelo abstracto concebido por el usuario con el modelo matricial necesario para la implementación de los cálculos.

En las consultas realizadas sobre los programas desarrollados se encontró que estos entregan rutinas programadas para que el usuario haga uso de ellas, de esta manera el programador tiene que haber desarrollado un programa de generación de modelos de Markov exigiéndole el uso de estructuras de memoria complejas y el desarrollo de rutinas para la construcción de matrices que le permitan entregar la matriz generadora o la matriz de probabilidades como parámetros de entrada al programa. Esto demanda un conocimiento en programación que transciende lo básico en el manejo de algoritmos y de un lenguaje, de esta manera el acceso a estos desarrollos queda limitado. En el desarrollo presentado en este trabajo se presenta un concepto que integra una utilidad de generación del modelo, la cual es tomada de

JMarkov, con rutinas que realizan los cálculos de probabilidades de estado estable y medidas de desempeño del proceso. El logro de este proyecto es el desarrollo de una herramienta que permite al usuario, a partir del desarrollo abstracto de su modelo, la utilización de una paquete que le facilita programar las reglas de comportamiento del sistema, con lo cual el programa genera el modelo, lo representa en forma matricial y realiza los cálculos de probabilidades de estado estable y de medidas de desempeño del sistema. Esto se logró mediante la integración de los algoritmos de solución de procesos de cuasi-nacimiento a la metodología de trabajo que ha sido desarrollada en JMarkov [13] por German Riaño. En este ambiente el usuario solo deberá conocer conceptos básicos de programación orientada a objetos, particularmente en JAVA, para el desarrollo de su modelo y a partir de rutinas simples programadas por él esté en capacidad de obtener los cálculos necesarios para el estudio de su modelo, tomando en consideración que la dimensión de los modelos que puede manejar utilizando este módulo superan razonablemente las dimensiones que se podrían manejar con una herramienta amigable como una hoja de cálculo. Es de anotar que el usuario no necesita programar rutinas complicadas de manejo de matrices ni preocuparse por el manejo completo de estructuras de datos para el almacenamiento de los estados y sus características.

1.1. Objetivos

El objetivo general de este trabajo es el desarrollo de una herramienta computacional para modelar y resolver procesos de Markov de cuasi nacimiento y muerte. Partiendo de este objetivo se considero el desarrollo de los siguiente aspectos puntuales:

- Buscar la adición de un módulo a la librería JMarkov [13] diseñada por Germán

Riaño, Ph.D., que permitiera el trabajo con este tipo de modelos y que tomara ventaja de la metodología propuesta en ella.

- El desarrollo de una herramienta de apoyo para proyectos de investigación sobre el comportamiento de sistemas susceptibles de ser modelados como procesos de Markov, considerando que el trabajo con modelos de situaciones reales exigen la utilización intensiva de recursos computacionales.

- Aprovechar el gran avance que en la actualidad se ha alcanzado en la capacidad de hardware para la utilización de algoritmos eficientes en los cálculos de la distribución de estado estable.

1.2. Estado del arte

Partiendo del hecho que nuestro interés esta centrado en el desarrollo de una herramienta computacional para el trabajo con procesos de Cuasi Nacimiento y Muerte, es lógico preguntarse cuales han sido los desarrollos que hasta el día de hoy se han realizado en este campo. De esta manera se hizo una búsqueda en la red y en la literatura tratando de encontrar desarrollos similares que pudieran servir como punto de partida y como resultado se obtuvo la referencia de los siguientes programas:

- MGMTool [1]: Este un programa con una gran cantidad de herramientas para resolver problemas de colas. Es fuerte en el trabajo con distribuciones tipo Fase. Trabaja con modelos con estructuras Geométrico matriciales. El algoritmo que utiliza para el calculo de la matriz $\mathbf{R}$ de Neuts [11] es el de iteraciones repetidas [1] ya que al momento de su desarrollado no se conocía el algoritmo de Latouche

[1]Estos algoritmos se explica en el capítulo 3.

y Ramaswami.

- Library for solving QBD and QBD-M processes and MM sum(CPP, K)/GE/c/L G-queue solver tool [12]: Es una librería que permite resolver procesos de QBD utilizando básicamente algoritmos de sustitución sucesiva, aunque en la información de desarrollo actual plantean estar implementado nuevos algoritmos de solución. No tiene herramientas para generación de la cadena.

- XTelpack [5]: Este e un programa que resuelve gran cantidad de modelos de cadenas de Markov de tiempo discreto o continuo, resuelve procesos de Cuasi Nacimiento y Muerte y utiliza diferentes algoritmos para calcular la matriz **R** de Neuts, sin embargo no tiene rutinas de generación de modelo, los datos de entrada son matrices.

Después de conocer estos trabajos, cabe anotar que no fue posible hallar otros que trabajaran con Procesos de Cuasi Nacimiento y Muerte.

Capítulo II

Procesos de Cuasi Nacimiento Y Muerte Homogeneos

El estudio de los procesos de Cuasi Nacimiento y Muerte se inicio en el artículo de R.V. Evans [6] y en la Tesis Doctoral de V. Wallace [17] quien fue el primero en dar el nombre de Cuasi Nacimiento y muerte (Quasi Birth and Death, **QBD**) a estos procesos[1] . Es razonable antes de entrar en la explicación de la implementación de los algoritmos utilizados para el tratamiento de los Procesos de **QBD** realizar una breve introducción a la teoría. En este capitulo se resumirán brevemente estos procesos tratando de ilustrar al lector al respecto de las características que reúnen estos para poder calcular su distribución estacionaria de probabilidad utilizando métodos analítico matriciales. En este capitulo no se pretende tratar a fondo el análisis de este tipo de cadenas, antes que eso es un resumen corto de la teoría estudiada en los libros de Latouche y Ramaswami [9], Nelson [10] y Stewart [14], cualquier intención de estudiar a fondo estos proceso aconsejamos al lector remitirse a estas referencias.

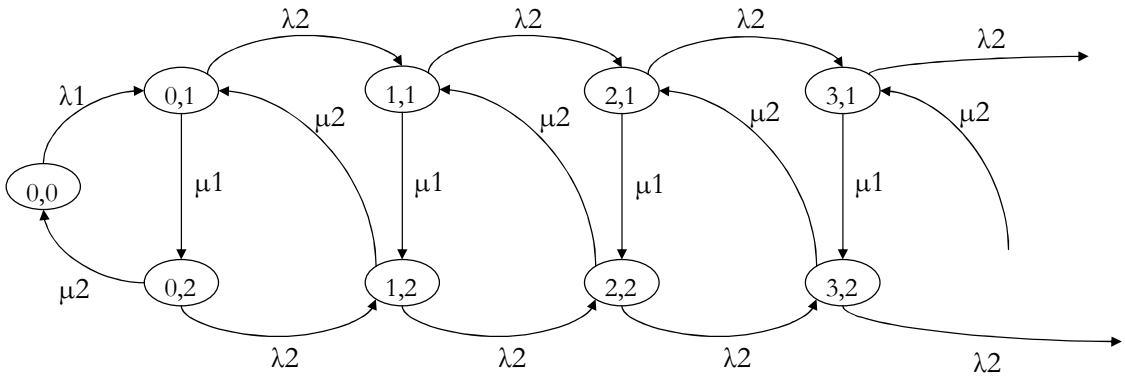

Figura 1: Cola $M/M/1$ Modificada

2.1. Definición

En la Figura 1 se puede observar una cola $M/M/1$ modificada, con llegadas Poisson con tasa λ_1 cuando el sistema esta vacío y tasa λ_2 cuando hay por lo menos un cliente en el sistema, los clientes requieren un servicio de dos fases exponenciales con tasas μ_1 para la primera fase y μ_2 para la segunda fase. En esta figura se puede observar una estructura que se repite a partir de los estados en los que el sistema tiene un cliente en cola; también puede verse que en la medida en la que el cliente que esta siendo servido progresa en al las fases de servicio la primera componente de los estado que representan este avance permanece invariante. Intuitivamente puede afirmarse que aquellos estados en los que el número de clientes n en cola permanece invariante mientras avance el servicio i del cliente que es servido en un instante determinado definen un nivel del proceso, es el caso de los estados $(1,1)$ y $(1,2)$; de igual manera se puede notar que las fases del servicio del cliente servido definen la fase del estado. Aquellos estados que se encuentran antes de empezar a darse la estructura repetida del sistema son estados de frontera dado que definen el comienzo

[1]En adelante se usará dentro de este texto la abreviatura **QBD** para referirse estos procesos.

del proceso como puede observarse en la Figura 1, en este caso estos estados son $(0,0)$, $(0,1)$, $(0,2)$. Ahora para explicar los procesos de **QBD** se debe comenzar por considerar una cadena de Markov de tiempo continuo $\{X(t) : t \geq 0\}$, similar a la de la Figura 1, en donde el espacio de estados es $S = \{(n,i) : n \geq 0, 0 \leq i \leq m\}$, tal que este espacio de estados S puede ser dividido en una serie de subconjuntos $\ell(n)$ de manera que se cumpla que $\bigcup_{n\geq 0} \ell(n) = S$. En los procesos que pueden ser descritos de este modo la primera componente n se conoce como el nivel de la cadena y la segunda componente i se conoce como la fase del estado. En cada nivel en el que se puede dividir la cadena el número m se refiere a la cardinalidad del conjunto de estados que contiene que puede ser finito o infinito. Para definir este clase de procesos como **QBD** el comportamiento de estos debe ser tal que para pasar del estado (i,n) al estado (i',n') se debe cumplir una de las siguientes condiciones: $n' = n$, $n' = n+1$ ó $n' = n-1$, es decir, en la cadena solo se pueden dar transiciones entre estados del mismo nivel, de un nivel superior a uno inmediatamente inferior o de un nivel inferior a uno inmediatamente superior, de esta manera un proceso de nacimiento y muerte clásico puede ser definido como un caso particular de un proceso **QBD**. Esta característica lleva a que la matriz generadora de proceso $\mathbf{Q}$ tenga una estructura tridiagonal de la forma

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B}_0 & \mathbf{A}_0 & & & \\ \mathbf{B}_1 & \mathbf{A}_1 & \mathbf{A}_0 & & \\ & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \\ & & \ddots & \ddots & \ddots \end{pmatrix}$$

donde las componentes de las filas de la matriz $\mathbf{Q}$ deben sumar a cero, por tal razón se debe cumplir que $(\mathbf{B}_0 + \mathbf{A}_0)e = (\mathbf{B}_1 + \mathbf{A}_1 + \mathbf{A}_0)e = (\mathbf{A}_2 + \mathbf{A}_1 + \mathbf{A}_0)\mathbf{e} = 0$ donde

e es un vector columna dimensionalmente apropiado. Las matrices $\mathbf{B}_1$, $\mathbf{A}_0$ y $\mathbf{A}_2$ son matrices no negativas y las matrices $\mathbf{B}_0$ y $\mathbf{A}_1$ tienen las componentes por fuera de diagonal no negativas y las componentes de la diagonal estrictamente negativas, esto se puede explicar ya que la diagonal de las matrices $\mathbf{B}_0$ y $\mathbf{A}_1$ componen la diagonal de la matriz $\mathbf{Q}$. El comportamiento de los estados de frontera no necesariamente es estrictamente idéntico al descrito en la matriz $\mathbf{Q}$, sin embargo las transformaciones necesarias para su manejo son simples y no modifican en absoluto las propiedades del proceso. En el contexto de esta exposición se trabajara con el supuesto de que el proceso $\mathbf{Q}$ es irreducible. En adelante los procesos referidos en este texto como **QBD** atenderán a la anterior definición.

2.2. Propiedad matriz-geométrica

Primero se debe partir del supuesto de que el proceso **QBD** es irreducible, esto implica que el vector de distribución estacionaria invariante $\boldsymbol{\pi}$ existe y que es la única solución al sistema $\boldsymbol{\pi}\mathbf{Q} = 0$, $\boldsymbol{\pi}\mathbf{1} = 1$. Si el vector $\boldsymbol{\pi}$ es particionado en vectores pequeños $\boldsymbol{\pi}_n$, $n \geq 0$ de acuerdo con los niveles del proceso, donde $\boldsymbol{\pi}_n$ tiene m componentes, entonces el sistema anteriormente planteado se puede descomponer de la siguiente manera

$$\begin{aligned}
&\boldsymbol{\pi}_0\mathbf{B}_0 = -\boldsymbol{\pi}_1\mathbf{B}_1 \\
&\boldsymbol{\pi}_n\mathbf{A}_1 = -\boldsymbol{\pi}_{n-1}\mathbf{A}_0 - \boldsymbol{\pi}_{n+1}\mathbf{A}_2 \qquad \text{para } n \geq 1 \\
&\sum_{n\geq 0} \boldsymbol{\pi}_n\mathbf{e} = 1,
\end{aligned}$$

donde este sistema es muy similar al que se plantea en el análisis de procesos de Nacimiento y Muerte continuos. Se puede observar que el valor de $\boldsymbol{\pi}_n$ queda en función de las tasas de transición entre los estados con $j+1$ y $j-1$ clientes y los

estados con j clientes en el sistema, como estas tasa de transición no dependen del nivel j es posible intuir que existe una matriz constante R tal que cumple el Teorema 2.2.1 [9].

Teorema 2.2.1 *Suponga que el procesos QBD de tiempo continuo es positivo recurrente, entonces la distribución de probabilidad estacionaria es tal que*

$$\boldsymbol{\pi}_n = \boldsymbol{\pi}_0 \mathbf{R}^n \qquad \textit{para } n \geq 0,$$

donde la matriz $\mathbf{R}$ *es tal que para cualquier* $n \geq 0$, $R_{ij} \quad (1 \leq i, j \leq m)$ *representa el valor esperado de visitas al estado* $(n+1, j)$ *antes de su primer retorno a los estados* $\ell(0) \bigcup \ldots \bigcup \ell(n)$, *dado que empezó en el estado* (n, i).

En el Teorema 2.2.1 se puede leer una interpretación de la matriz $\mathbf{R}$, adicionalmente las sucesivas potencias tienen una interpretación de manera tal que para cualquier $n \geq 0$, $k \geq 1$, $1 \leq i, j \leq m$ (R_{ij}^k) es el valor esperado del número de visitas al estado $(n + k, j)$ antes de su primer retorno a los estados $\ell(0) \bigcup \ldots \bigcup \ell(n)$, dado que empezó en el estado (n, i).

Con la intención de facilitar la evaluación numérica que se expondrá en el Capitulo 3 de la distribución estacionaria de probabilidad $\boldsymbol{\pi}$, se presentará la caracterización propuesta por Latouche y Ramaswami [9] en la que se introducen las Matrices $\mathbf{U}$ y $\mathbf{G}$. Como aclaran en su exposición, estas matrices no son solo un artificio matemático, estas matrices están relacionadas de manera fundamental con la dinámica del proceso.

Primero es necesario partir de la definición de $\mathbf{G}$ de la siguiente manera

$$\mathbf{G}_{ij} = P[\tau < \infty \ \ \& \ \ X(\tau) = (n-1, j) | X(0) = (n, i)],$$

donde τ es el tiempo de primera visita desde el nivel n al nivel $n-1$.

La matriz $\mathbf{U}$ se define por

$$\mathbf{U} = \mathbf{A_1} + \mathbf{A}_0\mathbf{G},$$

esta matriz se interpreta de la siguiente manera: empezando de de un estado $X(0) \in \ell(n)$, considere el proceso de Markov restringido a $\ell(n)$ antes de la primera visita a $\ell(n-1)$; este es un proceso transiente, y la matriz $\mathbf{U}$ es la matriz infinitesimal generadora de proceso. Dada la homogeneidad del proceso los valores de $\mathbf{G}$ y $\mathbf{U}$ no dependen de $n \geq 1$.

Después de tener las definiciones de las matrices $\mathbf{G}$ y $\mathbf{U}$ el Teorema 2.2.2[9] muestra como se relacionan estas con $\mathbf{R}$.

Teorema 2.2.2 *Las matrices* $\mathbf{R}$*,* $\mathbf{U}$ *y* $\mathbf{G}$ *satisfacen las siguientes ecuaciones:*

$$\begin{aligned}
\mathbf{U} &= \mathbf{A}_1 + \mathbf{A}_0\mathbf{G} && (2.2.1)\\
&= \mathbf{A}_1 + \mathbf{R}\mathbf{A}_2 \\
&= \mathbf{A}_1 + \mathbf{A}_0(-\mathbf{U})^{-1}\mathbf{A}_2, \\
\mathbf{R} &= \mathbf{A}_0(-\mathbf{U})^{-1}, && (2.2.2)\\
\mathbf{G} &= (-\mathbf{U})^{-1}\mathbf{A}_2, && (2.2.3)
\end{aligned}$$

y además

$$\begin{aligned}
\mathbf{A}_2 + \mathbf{A}_1\mathbf{G} + \mathbf{A}_0\mathbf{G}^2 &= 0, \\
\mathbf{A}_2 + \mathbf{R}\mathbf{A}_1 + \mathbf{R}^2\mathbf{A}_0 &= 0.
\end{aligned}$$

La distribución de frontera $\boldsymbol{\pi}_0$ esta determinada de acuerdo con el Lemma 2.2.3.

Lemma 2.2.3 *El vector* $\boldsymbol{\pi}_0$ *de distribución estacionaria para* $\ell(0)$ *es una solución única para el sistema*

$$\boldsymbol{\pi_0}(\mathbf{B}_0 + \mathbf{A}_0\mathbf{G}) = 0$$

$$\boldsymbol{\pi_0}(\mathbf{I} - \mathbf{R})^{-1} = 1.$$

Es importante anotar que del Teorema 2.2.2 se puede observar que es suficiente conocer una de las tres matrices para calcular cualquiera de las otras dos, esta es una característica definitiva en el algoritmo de calculo de $\mathbf{R}$ que se explica en el Capitulo 3.

2.3. Estabilidad

En la sección 2.1 se menciono el supuesto de que el proceso representado por la matriz$\mathbf{Q}$ es irreducible y recurrente. La idea de presentar el concepto de estabilidad del sistema es evaluar la validez de este supuesto en la cadena que se pretende modelar. Existe una característica sencilla que es necesaria y suficiente en la evaluación de la estabilidad del sistema cuando se tiene un conjunto de cardinalidad finita m en cada nivel. Esta condición toma ventaja de una propiedad fácil de verificar que poseen las matrices $\mathbf{A}_0$, $\mathbf{A}_1$, y $\mathbf{A}_2$ y los estados de la cadena; esta característica se refiere a que el sistema debe tener una tendencia mayor a moverse en dirección al nivel cero para que sea estable. Esta condición se puede entender de manera análoga a la condición de estabilidad en una cola $M/M/1$ donde λ es la tasa de llegada al sistema y μ es la tasa de servicio, de esta manera para que el sistema sea estable se debe cumplir que $\lambda < \mu$.

Para entender la manera en la que esta condición es evaluada en un proceso en forma matriz-geométrica, considere un nivel $n \gg 0$, es decir un nivel lo suficientemente lejano de las condiciones de frontera. Si se observa una transición del nivel n

al nivel $n-1$, se necesita conocer cual es la tasa del nivel n que se debe considerar para la transición. De esta manera si se conoce que la transición empieza en la fase j, $1 \leq j \leq m$, entonces se tiene la tendencia (drift) hacia los niveles inferiores del proceso estará dada por

$$\sum_{i=1}^{m} \mathbf{A}_2(j,i). \tag{2.3.4}$$

En la ecuación (2.3.4) $\mathbf{A}_2(j,i)$ es la tasa de transición del estado (n,j) al estado $(n-1,i)$. Para calcular el valor esperado de la tasa de transición del nivel n al nivel $n-1$ se necesita las probabilidades α_i de que el proceso en el nivel $n \gg 0$ de la sección de comportamiento típico del proceso este en la fase $1 \leq i \leq m$. De esta manera el valor esperado de la tasa de transición del nivel n al nivel $n-1$ esta dada por

$$\sum_{j=1}^{m} \alpha_j \sum_{i=1}^{m} \mathbf{A}_2(j,i), \tag{2.3.5}$$

de manera matricial la ecuación (2.3.5) se puede escribir

$$\boldsymbol{\alpha}\mathbf{A}_2\mathbf{e}.$$

Con un razonamiento similar se puede tener que el valor esperado de la tasa de transición del nivel $n \gg 0$ al nivel $n+1$ esta dada por

$$\boldsymbol{\alpha}\mathbf{A}_0\mathbf{e}.$$

Lo siguiente que queda por calcular es el valor de las probabilidades $\boldsymbol{\alpha}$. Para calcular estos valores considere un proceso de Markov derivado del proceso original con espacio de estados $1, 2, \ldots, m$ el cual solo identifica las transiciones en términos de las fases de los estados. Particularmente, mientras en el proceso original existe una transición del estado (n,i) al estado $(n+1,j)$ el proceso derivado solo considera la transición de la fase i a la fase j. La Matriz generadora de este proceso se puede

definir como $\mathbf{A} = \mathbf{A}_0 + \mathbf{A}_1 + \mathbf{A}_2$. De acuerdo con esto el vector α será una solución única al sistema

$$\boldsymbol{\alpha}\mathbf{A} = 0,$$

$$\boldsymbol{\alpha}\mathbf{e} = 1.$$

De esta manera el sistema será estable si se cumple la condición de estabilidad

$$\boldsymbol{\alpha}\mathbf{A_2}e > \boldsymbol{\alpha}\mathbf{A_0}\mathbf{e}.$$

El Teorema 2.3.1 [9] resume todo el anterior razonamiento.

Teorema 2.3.1 *Considere un proceso* ***QBD*** *continuo e irreducible. Suponga que el número de fases m en un nivel es finito y que la matriz* $\mathbf{A} = \mathbf{A}_0 + \mathbf{A}_1 + \mathbf{A}_2$ *es irreducible. Entonces, el proceso* ***QBD*** *continuo es recurrente positivo si y solo si* $\boldsymbol{\alpha}\mathbf{A_2}\mathbf{e} > \boldsymbol{\alpha}\mathbf{A_0}\mathbf{e}$*, donde* α *es la solución única al sistema* $\boldsymbol{\alpha}\mathbf{A} = \mathbf{0}$*,* $\boldsymbol{\alpha}\mathbf{e} = 1$*. Es recurrente nulo si* $\boldsymbol{\alpha}\mathbf{A_2}\mathbf{e} = \alpha\mathbf{A_0}\mathbf{e}$ *y transiente si* $\boldsymbol{\alpha}\mathbf{A_2}\mathbf{e} < \boldsymbol{\alpha}\mathbf{A_0}\mathbf{e}$*.*

Capítulo III

Calculo de la Matriz R

En el capitulo anterior se observaron las propiedades de los procesos **QBD** y su estructura en la distribución de probabilidad estacionaria. De acuerdo con esto la distribución de probabilidad estacionaria del proceso, cuando existe, es de la forma $\boldsymbol{\pi_n = \pi_0 R^n}$, de esta manera un paso transcendental es el calculo de la matriz $\mathbf{R}$. Del Teorema 2.2.2 se puede observar que el conocimiento de una de las tres matrices $\mathbf{R}$, $\mathbf{U}$ ó $\mathbf{G}$, permite el cálculo de cualquiera de las otras dos fácilmente utilizando las ecuaciones (2.2.1), (2.2.2) y (2.2.3). De esta manera es posible utilizar métodos para calcular una de estas matrices. Es común en la literatura encontrar métodos que definen algoritmos iterativos para realizar el cálculo numérico de la matriz $\mathbf{R}$, es el caso de Nelson [10] que describe el método dado por las ecuaciones

$$\mathbf{R}(0) = \mathbf{0}$$

$$\mathbf{R}(k+1) = -\sum_{\ell=0,\ell\neq 1}^{2} \mathbf{R}^{\ell}(k)\mathbf{A}_{\ell}\mathbf{A}_{1}^{-1}, \quad k = 0, 1, \ldots,$$

donde las iteraciones se repiten hasta el momento en el que las entradas de la matriz $\mathbf{R}(k+1)$ y $\mathbf{R}(k)$ difieren en valor absoluto en una constante ϵ muy pequeña. De igual manera Stewart [14] describe dos metodologías para el calculo de la matriz, que se mencionan a continuación. La primera la denomina **MG-Original** y utiliza el mismo

método iterativo descrito por Nelson. La segunda la denomina **MG-Improved**, esta basada en investigaciones orientadas a aplicar el método de Newton y métodos relacionados al calculo de $\mathbf{R}$, este método se describe por las ecuaciones

$$\mathbf{R}(0) = \mathbf{0}; \quad \mathbf{R}(k+1) = \mathbf{R}(k) + \mathbf{Y}(k), \quad k = 0, 1, \ldots,$$

donde

$$\mathbf{Y}(k) = \left[\sum_{i=0}^{\infty} \mathbf{R}^i(k)\mathbf{A}_i - \mathbf{R}(k)\right] + \mathbf{Z}(k)\mathbf{A}_1 + (\mathbf{R}(k)\mathbf{Z}(k) + \mathbf{Z}(k)\mathbf{R}(k))\mathbf{A}_2,$$

y

$$\mathbf{Z}(k) = -\left[\mathbf{R}(k) - \sum_{i=0}^{\infty} \mathbf{R}^i(k)\mathbf{A}_i\right](\mathbf{A}_1)^{-1}.$$

De otro lado las investigaciones de Latouche y Ramaswami [9] han concluido en la obtención de algoritmos que calculan primero la matriz $\mathbf{G}$ y luego utilizan las ecuaciones (2.2.1), (2.2.2) y (2.2.3) para el cálculo de la matriz $\mathbf{R}$, el calculo de la matriz $\mathbf{G}$ en primera instancia se justifica dada su propiedad de matriz estocástica cuando el proceso **QBD** es recurrente positivo, esto es de gran interés dado que permite tener a priori la condición de que $\mathbf{G}$ debe cumplir la ecuación $\mathbf{Ge} = \mathbf{0}$. Por otro lado, de la matriz $\mathbf{U}$ se sabe que debe ser subestocastica es decir, debe cumplir con la ecuación $\mathbf{Ue} \leq \mathbf{e}$ y de $\mathbf{R}$ solo se conoce que el máximo valor propio debe ser estrictamente menor que 1. De esta manera la propiedad de la matriz G facilita la verificación del convergencia del algoritmo. En su exposición Latouche y Ramaswami proponen dos algoritmos básicos: Algoritmo de Progresión Lineal en las ecuaciones y el algoritmo de reducción logarítmica.

3.1. Algoritmo de Progresión Lineal

En el Algoritmo 1 se puede observar la descripción de este, básicamente este

Algoritmo 1 Algoritmo de Progresión Lineal

$\mathbf{G}_\bullet = (I - \mathbf{A}_1)^{-1}\mathbf{A}_2$;
while $\|\mathbf{1} - \mathbf{G}_\bullet\mathbf{1}\|_\infty \leq \epsilon$ **do**
  $\mathbf{U}_\bullet = \mathbf{A}_1 + \mathbf{A}_0\mathbf{G}_\bullet$;
  $\mathbf{G}_\bullet = -\mathbf{U}_\bullet\mathbf{A}_2$
end while
$\mathbf{R} = \mathbf{A}_0(\mathbf{I} - \mathbf{U}_\bullet)^{-1}$

algoritmo se basa en las ecuaciones

$$\mathbf{U} = \mathbf{A}_1 + \mathbf{A}_2\mathbf{G}$$

$$\mathbf{G} = (-\mathbf{U})^{-1}\mathbf{A}_2$$

que corresponden a las ecuaciones (2.2.1) y (2.2.3). Estas ecuaciones pueden ser resueltas utilizando sustituciones sucesivas, comenzando arbitrariamente con la asignación $G(0) = 0$ se pueden calcular iterativamente las secuencias $\{\mathbf{U}(k), k \geq 1\}$ y $\{\mathbf{G}(k), k \geq 1\}$ de la siguiente manera:

$$\mathbf{G}(0) = 0, \tag{3.1.6}$$

$$\mathbf{U}(k) = \mathbf{A}_1 + \mathbf{A}_0\mathbf{G}(k-1), \tag{3.1.7}$$

$$\mathbf{G}(k) = (-\mathbf{U}(k))^{-1}\mathbf{A}_2 \quad \text{para todo } k \geq 1. \tag{3.1.8}$$

La convergencia de esta serie de sustituciones esta garantizada por el Teorema 3.1.1 [9].

Teorema 3.1.1 *Las secuencias* $\mathbf{U}(k), k \geq 1$ *y* $\mathbf{G}(k), k \geq 1$ *definida por*

$$\mathbf{U}(1) = \mathbf{A}_1,$$

$$\mathbf{G}(k) = (-\mathbf{U}(k))^{-1}\mathbf{A}_2,$$

$$\mathbf{U}(k+1) = \mathbf{A}_1 + \mathbf{A}_0\mathbf{G}(k)$$

para $k \geq 1$ *converge a las matrices* $\mathbf{U}$ *y* $\mathbf{G}$*, respectivamente. La secuencia* $\{G(k, k \geq 1)\}$ *es monótona.*

3.2. Algoritmo de Reducción Logarítmica

Algoritmo 2 Algoritmo de Reducción Logartimica

$\mathbf{H}^{\bullet} = (-\mathbf{A}_1)^{-1}\mathbf{A}_0$;
$\mathbf{L}^{\bullet} = (-\mathbf{A}_1)^{-1}\mathbf{A}_2$;
$\mathbf{G}_{\bullet} = \mathbf{L}^{\bullet}$;
$\mathbf{T} = \mathbf{H}^{\bullet}$;
while $\|\mathbf{1} - \mathbf{G}_{\bullet}\mathbf{1}\|_{\infty} \leq \epsilon$ **do**
  $\mathbf{U}^{\bullet} = \mathbf{H}^{\bullet}\mathbf{L}^{\bullet} + \mathbf{L}^{\bullet}\mathbf{H}^{\bullet}$;
  $\mathbf{M} = (\mathbf{H}^{\bullet})^2$;
  $\mathbf{H}^{\bullet} = (I - U)^{-1}\mathbf{M}$;
  $\mathbf{M} = (\mathbf{L}^{\bullet})^2$;
  $\mathbf{H}^{\bullet} = (I - U)^{-1}\mathbf{M}$;
  $\mathbf{G}_{\bullet} = \mathbf{G}_{\bullet} + \mathbf{T}\mathbf{L}^{\bullet}$;
  $\mathbf{T} = \mathbf{T}\mathbf{H}^{\bullet}$
end while
$\mathbf{U} = \mathbf{A}_1 + \mathbf{A}_0\mathbf{G}$;
$\mathbf{R} = \mathbf{A}_0(\mathbf{I} - \mathbf{U})^{-1}$

Para explicar el algoritmo se debe empezar por anotar que el calculo de la matriz G se puede realizar calculando la distribución de masa de probabilidad de todas las trayectorias que el proceso puede seguir antes de visitar el nivel 0. Si este procedimiento no se puede realizar de manera exacta, se puede escoger un ϵ arbitrariamente pequeño y, definiendo $L(\epsilon) = inf\{k : \|\mathbf{1} - \mathbf{G1}\|_{\infty} < \epsilon\}$, se puede iterar poniendo como límite el número de trayectorias que existen entre los niveles 1 y $L(\epsilon)$. Definido de esta manera, el algoritmo es lineal en L, ya que en cada iteración el número de niveles que la cadena debe visitar aumenta en 1 y el número de iteraciones es $L(\epsilon) - 1$.

En el algoritmo de reducción logarítmica en cada iteración se multiplica por 2 el

número de niveles que el proceso debe visitar. De esta manera el número de iteraciones que el algoritmo debe realizar es aproximadamente $\lg_2 L(\epsilon)$, lo cual significa una reducción logarítmica en el número de iteraciones.

Empezando por la ecuación $\mathbf{A}_2 + \mathbf{A}_1\mathbf{G} + A_0\mathbf{G}^2 = 0$ esta ecuación se puede transformar en

$$\mathbf{G} = (-\mathbf{A}_1)^{-1}\mathbf{A}_2 + (\mathbf{A}_1)^{-1}\mathbf{A}_0\mathbf{G}^2$$

la cual se puede escribir de la forma

$$\mathbf{G} = \mathbf{L} + \mathbf{H}\mathbf{G}^2$$

donde

$$\mathbf{L} = (-\mathbf{A}_1)^{-1}A_2, \quad \mathbf{H} = (-\mathbf{A}_1)^{-1}\mathbf{A}_0.$$

El algoritmo de reducción logarítmica esta basado en la ecuación

$$\mathbf{G} = \sum_{k\geq 0}\left(\prod_{0\leq i\leq k-1}\mathbf{H}(i)\right)\mathbf{L}(k),$$

donde

$$\mathbf{H}(0) = (-\mathbf{A}_1)^{-1}\mathbf{A}_0,$$

$$\mathbf{L}(0) = (-\mathbf{A}_1)^{-1}\mathbf{A}_2,$$

$$\mathbf{H}(k+1) = (\mathbf{I} - \mathbf{U}(k))^{-1}(\mathbf{H}(k))^2,$$

$$\mathbf{L}(k+1) = (\mathbf{I} - \mathbf{U}(k))^{-1}(\mathbf{L}(k))^2,$$

para $k \geq 0$ y con

$$\mathbf{U}^(k) = \mathbf{H}(k)\mathbf{L}(k) + \mathbf{L}(k)\mathbf{H}(k).$$

En la Figura 2 Stewart [14] realiza un paralelo de los cuatro algoritmos para mostrar el desempeño de cada uno de estos para el calculo de la matriz $\mathbf{R}$ de una

Iter	MG-Original	MG-Improved	Prog. Lineal	Red. Logarítmica
1	0.34312839769451	0.34312839769451	0.18168327834409	0.06923287740729
2	0.22574664252443	0.18947301357165	0.10877581067908	0.01452596126591
3	0.16258700550171	0.12703279214776	0.06923287740729	0.00084529885493
4	0.12254126626620	0.08998901062126	0.04565416814524	0.00000312417043
5	0.09495430686098	0.06572451436462	0.03077579745051	0.00000000004292
6	0.07496252810333	0.04896069923748	0.02104375591178	0.00000000000000
7	0.05997280511964	0.03697318497994	0.01452596126591	0.00000000000000
8	0.04845704417766	0.02819450539239	0.01009122379682	0.00000000000000
9	0.03944901267606	0.02165472972177	0.00704115825305	0.00000000000000
10	0.03230483823269	0.01672104887702	0.00492784817409	0.00000000000000

Cuadro 1: Resultados para la aproximación de la matriz R en una cola $M/C_2/1$ [14]

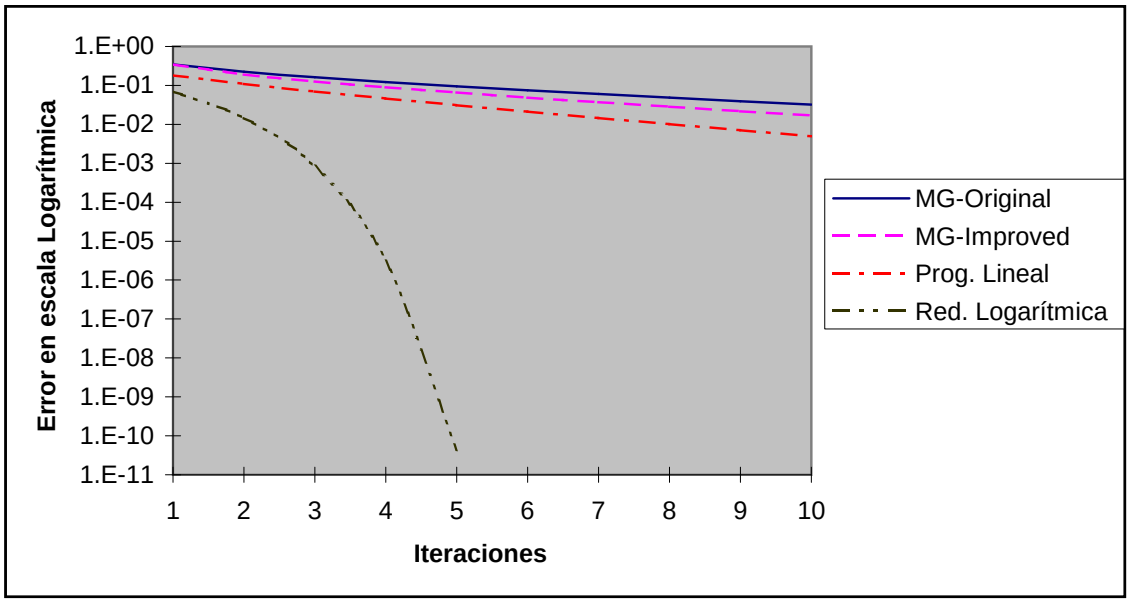

Figura 2: Resultados para la aproximación de la matriz R en una cola $M/C_2/1$ [14]

cola $M/C_2/1$, es decir una cola con arribos Poisson, un servidor Coxian, con dos fases, y una cola infinita. En el Cuadro 1 se pueden observar los resultados del error medido como $\|\mathbf{R}_{\text{exacta}} - \mathbf{R}_{\text{iter}}\|_2$, en ella se puede observar que el algoritmo de reducción logarítmica solo necesita de 6 iteraciones para calcular la matriz $\mathbf{R}$ utilizando la precisión completa del computador. Una comparación más formal en la eficiencia de los algoritmos puede ser observada en el trabajo de Hung y Tien [15], donde los resultados permiten observar el desempeño superior del algoritmo de Latouche y Ramaswami.

Se puede observar en la literatura la propuesta de diversas metodologías para el calculo de la distribución de probabilidad estacionaria del sistema. Un ejemplo de esto es es el algoritmo "Projection" propuesto por Ciardo y Smirni [3], en este algoritmo se resuelve el sistema utilizando las ecuaciones de Chapman-Kolmogorov para relacionar los flujos de entrada y de salida de cada uno de los estados en equilibrio. Sin embargo hasta la fecha no existen resultados avanzados en el desarrollo y comprobación de este algoritmo y su aplicación tiene restricciones sobre la estructura de la matriz $\mathbf{Q}$. Otra aproximación que cabe mencionar es la de Tüfekçi y Güllü [16], la idea básica del algoritmo es agrupar los niveles $\{n, n+1, \ldots\}$ en un supernivel l_n, de esta forma se define un sistema finito con el supernivel y los niveles restantes. La tasa de flujo condicional del desde el supernivel a los niveles restantes es aproximada simultáneamente con las probabilidades de estado estable. Este algoritmo no es tan rápido como el algoritmo de reducción logarítmica.

Por último cabe mencionar que igualmente en la literatura se han propuesto modificaciones al algortimo de reducción logarítmica. En el trabajo de Hung and Tien [15] se menciona la mejora hecha por Naoumov al algoritmo de Ramaswami, en donde la mejora pretende reducir la complejidad del algoritmo en cada iteración,

sin embargo al observar las comparaciones hechas en este trabajo, el desempeño no parece ser sustancialmente mejor. De otro lado se encuentra el trabajo de Ye [18], la idea de Ye es trabajar sobre los pasos que implican la inversión de la matriz $(-\mathbf{A}_1)^{-1}$, para esto el propone el algoritmo "GTH-like", los resultados mostrados muestran una efectiva mejora en la disminución del error en el calculo de $\mathbf{G}$ utilizando el mismo número de iteraciones, sin embargo la adición de este calculo aumenta la complejidad de cada una de las iteraciones.

Capítulo IV

Descripción del Programa

El módulo para modelar y resolver procesos de Markov con estructura de **QBD** fue desarrollado en JAVA [8], considerando que este desarrollo fue pensado como una adición a la librería JMarkov [13] que mantuviera las ventajas de la concepción que tiene ésta en la metodología para modelar los procesos de Markov. En este capítulo se hará una exposición detallada de la implementación de las clases que fueron desarrolladas para el módulo.

4.1. JMarkov, una breve descripción

La librería JMarkov fue desarrollada para modelar y resolver cadenas de Markov de tiempo discreto y continuo de gran escala utilizando las ventajas de la programación orientada a objetos que maneja el lenguaje JAVA. Para utilizar el software el usuario requiere un conocimiento básico del lenguaje Java. No es necesario que el usuario conozca el uso de estructuras de manejo dinámico de memoria ni necesita programar algoritmos de cómputo para calcular medidas de desempeño y probabilidades en estado transiente o estable, pues JMarkov es capaz de realizar estos cálculos. El diseño del programa permite subsanar el requerimiento de que los eventos de ocurrencia deben seguir distribuciones exponenciales con el uso de

distribuciones de fase [11], a costa de incrementar el tamaño del modelo. Con la facilidades computacionales este no debería ser un problema tan serio.

JMarkov provee objetos abstractos tales como State, Event y MarkovProcess, que el usuario debe extender para modelar el sistema específico de interés. Internamente JMarkov cuenta con una librería de funciones básicas que manipulan estos objetos abstractos para generar los estados y la matriz de transiciones así como funciones para la obtención de medidas de desempeño. Como se está usando un lenguaje de programación general, adicionalmente el usuario tiene la flexibilidad de definir en su programa características y métodos que considere necesarios en la descripción de cada elemento del sistema. De esta manera el programa flexibiliza la definición de los estados para el usuario a diferencia de otros programas que se han desarrollado en esta área, cabe mencionar entre ellos el trabajo que Stewart [14] realiza con el programa MARCA (MARKov Chain Analyzer) en la que el usuario debe ajustarse a una estructura precisa para otorgar al programa el archivo de entrada necesario para la generación del modelo. De otro lado cabe señalar que la idea de trabajo que se propone permite independencia total entre el modelo y los cálculos permitiendo que el usuario se concentre únicamente en la programación de las reglas de comportamiento del modelo y la representación apropiada de los estados. Esto gracias a que al extender las clases proporcionadas en la librería los objetos que el usuario cree, heredan los métodos necesarios para realizar los cálculos pertinentes del modelo.

4.1.1. Algoritmo de construcción de Cadenas de Markov BuildRS

JMarkov trabaja con una modificación del algoritmo BuildRS presentado por Ciardo [2]. Lo que el algoritmo busca es generar dinámicamente el conjunto de estados, haciendo una exploración del grafo partiendo de un estado inicial. Se supone que

existe un conjunto de Eventos posibles aunque no todos los eventos pueden ocurrir en cada estado. Para cada estado el algoritmo examina qué eventos pueden ocurrir y, en caso de que ocurra, qué estado destino genera la transición. De esta manera se van hallando nuevos estados y se actualiza la matriz de tasas correspondiente. Nótese que no es necesario saber de antemano cuántos estados tiene la cadena, pues el algoritmo los va a encontrar todos automáticamente. Si la cadena es finita el algoritmo termina. En el Algortimo 3 se explica en detalle el algoritmo. Los conjuntos $\mathcal{E}$, $\mathcal{U}$ y $\mathcal{S}$ representan, respectivamente, el conjunto de eventos, el conjunto de estados encontrados pero no explorados y el conjunto de estados explorados.

Algoritmo 3 Algoritmo de construcción de Cadenas de Markov BuildRS

$\mathcal{S} = \phi, \mathcal{U} = \{s_0\}$, $\mathcal{E}$ dado.
while $\mathcal{U} \neq \phi$ **do**
  for all $e \in \mathcal{E}$ **do**
    $i \in \mathcal{U}$
    if $\texttt{active}(i, e)$ **then**
      $j := \texttt{dest}(i, e)$
      if $j \notin \mathcal{S} \cup \mathcal{U}$ **then**
        $\mathcal{U} := \mathcal{U} \cup \{j\}$
      end if
      $R_{ij} := R_{ij} + \texttt{rate}(i, j, e)$
    end if
  end for
end while

El usuario de JMarkov debe seguir los siguientes pasos para modelar un problema específico:

- Codificar la representación de los estados.
- Determinar los eventos que pueden ocurrir.
- Implementar las siguientes 3 funciones que determinan el comportamiento del sistema:

1. `active(i,e)`, determina si el evento e está activo (puede ocurrir) en el estado i.
2. `dest(i,e)`, determina el estado destino de la transición generada por el evento e.
3. `rate(i,j,e)`, determina la tasa con la que se pasa del estado i al j si ocurre e.

Como de antemano no se sabe el número de estados que tendrá el modelo, es necesario utilizar una estructura de memoria dinámica para la generación de estados y el almacenamiento de la matriz de transiciones. JMarkov se encarga de esto y todo el proceso es transparente para el usuario.

4.2. El Módulo QBD

El módulo desarrollado en este proyecto mantiene las propiedades básicas de JMarkov. Básicamente el usuario debe concentrar su trabajo en una definición clara y compacta del problema que pretende modelar, relajando su preocupación por la programación de algoritmos que realicen los cálculos que involucran la solución del modelo. Este módulo esta compuesto por dos clases que deben ser extendidas por el usuario para la programación de su modelo, en esta sección se presentará una descripción de la implementación y funcionamiento de este, la documentación de las clases puede ser consultado en el Apéndice A.

La manipulación de matrices en la programación de las clases que componen este módulo fue una constante, por tal razón se hizo uso de la librería MTJ [7] para el manejo de estas matrices. La elección de esta herramienta en particular atendió a la orientación que tiene para manejar matrices dispersas, ya que en los modelos que se construyen para los procesos de **QBD** resultan en matrices altamente dispersas.

4.2.1. Clase `GeomProcess`

Esta es la clase principal del módulo ya que en ella esta codificado el algoritmo de reducción logarítmica, que de acuerdo con la exposición hecha en los capitulos 2 y 3 es básico en el calculo de la Matriz $\mathbf{R}$ y las probabilidades de estado estable. `GeomProces` extiende la clase `MarkovProcess` de JMarkov, por tal razón hereda todos los métodos existentes en ella. La idea al hacer esto es mantener al máximo la metodología que maneja JMarkov para modelar las cadenas de Markov procurando la mayor transparencia para el usuario en el uso de este módulo. Esta característica permite que al momento de modelar la cadena las consideraciones adicionales en la programación de las reglas de comportamiento de la cadena sean mínimas. De esta manera la clase `GeomProcess` tiene compatibilidad plena en la utilización del Algoritmo 3 de construcción de Cadenas de Markov BuildRS que esta implementado en la clase `MarkovProcess`, por tanto no se tuvo que trabajar en ninguna modificación del programa original.

En esta exposición se describirán los métodos críticos del programa, adicionales a estos existen algunos métodos sencillos que son utilizados como apoyo a la impresión de resultados pero que resulta irrelevante exponer.

4.2.1.1. Método `getGeomStates`

Este método construye un arreglo de objetos `GeomState` tomando los estados encontrados por el método `generate` de `MarkovProcess`, y que son almacenados en $\mathcal{S}$, una colección tipo TreeSet. Esto es necesario para que puedan ser manejados de una manera ordenada y eficiente en la manipulación que se tienen que hacer de los estados durante la construcción de las matrices que involucran los cálculos de las Matrices $\mathbf{G}$ y $\mathbf{R}$.

4.2.1.2. Método `getRatesA`

Este método fue hecho para la construcción de las submatrices $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$ y de las submatrices de los estados de frontera $\mathbf{B}$. Al momento de ordenar los estados en `getGeomStates` el programa identifica los estados de frontera y los estados que pertenecen a cada nivel relativo en los primeros tres niveles del proceso, delimitando las submatrices que se deben construir, luego esta información es argumento para este método que extrae los elementos de la matriz generadora $\mathbf{Q}$ que componen cada una de las submatrices. Este método construye cada matriz individualmente.

4.2.1.3. Métodos `getA1Complete` y `getB00Complete`

Básicamente estos métodos lo que hacen es calcular las diagonales de las matrices $\mathbf{A}_1$ y $\mathbf{B}_0$, esto es necesario ya que la diagonal de estas matrices no puede ser calculada si no hasta el momento de tener todas las matrices.

4.2.1.4. Método `getRmatrix`

Este método utiliza las matrices $\mathbf{A}_0$, $\mathbf{A}_1$ y $\mathbf{A}_2$ como argumento para el calculo de la matriz $\mathbf{R}$ utilizando el Algoritmo 2 de reducción logarítmica explicado en el Capitulo 3.

4.2.1.5. Método `GetInitialSol`

Aquí se calculan las probabilidades del vector π_0 utilizando las ecuaciones del Lema 2.2.3. Antes de comenzar el calculo de esta probabilidades este método hace un llamado al método `Stability` para verificar la estabilidad del sistema.

4.2.1.6. Método `steadyProbabilitysLevel`

Este método calcula los vectores $\boldsymbol{\pi_n}$ utilizando loa propiedad geométrico matricial del proceso $\boldsymbol{\pi_n} = \boldsymbol{\pi_{n-1}}\mathbf{R}$.

4.2.1.7. Método `ExpectedQueue`

En la ejecución del programa el módulo entrega como resultado la longitud promedio de la cola del sistema en estado estable, calculada así

$$\overline{N_q} = \boldsymbol{\pi}_1\mathbf{R}(\mathbf{I}-\mathbf{R})^{-2}\mathbf{e}.$$

4.2.1.8. Método `Stability`

Verifica el cumplimiento de la condición de estabilidad del proceso

$$\boldsymbol{\alpha}\mathbf{A}_2\mathbf{e} > \boldsymbol{\alpha}\mathbf{A}_0\mathbf{e}.$$

4.2.2. Clase `GeomState`

Esta clase del módulo es particularmente sencilla. Extiende `State` de JMarkov heredando los métodos de esta. Esta clase fue necesaria para establecer la identificación de los estados como estados de frontera o típicos y además establecer la pertenencia de los estados típicos a un nivel relativo. Sobreescribe el método `compareTo` de la clase `State` atendiendo a la necesidad de considerar la condición de frontera y el nivel de los estados para poder lograr un orden lexicográfico.

4.3. Metodología del programa

La metodología para modelar las cadenas de **QBD** utilizando este módulo se atiene estrictamente a la metodología planteada por JMarkov, por tanto el proceso es transparente para el usuario y el único requerimiento es tener un buena buena

habilidad en la tarea de modelar cadenas de Markov de una manera clara y compacta. Las consideraciones que debe tener claras son la limitación en el número de estados que debe generar, ya que si no trunca el proceso este no va terminar dado el carácter infinito de los procesos, además debe adicionar a la descripción de los estados el nivel relativo y el carácter de estado de frontera o típico.

Para utilizar este módulo el usuario debe crear tres clases en las que extiende las clases `GeomProcess`, `GeomState` y `Event`, en donde programará las reglas de comportamiento del sistema. Aunque la codificación de estas clases es bastante flexible, existe una estructura básica que el usuario debe seguir en cada una de las clases, esta estructura se describe en las siguientes secciones.

4.3.1. Estructura GeomState

En esta clase el usuario debe definir la estructura del arreglo descriptor de los estados del proceso, esto lo logra en la definición del constructor de la clase, al definir este constructor se debe establecer el tamaño del arreglo descriptor que en el caso de procesos **QBD** debe ser de 2 posiciones. En estos proceso se tiene la convención encontrada en la literatura general, es decir que la primera componente identifica el nivel n del proceso y la segunda componente identifica la fase i del estado. Igualmente se debe tener el parámetro que identifica el nivel relativo al que pertenece el estado, así como la definición de los estado de frontera o típicos. Estos son los requerimientos mínimos que debe contener la codificación de la clase. Adicionalmente esta clase hereda un método abstracto de la clase `State` en la que el usuario debe definir las etiquetas de los estados, esto es necesario para la impresión de los informes finales. Es potestad del usuario la definición de los métodos apropiados para la obtención de la información necesaria del estado en el proceso de programación de las reglas

de comportamiento del proceso.

4.3.2. Estructura Event

En la programación de esta clase el usuario debe básicamente identificar apropiadamente los posibles eventos que modifican los estados del proceso. Más allá de este mínimo requerimiento el usuario esta en libertad de definir cualquier otro método que le ayude con el manejo de los eventos.

4.3.3. Estructura GeomProcess

Esta clase es la que define el proceso concretamente. Esto es porque aquí es donde se especifican las reglas de comportamiento que gobiernan el proceso y que seguirá el Algoritmo 3 para la generación de los estados de este. La estructura básica de esta clase consiste en la implementación de los métodos abstractos que hereda de la clase `MarkovProcess`, estos métodos son:

- `active`: en este método se deben programar los criterios para definir cuando un evento esta activo cuando el proceso se encuentra en un estado determinado. Particularmente es en este método y en el siguiente donde debe ser considerado el hecho de que la generación de estados debe ser limitada a un conjunto finito, ya que el carácter infinito de la cadena puede generar un desborde en las pilas de memoria de la máquina si este factor no es manejado con cuidado.

- `dests`: en este método se definen las variaciones que se deben realizar sobre el estado actual para alcanzar un nuevo estado del proceso a partir de la ocurrencia de un evento específico.

- `rate`: en este método simplemente se definen las tasas a las cuales se mueve el proceso de un estado a otro dependiendo del evento que genera la transición.

Adicionalmente el usuario esta en libertad de implementar los métodos adicionales que considere necesarios para la obtención de información o manipulación del proceso. Hasta el momento ha sido costumbre en la utilización de la librería JMarkov la definición de un método `main` en el que el usuario crea una instancia de esta clase para ejecutarla como una aplicación y obtener los resultados. Esta no es una exigencia de la metodología de trabajo ya que la estructura de la librería y la orientación a objetos del lenguaje permiten que estas clases sean creadas para utilizarlas en un programa en el que estas sean parte de una rutina que involucra un proyecto mas general.

Después de observar la metodología de trabajo descrita es posible notar que las variaciones sobre la propuesta de JMarkov descrita en la Sección 4.1 son mínimas, lo cual facilita su utilización por parte de los usuarios que ya hayan tenido una aproximación a la librería en el trabajo con cadenas de Markov finitas.

4.4. Ejemplos

Para ilustrar el manejo de este módulo se presentan a continuación los resultados de dos ejemplos modelados y resueltos utilizando este metodología.

4.4.1. Ejemplo 1 Cola $M/M/1$ modificada

Este ejemplo es tomado de Nelson [10]. En la Figura 3 se puede observar una cola $M/M/1$ modificada con llegadas Poisson con tasa λ_1 cuando el sistema esta vacío y tasa λ_2 cuando hay por lo menos un cliente en el sistema. Los clientes requieren un servicio de dos fases exponenciales con tasas μ_1 para la primera fase y μ_2 para la segunda fase.

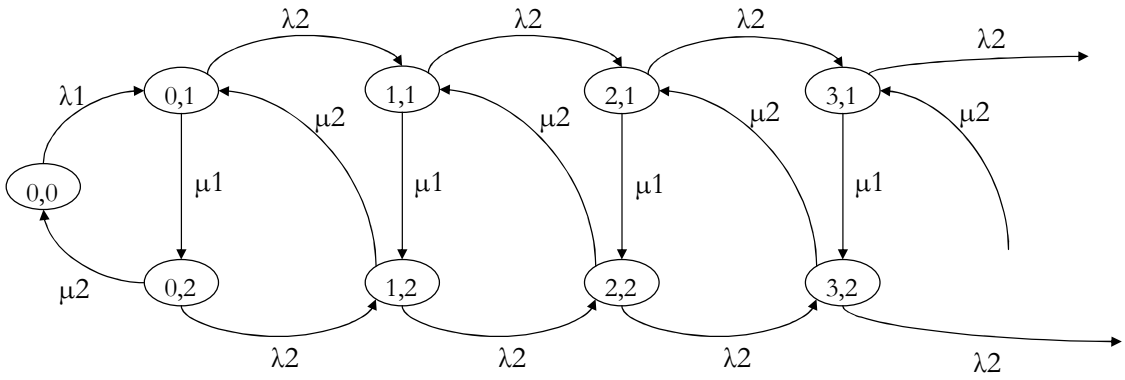

Figura 3: Cola $M/M/1$ Modificada

4.4.1.1. Modelo

Para modelar este ejemplo se van a considerar pares ordenados (n, i) donde la primera componente se refiere al nivel del proceso e indica el número de clientes en el sistema incluido el que esta en servicio, la segunda componente se refiere a la fase del servicio del cliente que esta siendo servido. Si los estados se ordenan de manera lexicográfica $(0,0), (1,1), (1,2), (2,1), \ldots$, de acuerdo con esta descripción la matriz generadora de este modelo es

$$\mathbf{Q} = \begin{pmatrix} -\lambda_1 & \lambda_1 & & & & & & & & \ldots \\ & -a_1 & \mu_1 & \lambda_2 & & & & & & \ldots \\ \mu_2 & & -a_2 & & \lambda_2 & & & & & \ldots \\ & & & -a_1 & \mu_1 & \lambda_2 & & & & \ldots \\ & \mu_2 & & & -a_2 & & \lambda_2 & & & \ldots \\ & & & & & -a_1 & \mu_1 & \lambda_2 & & \ldots \\ & & & \mu_2 & & & -a_2 & & \lambda_2 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

4.4.1.2. Modelo computacional

Para este modelo las clases se definieron de la siguiente manera:

1. Clase `QueueMM1ModifyState`: en esta clase se definió un arreglo de dos posiciones donde la primera componente es el número de entidades en el sistema y la segunda la fase de servicio del cliente que es servido. Adicionalmente se definieron tres métodos:

   - `getServerStatus` para consultar la fase del servicio del cliente que es servido en un estado particular.
   - `getQueueSize` para consultar el tamaño de la cola en un estado en el que se encuentre el proceso.
   - `getSystemNumber` para consultar el número de clientes que se encuentren en el sistema en un momento determinado.

   Por último se definió las etiquetas de los estados con el formato n, i, por convención el estado $0, 0$ significa que el sistema esta vació.

2. Clase `QueueMM1ModifyEvent`: En esta clase se definió la identificación de los eventos de la siguiente manera:

   - Llegada cuando el sistema esta vació = 0.
   - Llegada cuando hay al menos un cliente en el sistema = 1.
   - Terminación de la fase 1 de servicio = 2.
   - Partida del sistema (Terminación fase 2 del servicio)= 3.

3. Clase `QueueMM1Modify`: Esta es la clase que extiende `GeomProcess` y en la que se implementaron los métodos `active`, `dests`, `rate` del modo que se muestra en los algoritmos 4, 5, 6.

Algoritmo 4 Método `active` para la Cola $M/M/1$ modificada

```
Estado s_1 = (x, y) dado.
evento i dado.
result = false.
if i = 0 then
  if x = 0 y y = 0 then
    result = true.
  end if
else if i = 1 then
  if x ≤ 4 y x ≥ 1 then
    result = true.
  end if
else if i = 2 then
  if y = 1 then
    result = true.
  end if
else if i = 3 then
  if y = 2 then
    result = true.
  end if
end if
```

La codificación del programa para este modelo puede ser consultada en el apéndice B.

4.4.1.3. Resultados

Si le damos valores a los parámetros del modelo presentado de la siguiente forma, $\lambda_1 = 7$, $\lambda_2 = 4$, $\mu_1 =$ y $\mu_2 = 9$, en los resultados que se presentan en el Listing 4.1 se puede confirmar la coincidencia de la estructura de la matriz generadora con la matriz presentada en la sección 4.4.1.1, la presentación que se hace de los resultados

es una copia exacta de la pantalla de impresión de los cálculos obtenidos para este modelo.

Listing 4.1: Resultados Cola M/M/1 modificada.

```
Generating model...

   .........
  DONE. 9 states found.
   Building States...
   Printing States...
   Building States...
   Building Generator Matrix ...
   Building Rates Matrix ..
M/M/1 Modify

System has:
3 Boundary States.
2 States in level 1.
2 States in level 2.
2 States in level 3.

             EQUILIBRUM
STATE        PROBAB.       DESCRIPTION

0,0          0.06709
1,1          0.07531
1,2          0.05214
2,1          0.05859
2,2          0.05661
3,1          0.04874
3,2          0.05116
4,1          0.04135
4,2          0.04437
.            .
.            .
.            .

GENERATOR MATRIX:
                0,0       1,1       1,2       2,1       2,2       3,1       3,2       4,1       4,2
      0,0    -7.000     7.000
      1,1              -13.000     9.000     4.000
      1,2     9.000              -13.000               4.000
      2,1                                 -13.000     9.000     4.000
      2,2               9.000                       -13.000               4.000
      3,1                                                     -13.000     9.000     4.000
      3,2                                           9.000               -13.000               4.000
      4,1                                                                           -9.000     9.000
      4,2                                                     9.000                          -9.000
```

```
A0 MATRIX:

      4.000
                4.000
A1 MATRIX:

    -13.000      9.000
               -13.000
A2 MATRIX:


      9.000
R MATRIX:

      0.641      0.444
      0.197      0.444
Expected number of queued costumers: 0.001805636223658163
```

4.4.2. Ejemplo 2 Cola $M/C_2/1$

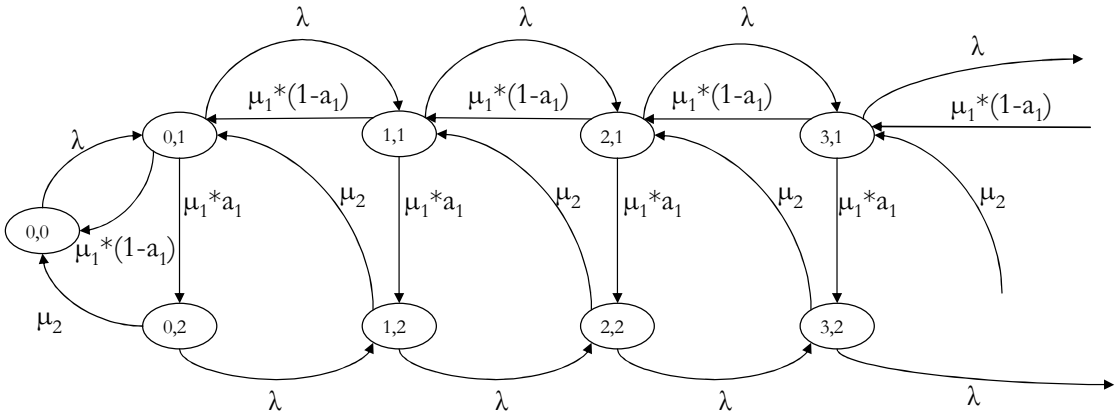

Figura 4: Cola $M/C_2/1$

En la Figura 4 se muestra un ejemplo tomado de Stewart [14], en el se puede observar una cola con llegadas Poisson con tasa λ, el servidor tiene un tiempo de servicio con una distribución Coxian de dos fases de distribución exponencial con tasas μ_1, μ_2 y con una probabilidad a de que al terminar la primera fase del servicio este tenga que continuar con la segunda fase, en caso contrario el servicio es terminado y el cliente sale del sistema; de otro lado al terminar la segunda fase el cliente

sale del sistema.

Este ejemplo fue generalizado a k fases en el ejercicio que se muestra el Cuadro 2, fue resuelto en un computador con un procesador Intel®Pentium 4 de 2.4 GHz, 512 MB en RAM y sistema operativo Windows®XP. En el se pueden observar para cada caso el número de fases del servidor, el número de estados encontrados, las dimensiones de las matrices manipuladas y el tiempo de ejecución del programa. En este Cuadro puede notarse que el programa alcanzo a manejar un servidor de 600 fases el cual generó un modelo con 2401 estados y matrices de dimensión 600×600, el tiempo de solución de este último ejemplo fue de aproximadamente 10 minutos.

Fases (k)	Estados	Dimensión Matrices	Tiempo de solución (milisegundos)
2	9	2×2	203
3	13	3×3	234
4	17	4×4	281
5	21	5×5	281
10	41	10×10	406
20	81	20×20	656
30	121	30×30	725
50	201	50×50	1109
100	401	100×100	4328
500	2001	500×500	330719
600	2401	600×600	584515

Cuadro 2: Resultados de la generalización de la cola $M/C_k/1$

Algoritmo 5 Método `dests` para la Cola $M/M/1$ modificada

Estado $s_1 = (x, y)$ dado.
rellevel = nivel relativo de s_1.
boundary = true si s_1 es de frontera, false en caso contrario.
evento i dado.
if $i = 0$ **then**
  if $y = 0$ **then**
    $x = 1$, $y = 1$.
  end if
else if $i = 1$ **then**
  if $x \leq 4$ **then**
    $x = x + 1$.
  end if
  if boundary = true **then**
    rellevel = -1, boundary = false.
  else
    rellevel = rellevel + 1.
  end if
else if $i = 2$ **then**
  y = y + 1.
else if $i = 3$ **then**
  if $x = 1$ **then**
    $x = 0$, $y = 0$.
  else
    $x = x - 1$, $y = y - 1$.
    if rellevel = -1 **then**
      boundary = true, rellevel = 3.
    else
      rellevel = rellevel -1.
    end if
  end if
end if
newstate = (x, y), newrellevel = rellevel, newboundary = boundary.

Algoritmo 6 Método `rate` para la Cola $M/M/1$ modificada

Estados s_1 y s_2 dados.
evento i dado.
rate $= 0$.
if $i = 0$ **then**
  rate $= \lambda_1$.
else if $i = 1$ **then**
  rate $= \lambda_2$.
else if $i = 2$ **then**
  rate $= \mu_1$.
else if $i = 3$ **then**
  rate $= \mu_2$.
end if

Capítulo V

Conclusiones

Este módulo ofrece al usuario una metodología que le permite mantener su concentración en el desarrollo de un modelo abstracto claro y compacto, esto gracias a la integración con la utilidad que JMarkov tiene desarrollada para la generación del modelo a partir de programar de una manera simple las reglas de comportamiento del sistema, de esta manera el usuario solamente debe que extender la clase GeomProcess para programar las reglas del sistema utilizando instrucciones básicas del lenguaje tales como `if ... then`, de este modo la integración con los algoritmos de solución de procesos **QBD** a JMarkov facilita el trabajo evitando el manejo de estructuras de datos complejas y el desarrollo de rutinas para realizar operaciones matriciales.

Puede observarse de los resultados obtenidos en el ejercicio presentado con el ejemplo de la cola $M/C_k/1$ que este programa permite el manejo de cadenas de tamaño considerable. Como punto de referencia se puede notar que la cadena más grande generada maneja matrices de dimensiones 600×600, lo cual supera significativamente la capacidad de una hoja de cálculo, hecho que significa una ventaja en la facilidad que esto implica para el manejo de matrices con estas dimensiones, al tiempo que disminuye la probabilidad de error en el manejo de las componentes

de estas matrices. Adicionalmente disminuye el tiempo destinado a la búsqueda de posibles errores dada la característica de generar el proceso a partir de las reglas de comportamiento ya que en general es más claro verificar la validez de éstas que realizar una búsqueda en las componentes de las matrices para encontrar el posible error. De otro lado, la posibilidad de realizar cambios en el modelo para consultar los efectos de estos se facilita en la metodología propuesta ya que estos cambios se darían en las reglas programadas lo cual es más claro y práctico que realizar cambios en las entradas de una matriz, aún más si el tamaño de esta es razonablemente grande. Por estas razones esta puede ser una herramienta útil en el desarrollo de trabajos de investigación que puedan involucrar el manejo de modelos complejos.

La aplicación de técnicas de simulación para la obtención de resultados que permitan apoyar la toma de decisiones en el desarrollo de modelos ha sido una práctica con resultados aceptables, sin embargo en este programa, aunque utiliza algoritmos de aproximación, sus resultados son prácticamente exactos ya que los criterios de aproximación permiten evaluar con alta precisión la diferencia entre el resultado obtenido y el resultado teórico.

5.1. Proyecto Futuro

Este módulo fue desarrollado para trabajar con procesos de Markov con estructuras de **QBD** en general, sin embargo es de anotar que existen estructuras de ciertos procesos de este tipo que pueden ser explotadas para obtener mecanismos de solución muy eficientes, un ejemplo claro de esto esta presentado en el trabajo de Dayar y Quessette [4], es por eso que a partir de este trabajo nace la inquietud de incorporar al módulo la posibilidad de utilizar los desarrollos que se hallan realizado

en este campo para buscar mejorar el desempeño del programa. Por otro lado existen dentro de los procesos de Markov infinitos muchas otras estructuras que pueden ser resueltas utilizando técnicas geométrico matriciales estrechamente relacionadas con las que han sido estudiadas para este trabajo, cabe mencionar en particular los paradigmas de Neuts [11], por eso otro aspecto a tener en cuenta como trabajo futuro es el desarrollo del módulo para trabajar con este tipo de estructuras.

JMarkov es una librería de objeto general para trabajar con cadenas de Markov de tiempo discreto o continuo, por tal razón es lógico pensar que es apropiada para resolver procesos **QBD** finitos, sin embargo, dada su estructura, estos procesos también pueden ser tratados de una manera eficiente utilizando métodos geométrico matriciales para su solución, por tal razón es otro elemento más de interés para desarrollos futuros.

Por último cabe anotar que uno de los aspectos que más inspira el interés en el trabajo con procesos que poseen estructuras repetitivas tipo **QBD** es la posibilidad de utilizar distribuciones tipo fase, esto dado que estas distribuciones pueden ser utilizadas para aproximar con facilidad varias distribuciones. Aunque en este momento existe una librería desarrollada por el profesor German Riaño para trabajar con estas distribuciones, un interés inmediato en nuestro trabajo es la incorporación definitiva de este módulo en JMarkov.

Apéndice A

Documentación

En este apéndice puede ser consultada la documentación de las clases en donde se presentan sus constructores y métodos con el fin de facilitar su utilización. Esta documentación fue generada utilizando la herramienta javadoc y de ninguna manera pretende ser un exhaustivo manual de usuario.

Apéndice B

Ejemplos

Listing B.1: QueueMM1Modify.java

```
import jmarkov.GeomState;
import jmarkov.GeomProcess;
import jmarkov.State;
import jmarkov.Event;
import jmarkov.EventsSet;




/**This class extends GeomState and define the properties of a system
 * whit one server and a queue whith infinite capacity, it has two properties,
 * namely the number of entities in the system and the server status.
 *
 * @author Julio Goez. Universidad de los andes.
 */
class QueueMM1ModifyState  extends GeomState {


  /**We identify the states whit the pair $(x,y)$ where $x$
   * is the number of costumers in the system (0,1,2,3,..)
   * and $y$ is the current service stage of the costumer in
   * service (1,2).
   * @param x Number of costumers in the system, including any receiving service
```

```
 * @param y Current stage of service of costumer.
 * @param rlevel Relative level of the state.
 */

QueueMM1ModifyState (int x, int y, int rlevel){
  super(2,rlevel);
  this.prop[1] = y;
  if (y == 0){
    this.prop[0] = 0;
  }
  else{
    this.prop[0] = x;
  }
}

/* (non-Javadoc)
 * @see jmarkov.State#computeMOPs()
 */
public void computeMOPs(){
  setMOP( "nada" ,0);
}



/**
 * @return The status of server when the system is in state "(x,y)"
 */
public int getServerStatus(){
   return this.prop[1];
}

/**
 * @return The Queue Size when the system is in state "(x,y)"
 */
public int getQueueSize(){
  if ( this.prop[0]== 0){
    return 0;
  }
  return (this.prop[0]-1);
```

```
  }

  /**
   * @return The number of costumers in the system in state "(x,y)"
   */
  public int getSystemNumber() {
      return this.prop[0];
  }


  /* (non-Javadoc)
   * @see jmarkov.State#label()
   */
  public String label(){
    String stg = prop[0] + "," + prop[1];
    return stg ;
  }

}


/**
 * This class define the events.
 * @author Julio
 *
 */
class QueueMM1ModifyEvent extends Event {

  final static int ARRIVAL1 = 0; // only for emtpy system.
  final static int ARRIVAL2 = 1; // When there are one or more costumers in the system.
  final static int SERVICE1 = 2; // the first stage of service.
  final static int DEPARTURE = 3; // the second stage of service.
  int type;

  /**
   * @param type Event type, e.g. Arrival1, Arrival2, Sevice1, departure.
   */
  QueueMM1ModifyEvent (int type){
```

```
    this.type = type;
  }

  /**
   * @return An array whit all the possible events in the process.
   */
  static EventsSet getAllEvents(){
    EventsSet E = new EventsSet();
    E.add(new QueueMM1ModifyEvent(ARRIVAL1));
    E.add(new QueueMM1ModifyEvent(ARRIVAL2));
    E.add(new QueueMM1ModifyEvent(SERVICE1));
    E.add(new QueueMM1ModifyEvent(DEPARTURE));
    return E;

  }

  /* (non-Javadoc)
   * @see java.lang.Object#toString()
   */
  public String toString(){
    String stg = "";
    switch (type){
      case (ARRIVAL1):
      stg = "Arrival when the system is empty";
      case (ARRIVAL2):
      stg = "Arrival when there are one or more items in the system";
      case (SERVICE1):
      stg = "When one item finish the first stage of service";
      case (DEPARTURE):
      stg = "When one item leave the system";
    }
    return stg;
  }
}
```

```
// The main class
/** This example represents a queueing system with
 * Poisson arrivals at rate $\lambda$ if the system
 * is empty, and a rate $\lambda$ otherwise. Costumers
 * require two exponential stages of service: the first
 * at rate $\mu_1$ and the second at rate $\mu_2$.
 * A Matrix Geometric Example (Nelson chapter 9).
 * @author Julio Cesar Goez. Universidad de los Andes.
 */
public class QueueMM1Modify extends GeomProcess{

  // only for emtpy system.
  final int ARRIVAL1 = QueueMM1ModifyEvent.ARRIVAL1;
  // When there are one or more costumers in the system.
  final int ARRIVAL2 = QueueMM1ModifyEvent.ARRIVAL2;
  // the first stage of service.
  final int SERVICE1 = QueueMM1ModifyEvent.SERVICE1;
  // the second stage of service.
  final int DEPARTURE = QueueMM1ModifyEvent.DEPARTURE;
  // Arrivals rates
  private double lambda1,lambda2;
  // service rates
  private double mu1  , mu2;

  /** Constructs a M/M/1 queue whit arrival sates lambda1 and lamdda2
   *  and service rates mu1 and mu2.
   * @param lambda1 Arrival rate when the system is empty.
   * @param lambda2 Arival rate otherwise.
   * @param mu1 Service rate for the first service stage.
   * @param mu2 Service rate for the second service stage.
   */
  public QueueMM1Modify(int lambda1, int lambda2, double mu1, double mu2) {
      super((new QueueMM1ModifyState(0,0,3)),
              QueueMM1ModifyEvent.getAllEvents());
      this.lambda1 = lambda1;
      this.lambda2 = lambda2;
      this.mu1 = mu1;
```

```
    this.mu2 = mu2;
  }



/* (non-Javadoc)
 * @see jmarkov.GeomProcess#active(jmarkov.State, int)
 */
protected boolean active(State i, Event e){

  QueueMM1ModifyState s = (QueueMM1ModifyState ) i;
  QueueMM1ModifyEvent ev = (QueueMM1ModifyEvent) e;
  boolean result = false ;

  switch(ev.type) {
    case ARRIVAL1:
    result = (s.getServerStatus() == 0) && (s.getSystemNumber() == 0) ;
    break;
    case ARRIVAL2:
    result = (s.getSystemNumber() <= 4) && (s.getSystemNumber() >= 1);
    break;
    case SERVICE1:
    result = (s.getServerStatus() == 1);
    break;
    case DEPARTURE:
    result = (s.getServerStatus() == 2);
    break;
  }

  return result;
}

/* (non-Javadoc)
 * @see jmarkov.GeomProcess#dest(jmarkov.State, int)
 */
protected State[] dests(State i, Event e){
  QueueMM1ModifyState s = (QueueMM1ModifyState) i;
  QueueMM1ModifyEvent ev = (QueueMM1ModifyEvent) e;
```

```
int newx = s.getSystemNumber();
int newy = s.getServerStatus();
int newlevel = s.getlevel();

switch (ev.type){
  case ARRIVAL1:
  if(s.getServerStatus()==0){
    newy = 1;
    newx = 1;
  }
  break;
  case ARRIVAL2:
  if (s.getSystemNumber() <4){
    newx += 1;
  }
  if(s.getlevel() == 3){
    newlevel = -1;
  }
  else{
    newlevel += 1;
  }
  break;
  case SERVICE1:
  newy += 1;
  break;
  case DEPARTURE:
  if(s.getSystemNumber() == 1 ){
    newx = 0;
    newy = 0;
  }
  else{
    newx -= 1;
    newy -= 1;
    if(s.getlevel() == -1){
      newlevel = 3;
    }
    else{
      newlevel -= 1;
```

```
      }
    }
    break;
  }
  return (new State[] {new QueueMM1ModifyState (newx, newy, newlevel)});
}


/* (non-Javadoc)
 * @see jmarkov.GeomProcess#rate(jmarkov.State, int)
 */
protected double rate(State i, State j, Event e){
  QueueMM1ModifyState s1 = (QueueMM1ModifyState) i;
  QueueMM1ModifyState s2 = (QueueMM1ModifyState) i;
  QueueMM1ModifyEvent ev = (QueueMM1ModifyEvent) e;
  double tasa = 0;
  switch (ev.type){
    case ARRIVAL1:
    tasa = lambda1;
    break;
    case ARRIVAL2:
    tasa = lambda2;
    break;
    case SERVICE1:
    tasa = mu1;
    break;
    case DEPARTURE:
    tasa = mu2;
    break;
  }
  return tasa;
}


/* (non-Javadoc)
 * @see jmarkov.MarkovProcess#description()
 */
public String description(){
```

```
    return "M/M/1 Modify";
  }

  public static void main ( String[] a ){
    QueueMM1Modify theQueue = new QueueMM1Modify( 7 , 4 , 9 , 9);
      System.out.print(theQueue);

  }

}
```

Referencias

[1] A.P.A. van Moorsel B.R. Haverkort and A. Dijkstra. *MGMTool: A performance modelling tool based on matrix geometrix*, pages 397–401. Antony Rowe ltd., Chippenham, England, 1992.

[2] Gianfranco Ciardo. *Tools For Formulating Markov Models*, pages 11–41. Kluwer Academic, 2000.

[3] Gianfranco Ciardo and Evgenia Smirni. Etaqa: An efficient technique for the analysis of qbd-processes by aggregation. 2002.

[4] Tugrul Dayar and Frank Quessete. Quasi-birth-and-death processes with level-geometric distribution. *SIAM J. Matrix Anal. Appl.*, 24(1):281–291, July 2002.

[5] Nihat C. Oguz et all. *TELPACK Version 2 Teletraffic Analysis Package*. School of Interdisciplinary Computing and Engineering, University of Missouri - Kansas City, http://www.sice.umkc.edu/telpack/index.html, 2.0 edition, 2002.

[6] R. V. Evans. Geometric distribucin in some two-dimensional queueing systems. *Operations Researchs Journal*, 15(46):830–+, 1967.

[7] Bjrn-Ove Heimsund. *Matrix Toolkits for Java (MTJ)*. University of Bergen : Fac. of Math and Nat. Sci., http://www.math.uib.no/ bjornoh/mtj/, version: 1.0 edition, 2004.

[8] Sun Microsystems Inc. *JavaTM 2 SDK, Standard Edition Documentation*. Sun Microsystems, Inc., http://java.cun.com/j2se/1.5.0/download.html, version 1.5.0 beta 1 edition, 2004.

[9] Guy Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastics Modeling*, volume 1 of *Statistics and Applied Probability*, chapter III and IV, pages 81–256. ASA-SIAM, Philadelphia, Pensylvania, first edition, 1999.

[10] Rnadolph Nelson. *Probability, Stochastics Processes and Queueing Theory. The Mathematics of Computer Performance Modeling*, volume 1. Springer - Verlag, New York, first edition, 1995.

[11] Marcel F. Neuts. *Matrix-Geometric Solutions in Stochastics Models. An Algorithmic Approach*, volume 1 of *Johns Hopkins Series in the Mathematical Sciences*. Johns Hopkins University Press, Baltimore, Maryland, first edition, 1981.

[12] T. V. Do R. Chakka and Zs. Pndi. *Library for solving QBD and QBD-M processes and MM sum(CPP, K)/GE/c/L G-queue solver tool*. http://www.hit.bme.hu/ pandi/download/solve/, 1.0 edition, 2004.

[13] German Riaño. *JMarkov Java Package. Instructions and Reference Manual. Universidad de los Andes.* Universidad de los Andes, http://copa.uniandes.edu.co., 2004.

[14] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*, volume 1, section 5.3, pages 258–269. Princeton University Press, Philadelphia, Pensylvania, first edition, 1994.

[15] Hung T. TRAN and Tien V. DO. Computational aspects for steady state analysis of qbd processes. *PERIODICA POLYTECHNICA SER. EL. ENG.*, 44(2):179–200, Nov 2000.

[16] Tolga Tüfekçi and Refik Güllü. An iterative aproximation scheme for repetitive markov processes. *Journal of Applied Probability*, 36:654–667, 1999.

[17] V. Wallace. *The solution of Quasi Birth ans Death processes arising from multiple acces computer system*. PhD dissertation, University of Michigan, System Engineering Laboratory, 1969.

[18] Qiang Ye. On latouche-ramaswami's logarithmic reduction algortihm for quasi-birth-and-death processes. *Stochastics Models*, 18(3):449–467, 2002.