Este exemplar corresponde à redação final da Tese / Dissertação devidamente corrigida e defendida por: André Luiz Garcia Pereira

e aprovada pela **Banca Examinadora.**

Campinas, 29 de Julho de 1999

COORDENADORIA DE PÓS-GRADUAÇÃO
CPG-IC

Uma Biblioteca para a Simulação de Tráfegos e de Eventos Raros em Redes ATM

André Luiz Garcia Pereira

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Uma Biblioteca para a Simulação de Tráfegos e de Eventos Raros em Redes ATM

André Luiz Garcia Pereira

dezembro de 1998

Banca Examinadora:

- Prof. Dr. Nelson Luis Saldanha da Fonseca[1] (orientador)
- Prof. Dr. Sibelius Lellis Vieira[2]
- Prof. Dr. Edmundo Roberto M. Madeira[1]
- Prof. Dr. Ricardo de Oliveira Anido[1] (suplente)

[1] Instituto de Computação, Unicamp.
[2] Departamento de Informática, Universidade Federal de Goiás.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Pereira, André Luiz Garcia

P414b Uma biblioteca para a simulação de tráfegos e de eventos raros em redes ATM / André Luiz Garcia Pereira -- Campinas, [S.P. :s.n.], 1999.

Orientador : Nelson Luis Saldanha da Fonseca

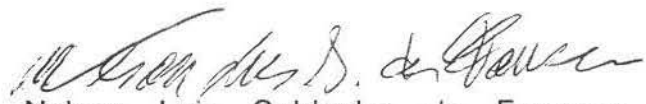
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Modo de transferência assíncrona. 2. Markov, Processos de. 3. Processos estocásticos. 4. Movimento browniano. I. Fonseca, Nelson Luis Saldanha da. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Uma Biblioteca para a Simulação de Tráfegos e de Eventos Raros em Redes ATM

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e definida por André Luiz Garcia Pereira e aprovada pela Banca Examinadora.

Campinas, 11 de dezembro de 1998



Nelson Luis Saldanha da Fonseca.
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Dissertação defendida e aprovada em 11 de dezembro de 1998, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Sibelius Lellis Vieira
UFG

Prof. Dr. Edmundo Roberto Mauro Madeira
IC - UNICAMP

Prof. Dr. Nelson Luís Saldanha da Fonseca
IC - UNICAMP

"Respondeu-lhe Jesus: Eu sou o caminho,
e a verdade, e a vida;
ninguém vem ao Pai senão por mim."
João 14:6

Dedico este trabalho ao meu
Senhor Jesus Cristo.

Agradecimentos

Ao meu Deus Único e Onipotente!

À minha mãe Maria do Socorro que sempre esteve ao meu lado orando e aconselhando-me com a sua sabedoria.

Ao meu pai Cid Ney e à tia Graça que me apoiaram e me acolheram nos momentos mais difíceis.

Aos meus irmãos Fabio, Daniela e Adriana que através de seus incentivos, não desanimei.

À minha tia Myrtes que esteve sempre me orientando em minha vida.

Aos Tios Walber e Iran que me ajudaram a superar dificuldades.

Ao Orientador Professor Dr. Nelson Luis Saldanha da Fonseca que apesar das minhas dificuldades, demonstrou paciência e apoio, mesmo quando as coisas não pareciam estar bem.

Ao amigo Professor Dr. Cândido Xavier de Mendonça, que me incentivou durante todo este trabalho.

Aos membros da Igreja Batista Nova Aliança e em especial ao Rogério, ao Alexandre, e ao Franklin que me apoiaram e oraram para que este trabalho fosse realizado.

Aos Pastores Ebenézer, Daniel, Calvin e Bill pelas suas orientações e orações.

Aos professores e funcionários do Instituto de Computação, que direta ou indiretamente contribuíram para que este trabalho fosse realizado.

À CAPES e à Universidade Federal do Maranhão, pelo apoio financeiro.

Prefácio

Será apresentado neste trabalho uma coletânea de técnicas e teorias úteis para o desenvolvimento de simuladores capazes de representar tráfegos de redes multimídia. Além de um estudo detalhado destas técnicas, foi desenvolvido uma biblioteca de rotinas voltada para a geração de tráfegos e para a simulação de eventos raros.

Abstract

In this dissertation a collection of useful techniques and theories for the development of simulators capable of handling network multimedia traffic. Besides a detail study of this techniques, a library of routines was develop in order to generate traffic and to simulate rare events in networks.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

A tecnologia digital sofreu um avanço considerável ao longo do anos, permitindo o desenvolvimento de técnicas de transmissão digital que possibilitam hoje, o transporte de tráfegos multimídia.

Como resultado desta tecnologia, surgiu o padrão Redes de Serviços Digitais Integrados de Faixa Larga (Broadband - Integrated Services Digital Network - B-ISDN)[40].

Redes com o padrão B-ISDN baseiam-se no modo de transferência Asyncronous Transfer Mode - ATM , meio pelo qual, texto, áudio e vídeo coexistem em uma forma simples, o que possibilita vantagens consideráveis ao tratar tráfegos multimídia. Utiliza uma forma de transmissão baseada em pequenas unidades de informação de tamanho fixo e formato padronizado denominadas células, são transmitidas através de conexões com circuitos virtuais, sendo orientadas pela informação contida em seu cabeçalho em cada célula. Isto é vantajoso por sua simplicidade. Por outro lado, modelos tradicionais que possuem unidades de informação de tamanho variável, a complexidade de equipamentos que possam suportar tais informações como "switches", cresce consideravelmente com o aumento do tamanho de cada pacote, além de gerar retardos com este empacotamento.

Para o dimensionamento de redes multimídia utiliza-se modelos analíticos e simulação. Modelos analíticos são complexos e muitas vezes intratáveis.O sucesso da simulação depende da identificação correta da medida de desempenho, ou seja, necessita-se saber quais elementos do sistema avaliado podem ser usados para ter-se resultados de acordo com a realidade. Tem-se como exemplo, a taxa de células perdidas em redes ATM como uma função do conjunto de parâmetros do controle de congestionamento.

A correlação também deve ser levada em consideração quando realizar-se estimativas de performance estatística. Considera-se a autocorrelação uma medida de dependência entre cada elemento da medida observada, existindo uma Autocovariância determinada pela dependência de valores em um determinado intervalo de tempo, verificando-se assim, a ligação entre as taxas de tráfegos entre estes intervalos. Há a existência de Autocorrelações positivas e negativas. A primeira é levada em consideração na simulação de tráfegos de redes. Autocorrelação positiva na seqüência de tempo de atrasos manifestam-se como quebras de longos ou pequenos atrasos e, a natureza autocorrelacionada das amostras tem complicado consideravelmente a tarefa de formar predições de performance.

Neste trabalho, foi desenvolvido então uma biblioteca contendo diversas rotinas voltadas para a geração de tráfegos e simulação de eventos raros.

1.2 Organização da Dissertação

Este trabalho está organizado da seguinte maneira:

No Capítulo 2 será apresentado alguns conceitos teóricos sobre os modelos de tráfegos básicos que frequentemente são utilizados.

No Capítulo 3 será descrito técnicas que servirão como arcabouço para a simulação de eventos raros.

No Capítulo 4 serão mostrados os procedimentos para a aplicação da simulação de geradores de tráfegos de Processos Markovianos gerais à modelos Auto-Semelhantes.

No Capítulo 5 apresentar-se-á a aplicação da teoria revista no Capítulo 3 descrevendo-se como criar simuladores de eventos raros.

No Capítulo 6 serão mostrados as contribuições, conclusão e motivação de trabalhos futuros.

Capítulo 2

Modelos de tráfegos

Neste capítulo serão apresentados alguns conceitos teóricos relativos a modelos de tráfegos que possuem uma considerável importância na retratação de tráfegos de redes reais.

Na atitude de gerenciar-se QoS, decisões de projetos baseados em previsões de desempenho de redes são realizadas, e, quando mal elaboradas, afetam sobremaneira o desempenho dos sistemas de comunicação em geral. Como uma base, técnicas analíticas, simulações e experimentos são utilizados para avaliação do desempenho de diferentes elementos tais como protocolos e equipamentos quando os mesmos são configurados e agregados, afim de que o objetivo final, que é o sistema desejado, atenda realmente os requisitos de cada cliente.

Tráfegos de dados tradicionalmente eram modelados através de modelos estatísticos bem simplificados, como Processos de Poisson. Com a integração de tráfegos multimidia, novos desafios surgiram. Apareceram características adicionais tais como em tráfegos correlacionados entre o tempo de chegada de células geradas por outros tipos de informações integradas, como áudio e vídeo digitalizados. E mais, o tráfego em rajadas ou "Bursty" surgem em aplicações como compressão de áudio ou transferência de arquivos. Ao se caracterizar uma

fonte com tráfego em rajadas, a utilização da taxa média de geração de células não é suficiente, pois a mesma não representa o seu comportamento em um momento especifico, e os parâmetros considerados mais adequados seriam a duração média dos períodos de atividade e a explosividade (burstiness) da fonte.

Tráfego auto-semelhantes é um tipo de tráfego comum encontrado em redes. É representado pela ausência de um comprimento natural de uma rajada de tráfego que pode variar de segundo à horas e sua característica estatística não muda em decorrer do tempo, ou seja, independentemente da amostra observada de milisegundos ou horas de tráfegos, o padrão encontrado é o mesmo.

Três modelos que são empregados ao representarem redes de comunicação de dados considerados mais utilizados, Processo de Poisson, Modelos Markovianos Modulados de Poisson e Movimento Browniano Fracional.

2.1 Processos Poisson

Um processo de Poisson é um processo de contagem de eventos randômicos que assumem apenas valores inteiros *(K(t) ∈ Z)*, representando o número total de eventos ocorridos entre o intervalo de tempo *0* e *t*, como o número de partículas radioativas emitidas por uma fonte durante este intervalo. Sendo assim, aparece como um modelo estocástico adequado para um grande número de fenômenos observáveis. Em relação à chegadas de pacotes, o modelagem de redes de dados é realizado dentro de um auto grau de precisão e tem sido empregado em larga escala.

A distribuição de Poisson é dada pela expressão:

$$P_k(t) = \frac{(\lambda t)^2}{k!} e^{-\lambda t} \qquad (1)$$

O Processo de Poisson é determinado como um processo puro de nascimento em relação a variável tempo com λ sendo a taxa média em que estes clientes chegam. Dado uma condição inicial (assume-se que o sistema começa no tempo *0* com *0* clientes):

$$P_K(0) = \begin{cases} 1 & k=0 \\ 0 & k\neq 0 \end{cases} \qquad (2)$$

$P_k(t)$ determina a probabilidade que *k* clientes cheguem no intervalo *(0,t)*. Se a taxa de chegada média é λ por segundo, o número médio de chegadas no intervalo *(0,t)* é λt. Isto pode ser demonstrado da seguinte forma. Dado *K* como o número de chegadas entre *(0,t)* :

$$E[K] = \lambda t \qquad (3)$$

A variância de um processo de Poisson de um número de chegadas também é λt. Para encontrar-se a variância, calcula-se o seguinte momento:

$$\sigma_K^2 = \lambda t \qquad (4)$$

é a variância encontrada.

Chama-se a atenção para a relação entre o processo de Poisson e a distribuição exponencial. Considere uma variável aleatória t' ser o tempo entre chegadas adjacentes em um sistema de filas, e que respectivamente *A(t)* e *a(t)* sejam suas Função de Densidade de Probabilidade e Função de Distribuição de

Probabilidade. Para tal definição, $a(t)\Delta t + o(\Delta t)$ é a probabilidade de que a próxima chegada ocorra entre t e $(t + \Delta t)$ segundos do tempo da última chegada. Por esta definição, $A(t)$ é a probabilidade que o tempo entre as chegadas seja $\leq t$, e é dada por:

$$A(t) = 1 - P[t' > t] \qquad (5)$$

Porém, $P[t' > t]$ é considerada uma probabilidade de não acontecer chegadas no intervalo $(0,t)$, dado por $P_0(t)$, tem-se pela modificação em *(5)*.

$$A(t) = 1 - P_0(t) \qquad (6)$$

Aplica-se a propriedade Poisson em *(1)*, tem-se:

$$A(t) = 1 - e^{-\lambda t} \qquad t \geq 0 \qquad (7)$$

logo, por diferenciação obtém-se

$$a(t) = \lambda e^{-\lambda t} \qquad t \geq 0 \qquad (8)$$

que é a distribuição exponencial.

Observa-se portanto, em *(7)* e *(8)* que para um processo de chegada Poisson, o tempo entre-chegadas é exponencialmente distribuído.

2.2 Processos Markovianos Modulados de Poisson

Processos Markovianos Modulados são processos que apresentam uma taxa de chegada dependente do estado de uma cadeia de Markov embutida e os intervalos de tempo entre cada chegada são correlacionados, ou seja, um Processo Markoviano Modulado é um processo não renovável com a flexibilidade de se poder escolher diferentes valores para as taxas de chegada em cada estado bem como as taxas de transição entre estados da cadeia de Markov embutida. Constitui também, uma classe importante de modelos de tráfegos, e geralmente são utilizados para descreverem tráfegos de vídeo e voz. Porém, outros tipos de tráfegos como dados podem também ser representados desta forma.

Processos Markovianos Modulados de Poisson são uma generalização dos processos de Poisson e um tipo particular de Processo Markoviano Modulado com chegadas em tempo continuo porém com espaço de estados discreto. Para este tipo de processo, independentemente do tempo corrente, o fluxo de chegada de células sempre será do tipo Poisson, porém, a taxa média do fluxo de chegada, é função de uma cadeia de Markov. A taxa média é uma variável aleatória determinada pelo estado de uma cadeia de Markov é irredutível e contínua no tempo, onde, esta cadeia modula a taxa média do processo markoviano modulado.

Para um Processo Markoviano Modulado de Poisson, a cadeia de Markov embutida é implementada através de uma matriz de transição de estados:

$$P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix} \qquad (9)$$

As chegadas de pacotes ocorrem segundo um processo de Poisson cuja a taxa média é uma variável determinada pelo estado atual da cadeia de Markov, ou seja, se o estado do sistema é i, então as chegadas ocorrem de acordo com um processo de Poisson com taxa λ_i.

Processos Markovianos Modulados de Poisson são modelos de tráfegos que tem tido uma grande importância devido a possuírem uma série de vantagens quais sejam, é analiticamente tratável, exibe as correlações entre chegadas sucessivas, permite a caracterização de tráfegos agregados ou não agregados de fontes homogêneas, sobrepor-se dois ou mais Processos Markovianos Modulados de Poisson como resultado tem-se um novo Processo Markoviano Modulado de Poisson e mesmo que exista um grande aumento na quantidade de fontes o seu nível de complexidade pode ser mantido através de uma combinação das técnicas de superposição de tráfegos agregados para gerar um Processo Markoviano Modulado de Poisson e superposição de Processos Markovianos Modulados de Poisson.

Fontes On-Offs são um caso particular de Processo Markoviano Modulado de Poisson com uma matriz de transição com apenas dois estados. O estado da cadeia de Markov resultante, corresponde ao ponto de quantização da taxa agregada, e o degrau desta quantização mais o número de estados e as taxas de transição são ajustados de maneira a casar com os parâmetros probabilísticos que são obtidos na observação do tráfego real. Tráfegos em rajadas são adequadamente representados por este modelo. Para os períodos ativos, são distribuídos exponencialmente com média T e os períodos de silêncio com média $S = T(b - 1)$. Uma única fonte em rajada pode ser modelada através de uma cadeia de Markov onde $\lambda = 1/S$ e $\mu = 1/T$.

A cadeia de Markov associada ao processo para uma fonte contendo N estados é vista à seguir.

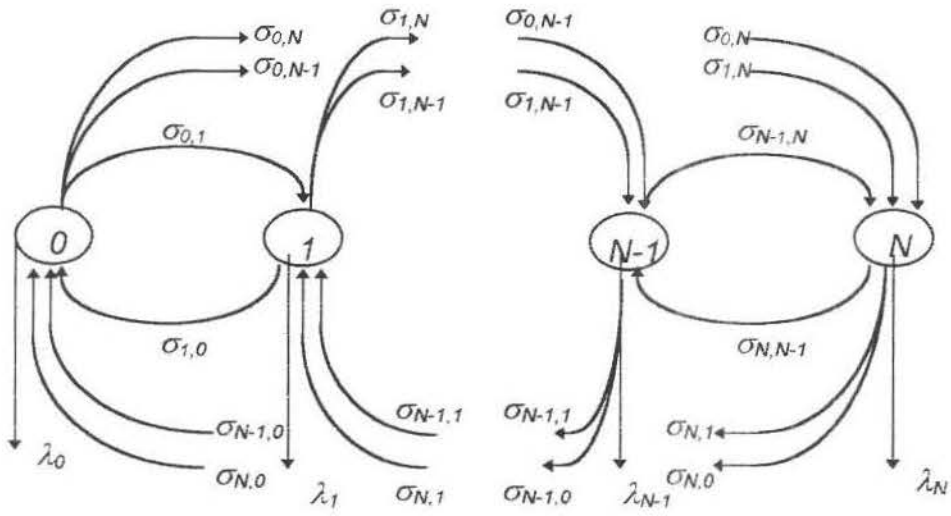


Figura 2.1: Processo Markoviano Modulado de Poisson - Cadeia de Markov

Um processo de nascimento e morte de um Processo Markoviano Modulado de Poisson representa o número de minifontes ativas:

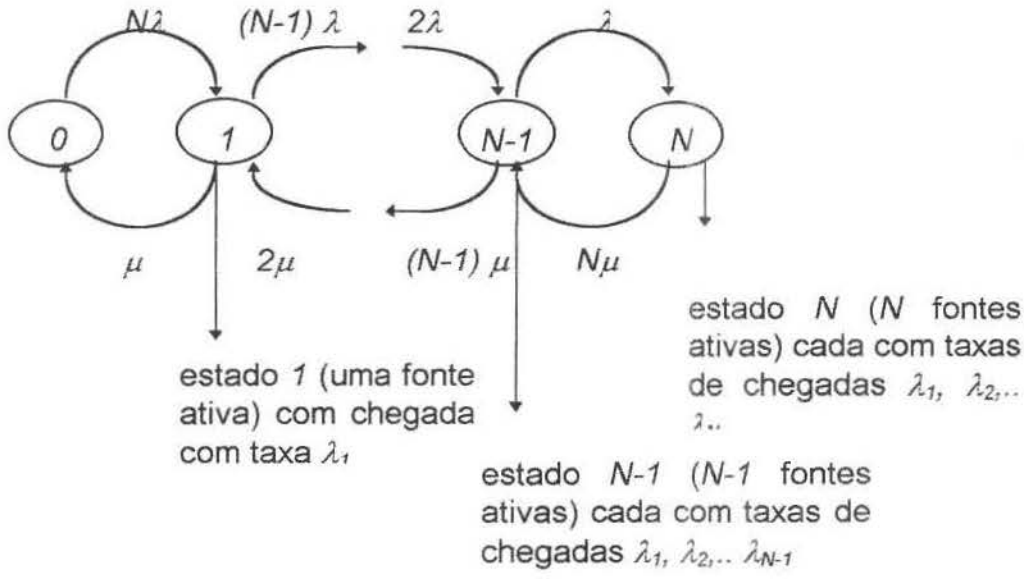


Figura 2.2: Processo Markoviano Modulado de Poisson - Nascimento e Morte

2.3 Movimento Browniano Fracional (Fractal Brownian Motion)

Para uma grande maioria de tráfegos, Processos de Poisson e Processos Markovianos Modulados de Poisson servem para uma modelagem segura e abrangente, e até mesmo auto-semelhança pode ser modelada em alguns casos. Entretanto, diversos tipos de tráfegos como de tráfegos de redes locais tem mostrado dependência de longa duração e também a presença de altas variâncias e até de variâncias infinitas. Por este fato, os modelos de Poisson não se adequam, pois não conseguem capturar tais características.

O modelo de tráfego chamado Movimento Browniano Fracional (Fractional Brownian Motion - FBM) estabelece um ganho de um melhor entendimento de questões relacionadas a performances de redes e filas quando as mesmas não obedecem a tradicionais modelos observando-se assim uma natureza fractal.

Antes de se definir Movimento Browniano Fracional, necessita-se contudo, ter-se bem conhecido a definição de auto-semelhança. A presença de auto-semelhança significa que ao observar-se o tráfego durante um longo tempo, pode ser observado que ao se encontrar as chamadas rajadas de tráfegos, não existe um padrão para a duração média das mesmas que podem durar desde milisegundos à horas.

Além do mais, auto-semelhança apresenta também uma dependência de longa duração associada a uma forte evidência de um sistema com memória possuindo uma função de correlação positiva (Uma correlação positiva significa que se uma série de tráfego aumentar durante um certo intervalo, a probabilidade de aumento em um próximo intervalo será alta) influenciando fortemente os próximos passos do tráfego, o que gera um impacto no comportamento das filas e seu estudo é de crucial importância ao solucionar problemas de engenharia de tráfegos de pacotes tais como estimação de tamanho de buffer, controle de

admissão e congestionamento. Se ignorado, pode causar previsões imprecisas de performances e alocações de recursos inadequados.

Analiticamente, define-se auto-semelhança da seguinte forma. Dado $X = (x_t : t = 1,2,3,...)$ ser uma covariância de um processo estocástico estacionário com média $\mu = E[x_t]$ e variância $\sigma^2 = E[(x_t - \mu)^2]$ e uma função de autocorrelação

$$t\ (k) = \frac{E[(x_t - m)(x_{t+k} - m)]}{E[(x_t - m)^2]} \qquad (10)$$

para $k = 0,1,2,3,...$. A função acima respeita a forma $\tau\ (k) \sim a^{k-\beta}$ com $k \to \infty$, $0 < \beta < 2$ e uma constante a. Dado $X_m = (x^m_k : k = 1,2,3,...)$ ser uma série de blocos de tamanho m:

$$x^m_k = \frac{1}{m} \sum_{i = km - m+1}^{km} x_i \qquad (11)$$

com $k = 1,2,3,...$ e $m = 1,2,3,...$.

Para cada m, X^m é uma covariância de processo estacionário. X é assim chamado auto-semelhante de segunda ordem com um parâmetro de Hurst $H = 1 - \beta/2$. Para que haja uma correlação positiva, o valor de H deve ficar entre $½$ e 1. O processo X é considerado um processo assintoticamente de segunda ordem com parâmetro de Hurst $H = 1 - \beta/2$, se

$$\tau^{(n)}\ (1) \to 2^{1-\beta} - 1, \text{ com } n \to \infty \qquad (12)$$

$$\tau^{(n)}\ (k) \to 1/2\delta^2(k^{2-\beta}), \text{ com } n \to \infty\ (k = 2,3,...) \qquad (13)$$

onde

$$\delta^2 (f(k)) = f(k + 1) - 2 f(k) + f(k - 1) \qquad (14)$$

Um Movimento Browniano Fracional representa um modelo de tráfego capaz de retratar auto-semelhança de forma acurada e realística para redes de altas velocidades baseadas em ATM. Um Movimento Browniano Fracional é um modelo que foi muito usado para descrever outros fenômenos como por exemplo a descrição de um movimento de uma partícula em um líquido sujeita a colisões.

Dado um Movimento Browniano Ordinário representando por $B(t)$, tem-se:

$$E[B(t + s) - B(t)] = 0$$

e

$$Var[B(t + s) - B(t)] = \sigma|s| \qquad (15)$$

A definição de Movimento Browniano Fracional é feita pela relação com a média de $dB(t)$ no qual incrementos passados de $B(t)$ são determinados pelo peso $(t - s)^{h - 1/2}$. Também por definição, um Movimento Browniano Fracional é um processo contínuo $B_H = (B_H(s) : s \geq 0)$, $0 < H < 1$, com função de correlação:

$$R(s,t) = \frac{1}{2(s^{2H} + t^{2H} - |s - t|^{2H})} \qquad (16)$$

possuindo para todo $a > 0$,

$$B_H(at) = a^H B_H(t) \qquad (17)$$

com

$$B_H(t) = \frac{1}{G(H + 1/2)}\int_{-\infty}^{0}((t - s)^{H-1/2} - (-s)^{H-1/2})dB(s) + \int_{0}^{t}(t - s)^{H-1/2}dB(s) \quad (18)$$

Esta equação (18) leva ao Movimento Browniano Ordinário se $H = 1/2$. O incremento do Movimento Browniano Fracional, X_j de uma sequência estacionária, chama-se Ruído Gaussiano Fracional (Fractional Gaussian Noise - FGN).

Um Ruído Gaussiano Fracional pode ser encarado como uma considerável aproximação de um complexo processo que apresenta dependência de longo alcance. Dado $X = \{X_k: k = 1,2, ...\}$ é um processo Gaussiano estacionário com média $m = E[X_k]$ e variância $\sigma^2 = E[(X_k - m)^2]$ com função de autocorrelação

$$\tau(k) = 1/2(|k+1|^{2H} - 2|k|^{2H} + |k-1|^{2H}), K=1,2,3,... \quad (19)$$

Assintoticamente, $\tau(k) \sim H(2H - 1)|k|^{2H-2}$, com $½ < H < 1$ exibindo desta forma, uma dependência de longo alcance caracterizando um processo auto-semelhante.

Na tentativa pode-se obter uma aproximação de amostras que gerem uma característica auto-semelhante através da simulação de um Ruído Gaussiano Fracional. Pode-se apontar como uma boa abordagem, o algoritmo recursivo descrito por Huang chamado de método de Hosking[12].

Uma outra abordagem, é o algoritmo Deslocamento de Ponto Médio Aleatório (Random Midipoint Displacement - RMD) que é apontado, como sendo um algoritmo rápido, simplificado, eficiente e bastante adequado qualitativamente. No tocante ao quantitativo, os autores também dizem que o processo é satisfatório para valores de $H < 0.9$.

Capítulo 3

Simulação de Eventos Raros

Neste capítulo serão introduzidas técnicas para a simulação de Eventos Raros.

Tráfegos de redes multimídia (dados, vídeo e áudio) requerem condições para se obter uma qualidade de serviço bastante considerável. Devido a estas exigências, a tecnologia nos levou a elevados controles e tolerâncias à falhas para que sejam satisfeitos cada requisito imposto por estes diferentes tipos de tráfegos. Devido a alta confiabilidade das novas redes de comunicação de dados baseadas em RDSI-FL (Rede Digital de Serviços Integrados de Faixa Larga), perdas de células causadas por bits com erro, congestionamentos ou erros de roteamentos de células entre outros, são cada vez mais difíceis de acontecer. Por exemplo em redes ATM, a probabilidade de uma célula ser descartada devido a erros de transmissão esta em torno de 10^{-6} ou inferior.

De maneira a buscar soluções para estimar taxas de perdas de células, são desenvolvidos diversas técnicas as quais muitas delas, são baseadas em modelos analíticos muitos deles são impossíveis de serem derivados. No intuito de encontrar uma solução, faz-se do uso de simulação. No entanto, para a utilização da simulação baseada na abordagem Monte Carlo, aparecem dificuldades variadas, e a maior delas é a complexidade computacional da

simulação. Um simulador baseado em Monte Carlo necessitaria pelo menos 10^{11} ciclos para fazer a estimativa da probabilidade de perda de células com uma probabilidade de 10^{-9}. Dependendo da complexidade do sistema, isto poderia consumir um alto tempo de simulação, mesmo para probabilidades maiores em torno de 10^{-4} à 10^{-3}. Para sistemas baseados em altas capacidades como fibras-óticas, encontram-se taxas de perdas da ordem de 10^{-10}, o que conclui-se, há a necessidade de pelo menos 10^{12} ciclos de simulação para achar-se uma estimativa.

3.1 Amostragem Importante (Importance Sampling).

Amostragem Importante foi concebido por volta de 1950 com trabalhos baseados em atenuação de problemas de partículas de neutrons. À partir de então, Amostragem Importante teve um maior impacto começando a aparecer na literatura das comunicações. Diversos autores como Shanmugan e Balaban[5] apresentaram os primeiros trabalhos com taxas de erros em comunicação de dados e métodos de Amostragem Importante foram essencialmente explorados. Estas idéias foram posteriormente discutidas por Jeruchim à partir de 1980[26]. Em 1983 um trabalho pioneiro foi realizado por Cottrel et all[14], onde utilizaram a teoria dos Desvios Largos para projetar simulações de eventos raros com Amostragem Importante em novos algoritmos de comunicações. À partir de 1990, outras importantes contribuições foram dadas à simulação de eventos de erros [1], [3], [7], [8] e outros.

Amostragem Importante é uma técnica que permite com bastante acurácia, fazer a estimativa de pequenas probabilidades em um espaço de tempo reduzido e que pode ser realizada sem a necessidade de muitos recursos computacionais, ou seja, pode-se simular aplicando Amostragem Importante com um simples microcomputador com pouca capacidade de memória.

A idéia fundamental é a redução da variância ou o "custo computacional" para buscar-se a probabilidade da ocorrência de eventos raros (perda de células). Possui um grande potencial para oferecer um substancial ganho de tempo de execução em comunicação digital. Para este objetivo, toma-se uso de uma evidência. Durante a simulação, alguns valores associados às variáveis aleatórias possuem mais impacto que outros, e, se estes valores importantes são enfatizados por um amostragem mais freqüente, então a variância do estimador da probabilidade pode ser reduzida. Por este motivo que a metodologia básica da técnica Amostragem Importante é a escolha de uma distribuição que encoraje o surgimento de valores "importantes". Este uso de uma distribuição modificada irá resultar em um estimador da probabilidade modificado.

A aplicação da técnica consiste em uma forma diferente do uso do simulador, modificando a sua estrutura original baseada em Monte Carlo em duas etapas, a primeira é referente à geração de células de um determinado processo de chegada, que é a modificação da função original, gerando uma função parcial ("bias") ou função torcida ("twisted"). A segunda refere-se a saída da simulação, uma vez que foi aplicado uma modificação para se acelerar a simulação na primeira etapa, o estimador final da probabilidade deve ser corrigido, limpo de qualquer intervenção causada pela alteração anterior, também chamado de estimador "unbiased".

A escolha de uma boa distribuição de probabilidade modificada é a arte da Amostragem Importante. O resultado esperado para uma boa distribuição pode ter um enorme ganho de tempo computacional, do contrário, uma escolha infeliz, pode ter um gasto de tempo muito longo na simulação e até mesmo um tempo maior do que o modelo de simulação tradicional Monte Carlo. Felizmente, existem orientações que podem ser seguidas ao diagnosticar as melhores alterações na distribuição de probabilidade aplicada ao simulador para realmente encontrar de maneira eficiente a probabilidade final do acontecimento do evento raro.

Geralmente, a distribuição modificada ótima para uma simulação é conhecida teoricamente em muitos casos. Durante os últimos 20 anos, tem aparecido uma grande proliferação de idéias engenhosas ao desenvolvimento da simulação com Amostragem Importante, e várias, tem tentado aproximar características de um distribuição ótima. Todavia, existem diferenças na aplicação consideradas bastante complicadas em sistemas de filas, apesar de existir uma solução razoavelmente simples para um "bias" em processos de chegadas do tipo Poisson, em Processos Poisson Markovianos Modulados, a solução é considerada mais difícil. Por outro lado, o "bias" encontrado, mesmo não sendo ótimo, pode reduzir a complexidade computacional em diversas ordens de magnitude abrindo variadas possibilidades para o uso de Amostragem Importante na determinação de células perdidas.

Existem diversos tipos de alterações que podem ser feitas. Para certos processos de Markov, alterações exponenciais baseadas na teoria dos Desvios Largos tem demonstrado serem melhor porque o processo envolve passos infinitesimais, além de que, para estes casos, geralmente de forma analítica, não é possível minimizar a variância do estimador da probabilidade. Parâmetros ótimos desta alteração são difíceis de se determinar exceto para processos de Markov de uma única dimensão. Infelizmente, um processo de chegada de células ATM muitas vezes envolve pelo menos um processo de Markov que possui no mínimo duas dimensões devido a natureza correlacionada do tráfego. Como exemplo, é possível descrever um modelo de simulador para se gerar um evento raro baseado em uma fila com um processo de chegada do tipo Processo Markoviano Modulado de Poisson utilizando um "bias" exponencial na matriz de transição de estados para a origem de dados[37], e testar diferentes valores para este "bias" em uma Amostragem Importante sub-ótima. É possível encontrar variadas reduções de variâncias em diversas ordens de magnitude para a aplicação.

Como já descrito anteriormente, a saída da simulação também deve ser alterada, ou melhor, corrigida. O resultado final de uma simulação baseada em Amostragem Importante, deve ser imparcial. Coloca-se um peso ao corrigir o cálculo da taxa de probabilidade do evento assegurando que o resultado final não seja alterado devido a modificação feita na distribuição durante toda a simulação. O peso é uma derivação de uma distribuição subalterna com respeito a distribuição da simulação. Toma-se o uso de parâmetros que devem ser escolhidos muito cuidadosamente para aplicá-los na função de densidade de probabilidade associada na simulação. Em suma, este resultado deve ser livre de qualquer parcialidade, a modificação da distribuição por um determinado processo (por exemplo, Teoria dos Desvios Largos), não deve influenciar no resultado final que é mostrado sem as influências causadas no simulador para a aceleração do processo.

Analiticamente, define-se Amostragem Importante da seguinte forma. Dado U como sendo uma variável aleatória que possui uma função de densidade de probabilidade $p(u)$, considera-se a estimativa da probabilidade P tal que U esteja em algum conjunto A, então:

$$P = \int_{-\infty}^{\infty} I_A(t)p(t)dt = E_p[I_A(U)] \qquad (1)$$

onde $I_A(.)$ é a função indicadora do evento A.

Assume-se que $p'(u)$ é outra função densidade e que $p(u) = 0$ se $p'(u) = 0$, portanto:

$$P = \int_{-\infty}^{\infty} I_A(t)\frac{p(t)}{p'(t)}p'(t)dx$$

$$= E_{p'}\left[I_A(U)\frac{p(U)}{p'(U)} \right] = E_{p'}\left[I_A(U)\, L(U) \right] \qquad (2)$$

onde:

$$L(U) = \frac{p(U)}{p'(U)} \qquad (3)$$

É a chamada taxa de probabilidade ou função peso e a notação p' denota a amostragem de uma densidade $p'(U)$. Isto sugere um esquema de estimação de redução de variância (Amostragem Importante). Dado então, N amostras $u_1, \ldots, u_N$ usando a densidade p', pela equação (2), uma estimativa não modificada de P é dada por:

$$P'_N = \frac{1}{N}\sum_{n=1}^{N} I_A(u_n) L(u_n) \qquad (4)$$

P pode ser estimado pela simulação de uma variável aleatória com uma densidade diferente e então retirar a modificação da saída $I_A(u_n)$ pela multiplicação da taxa de probabilidade. Chama-se $p'(u)$ de "Densidade Torcida" ou "Densidade Biased".

Para a equação em (3), alguns dos maiores tratamentos matemáticos de Amostragem Importante usam uma outra abordagem, a "Radon-Nikodym derivative" (RND)[5] em vez da função peso permitindo que a técnica Amostragem Importante seja discutida para problemas mais complexos, como por exemplo situações onde as densidades não existem ou as entradas são processos estocásticos sobre tempos contínuos. Dado (Ω, ξ, P) ser um espaço de probabilidade com Ω o espaço amostral de interesse, ξ é o evento encontrado em Ω onde é medida a probabilidade de seu acontecimento e P é a medida de

probabilidade associada a números entre *0* e *1* aos eventos em ξ. Dado *Q* sendo uma segunda medida de probabilidade de eventos em ξ possuindo a propriedade de para cada evento *A* dado que *Q(A) = 0*, tem-se *P(A) = 0*. Logo, a taxa de probabilidade de *P* em relação a *Q* encontrada é dada por:

$$P(E) = \int_E \left(\frac{dP}{dQ}\right) dQ \qquad (5)$$

Para chegar a verdadeira probabilidade de erro em Amostragem Importante, a simulação é apropriadamente arrumada de maneira a obter-se um resultado fiel além de um ganho computacional considerável e, para isso acontecer, escolhas de parâmetros "bias" são uma preocupação à parte. Parâmetros de parcialidade "bias" são valores que são utilizados como entrada para alimentar a função modificada de forma a estimular a aceleração do simulador com o objetivo de chegar-se mais rápido ao evento raro. Trata-se de uma questão bastante importante de como o simulador será alterado de modo a comportar-se como desejado. O valor a ser injetado na simulação possui um grau de importância tão grande que se for mal escolhido, levará a resultados bem piores que a simulação normal de Monte Carlo, e é óbvio, caso encontre-se o valor adequado, vários ciclos de execução serão poupados.

A mais importante técnica utilizada para encontrar-se o valor "bias" tem sido a teoria dos Desvios Largos[8], [14], [15]. A base central estabelece que erros são determinados por um grande desvio das variáveis inerentes ao sistema em relação à suas médias. E durante a aplicação sobre a técnica Amostragem Importante, a teoria dos Desvios Largos é usada para se estudar a eficiência assintótica do estimador da probabilidade. A chave deste estudo, é que tal estimador requer um número de tentativas de simulação que não seja tão grande ou que não tenha um crescimento exponencial, e, a teoria dos Desvios Largos é capaz de identificar estes valores que tornam uma simulação mais demorada, e

por outro lado, identificar valores que façam o simulador chegar ao objetivo. A teoria dos Desvios Largos será mais detalhada mais adiante.

Apesar de diversos esforços baseados na teoria dos Desvios Largos terem sido realizados ao apresentarem maneiras de encontrar-se o valor "ótimo" para um conjunto de parâmetros que devem levar a simulação Amostragem Importante à sua melhor performance, lembrando-se de que, esquemas úteis e práticos de Amostragem Importante são paramétricos, ou seja, encontrar-se parâmetros ótimos podem ser considerados um problema de otimização não linear multidimensional. Os valores dos parâmetros devem ser marcados ao otimizarem alguma medida de performance, casualmente a variância do estimador, $\sigma^2_{AI}(p,p^*)$. Assume-se que uma representação exata de uma variância não seja disponível, tem-se proposto o uso de medidas de performance que são estimativas estatísticas da variância do estimador, $\sigma^2_{AI}(p,p^*)$.

Um outro método eficiente a se obter um valor favorável para o "bias" próximo ao ótimo baseia-se na procura heurística e uma abordagem analítica aproximada, onde através desta técnica, tem-se realizados vários experimentos que tem tido bons resultados em modelos para tráfegos Auto-Semelhantes[12][16]. Como exemplo, Huang et ali [12] observou as propriedades das amostras obtidas bem como a variância do estimador de $Pr(Q_k > b)$ onde Q_k é a fila e b é o tamanho máximo da fila, são altamente afetadas pela escolha do "bias".

Um fato importante nestas técnicas está no que se refere a redução de variância do estimador da probabilidade de encontrar-se um evento raro. Ao comparar-se um estimador de um simulador Monte Carlo com um estimador Amostragem Importante verifica-se a diferença. Observa-se abaixo a relação entre os estimadores de Monte Carlo e de Amostragem Importante, respectivamente *MC* e *AI*:

$$VAR(P_{MC}) = s^2_{MC} = \frac{P_E(1-P_E)}{n} \qquad (6)$$

$$VAR(P_{AI}) = s^2_{AI} = \frac{1}{n}\left\{\int 1_E(x)w(x)f(x)dx - P_E^2\right\} \qquad (7)$$

Na maioria dos trabalhos em Amostragem Importante a distribuição modificada ótima é dada por:

$$f^*_{OPT}(x) = \frac{1_E(x)f(x)}{P_E} \qquad (8)$$

Desde que a substituição de (8) em (7) o resultado seja $\sigma^2_{AI} = 0$.

Não é possível utilizar a equação (8) na prática, serve como um referencial para confecções de distribuições modificadas adequadas. A idéia central para se alcançar boas distribuições está na modificação de parâmetros associados às distribuições originais. Muito comum a alteração por meio da modificação da média de modo a se obter uma nova distribuição com valores de erro mais rapidamente, ou seja, a alteração fará com que a distribuição original torne-se uma distribuição modificada e comporte-se de maneira a gerar valores de erros assim esperados como raros. Isto é mostrado em dois exemplos.

Na aplicação de Amostragem Importante em uma fila M/M/1[6], [18], com taxas de chegada e serviço respectivamente λ e μ, para a estimação da probabilidade de erro, o estímulo ao aparecimento do evento de erro é feito pela troca das taxas médias λ por μ e μ por λ.

Outro exemplo, agora não tão simples, ao aplicar Amostragem Importante na simulação de tráfegos Auto-Semelhantes[12], [16]. Dado um processo

Fractional Gaussian Noise (FGN) X com $m=0$, a média e a variância de X_k, dados valores obtidos $x_{k-1}, x_{k-2}, \ldots, x_1$ pode serem dadas como:

$$m_k = E(X_k | x_{k-1}, x_{k-2}, \ldots, x_1) = \sum_{j=2}^{k} F_{kj} x_{k-j} \qquad (9)$$

e

$$v_k = Var(X_k | x_{k-1}, x_{k-2}, \ldots, x_1) = s^2 \prod_{j=2}^{k} (1 - F_{jj}^2) \qquad (10)$$

Afim de se gerar a densidade modificada aplica-se uma média considerada "torcida" determinada através de uma técnica analítica denominada Procura Heurística que busca estimar a média considerada ótima ou com proximidade à ótima, pode ser encontrada através da relação:

$$m^*_{opt} = \frac{m}{H - m} \qquad (11)$$

onde H é o parâmetro de Hurst e μ é a média normalmente distribuída.

Através do valor m^*, pode-se chegar a distribuição modificada para o tráfego FGN. Seja $Y'=\{Y'(k) : Y'(k) = X(k) + m^*, k=1,...\}$ o processo FGN modificado, são realizadas novas amostras utilizando a distribuição modificada $(y'_1,\ldots,y'_{k-1})$ do processo Y' com média e variância:

$$m_k = m^* + \sum_{j=2}^{k} F_{kj}(y'_{k-j} - m^*) \qquad (12)$$

e

$$v_k = Vary'(Y'_{k-1} | y'_{k-1}, \ldots, y'_1) = Varx(X_k | x_{k-1}, \ldots, x_1) \qquad (13)$$

Novas considerações são vistas. Dado um espaço amostral do sistema envolvido, uma função de densidade modificada deveria encaixar-se precisamente em cima de uma região deste espaço pela qual existe o evento raro (erro). Na maioria das vezes torna-se um tanto difícil para enquadrar estes valores, porém, considera-se uma boa distribuição modificada como sendo aquela pela qual esteja enquadrada ao máximo possível na região onde os erro são mais freqüentes e que a mesma não represente nada nas demais regiões, ou seja, o seu valor seja zero em regiões não envolvidas.

Diversas visões podem ser obtidas à partir de um estudo mais detalhado. Acredita-se que distribuições consideradas como boas distribuições podem ser encontradas por diversos métodos. Pode ser citado dois deles, Tradução de Média (Mean Translation - MT) e Mais Provável Caminho para o Erro (Most Probable Error Path - MPEP)[36]. O primeiro trata-se de mudar a média da função de densidade original de forma a forçar a mesma a cair na região de erro, enquanto que a segunda, como uma extensão da primeira, leva-se em consideração a vizinhança dos pontos da região de erro como boas.

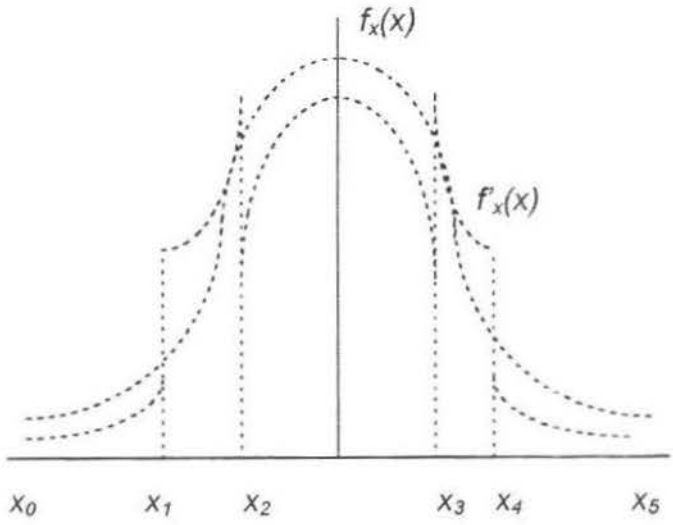


Figura 3.1: Relação entre a função atual e sua modificação

Dada acima o exemplo de uma função de densidade de probabilidade $f_x(x)$, a mesma é modificada para uma forma $f^*_x(x)$ que é aplicada obtendo valores que graficamente pode-se perceber que aparecem mais amostras para os intervalos $[x_1, x_2]$ e $[x_3, x_4]$.

Selecionando-se uma amostra x de $f^*_x(x)$, sua probabilidade é aumentada por uma "bias" $B(x)$:

$$B(x) = \frac{f^*_X(x)}{f_X(x)} \qquad (14)$$

e o peso da amostra é dado por:

$$W(x) = \frac{1}{B(x)} \qquad (15)$$

e a probabilidade final não modificada é obtida por:

$$P_j = \frac{1}{N} \sum_{i=1}^{n_j} \frac{1}{B(X_{ij})} \qquad (16)$$

A análise do fator de "bias" é referido no que é capaz de diminuir a variância do estimador. No caso, a obtenção do valor de $B(x)$ para que esta variância seja menor do que a variância da função original, escolhe-se um valor de $B(x) > 1$.

É descrito analiticamente um valor "ótimo" para o caso avaliado. Considera-se a estimação da probabilidade da cauda $Pe = P(X > T)$ onde X é uma variável Gaussiana de média zero variando entre 0 e T, $T > 0$. Dado:

$$B(X) = \frac{c}{[f_X(x)]^a} \tag{17}$$

onde as constantes c e α são escolhidas tal que:

$$\int_{-\infty}^{\infty} f_X^*(x)dx = \int_{-\infty}^{\infty} B(x)f_X(x)dx = 1 \tag{18}$$

e

$$t = \frac{N_C}{N_{AI}} \approx \frac{\int_{I_j} f_X(x)dx}{\int_{I_j} W(x)f_X(x)dx} \tag{19}$$

N_c = Número de amostras requeridas para a análise da simulação Monte Carlo

e

N_{AI} = Número de amostras requeridas para a análise da simulação Amostragem Importante

seja maximizado.

A equação dada em (18) implica que o valor de c é dado por:

$$c = \sqrt{\frac{(1-a)}{(2p)^a}} \tag{20}$$

onde α tem um valor no intervalo $(0,1)$.

o fator de ganho do tamanho da amostra para uma fdp Gaussiana em acordo com (17) é demonstrado através de:

$$t = \sqrt{1-a^2}\,\frac{Q(T)}{Q(T\sqrt{1+a})} \qquad (21)$$

onde

$$Q(y) = \frac{1}{\sqrt{2p}}\int_y^{\infty} e^{\frac{-x^2}{2}}\,dx \qquad (22)$$

Para valores altos de y, $Q(y)$ pode ser aproximado por

$$Q(y) \approx \frac{e^{\frac{-y^2}{2}}}{y\sqrt{2p}} \qquad (23)$$

e , uma vez feito isto, pode-se aproximar também τ por:

$$t \approx \sqrt{(1+a)(1-a^2)}\; e^{\frac{aT^2}{2}} \qquad , T >> 1 \qquad (24)$$

Da equação (24), obtém-se o valor de α que maximiza τ como:

$$a_{opt} = \frac{-3+\sqrt{9+4T^2(1+T^2)}}{2T^2} \qquad (25)$$

A nova distribuição é verificada na figuraà seguir:

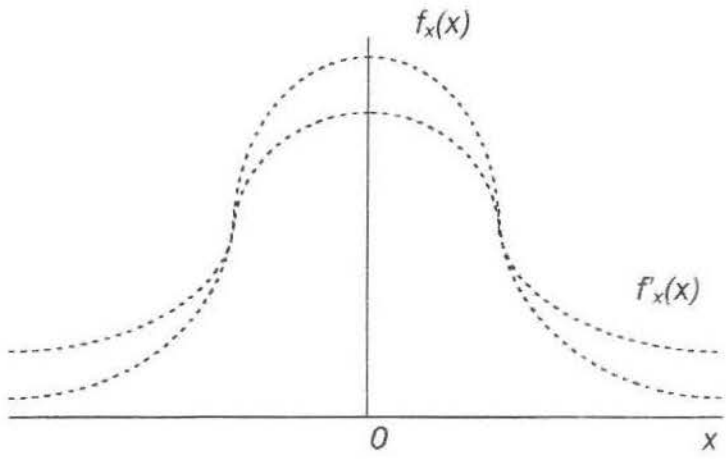


Figura 3.2: Modificação de uma distribuição Gaussiana

Existem para a abordagem Amostragem Importante três tipos de distribuições "bias" que podem ser encontradas além do "ótimo". O primeiro caso, chamado de Partial Biasing (P), descrito como uma distribuição que se aproxima do erro mas às vezes não é suficiente para o encontrar-se o mesmo. A segunda e muito utilizada, Most Probable Error Path (MPEP), como já vista anteriormente, o caminho mais provável para o "ótimo", quando não há possibilidade de se encontrar o "bias" ótimo, utiliza-se valores pertencentes à vizinhança da região de erro. O terceiro caso, OVER biasing, as variáveis aleatórias são quase sempre geradas em um subconjunto da região de erro onde a distribuição "bias" é considerada muito larga em relação a distribuição original superestimando o resultado causando também resultados não reais. Todos os casos relacionados acima baseiam-se no método Mean Translation. O efeito de não encontrar-se o ótimo pode-se levar a uma larga variância e a problemas de subestimação de resultados.

O problema de subestimação associado com o da superestimação são difíceis de se detectarem e não deveriam ser ignorados. Pode-se facilitar as coisas experimentando sempre vários tipos de "bias". O caso é ilustrado à seguir:

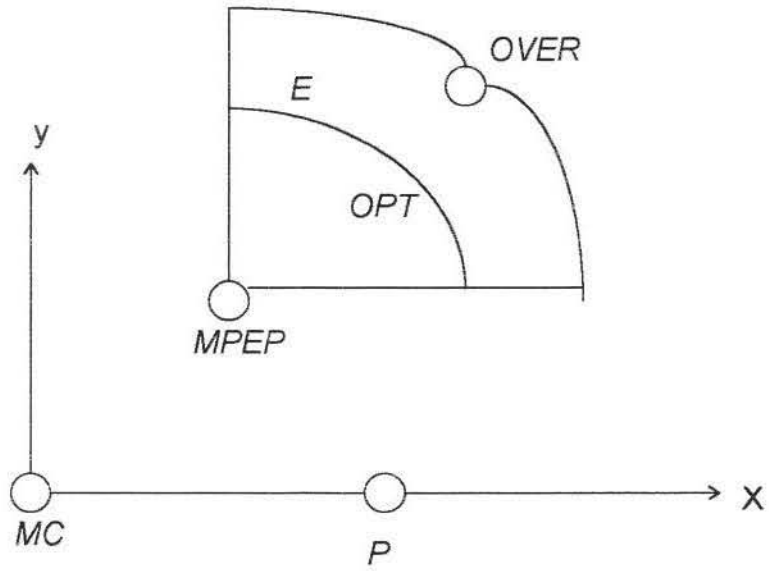


Figura 3.3: Possíveis valores de alcance de uma simulação

Amostragem Importante foi concebida para diversas situações de complexidade, todavia, apareceram muitos e muitos casos e os sistemas ficaram ainda muito mais complexos dificultando cada vez mais o uso de Amostragem Importante. Esta abordagem se complica consideravelmente nestes sistemas onde por exemplo, apresentam-se por terem uma grande extensão de memória, memória infinita ou é desconhecida. A memória é, por definição, a dimensão das variáveis de entrada, pode-se imaginar um sistema com mais de N dimensões, $X_i = \{X_{1i}, X_{2i}, \ldots, X_{Mi}\}$, um vetor de variáveis com $M > N$ e $N >> 1$, por esta razão, é considerada uma boa abordagem para combater a complexidade de memória de um sistema, a divisão da simulação em diversas partes tratando-se como sub-problemas.

3.2. Amostragem Importante em Sistemas com Memória

De maneira a encontrar uma saída para sistemas que apresentam memória, diversos trabalhos tem sido desenvolvidos e diversos problemas tem sido atacados durante os últimos anos em todas as áreas que tem sido possível a aplicabilidade de Amostragem Importante[7], [23], [24], [36]. A dificuldade do uso

de distribuições "bias" em relação ao se encontrar o "ótimo" ou "próximo ao ótimo" aumenta consideravelmente com o crescimento da memória de um sistema. Isto enfatiza a idéia de que ao se desenhar uma Cadeia de Markov modificada, a sua matriz de transição deve ter uma forma a mais próxima possível da matriz de transição original, pois, caso contrário, para longas memórias pode-se levar a problemas de "Overbiasing". E mais, memórias variáveis, desconhecidas ou até infinitas são uma grande preocupação, como por exemplo, complexos sistemas codificados: receptores usando decodificação modificável ou equalizadores de sinais adaptativos são sistemas que fazem parte desta categoria.

Diversas técnicas são consideradas apropriadas para solucionar muitos problemas de memória na simulação. Entre elas, duas delas possuem uma certa ênfase, uma, a Simulação Regenerativa, voltada para a atuação em sistemas de filas de redes de computadores onde a cada final de ciclo ou tomada de tempo, o sistema recomeça de uma outra forma probabilística. A combinação de Simulação de Evento de Erro (Error Event Simulation - EES) com Condicionamento (Conditioning) responsável por tratar outros problemas tais como a simulação de canais de comunicação de satélites com uma estrutura não linear. Tratando-se de redes ATM, a questão da complexidade de sistemas de filas é feita através da simulação regenerativa como uma técnica capaz de evitar o aparecimento de valores para o "bias" que não sejam apropriados que tornariam a simulação pobre e ineficiente. Técnicas regenerativas fazem com que a aplicação de Amostragem Importante torne-se viável e eficiente, além de serem indispensáveis para a correção de certos intervalos de confidência para as estimativas envolvidas. Entretanto, usando-se até mesmo os mais favoráveis parâmetros de Amostragem Importante muitas vezes tem-se longos ciclos de simulação quando não infinitos.

A idéia principal para a Simulação Regenerativa é o uso inicial, em cada regeneração de ciclo de simulação, a configuração dos parâmetros que levarão a uma estimativa detalhada com o máximo de eficiência e então alterá-los durante a

simulação de maneira que o sistema seja dirigido a regenerar-se o mais rápido possível.

Desta forma, os resultados de um Amostragem Importante ótima e a presença de pequenos ciclos de regeneração serão alcançados simultaneamente. Este tratamento é importante porque existem pontos na simulação no qual Amostragem Importante mostra-se ineficiente e deste modo, a alteração de parâmetros é imprescindível, e que isto seja feito de maneira suficientemente rápida para não comprometer a performance do simulador.

Além disso, existem dificuldades na utilização de métodos regenerativos. Uma delas tem haver com o comprimento de cada ciclo de regeneração potencial, especialmente para redes mais complexas. Outra dificuldade é a questão de eficiência de Amostragem Importante no esquema de "biasing" que em alguns casos, não há possibilidade de alteração de estatísticas observáveis de uma simulação em torno de um certo limite. Em outros, funções pesos são geradas com valores muito pequenos. Isto tudo pode levar a estimativa de Amostragem Importante a ser subestimada.

Isto é evidenciado por uma questão razoável. A eficiência da abordagem Amostragem Importante dependende considerávelmente da memória do sistema em que se encontra aplicada. Com o aumento desta memória ou dimensão das variáveis de entrada do sistema, é causado um aumento do número de eventos aleatórios que podem contribuir para a geração de um evento raro bem como do número de componentes que são combinados para produzirem um peso total afim de corrigir o estimador da probabilidade que se deseja encontrar. Contudo, todo este crescimento é considerado perigoso, pois, ao combinar muitos eventos aleatórios, isto pode causar um novo efeito, considerado dissolução do "bias". Este efeito é gerado pelo distanciamento evento raro causado pela alta combinação de parâmetros de entrada.

Ao tentar contornar estes problemas, a metodologia Amostragem Importante Dinâmica propõe-se a combinar a Amostragem Importante com a

simulação regenerativa. Esta metodologia baseia-se na observação que parâmetros de Amostragem Importante podem ser modificados durante um ciclo de regeneração dependentemente de uma considerada "estimação eficiente" ou "aceleração de regeneração" é desejada. Isto porque durante toda a simulação, haverá casos para mudanças de parâmetros e isto deve ser feito de maneira rápida.

Analiticamente, dado P^* como sendo uma matriz de transição de amostragem alternativa, com $P^*(s)$ sendo função de probabilidade. Tem-se como observável:

$$E_P[G(s)] = E_P{}^*[G(s)L^*(s)] \qquad (26)$$

onde $L^*(s) = P(s)/P^*(s)$, e $P^*(s)<>0$ se $G(s)P(s) <> 0$.

L^* é a taxa de probabilidade ou função peso. $E[f]$ é definido como:

$$\hat{E}_*[f] = \frac{\frac{1}{N}\sum_{k=1}^{N} H_k(s)L_k^*(s)}{\frac{1}{M}\sum_{k=1}^{M} G_k(s)L_k^*(s)} \qquad (27)$$

onde $\hat{E}^*[f]$ refere-se a uma estimativa de $E[f]$ usando aplicando-se Amostragem Importante. Dado:

$$H^*(s) = \sum_{i=0}^{t_1-1} h(X_i)L_i^*$$

e

$$G^*(s) = \sum_{i=0}^{t_1-1} g(X_i)L_i^* \qquad (28)$$

Logo, o estimador baseado em Amostragem Importante é:

$$\hat{E}_*[f] = \frac{\frac{1}{N}\Sigma_{k=1}^{N} H_k^*(s)}{\frac{1}{M}\Sigma_{k=1}^{M} G_k^*(s)} \qquad (29)$$

onde

$$L_{ik}^* = \frac{P(X_{ok},\ldots,X_{ik})}{P^*(X_{ok},\ldots,X_{ik})} \qquad (30)$$

A variância do estimador (29) é dada por $\sigma^2(P,P^*)$. Considera-se no entanto, uma Cadeia de Markov associada. Assume-se então que a distribuição inicial seja não modificada. Tem-se:

$$\frac{P(X_{ok},\ldots,X_{ik})}{P^*(X_{ok},\ldots,X_{ik})} = \frac{\prod_{j=0}^{i-1} p(X_{jk},X_{j+1,k})}{\prod_{j=0}^{i-1} p^*(X_{jk},X_{j+1,k})} \qquad (31)$$

onde $p(X_j,X_{j+1})$ é a probabilidade de transição da Cadeia de Markov.

A simulação depende de todas transições aleatórias que previamente ocorreram no mesmo ciclo de regeneração. Desta forma, verifica-se que a considerada "memória" de um sistema é incrementada dentro de cada ciclo de regeneração. Isto é considerado um efeito devastador fazendo com que ciclos sejam cada vez mais longos, portanto, evita-se tais efeitos com quebras adequadas na simulação.

A estrutura baseada em *(29)*, é considerada uma forma estática da aplicabilidade de Amostragem Importante onde a probabilidade de transição p^* modificada não depende de um estado X_i em um determinado momento i. Considera-se que, dentro de certas condições, a simulação de uma cadeia de Markov, a melhor simulação deve ser realizada com Amostragem Importante Dinâmica. Neste contexto, quando o uso desta nova abordagem, a probabilidade de transição modificada $p^*(X_j,X_{j+1})$ torna-se:

$$p^*_{X_j X_{j+1}}(X_j, X_{j+1}) \qquad (32)$$

onde o mesmo evidencia a dependência da probabilidade modificada em uma transição de estado determinada. Para a Amostragem Importante Dinâmica, a abordagem é demonstrada e justificada matematicamente da seguinte maneira. Esta nova forma de tratamento de Amostragem Importante fornece um equilíbrio considerável entre a estimativa não modificada e a função peso associada. Dado $E_P[H(s)]$:

$$\hat{E}_{P^*}[H^*(s)] = \frac{1}{N}\sum_{k=1}^{N}\sum_{i=0}^{t_1-1} h(X_{ik})L^*_{ik} \qquad (33)$$

Durante a fase do evento de erro do ciclo de regeneração, a taxa de probabilidade da trajetória do estado de um sistema observado, $\{X_i\}_{i\geq 0}$, torna-se inferior em relação à taxa de probabilidade da trajetória do estado dentro de uma medida não modificada. Contudo, a função peso decai dentro de cada ciclo de regeneração desde que o peso seja menor que *1*, e o efeito de eventos importantes sucessivos em uma estimativa cumulativa decai também.

Eventualmente, a função peso diminui e cada chegada bloqueada contribui de forma insignificante com o valor encontrado pelo estimador (33).

A função acumulativa peso para uma Cadeia de Markov discreta é da forma:

$$L_i^* = \prod_{j=0}^{i-1} \frac{p(X_j, X_{j+1})}{p^*_{X_j, X_{j+1}}(X_j, X_{j+1})} \qquad (34)$$

E este valor chega a zero quando $i \to \infty$ bem como $L^*_{ik} \to \infty$ e $i^2 L^*_{ik} \to \infty$.

Para exemplificar, basta verificar uma fila de um único servidor[23]. Considera-se uma fila com o comprimento K. Dado o número de células no sistema em um dado instante k, X_k, assume-se que o sistema passe por diversos estados, onde, em cada um deles, aparecerão dois tipos de ciclos a serem regenerados. Baseando-se no contexto que uma regeneração é realizada quando a fila está vazia, $X_k = 0$, o primeiro tipo de ciclo, ocorre quando o sistema, iniciando-se de um estado i, passe entre outros estados e retorne ao estado i sem estourar a fila $\{X_k < K\}$. O segundo, ocorre quando, ao realizar o mesmo percurso do primeiro caso, haja pelo menos um estouro $\{X_k = K\}$ durante o mesmo.

Em uma simulação normal, o primeiro caso será mais evidente. É conhecido que para uma fila do tipo M/M/1/K a taxa média de chegada é $1/\lambda$ e a taxa média de serviço é $1/\mu$. E em condições normais, a taxa média de serviço é maior que a taxa média de chegada possuindo um fator de utilização $p = \lambda/\mu$ onde $p << 1$ consequentemente a fila esvazia constantemente e o estouro seja considerado um evento raro. Contudo, a fazer o caso segundo mais evidente, basta trocar as taxas entre as médias de serviço e chegada e vice-versa utilizando-se a teoria dos Desvios Largos. Uma outra forma, e de certa maneira considerada balanceada, é aumentar suavemente a taxa média de chegada

diminuindo também na mesma proporção a taxa média de serviço. Dependendo da modificação, ter-se-á duas coisas, a primeira e esperada, a freqüência do estouro da fila $\{X_k > K\}$, a outra, largos ciclos regenerativos serão esperados já que $X_k = 0$ será mais difícil de acontecer.

Com relação a uma fator de utilização modificado ρ^*, se a relação enquadra-se em $(\rho < \rho^* < 1.0)$, o sistema irá esvaziar-se ainda com uma certa freqüência $\{X_k = 0\}$ embora com reduzida freqüência. Aumenta-se assim a possibilidade de estouro bem como o tamanho do ciclo. Para fatores largos $(\rho^* > 1.0)$ o comprimento de cada ciclo de regeneração cresce de forma impraticável.

Utilizando-se a metodologia Amostragem Importante Dinâmica, ao começar um ciclo de regeneração, utiliza-se um valor de ρ^* considerado ótimo ou próximo (utiliza-se técnicas para encontrar o melhor ρ^*) causando um aparecimento rápido do evento. Verifica-se que $h(X_{ik})L^*_{ik}$ converge em cada ciclo k. Observa-se um número finito de estouros (isto é levado em conta para uma simulação rápida) e em seguida, acelera-se a simulação à regeneração alterando-se ρ^* para um valor $\rho^* << 1$.

A prova de que $h(X_{ik})L^*_{ik}$ é demonstrado da seguinte forma. Dado $\{X_i\}_{i \geq \infty}$ ser uma Cadeia de Markov irredutível em um espaço finito ε dentro de uma matriz de transição P. Dado P^* a matriz de transição modificada. Tem-se:

$$L_i^* = \frac{P(X_0,\ldots,X_i)}{P^*(X_0,\ldots,X_i)} = \frac{\prod_{j=0}^{i-1} p(X_j,X_{j+1})}{\prod_{j=0}^{i-1} p^*(X_j,X_{j+1})} \qquad (35)$$

então, a menos que $p(.,.) = p^*(.,.)$, $lim_{i\to\infty} i^2 L^*_i = 0$.

Se $p(u,v)$ é insignificante quando $p^*(u,v)$ é positivo para qualquer estado $(u,v) \in \mathcal{E}$, o resultado é imediato desde que o espaço de estados finito e a irredutibilidade de uma Cadeia de Markov garantirá que o sistema passará pelo estado (u,v). Do contrário, observa-se que:

$$\lim_{i\to\infty} i^2 L_i^* = \lim_{i\to\infty} exp\left[\sum_{j=0}^{i-1} f(X_j, X_{j+1}) + 2\log i\right] \qquad (36)$$

onde

$$f(X_j, X_{j+1}) = log\left(\frac{p(X_j, X_{j+1})}{p^*(X_j, X_{j+1})}\right) \qquad (37)$$

contudo, desde que $\frac{2\log i}{i} \to 0$,

$$\frac{1}{i}\sum_{j=0}^{i-1} f(X_j, X_{j+1}) + \frac{2\log i}{i} \to \sum_{u,v} p(u)p^*(u,v)f(u,v) \qquad (38)$$

onde $(\pi(u) : u \in \mathcal{E})$ é a probabilidade estacionária de $p^*(.,.)$.

Pela rigorosa concavidade de $log(.)$,

$$\sum_{u,v} p(u)p^*(u,v)log\left(\frac{p(u,v)}{p^*(u,v)}\right) < log\left(\sum_{u,v} p(u)p(u,v)\right) = 0 \qquad (39)$$

desde que $p(u,v) \neq p^*(u,v)$. Por *(36)*,

$$\sum_{j=0}^{i-1} f(X_j, X_{j+1}) + 2\log i \to -\infty \qquad (40)$$

e, assim, $\lim i \to \infty \; i^2 L^*_i = 0$ por (38).

Isto prova que $\sum_{i=0}^{\infty} L_i^*$ converge e, desde que $0 \leq h(X_i) \leq 1$, $\sum_{i=0}^{M} h(X_i) L_i^*$ também converge em P^* como $M \to \infty$. A implicação prática para esta metodologia de simulação é um balanceamento entre fases de aceleração de regeneração de forma que a diferença entre os valores resultantes em dois instantes sucessivos de chegada onde houveram o estouro, M_1 e M_2 dentro de um k-ésimo ciclo de regeneração é menor que uma determinada tolerância ε.

$$0 < \sum_{i=0}^{M_2} h(X_{ik}) L_{ik}^* - \sum_{i=0}^{M_1} h(X_{ik}) L_{ik}^* = L_{M_2 k}^* < \varepsilon \qquad (41)$$

Para outras questões, uma outra técnica que aparece, contudo, voltada para outros sistemas que por sua vez apresentam memória, Simulação de Evento de Erro (Error Event Simulation - EES) e Condicionamento (Conditioning). Tais técnicas em uma combinação com a teoria dos Desvios Largos tem tido um grande impacto em cima da abordagem Amostragem Importante. Entre as áreas que podem ser mencionadas estão entre outras Sistemas TCM usando Receptores de Viterbi, Decodificadores de Viterbi e Canais de Comunicação de Satélites [36], onde estes casos apresentam memória além de terem uma certa complexidade inerente envolvida, quando não uma estrutura não linear.

Estes métodos podem ser descritos da seguinte forma. Dado $\{D_k\}$ sendo uma seqüência de dados com $D_k = \pm 1$ equiprovavelmente, e dado $\{Y_k\}$ sendo uma seqüência de uma variável aleatória Gaussiana de média zero.

O sistema é linear, e a decisão estatística do receptor é

$$v_k = \sum_{j=-m}^{m} a_j (D_{k-j} + Y_{k-j}) \qquad (42)$$

Dado $x_k = (D_{k-m}, \ldots, D_{k+m}, Y_{k-m}, \ldots, Y_{k+m})$ como entrada, a memória é dada por $M = 4m + 2$, e $v_k = g(x_k)$ dado por (42). Dado uma região de erro E descrita por:

$$E = \left\{ (d,y) : d_0 \sum_{j=-m}^{m} a_j (d_{-j} + y_{-j}) \leq 0 \right\} \qquad (43)$$

onde $d = (d_{-m}, \ldots, d_m)$ e $y = (y_{-m}, \ldots, y_m)$.

A probabilidade de erro é dada por $P_E = P(v_0 \geq | \; D_0 = -1)$. Em um fluxo de simulação tradicional, ao estimar P_E, uma longa seqüência $\{D_k, Y_k\}$ é gerada, e erros são checados para cada D_k. Em uma simulação de evento de erro, procede-se de uma forma muito diferente.

Dado um padrão de ruído de comunicação ISI (Intersymbol Interference) descrito como $D = (D_{-m}, \ldots, D_{-1}, D_1, \ldots, D_m)$ e $Y = (Y_{-m}, \ldots, Y_m)$, fixando-se $D = d$, então:

$$P_E = \sum_d P(v_0 \geq 0 | D_0 = -1, d) P(D = d)$$

$$= 2^{-2m} \sum_{d} P(v_0 \geq 0 | D_0 = -1,d) \qquad (44)$$

leva-se então ao estimador:

$$\hat{P}_E = 2^{-2m} \sum_{d} P(v_0 \geq 0 | D_0 = -1,d) \qquad (45)$$

uma vez que, cada evento de erro da forma $\{v_0 \geq 0 \mid D_0 = -1,d\}$ é estimado separadamente, e os resultados são combinados ao dar $\hat{P}_E$.

A principal vantagem é que eventos de erros são considerados subproblemas, e, se um subproblema é governado por um desvio largo, então o estimador pode ser encontrado facilmente em cada subproblema. A idéia básica desta junção de técnicas está na execução paralela da simulação Monte Carlo junta à simulação de Amostragem Importante, e após a um apropriado período de tempo, o estado do sistema Amostragem Importante é reiniciado ao igualar-se ao estado do sistema Monte Carlo, pois a memória é limitada a cada reinicio. Cada pedaço que foi reiniciado representa cada subproblema tratado, e, em nível de referência para o simulador de Amostragem Importante, a utilização do simulador Monte Carlo é de fundamental importância já que memórias em uma simulação tradicional Monte Carlo não são tão evidentes quanto a abordagem de Amostragem Importante em diversos casos. A essência principal desta abordagem está na limitação do tamanho da memória envolvida. Isto evita um problema bastante evidente, blocos de memória sobrepostos causam modificações imprecisas e ao impedir que isto aconteça, EES/Condicionamento isolam estes instantes através da quebra da simulação em pontos de reinicializações.

3.3. Teoria dos Desvios Largos

A Teoria dos Desvios Largos (Large Deviation) refere-se a uma coleção de técnicas para a estimação de propriedades de eventos raros, bem como suas freqüências e como isto pode acontecer. Considera-se a existência de uma grande lei chamada de "A Lei Forte de Eventos Raros". Isto significa que eventos raros, de uma maneira genérica, somente acontecem de uma única forma, precisamente, se existe um caminho único de menor custo que cause um evento raro, então o parâmetro assintótico obtido é largo e condicionado na ocorrência do evento raro. Com elevada probabilidade o sistema segue pelo caminho menos custoso em qualquer intervalo limitado em direção ao acontecimento do evento. A teoria dos Desvios Largos transforma problemas de probabilidade em problemas de otimização determinísticos além de envolver estimativas em cima de cada período de tempo finito, ou em cima de longos períodos de tempo considerando somente eventos que são funções de médias de amostras.

A Teoria dos Desvios Largos é uma ferramenta analítica ao medir a taxa que certas probabilidades tendem a zero. Além disso, esta técnica fornece um potencial bastante grande de oferecer ganhos computacionais na aplicação em diversas áreas, tais como, redes de filas, algoritmos estocásticos, cadeia de Markov, testes seqüenciais, sistemas de ondas de luz digitais (fibras óticas) e outros. Ademais, se a mesma pode ser aplicada, é considerada a mais poderosa abordagem em junção com a técnica Amostragem Importante em termos de simulação.

A Teoria dos Desvios Largos no estudo de redes estocásticas evidenciam duas grandes aplicações. A primeira refere-se a dualidade analisada em redes. Redes comutadas por circuito com algoritmos alternados de roteamento são usualmente consideradas duais. A teoria dos Desvios Largos fornece um caminho útil ao analisar e ganhar um intuitivo entendimento da dualidade.

A segunda aplicação da teoria dos Desvios Largos está em redes comutadoras de pacotes como ATM. Com já referenciado, taxas de erro em redes

ATM são da ordem de 10^{-9} à 10^{-12}. O significado destes valores indicam que em muitos aspectos operacionais de redes ATM necessitam que as taxas de erro sejam muito pequenas. Assim, o controle de admissão, a medição de buffers internos e a simulação de modelos ATM possuem domínios que vão de encontro com a característica de Desvios Largos. Ou seja, a Teoria dos Desvios Largos são aplicados em uma variada gama de tipos de eventos de erros, e, estes eventos são causados por um largo número de diferentes coisas acontecendo juntas, e com a teoria é possível identificar o melhor meio de encontrar-se o caminho menos custoso ao chegar ao evento dependendo do sistema envolvido.

A teoria dos Desvios Largos é uma abordagem voltada à demanda do sistema envolvido. Assim sendo, diversas formas da aplicabilidade desta técnica vem sido realizadas obtendo-se resultados consideráveis em diversos tipos de sistemas possuindo eles uma certa simplicidade como sistemas unidimensionais como por exemplo, a procura do valor ótimo para uma fila do tipo M/M/1[6],[18], ou até mesmo os mais complexos como sistemas multidimensionais, como por exemplo, o valor de modificação para uma matriz de transição de estados de uma fonte On-Off[37].

Por sua aplicabilidade, a teoria dos Desvios Largos é considerada uma eficiente ferramenta para a simulação. Esta abordagem, fornece uma otimalidade assintótica no limite dos eventos de interesse, os quais são considerados raros. A simulação é considerada ótima quando há a minimização da variância do estimador da probabilidade, o que por sua vez haja uma minimização da simulação. Ao se aplicar a técnica Amostragem Importante, ao se definir a função modificada, os parâmetros considerados ótimos ou próximo ao ótimo são previamente estabelecidos através de técnicas que podem estabelecer este tipo de análise e a teoria dos Desvios Largos apresenta-se como a principal devido a sua flexibilidade de emprego.

Considera-se uma seqüência de variáveis aleatórias $Y_1, Y_2, \ldots, Y_n$ convergindo em sua probabilidade à uma constante real y_0. tal qual:

$$Pr\{|Yn - y0| > \varepsilon\} \to 0 \qquad \text{com } n \to \infty \tag{46}$$

Existe o caso que (46) não somente converge à zero, mas realiza isto de forma rápida ou exponencial. Considera-se um intervalo ε sobre y, $(y, -\varepsilon, y+\varepsilon)$ que por sua vez não necessariamente contém o ponto de limite y_0, e então investiga a probabilidade $Pr\{ Y_n \in (y - \varepsilon, y + \varepsilon)\}$ com $n \to \infty$.

Muitas vezes vale a relação:

$$Pr\{ Y_n \in (y - \varepsilon, y + \varepsilon)\} \approx L(\varepsilon, y, n) \exp[-nI_0(y,\varepsilon)] \tag{47}$$

onde $L(.,.,.)$ é uma função de variação super lenta de n e $I_0(.,.)$ é uma função não negativa que fica zero quando $y = y_0$. Para uma valor fixo de $\varepsilon > 0$ e uma grande índice n, se $|Y_n - y_0| > \varepsilon$, então Y_n pode ser encarado como um grande desvio de um valor nominal y_0. Considera-se uma nova expressão de (47):

$$\frac{\log Pr\{Y_n \in (y - e, y + e)\}}{n} \approx \frac{\log L(e,y,n)}{n} - I_0(y,e) \tag{48}$$

dentro de uma geral condição, pode ser expressado como:

$$\lim_{n\to\infty} \frac{1}{n} \log Pr\{Y_n \in E\} = - \inf_{y \in E} I(y) \tag{49}$$

onde $E \in B(\Re)$ e $I(y)$ é uma função convexa não negativa chamada função taxa tal que $I(y_0) = 0$. Grande parte da teoria dos Desvios Largos é voltada a determinar

condições referentes a formação da função taxa $I(y)$. Sabe-se no entanto, que o valor de (49) Y_n é baseado na média de n valores escalares independentes e identicamente distribuídos, no entanto, Y_n pode ser descrito de forma vetorial dado que $E \in B(\mathfrak{R}^M)$. Os autores acrescentam que seqüências de diferentes espaços de probabilidades podem ser manipulados, e a normalização estabelecida por $1/n$ pode ser substituída por $1/a_n$ onde $\{a_n: n=1,2,...\}$ é a seqüência de números positivos tendendo para a infinidade.

Na aplicação da teoria de Desvios Largos constrói-se Y_n e E tal que (49) relata o problema da simulação. A escolha de Y_n e E pode muitas vezes ser descrito como $P(Y_n \in E)$ correspondendo a taxa de erro. Dentro de determinadas condições, pode ser descrito como:

$$I(y) = \sup_{q \in R} \{qy - m(q)\} \qquad (50)$$

onde $\mu(q) = lim_{n \to \infty} \mu_n(q)$ é suposto da forma:

$$m_n(n) = \frac{1}{n} \log E(\exp(nqY_n)) \qquad (51)$$

Deve-se notar a que categoria $\mu(q)$ pertence, ou seja, de forma a aplicar a teoria dos Desvios Largos, condições e valores associados devem ser identificados de maneira que se possa montar o problema em questão, a identificação do valor de $\mu(q)$ é um exemplo e uma tarefa necessária, saber se $\mu(q)$ existe ou é uma função convexa fechada ou diferenciável no interior do domínio de $\mu(q)$, e outros. A questão fundamental é assegurar que $\mu(.)$ é uma função considerada "bem comportada" além de $I(.)$ e E.

Exemplificando-se, dado $Y_n \approx N(0,1/n)$ e $E = [1,\infty)$, à partir deste valores, aplicar-se-á as fórmulas descritas acima. Considera-se que a função Y_n é Gaussiano e sua função de geração de momento é dada por:

$$E(exp(nqY_n)) = exp\left(\frac{n^2q^2n^{-1}}{2}\right) \tag{52}$$

uma vez que

$$I(y) = \sup_{q \in R}\left\{qy - \frac{q}{2}\right\} = \frac{y^2}{2} \tag{53}$$

desde que, de uma simples diferenciação, $qy-q^2/2$ é maximizado por $q=y$. Uma vez que a propriedade dos Desvios Largos da equação (49) é dada por:

$$\lim_{n\to\infty} \frac{1}{n} \log P\{X > 1\} = -\inf_{y \in (1,\infty)}\left\{\frac{y^2}{2}\right\} = -\frac{1}{2} \tag{54}$$

o que estabelece $P(X > 1) \approx e^{-(n/2)}$ para um n grande, e isto se adequa com uma aproximação bem conhecida de uma cauda Gaussiana. Para este caso, com valores de E e Y_n conhecidos, é possível com clareza obter-se o estimador, contudo, existem casos que estes valores são difíceis de estimar devido a complexidade do sistema envolvido, e sendo assim, pode-se estimar parâmetros não ótimos.

Existe um outro exemplo muito interessante na aplicação da teoria dos Desvios Largos. À partir de um único sistema, chega-se a variados tipos de Desvios Largos. Toma-se um grupo de n independentes e identicamente

distribuídos processos aleatórios $x_i(t)$. Cada processo pertence a classe da chamada categoria On-Off, onde o estado On é representado por $x_i(t) = 1$ e Off por $x_i(t) = 0$. Cada processo permanece em cada estado em um tempo exponencialmente distribuído antes de mudar o estado atual que se encontram. A média de tempo no estado On é $1/\mu$, e o tempo médio no estado Off é $1/\lambda$. Considera-se então $x_i(t)$ são variáveis aleatórias de Bernoulli com:

$$P_{ss}(x_i(t) = 1) = \frac{l}{l+m}$$

$$P_{ss}(x_i(t) = 0) = \frac{m}{l+m} \qquad (55)$$

contudo

$$E_{ss}(x_i(t) = 1) = \frac{l}{l+m} \qquad (56)$$

Interessa-se no número de origens que permanecem no estado On em qualquer momento. Este processo é verificado por:

$$S_n(t) \equiv \sum_{i=1}^{n} x_i(t) \qquad (57)$$

A forma considerada mais simples para a questão é:

$$R\left(\frac{x_1+\ldots+x_n}{n} \geq a\right) \qquad (58)$$

Dado agora variáveis aleatórias independentes e identicamente distribuídas $x_1, x_2, \ldots$, possuindo uma distribuição comum F, assume-se que a média $E(x_1)$ exista. Define-se:

$$M(\theta) = E[e^{\theta x 1}] \qquad (59)$$

$$I(a) = -\log(\inf_\theta e^{-\theta a} M(\theta))$$

$$I(a) = \sup_\theta(\theta a - \log M(\theta)) \qquad (60)$$

Note que $I(a)$ é não negativo e convexo e representa a função baseada no princípio dos Desvios Largos. A transformada aplicada ao *logM* em *(60)* é denominada de transformada convexa. Através da aplicação de diversos teoremas, pode-se chegar a taxas para algumas funções conhecidas:

$$\text{Normal}(\mu,\sigma) \qquad I(a) = \frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2$$

$$\text{Exponencial}(\lambda) \qquad I(a) = a\lambda - 1 - \log a\lambda$$

$$\text{Poisson}(\lambda) \qquad I(a) = a\log\frac{a}{\lambda} + \lambda - a \qquad (61)$$

A utilização da teoria dos Desvios Largos é bastante dependente do sistema envolvido como já fora dito, e, por sua vez, possui um tratamento matemático muito particular que, para alguns casos, representa um alto grau de complexidade em envolver-se com o caso. Por outro lado, quando encontra-se a função da taxa de probabilidade ideal para a aplicação na técnica de Amostragem Importante, os resultados são bastante satisfatórios.

Capítulo 4

Procedimentos de Geração de Tráfegos

Serão apresentados neste capítulo, procedimentos para a geração de tráfegos. A primeira parte está relacionada com sistemas Markovianos de modelagem de tráfegos. A segunda está baseada na característica de Auto-Semelhança do tráfego.

4.1 Geração de tráfegos markovianos

O tráfego consiste de chegadas de entidades discretas (pacotes) e pode ser matematicamente descrito como um processo pontual, consistindo de uma seqüência de instantes de chegada $T_1, T_2, \ldots, T_n, \ldots$ à partir de uma origem *0*. Convenciona-se, $T_0 = 0$. Existem duas formas equivalentes de descrever um processo pontual. O processo de contagem $\{N(t)\}_{n=0}^{\infty}$ subtende-se como um tempo contínuo, processo estocástico de valor inteiro não negativo, onde $N(t) = max\{n: T_n <= t\}$ é o número da chegada no intervalo *(0,t]*. O processo de tempo entre chegadas é uma seqüência aleatória $\{N(t)\}_{n=1}^{\infty}$ não negativa, onde $A_n = T_n - T_{n-1}$ sendo a duração do intervalo de tempo entre a n-ésima chegada e a n-ésima - 1. Assim sendo, tem-se:

$$\{ N(t) = n\} = \{ T_n <= t < T_{n+1}\} = \left\{ \sum_{1k=1}^{n} A_K \leq t \langle \sum_{k=1}^{n+1} A_k \right\}$$

onde $$T_n = \sum_{k=1}^{n} A_k \qquad (1)$$

Alguns tráfegos consistem de chegadas em grupo. Para descrevê-los, especifica-se uma seqüência aleatória não negativa $\{B_n\}_{n=1}^{\infty}$, onde B_n é um número aleatório correspondente ao tamanho do grupo.

A geração de tráfego pode ser representada por um algoritmo que gera números aleatórios a partir de um tempo $T_0 = 0$, sendo que cada evento de chegada é gerado aleatoriamente e colocado em uma lista de eventos.

Tráfegos de áudio são caracterizadas por possuírem dois períodos distintos. Isto porque em um áudio, existe som em um dado momento e em um outro não, havendo presença de pacotes quando há algum som, por exemplo, a voz humana, onde existe o período de fala, e um período de silêncio correspondendo a uma pausa em uma conversação.

Um tráfego de áudio é contínuo apresentando taxas constantes, o que se pode caracterizar como do tipo CBR(Constant Bit Rate). Contudo, quando se aplica alguma técnica de compactação, o tráfego de áudio muda a sua característica para VBR(Variable Bit Rate) ou quando ainda, se houver código para detecção de silêncio, será em rajadas.

A utilização de processos estocásticos denominados On-Offs, são os utilizados por capturarem com precisão o padrão de geração de células de uma fonte de voz, no caso, o período de áudio associado ao estado On e o período de silêncio, ao estado Off. Na verdade, On-Off é uma caso especial de um processo Markoviano Modulado com apenas 2 estados (On-Off).

Um tráfego de Vídeo é um tráfego continuo com taxa constante. Contudo, necessita-se para o mesmo uma banda larga de passagem devido a quantidade considerável de informação a ser trafegada. Isto obriga a utilização de técnicas que possam reduzir a demanda baseadas em compressão, ocasionando uma modificação na característica do tráfego aparecendo taxas variáveis de transmissão.

Apesar de tais técnicas de compressão caracterizarem um tráfego gerado como um tráfego com taxas variáveis, o sinal deve ser reproduzido no destino a uma taxa constante como já verificado com fontes de áudio. Geralmente tráfegos de vídeo vem acompanhado de tráfegos áudio [41].

Tráfegos de vídeo VBR que em recentes estudos, tem-se demonstrado o aparecimento novas formas de tráfegos baseadas características fractais ou auto-semelhantes[12],[16].

No nível do modelamento de tráfego, surge então duas formas de correlação: uma com dependência de pequeno alcance com o nível de atividade uniforme possuindo uma duração de centenas de milisegundos e uma outra com uma dependência de longo alcance correspondendo a mudanças no nível de atividade, durando de alguns segundos à horas.

Muitos modelos analíticos foram propostos para modelarem tráfegos de vídeo, chamando-se a atenção para modelos autoregressivos que capturam muito bem o padrão de transmissão durante uma cena, representando adequadamente a estrutura de autocorrelação do tráfego. Contudo, pecam ao tentarem prever o tempo entre chegadas de células.

Em fluxos de vídeo quando não houver mudança de cena, a taxa de transmissão é modelada como um processo contínuo de nascimento e morte no qual a taxa de transmissão varia apenas entre valores discretos. As taxas de transmissão e as taxas de transição do processo de Nascimento e Morte são computadas utilizando-se a autocorrelação. A modelagem de vídeo com mudança de cena é feita através do uso de uma cadeia de Markov bidimensional. Uma dimensão correspondendo a variações da taxa devido a mudança do nível de atividade numa cena e a outra dimensão On. Utiliza-se também, modelos Markovianos Modulados para a modelagem de fontes de vídeo onde os quais podem ser adequados à simulação de fontes de vídeo com a utilização de uma matriz de transição de dois estados para cada fonte.

Processo Markovianos Modulados podem ser descritos como aqueles que possuem uma taxa de chegada dependente do estado de uma cadeia de Markov

embutida, caracterizando-se por serem processos não renováveis onde os tempos de chegada são correlacionados. Oferecem a flexibilidade de se poder atribuir arbitrariamente diferentes valores para as taxas de chegada em cada estado bem como as taxas de transição entre os estados da cadeia de Markov embutida de maneira a imitar o comportamento do modelo de tráfego seja áudio, vídeo ou texto.

Um Processo Modulado de Markov é definido desta forma. Dado $M=\{M(t)\}^{\infty}_{t=0}$ ser um processo de Markov de tempo contínuo, com espaço de estado $\{1,2,...m\}$, assume-se que enquanto M é um estado k, a lei da probabilidade das chegadas de tráfego é completamente determinado por k, onde $1 <= k <= m$. Nota-se que quando M passa por uma transição, ou seja, a um estado j, então a nova lei de probabilidade para a chegada é determinada pelas condições de probabilidades efetuadas pelo estado j enquanto durar no mesmo, e assim por diante. Assim, a lei de probabilidade para chegadas é modulado pelo estado que M se encontra. O processo de modulação pode ser mais complicado na situação de modelamento de tempos discretos onde o tempo de delimitação não necessita ser restrito à variáveis aleatórias exponenciais.

Existem diversos modelos de processos Modulados de Markov em tempo discreto e contínuo, tais como um Processo de Poisson Modulado de Markov para o caso contínuo e um Processo de Bernoulli Modulado de Markov para o caso discreto.

Processos de Poisson foram bastante usados no passado. Poisson é um processo renovável onde o tempo entre chegadas $\{An\}$ é exponencialmente distribuído com taxa de parâmetros:

$$\lambda : P\{A_n \leq t\} = 1 - exp\ (-\lambda t) \qquad (2)$$

Equivalentemente, isso é um processo de contagem, satisfazendo:

$$P\{N(t) = n\} = \frac{exp(-\lambda t)(\lambda t)^2}{n!} \qquad (3)$$

onde o número de chegadas em intervalos disjuntos é estatisticamente independente.

Processos de Poisson possui propriedades analíticas elegantes. A primeira, a superposição de processo de Poisson resultam em um novo processo onde a taxa é a soma das taxas componentes. A segunda, o independente incremento da propriedade traz a Poisson um processo sem memória, simplificando filas de chegadas tipo Poisson. A terceira, são com justiça comuns em aplicações de tráfegos que fisicamente comprimem um grande número de fluxos de tráfegos independentes, cada qual pode ser bastante geral.

Processo Bernoulli, são processos análogos à Poisson só que em tempo discreto. A probabilidade do tempo de chegada em qualquer intervalo(slot) de tempo é p, independente de qualquer outro. Segue-se para o intervalo k, o correspondente número de chegadas do tipo binomial:

$$P\{N_k = n\} = \binom{k}{n} p^n (1-p)^{k-n} \qquad (4)$$

com n entre 0 e k.

O tempo entre chegadas é geométrica com parâmetro p:

$$P\{A_n = j\} = p(1-p)^j \qquad (5)$$

com j iniciando como um não negativo inteiro.

4.1.1 Simulação de uma fonte On-Off - Caso Contínuo

Considera-se uma única fonte On-Off com taxa de chegada para o estado On, λ. Estabelece-se uma matriz de transição de apenas 2 estados. O processo On-Off é um caso especial de um Processo Markoviano Modulado. A matriz de transição, pode ser representada da forma:

$$P_{2,2} = \begin{bmatrix} \theta & \alpha \\ \beta & \gamma \end{bmatrix}$$

Assume-se que quando a fonte encontrar-se no estado Off, não ocorrerá nenhuma chegada, no estado On haverá chegada de pacotes. Esta chegada é governada pela distribuição exponencial com uma taxa de chegada λ. Ao descrever-se uma cadeia de Markov para uma fonte On-Off, a mesma pode ser representada pela forma:

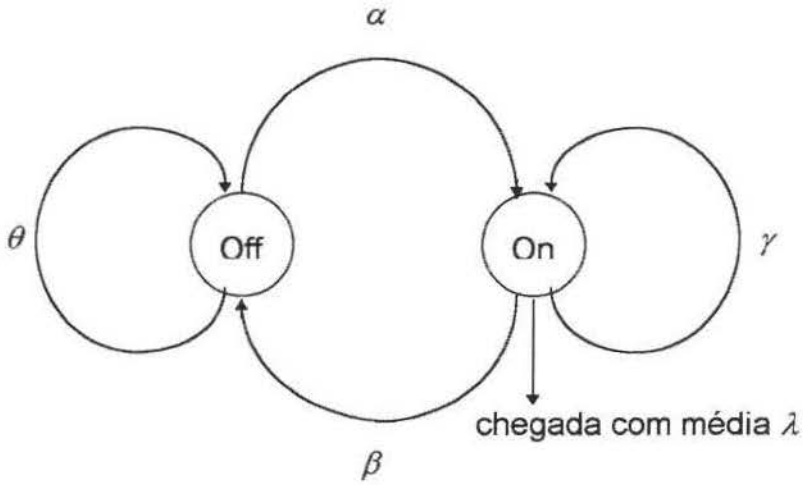


Figura 4.1:Cadeia de Markov - Fonte On-Off - Caso Contínuo

Chama-se a atenção da matriz de probabilidades, onde, cada linha representa a taxa de probabilidade de um determinado estado permanecer nele ou passar para outro. Para simular o comportamento de uma fonte On-Off de caso contínuo, basta seguir o tratamento adequado para a matriz de transição de estados. Um número aleatório é sorteado entre o intervalo *0* e *1* e, utilizando-se a distribuição de probabilidade acumulativa, é possível determinar-se se houve ou não transição de estados, por exemplo, dado a matriz de transição:

$$P_{2,2} = \begin{bmatrix} 0,3 & 0,7 \\ 0,5 & 0,5 \end{bmatrix}$$

apresentando uma taxa de chegada $\lambda = 0.6$ para o estado On. Soma-se cada linha, a formando dois vetores, tais:

$0 + 0,3 = 0,3$ e $0,3 + 0,7 = 1$ geram Off: $(0 ; 0,3 ; 1)$

e

$0 + 0,5 = 0,5$ e $0,5 + 0,5 = 1$ geram On: $(0 ; 0,5 ; 1)$

Com os dois vetores Off: $(0 ; 0,3 ; 1)$ e On: $(0 ; 0,5 ; 1)$ durante a simulação, quando no estado Off, se o valor sorteado estiver entre $[0$ e $0,3]$, permanecerá inalterado o estado, caso o valor estiver entre $[0,3$ e $1]$, haverá mudança de estado para On. O mecanismo também é válido estando-se inicialmente no estado On. No entanto, em On, haverá chegadas de pacotes baseada na taxa de chegada 0.6 utilizando-se uma saída exponencialmente distribuída.

4.1.2 Simulação de uma fonte On-Off - Caso Discreto

Dado uma única fonte On-Off, com uma matriz de transição de 2 estados:

$$P_{2,2} = \begin{bmatrix} \theta & \alpha \\ \beta & \gamma \end{bmatrix}$$

Como o caso é discreto, o que significa que a distribuição das amostras de chegadas e mudanças de estados é considerada binomial, ou seja, tem-se no entanto, um caso particular de um Processo de Bernoulli Modulado de Markov. Considera-se portanto, uma taxa de probabilidade ρ de haver uma chegada em um determinado tempo t. O tempo t é uma variável inteira que acrescida estabelecendo-se assim, cada ciclo da simulação. Assume-se também que o estado Off e o estado On.

A matriz de transição é idêntica ao caso contínuo, todavia, chama-se a atenção para o processo de chegada. Dado a probabilidade de haver chegada μ,

haverá sempre um chegada quando o valor de ρ for maior que a amostra compreendida entre o intervalo *0* e *1* que é retirada de uma função geradora de números aleatórios.

Para simular o comportamento de uma fonte On-Off de caso discreto, o tratamento para a matriz de transição de estados é o mesmo do caso contínuo. A cadeia de Markov que representa este processo é mostrada da forma:

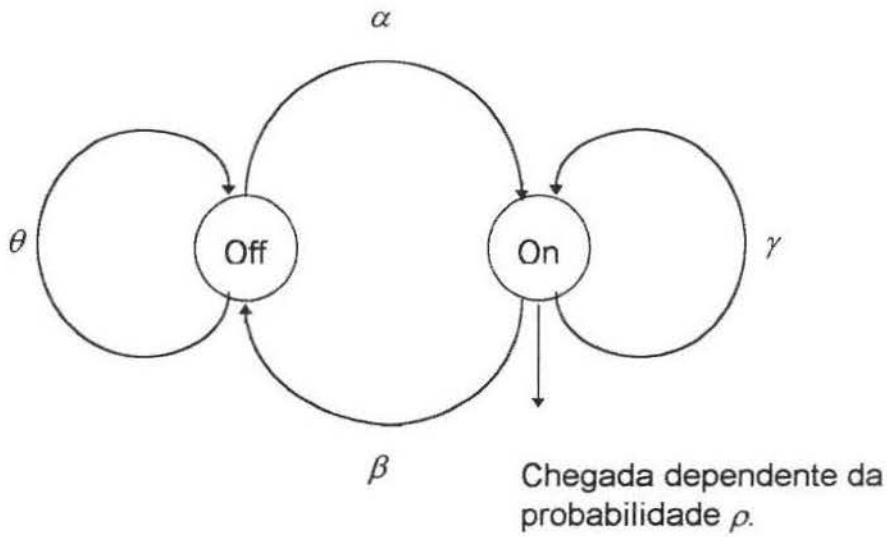


Figura 4.2:Cadeia de Markov - Fonte On-Off - Caso Discreto

Um número aleatório é sorteado entre o intervalo *0* e *1* e, utilizando-se a distribuição de probabilidade acumulativa, é possível determinar-se se houve ou não transição de estados, por exemplo, dado a matriz de transição:

$$P_{2,2} = \begin{bmatrix} 0,4 & 0,6 \\ 0,3 & 0,7 \end{bmatrix}$$

Soma-se cada linha, a formando dois vetores, tais:

0 + 0,4 = 0,4 e 0,4 + 0,6= 1 geram Off: *(0 ; 0,4 ; 1)*

e

0 + 0,3 = 0,3 e 0,3 + 0,7 = 1 geram On:*(0 ; 0,3 ; 1)*

Entretanto, para as chegadas, há uma mudança. Com uma taxa de probabilidade $\rho = 0,5$ e um conjunto de amostras retiradas ao acaso $A = \{0,35 ; 0,76 ; 0,82 ; 0,41 ; 0,4\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,5 > 0,35 ; 0,5 > 0,41 ; 0,5 > 0,4\}$.

4.1.3 Simulação de várias fontes On-Off's - Caso Contínuo

Dado um conjunto de fontes On-Off's $F_1, F_2, \ldots, F_n$. Assume-se que a taxa de chegada para o estado On para cada fonte é dada pelo vetor: λ: $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, taxas de mudança μ: $\{\mu_1, \mu_2, \ldots, \mu_n\}$, e que todas as fontes tenham matrizes de transição de apenas 2 estados:

$$MF_1 = \begin{bmatrix} \theta_1 & \alpha_1 \\ \beta_1 & \gamma_1 \end{bmatrix}, MF_2 = \begin{bmatrix} \theta_2 & \alpha_2 \\ \beta_2 & \gamma_2 \end{bmatrix}, \ldots, MF_n = \begin{bmatrix} \theta_n & \alpha_n \\ \beta_n & \gamma_n \end{bmatrix}$$

Também, para cada fonte, a cadeia de Markov que representa este processo é mostrada da forma:

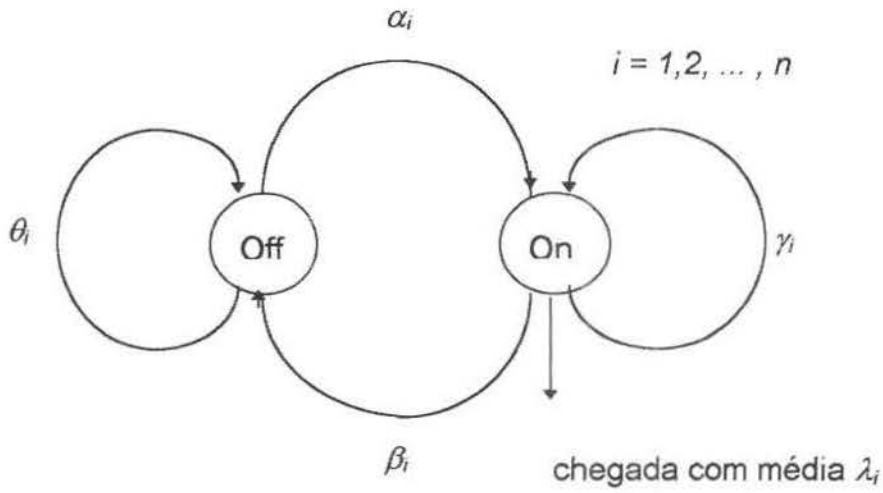


Figura 4.3:Cadeia de Markov - N Fontes On-Off - Caso Contínuo

E, o seu processo de nascimento e morte resultante da agregação de *N* fontes On-Off's Idênticas representando o número de fontes ativas é modelado da forma:

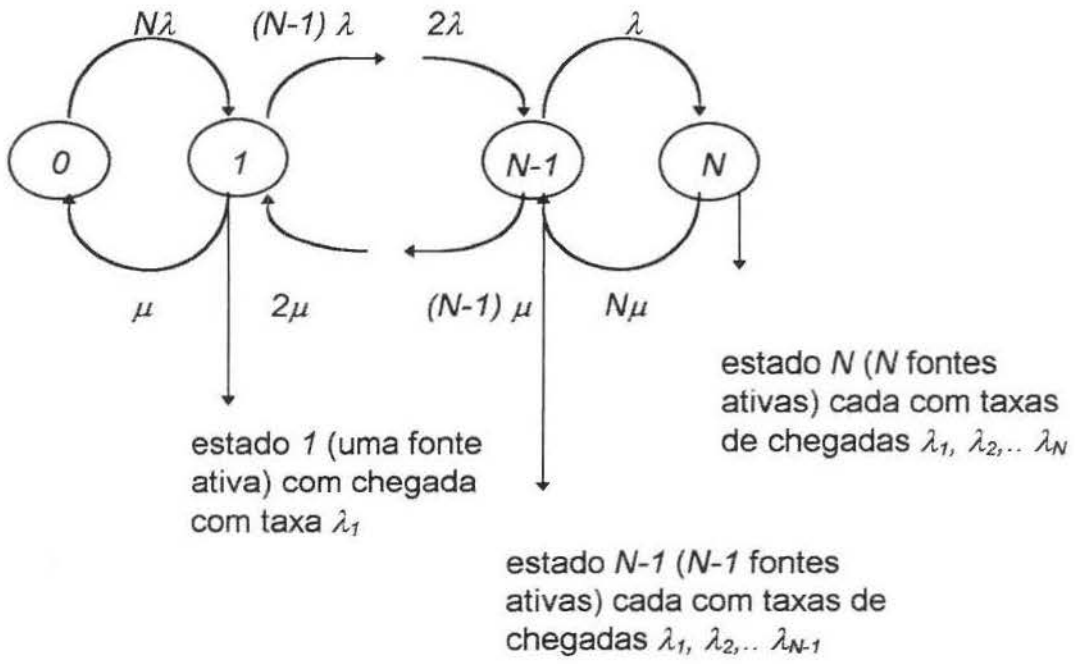


Figura 4.4:Nascimento e Morte - N Fontes On-Off - Caso Contínuo

Por exemplo, para se simular a transição de estado de cada fonte, o processo é idêntico ao caso de fonte única. A única diferença está em verificar-se se cada fonte está pronta para enviar célula, ou seja, em estado On. Na mudança de estado, sorteia-se um número aleatório entre *0* e *1* e utiliza-se a distribuição de probabilidade acumulativa para se determinar se houve ou não transição. Dadas 4 matrizes de transição para 4 fontes:

$$F_1P_{2,2} = \begin{bmatrix} 0,3 & 0,7 \\ 0,5 & 0,5 \end{bmatrix} \qquad F_2P_{2,2} = \begin{bmatrix} 0,2 & 0,8 \\ 0,6 & 0,4 \end{bmatrix}$$

$$F_3P_{2,2} = \begin{bmatrix} 0,9 & 0,1 \\ 0,3 & 0,7 \end{bmatrix} \qquad F_4P_{2,2} = \begin{bmatrix} 0,5 & 0,5 \\ 0,4 & 0,6 \end{bmatrix}$$

Cada fonte com taxas de chegadas respectivamente $\lambda_1 = 0,6$; $\lambda_2 = 0,3$; $\lambda_3 = 0,5$ e $\lambda_4 = 0,7$ quando encontram-se no estado On. Gera-se então vetores contendo as somas de cada linha, criando-se assim a estrutura de probabilidades acumulativas para cada fonte, tais:

Para Fonte1:

0 + 0,3 = 0,3 e 0,3 + 0,7= 1 geram Off: *(0 ; 0,3 ; 1)*

e

0 + 0,5 = 0,5 e 0,5 + 0,5 = 1 geram On:*(0; 0,5 ; 1)*

Para Fonte2:

0 + 0,2 = 0,2 e 0,2 + 0,8 = 1 geram Off: *(0 ; 0,2 ; 1)*

e

0 + 0,6 = 0,6 e 0,6 + 0,4 = 1 geram On:*(0 ; 0,6 ; 1)*

Para Fonte3:

0 + 0,9 = 0,9 e 0,9 + 0,1 = 1 geram Off: *(0 ; 0,9 ; 1)*

e

0 + 0,3 = 0,3 e 0,3 + 0,7 = 1 geram On:*(0 ; 0,3 ; 1)*

Para Fonte4:

0 + 0,5 = 0,5 e 0,5 + 0,5 = 1 geram Off: *(0 ; 0,5 ; 1)*

e

0 + 0,4 = 0,4 e 0,4 + 0,6 = 1 geram On:*(0 ; 0,4 ; 1)*

As 4 fontes procedem da mesma forma. Durante a simulação, os vetores encontrados para cada fonte servem para conduzir durante a simulação, quando no estado Off, para no caso a fonte4:*(0 ; 0,5 ; 1)* se o valor sorteado estiver entre *0* e *0.5*, permanecerá inalterado o estado, caso o valor estiver entre *0,5* e *1*, haverá mudança de estado para On. O mecanismo também é válido estando-se inicialmente no estado On. No entanto, em On, haverá chegadas de pacotes baseada na taxa de chegada $\lambda_4 = 0,7$ utilizando-se uma saída exponencialmente distribuída.

4.1.4 Simulação de várias fontes On-Off's - Caso Discreto

Dado um conjunto de fontes On-Off's $F_1, F_2, ..., F_n$. Assume-se um vetor de taxas de probabilidades de haver chegada ρ: $\{\rho_1, \rho_2, ..., \rho_n\}$. Assume-se também, para cada fonte, uma matriz de transição de estados, dadas:

$$MF_1 = \begin{bmatrix} \theta_1 & \alpha_1 \\ \beta_1 & \gamma_1 \end{bmatrix}, MF_2 = \begin{bmatrix} \theta_2 & \alpha_2 \\ \beta_2 & \gamma_2 \end{bmatrix}, ..., MF_n = \begin{bmatrix} \theta_n & \alpha_n \\ \beta_n & \gamma_n \end{bmatrix}$$

Este modelo é bastante utilizado quando se deseja caracterizar um tráfego com agregação de fontes idênticas como no caso de tráfegos VBR. A cadeia de Markov que representa este processo para cada fonte, é mostrada da forma:

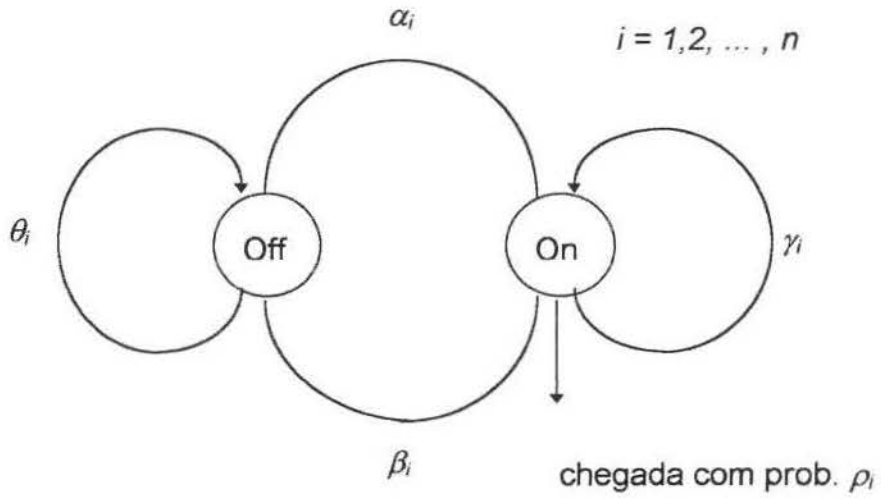


Figura 4.5:Cadeia de Markov - N Fontes On-Off - Caso Discreto

E, o seu processo de nascimento e morte resultante da agregação de *N* fontes On-Off's Idênticas representando o número de fontes ativas é modelado da forma:

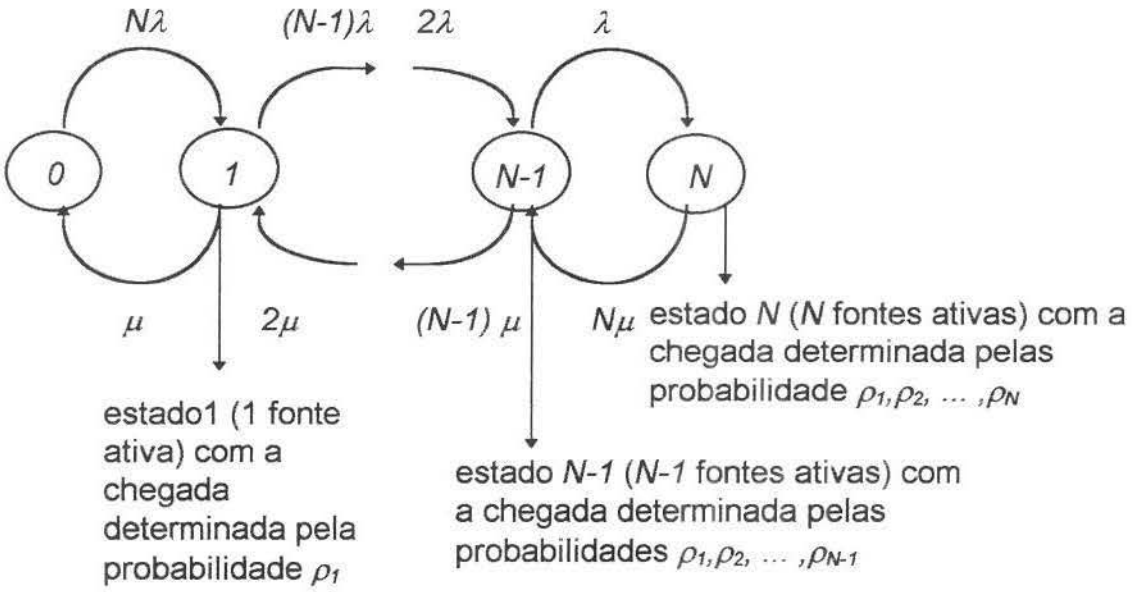


Figura 4.6:Nascimento e Morte - N Fontes On-Off - Caso Discreto

As matrizes de transição bem como os vetores de probabilidades são da mesma forma que para uma fonte única. Dadas 4 matrizes de transição para 4 fontes:

$$F_1P_{2,2} = \begin{bmatrix} 0,5 & 0,5 \\ 0,6 & 0,4 \end{bmatrix} \qquad F_2P_{2,2} = \begin{bmatrix} 0,3 & 0,7 \\ 0,4 & 0,6 \end{bmatrix}$$

$$F_3P_{2,2} = \begin{bmatrix} 0,2 & 0,8 \\ 0,5 & 0,5 \end{bmatrix} \qquad F_4P_{2,2} = \begin{bmatrix} 0,7 & 0,3 \\ 0,3 & 0,7 \end{bmatrix}$$

Gera-se então vetores contendo as somas de cada linha, criando-se assim a estrutura de probabilidades acumulativas para cada fonte, tais:

Para Fonte1:

$0 + 0,5 = 0,5$ e $0,5 + 0,5 = 1$ geram Off: $(0 ; 0,5 ; 1)$

e

$0 + 0,6 = 0,6$ e $0,6 + 0,4 = 1$ geram On:$(0 ; 0,6 ; 1)$

Para Fonte2:

$0 + 0,3 = 0,3$ e $0,3 + 0,7 = 1$ geram Off: $(0 ; 0,3 ; 1)$

e

$0 + 0,4 = 0,4$ e $0,4 + 0,6 = 1$ geram On:$(0 ; 0,4 ; 1)$

Para Fonte3:

$0 + 0,2 = 0,2$ e $0,2 + 0,8 = 1$ geram Off: $(0 ; 0,2 ; 1)$

e

$0 + 0,5 = 0,5$ e $0,5 + 0,5 = 1$ geram On:$(0 ; 0,5 ; 1)$

Para Fonte4:

$0 + 0,7 = 0,7$ e $0,7 + 0,3 = 1$ geram Off: $(0 ; 0,7 ; 1)$

e

$0 + 0,3 = 0,3$ e $0,3 + 0,7 = 1$ geram On:$(0 ; 0,3 ; 1)$

O processo de mudança de estado, é idêntico ao caso contínuo. Para o estado On de cada fonte, a geração de células é baseada no sorteio de valores aleatórios entre 0 e 1, e, estes valores devem exceder ao valor da probabilidade para que haja geração de células. Dadas taxas de probabilidades de cada fonte respectivamente $\rho_1 = 0,5$; $\rho_2 = 0,3$; $\rho_3 = 0,6$; e $\rho_4 = 0,2$.

Dado um conjunto de amostras retiradas ao acaso para cada fonte:

Fonte1:

$A = \{0,35 ; 0,76 ; 0,82 ; 0,41 ; 0,4\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,5 > 0,35 ; 0,5 > 0,41 ; 0,5 > 0,4\}$.

Fonte2:

$A = \{0,4 ; 0,55 ; 0,2 ; 0,9 ; 0,23\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,3 > 0,2 ; 0,3 > 0,23\}$.

Fonte3:

$A = \{0,53 ; 0,69 ; 0,80 ; 0,45 ; 0,25\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,6 > 0,53 ; 0,6 > 0,45 ; 0,6 > 0,25\}$.

Fonte4:

$A = \{0,35 ; 0,9 ; 0,7 ; 0,5 ; 0,1\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,2 > 0,1\}$.

4.1.5 Processo Markoviano Modulado de Poisson - 1 Fonte

Um Processo Markoviano Modulado de Poisson caracteriza-se por possuir uma matriz embutida $P_{i,j}$ representando cada estado e a transição entre os mesmos.

$$P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix}$$

Em um Processo Markoviano Modulado de Poisson se o estado do sistema é *i*, então as chegadas ocorrem de acordo com um processo de Poisson com taxa λ_i com *i=0,1,...N*. A cadeia de Markov associada ao processo para uma fonte contendo N estados:

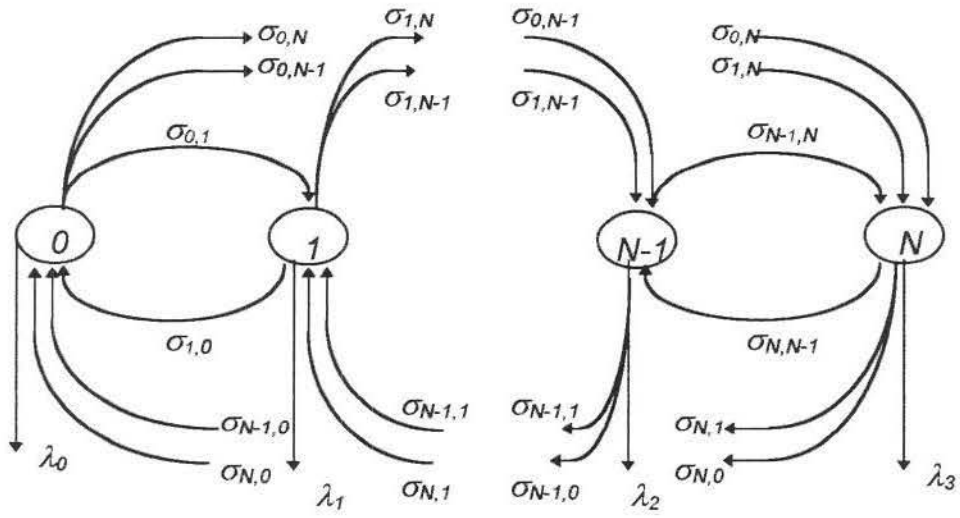


Figura 4.7:Cadeia de Markov - Processo Markoviano Modulado de Poisson

A dimensão de uma fonte para este tipo de processo é bem maior, ou seja, *NxN* , como já referenciado, *2x2* é um caso especial como as fontes On-Offs. Pode-se exemplificar sua simulação da seguinte forma. Dada a matriz de transição de estados de uma fonte única com 4 estados:

$$P_{4,4} = \begin{bmatrix} 0,2 & 0,2 & 0,2 & 0,4 \\ 0,1 & 0,5 & 0,3 & 0,1 \\ 0,3 & 0,4 & 0,2 & 0,1 \\ 0,7 & 0,1 & 0,1 & 0,1 \end{bmatrix}$$

Diferentemente de uma fonte On-Off, todos os estados geram células. Logo, respectivamente para todos os estados, as taxas de chegada para a distribuição exponencial são $\lambda_0 = 0,3$; $\lambda_1 = 0,6$; $\lambda_2 = 0,5$ e $\lambda_3 = 0,8$. Obtém-se então uma nova matriz contendo as somas de cada linha, criando-se assim a estrutura de probabilidades acumulativas para a fonte:

0 + 0,2 = 0,2 ; 0,2 + 0,2 = 0,4 ; 0,4 + 0,2 = 0,6 e 0,6 + 0,4 = 1

0 + 0,1 = 0,1 ; 0,1 + 0,5 = 0,6 ; 0,6 + 0,3 = 0,9 e 0,9 + 0,1 = 1

0 + 0,3 = 0,3 ; 0,3 + 0,4 = 0,7 ; 0,7 + 0,2 = 0,9 e 0,9 + 0,1 = 1

0 + 0,7 = 0,7 ; 0,7 + 0,1 = 0,8 ; 0,8 + 0,1 = 0,9 e 0,9 + 0,1 = 1

gera-se:

estado 0: *(0 ; 0,2 ; 0,4 ; 0,6 ; 1)*

estado 1: *(0 ; 0,1 ; 0,6 ; 0,9 ; 1)*

estado 2: *(0 ; 0,3 ; 0,7 ; 0,9 ; 1)*

estado 3: *(0 ; 0,7 ; 0,8 ; 0,9 ; 1)*

Sorteia-se um valor entre *0* e *1*. Se o estado atual é o *2*, com o seu vetor de probabilidades: *(0 ; 0,3 ; 0,7 ; 0,9 ; 1)*. Este valor, se estiver entre (*0* e *0,3)*, o estado mudará para *estado = 0* com geração de célula com taxa *(λ_0 = 0,3)*, se o valor estiver entre *(0,3 e 0,7)* o estado mudará para *estado = 1* com geração de célula com taxa *(λ_1 = 0,6)*, se o valor estiver entre *(0,7 e 0,9)* ele permanecerá no *estado = 2* com geração de célula com taxa *(λ_2 = 0,5)* e se o valor estiver entre *(0,9 e 1)*, haverá mudança para o *estado = 3* com geração de célula com taxa *(λ_3 = 0,8)*.

4.1.6 Processo Markoviano Modulado de Bernoulli - 1 Fonte

Um Processo Markoviano Modulado de Bernoulli caracteriza-se por possuir uma matriz embutida $P_{i,j}$ representando cada estado, onde a mesma determina a transição entre os estados.

$$P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix}$$

A diferença encontra-se na chegada de pacotes, para um determinado intervalo *k*, o número de chegadas pode ser avaliado através de uma distribuição binomial:

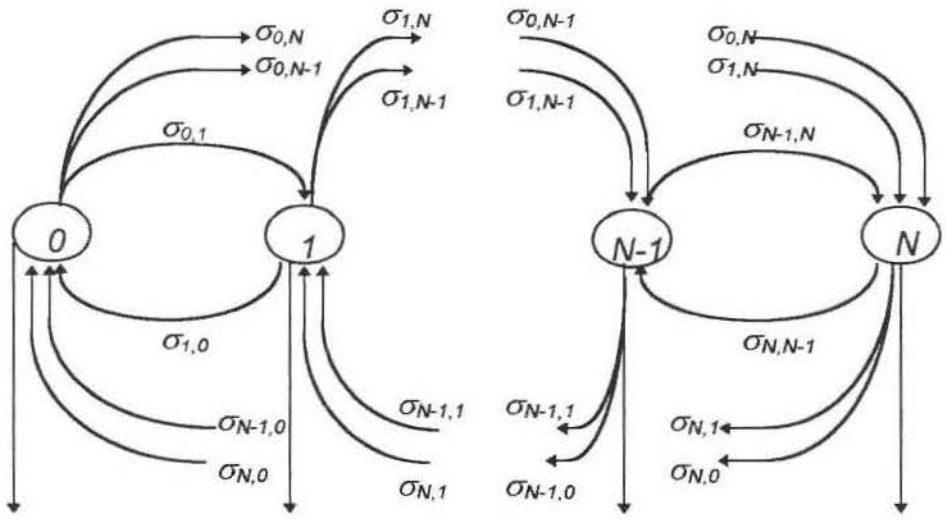
$$P\{N_k = n\} = \binom{k}{n} p^n (1-p)^{k-n} \qquad (6)$$

com *n* entre *0* e *k*. E o tempo entre as chegadas pode ser visto como uma distribuição geométrica com parâmetro *p*:

$$P\{A_n = j\} = p(1-p)^j \qquad (7)$$

com *j* iniciando como um não negativo inteiro.

A cadeia de Markov associada a este processo é demonstrada da seguinte forma:



Chegadas de acordo com as probabilidades $\rho_0, \rho_1, \ldots, \rho_{N-1}, \rho_N$

Figura 4.8:Cadeia de Markov - Processo Markoviano Modulado de Bernoulli

A dimensão de uma fonte para este tipo de processo também é bem maior, *NxN*. Pode-se exemplificar sua simulação da seguinte forma. Dada a matriz de transição de estados de uma fonte única com 4 estados:

$$P_{4,4} = \begin{bmatrix} 0,6 & 0,2 & 0,1 & 0,1 \\ 0,2 & 0,4 & 0,3 & 0,1 \\ 0,4 & 0,4 & 0,1 & 0,1 \\ 0,1 & 0,1 & 0,4 & 0,4 \end{bmatrix}$$

O processo de mudança de estados é idêntico ao Processo Markoviano Modulado de Poisson, por exemplo, à partir da matriz de transição original, obtém-se uma nova matriz contendo as somas de cada linha, criando-se assim a estrutura de probabilidades acumulativas para a fonte:

$0 + 0,6 = 0,6$; $0,6 + 0,2 = 0,8$; $0,8 + 0,1 = 0,9$ e $0,9 + 0,1 = 1$

$0 + 0,2 = 0,2$; $0,2 + 0,4 = 0,6$; $0,6 + 0,3 = 0,9$ e $0,9 + 0,1 = 1$

$0 + 0,4 = 0,4$; $0,4 + 0,4 = 0,8$; $0,8 + 0,1 = 0,9$ e $0,9 + 0,1 = 1$

$0 + 0,1 = 0,1$; $0,1 + 0,1 = 0,2$; $0,2 + 0,4 = 0,6$ e $0,6 + 0,4 = 1$

gera-se:

estado 0: $(0 ; 0,6 ; 0,8 ; 0,9 ; 1)$

estado 1: $(0 ; 0,2 ; 0,6 ; 0,9 ; 1)$

estado 2: $(0 ; 0,4 ; 0,8 ; 0,9 ; 1)$

estado 3: $(0 ; 0,1 ; 0,2 ; 0,6 ; 1)$

Também com todos os estados gerando células com um porém, células são geradas dependentemente da probabilidade deste acontecimento associada com cada estado, ou seja, é baseada no sorteio de valores aleatórios entre *0* e *1*, e, estes valores devem exceder ao valor da probabilidade para que haja geração de células. Por exemplo, dadas taxas de probabilidades de cada estado respectivamente $\rho_0 = 0,3$; $\rho_1 = 0,4$; $\rho_2 = 0,7$; e $\rho_3 = 0,1$.

Dado um conjunto de amostras retiradas ao acaso para cada estado:

Estado1:

$A = \{0,1 ; 0,3 ; 0,6 ; 0,32 ; 0,8\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,3 > 0,1\}$.

Estado2:

$A = \{0,2 ; 0,6 ; 0,5 ; 0,1 ; 0,9\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,4 > 0,2 ; 0,4 > 0,1\}$.

Estado3:

$A = \{0,5 ; 0,9 ; 0,8 ; 0,3 ; 0,7\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,7 > 0,58 ; 0,7 > 0,3\}$.

Estado4:

$A = \{0,03 ; 0,9 ; 0,7 ; 0,5 ; 0,1\}$. Haverá chegada para os casos onde μ for maior ou seja, $\{0,1 > 0,03\}$.

4.1.7 Processo Markoviano Modulado de Poisson - N Fontes

Considera-se agora um conjunto de fontes $F: \{F_1, F_2, ..., F_n\}$. Para cada fonte, está associado um Processo Markoviano Modulado de Poisson com uma matriz de transição de estados :

$$F_1P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix}, \qquad F_2P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix},$$

$$. \quad . \quad . \qquad F_nP_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix}$$

Considera-se também, vetores que representam as taxas de chegadas e de mudança de estados de cada fonte, respectivamente:

$$F_1\lambda: \{\lambda_0, \lambda_1, \ldots, \lambda_n\}, F_2\lambda: \{\lambda_0, \lambda_1, \ldots, \lambda_n\}, \ldots, F_n\lambda: \{\lambda_0, \lambda_1, \ldots, \lambda_n\}$$

e (8)

$$F_1\mu: \{\mu_0, \mu_1, \ldots, \mu_n\}, F_2\mu: \{\mu_0, \mu_1, \ldots, \mu_n\}, \ldots, F_n\mu: \{\mu_0, \mu_1, \ldots, \mu_n\}$$

Para uma fonte F_k onde $k = 1,\ldots,n$, se F_k estiver no estado i as chegadas ocorrem segundo um processo de Poisson com taxa $\lambda_{i,k}$. A cadeia de Markov associada ao processo para cada uma das fontes contendo N estados:

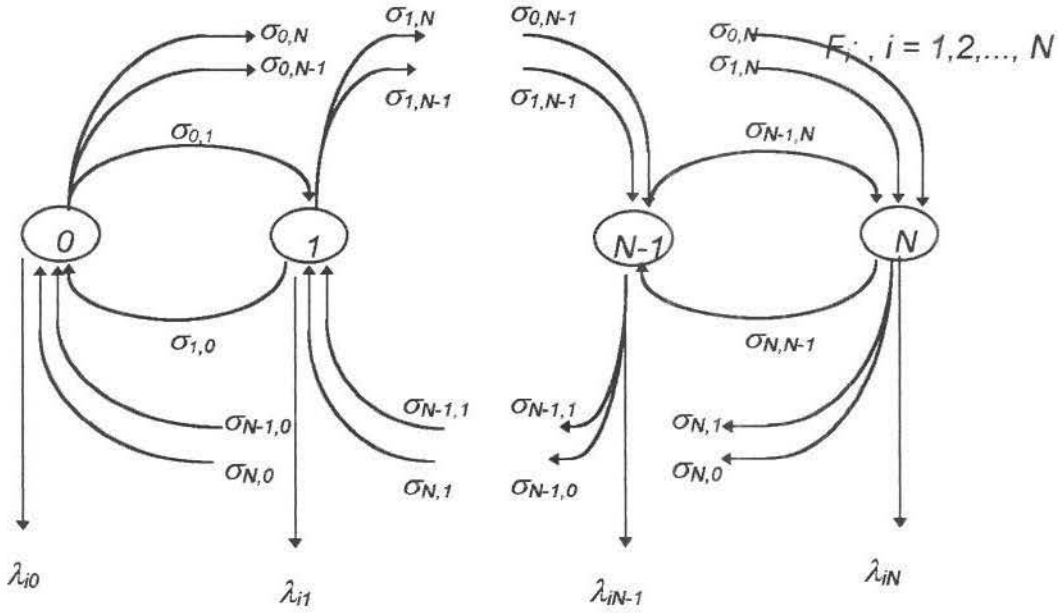


Figura 4.9:Cadeia de Markov - Processo Markoviano Modulado de Poisson - N fontes

Como uma extensão do Processo Markoviano Modulado de Poisson, agora para M Fontes e uma dimensão NxN. Pode-se exemplificar sua simulação da seguinte forma. Dado 2 Fontes:

Para a Fonte1:

$$F_1P_{4,4} = \begin{bmatrix} 0,2 & 0,3 & 0,2 & 0,3 \\ 0,3 & 0,3 & 0,3 & 0,1 \\ 0,2 & 0,4 & 0,2 & 0,2 \\ 0,6 & 0,1 & 0,1 & 0,2 \end{bmatrix}$$

Para a Fonte2:

$$F_2P_{2,2} = \begin{bmatrix} 0,5 & 0,5 \\ 0,6 & 0,4 \end{bmatrix}$$

Respectivamente para as duas fontes, tem-se $F_1\lambda$:{0,2 ; 0,5 ; 0,6 ; 0,8} e $F_2\lambda$:{0,5 ; 0,4} como vetores de suas taxas de chegada exponencial. Para cada fonte, novas matrizes transformadas da forma:

Fonte1:

$0 + 0,2 = 0,2$; $0,2 + 0,3 = 0,5$; $0,5 + 0,2 = 0,7$ e $0,7 + 0,3 = 1$

$0 + 0,3 = 0,3$; $0,3 + 0,3 = 0,6$; $0,6 + 0,3 = 0,9$ e $0,9 + 0,1 = 1$

$0 + 0,2 = 0,2$; $0,2 + 0,4 = 0,6$; $0,6 + 0,2 = 0,8$ e $0,8 + 0,2 = 1$

$0 + 0,6 = 0,6$; $0,6 + 0,1 = 0,7$; $0,7 + 0,1 = 0,8$ e $0,8 + 0,1 = 1$

gera-se:

estado 0: (0 ; 0,2 ; 0,5 ; 0,7 ; 1)

estado 1: (0 ; 0,3 ; 0,6 ; 0,9 ; 1)

estado 2: (0 ; 0,2 ; 0,6 ; 0,8 ; 1)

estado 3: (0 ; 0,6 ; 0,7 ; 0,8 ; 1)

e

Fonte2:

$0 + 0,5 = 0,5$ e $0,5 + 0,5 = 1$

$0 + 0,6 = 0,6$ e $0,6 + 0,4 = 1$

gera-se:

estado 0: *(0 ; 0,5 ; 1)*

estado 1: *(0 ; 0,6 ; 1)*

O processo de execução é idêntico ao caso com uma única fonte e os demais casos estudados de forma contínua, com mudanças de estados e a cada estado havendo geração de células de acordo com as suas respectivas taxas de chegadas.

4.1.8 Processo Markoviano Modulado de Bernoulli - N Fontes

Considera-se novamente um conjunto de fontes F: $\{F_1, F_2, ..., F_n\}$. Para cada fonte, está associado um Processo Markoviano Modulado de Bernoulli com uma matriz de transição de estados :

$$F_1P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix}, \qquad F_2P_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix},$$

$$. \quad . \quad . \qquad F_nP_{i,j} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & .. & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & .. & \sigma_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{n0} & \sigma_{n1} & .. & \sigma_{nn} \end{bmatrix}$$

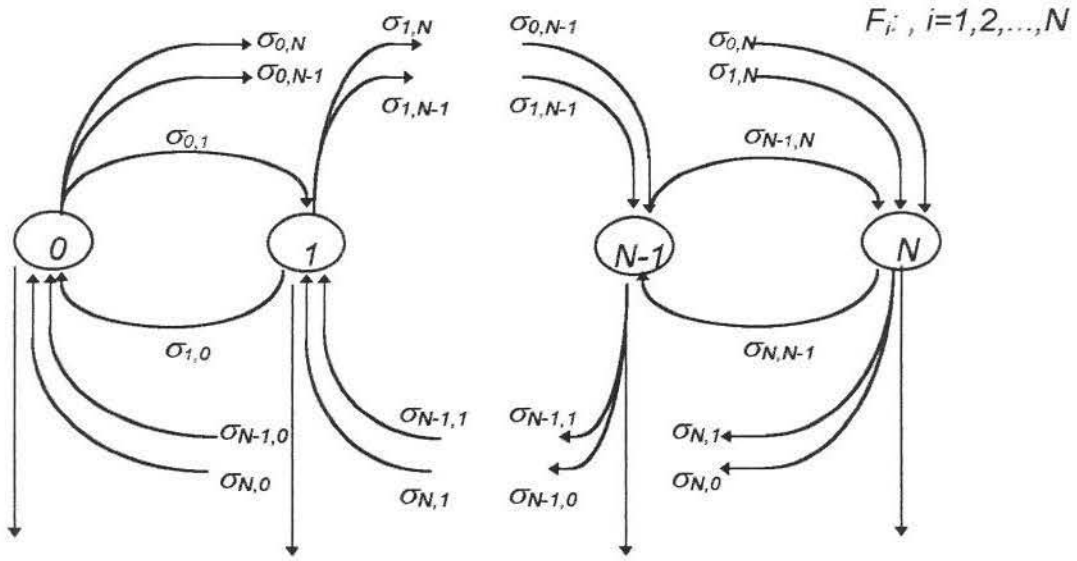
Observa-se, o conjunto de vetores que guardam as probabilidades de haver chegadas em cada fonte estando elas em um determinado estado e probabilidades de mudança de cada estado:

$F_1\rho$: $\{\rho_0, \rho_1, ..., \rho_n\}$, $F_2\rho$: $\{\rho_0, \rho_1, ..., \rho_n\}$, . . . , $F_n\rho$: $\{\rho_0, \rho_1, ..., \rho_n\}$

e (9)

$F_1\mu$: { μ_0, μ_1, ..., μ_n} , $F_2\mu$: { μ_0, μ_1, ..., μ_n} , . . . , $F_n\mu$: { μ_0, μ_1, ..., μ_n}

Para uma fonte F_k onde $k = 1,...,n$, se F_k estiver no estado i as chegadas ocorrem segundo um processo de Bernoulli com a caracterização da utilização das taxas de probabilidades associadas com cada estado de cada fonte. A cadeia de Markov associada ao processo para cada uma das fontes contendo N estados:



Chegadas de acordo com as probabilidades ρ_{i0},ρ_{i1}, ..., ρ_{iN-1},ρ_{iN}

Figura 4.10:Cadeia de Markov - Processo Markoviano Modulado de Bernoulli - N fontes

Como uma extensão do Processo Markoviano Modulado de Bernoulli, com M Fontes e dimensão NxN. Pode-se exemplificar sua simulação da seguinte forma. Dado 2 Fontes:

Para a Fonte1:
$$F_1P_{4,4} = \begin{bmatrix} 0,4 & 0,1 & 0,1 & 0,4 \\ 0,1 & 0,4 & 0,4 & 0,1 \\ 0,2 & 0,2 & 0,4 & 0,2 \\ 0,5 & 0,1 & 0,1 & 0,3 \end{bmatrix}$$

Para a Fonte2: $F_2P_{2,2} = \begin{bmatrix} 0,3 & 0,7 \\ 0,2 & 0,8 \end{bmatrix}$

Respectivamente para as duas fontes, tem-se $F_1\rho$:{0,5 ; 0,3 ; 0,4 ; 0,7} e $F_2\rho$:{0,6 ; 0,2} como vetores de suas taxas de probabilidades. Para cada fonte, novas matrizes transformadas da forma:

Fonte1:

$0 + 0,4 = 0,4$; $0,4 + 0,1 = 0,5$; $0,5 + 0,1 = 0,6$ e $0,6 + 0,4 = 1$

$0 + 0,1 = 0,1$; $0,1 + 0,4 = 0,5$; $0,5 + 0,4 = 0,9$ e $0,9 + 0,1 = 1$

$0 + 0,2 = 0,2$; $0,2 + 0,2 = 0,4$; $0,4 + 0,4 = 0,8$ e $0,8 + 0,2 = 1$

$0 + 0,5 = 0,5$; $0,5 + 0,1 = 0,6$; $0,6 + 0,1 = 0,7$ e $0,7 + 0,3 = 1$

gera-se:

estado 0: (0 ; 0,4 ; 0,5 ; 0,6 ; 1)

estado 1: (0 ; 0,1 ; 0,5 ; 0,9 ; 1)

estado 2: (0 ; 0,2 ; 0,4 ; 0,8 ; 1)

estado 3: (0 ; 0,5 ; 0,6 ; 0,7 ; 1)

e

Fonte2:

$0 + 0,3 = 0,3$ e $0,3 + 0,7 = 1$

$0 + 0,2 = 0,2$ e $0,2 + 0,8 = 1$

gera-se:

estado 0: (0 ; 0,3 ; 1)

estado 1: (0 ; 0,2 ; 1)

O processo de execução é idêntico ao caso com uma única fonte e os demais casos estudados de forma contínua, com mudanças de estados e a cada estado havendo geração de células de acordo com as suas respectivas taxas de probabilidades.

4.2 Geração de Tráfegos Auto-Semelhantes

Para dimensionar os recursos de uma rede, geralmente são desenvolvidos modelos de tráfegos. Tais modelos são geralmente baseados em processos markovianos, e, também são utilizados para retratarem uma variedade de tráfegos de rede encontrados nas antigas redes (tradicionais). Porém, com o advento de uma nova tecnologia digital voltada à tráfegos multimidia como ATM, estes modelos de tráfegos são considerados de aplicabilidade restrita e não são suficientes para retratarem tráfegos oriundos da superposição multimídia que apresentam uma nova característica que não fora encontrada anteriormente.

Diante destas evidências, novas classes de tráfegos estão surgindo [4], [12],[29],[34],[42] e [43]. Nos últimos anos, as análises de tráfegos atuais. demonstraram que várias origens estão produzindo fluxos de informações com uma característica bem diferente do que se costumava observar. Estes tráfegos multimídias agregados apresentam uma característica não convencional de tráfego, ou seja, uma forma caótica denominada Auto-Semelhante. E mais, tráfegos de redes locais Ethernet e também tráfegos de vídeo(VBR)[12],[16], apresentam também as mesmas características Auto-Semelhantes quando geradas longas amostras dos mesmos.

O termo Auto-Semelhança significa que a característica estatística do tráfego é invariante no tempo, ou seja, as mesmas propriedades estatísticas são observadas quando a escala de tempo muda, por exemplo, de segundos para horas. Quando amostras desse tráfego são agrupadas e comparadas em relação a intervalos de tempo diferentes, verifica-se o mesmo padrão de tráfego observado à partir do tráfego original. Esta característica de Auto-Semelhança pode ser considerada tanto no tráfego de dados com rajadas quanto no tráfego de vídeo.

Pode-se encontrar estruturas semelhantes em diversas áreas do conhecimento como por exemplo na Hidrologia, a quantidade de precipitação atmosférica diária, na Astronomia em relação ao comportamento de afastamento de galáxias, e até mesmo em muitos comportamentos associados com a

economia de um país. Também é verdade que este tipo de fenômeno se apresente nas comunicações de dados[42].

Auto-Semelhança envolve uma forte relação com a memória do sistema (correlação do sistema) envolvido. Uma correlação considerada positiva significa que quando um fluxo de pacotes aumenta em um determinado intervalo, a probabilidade de haver aumento no próximo intervalo também será alta. Uma correlação é negativa quando em um intervalo houver queda do fluxo, a probabilidade de queda no próximo intervalo também será evidente. Um sistema que possui correlação positiva ou negativa é considerado por possuir uma memória, e um sistema que não existe correlação é considerado sem memória.

Para sistemas de tráfegos, a correlação positiva é observada com atenção. Juntamente com estes tráfegos, observa-se uma dependência de longo alcance em toda a escala de tempo envolvida com os mesmos, o que torna difícil o modelamento de tais tráfegos. Quando une-se várias fontes On/Off de forma agregada estabelecendo um tráfego correlacionado, naturalmente aparece rajadas "burstys" e as mesmas persistem por um longo tempo.

Uma dependência de longo alcance significa que sua função de Autocorrelação associada ao tráfego, ao ser evidenciada graficamente em relação as amostras temporais, vê-se uma queda suave na curva do gráfico, o que significa na prática que situações de intenso tráfego pode acontecer entre segundos horas ou até dias, dificultando previsões à respeito.

Huang et al[12],[16], ao analisarem as curvas variância x tempo e análise R/S de diversos traços de vídeo VCR verificaram que nos primeiros pontos de amostragem, uma queda muito rápida da curva estabelecida do gráfico de maneira exponencial. Ou seja, a curva decai muito rapidamente, evidenciando uma dependência de pequeno alcance. Contudo, para os últimos pontos, já havia um suave declínio da curva, evidenciando uma dependência de longo alcance. Logo, tráfegos Auto-Semelhantes que possuem uma dependência de longo alcance, tem funções de autocorrelação decaem mais lentamente que de maneira exponencial.

Uma dependência de longa duração é uma característica que possui impacto mensurável e prático em comportamentos de filas e possui uma importância crucial para um grande número de problemas de engenharia de tráfegos de pacotes, tamanho de buffers, controle de admissão e congestionamento. Portanto, é um detalhe que não pode ser ignorado e pode resultar em previsões erradas de desempenho e alocações de recursos de rede inadequadamente.

Desta forma, a simulação de um tráfego Auto-Semelhante utilizando um modelo tradicional de tráfegos não é possível devido a longos ciclos de simulação que seria necessário para se gerar amostras adequadas destes tráfegos, e, por não existir geradores de números aleatórios eficazes para longos ciclos de simulação, toma-se o uso de outros modelos atuais propostos. Estes modelos são baseados em dois tipos de processos: Fractional Brownian Motion (FBM) e Fractional Gaussian Noise (FGN).

4.2.1 Modelo de tráfego Auto-Semelhante I

Lau et al[29], propuseram um método chamado Random Midpoint Displacement (RMD) para simulação de um processo de Fractional Brownian Motion.

Um Fractional Brownian Motion é um processo Gaussiano $B_H = (B_H(s) : s \geq 0)$, $0 < H < 1$, com uma função de correlação $R(s,t) = 1/2(s^{2H} + t^{2H} - |s - t|^{2H})$. Para todo $a > 0$, tem-se:

$$B_H(at) = a^H B_H(t), \qquad (10)$$

onde para $H = 1/2$, o processo torna-se ordinariamente um Brownian Motion. Ademais, com o processo de incremento $X_H = (X_H(k) = B_H(k + 1) - B_H(k) : k \geq 0)$ e com uma função de autocorrelação $\tau(k) = ½ ([k+1]^{2H} - 2[k]^{2H} + [k-1]^{2H})$, $k>=1$. Verifica-se que assintoticamente, $\tau(k) \approx H(2H -1)[k]^{2H-2}$, $½ < h < 1$ estabelecendo-

se uma dependência de longo alcance. É estabelecido que o processo agregado $X^{(m)} = (X^{(m)}(k) = m^{-H}(X_H(km-m+1)+...+X_H(km)) : k>=1), m>0$, todos possuem a mesma distribuição de X_H que é Auto-Semelhante.

O algoritmo RMD pode ser descrito da seguinte maneira: deseja-se gerar amostras FBM em um determinado intervalo de *0* à *T*. Divide-se este intervalo em subintervalos de maneira recursiva gerando novos pontos medianos entre *0* e *T* e entre os pontos intermediários. Dado dois pontos intermediários *[a e b]* onde, a construção de um ponto mediano entre os mesmos é dado pela relação:

$$\frac{Z(a)+Z(b)}{2} \qquad (11)$$

onde *Z(a)* e *Z(b)* são variáveis aleatórias Normalmente distribuídas *N (0,s)*.

Seja s_k o desvio padrão usado para a geração de ponto mediano no passo *k* e σ_0 o desvio padrão do deslocamento das amostras de um Fractional Brownian Motion na escala de tempo *T*, logo $\sigma_0 = T^{2H}$. Assumindo-se que $T = 2^n$, tem-se:

$$Var\left[Z\left(\frac{1}{2^k}\right) - Z(0)\right] = \left(\frac{1}{2^k}\right)^{2H}\sigma_0^2 \qquad (12)$$

onde *Var(X)* denota a variância de *X*. Lau et al [29] modificam a expressão acima para o algoritmo RMD de forma:

$$Var\left[Z\left(\frac{1}{2^k}\right) - Z(0)\right] = \frac{1}{4}Var\left[Z\left(\frac{1}{2^{K-1}}\right) - Z(0)\right] + S_K^2 \qquad (13)$$

Substituindo-se a equação *(4)* em *(3)* tem-se:

$$\left(\frac{1}{2^K}\right)^{2H}\sigma_0^2 = \frac{1}{4}\left(\frac{1}{2^{K-1}}\right)^{2H}\sigma_0^2 + S_K^2 \qquad (14)$$

modificando-se para:

$$S_K^2 = \left(\frac{1}{2^k}\right)^{2H}\left(1 - 2^{2KH-2-2KH+2H}\right)\sigma_0^2$$

$$= \left(\frac{1}{2^K}\right)\left(1 - 2^{2H-2}\right)\sigma_0^2 \qquad (15)$$

Contudo tem-se:

$$S_K = \left(\frac{1}{2^K}\right)^{2H}\sqrt{1 - 2^{2H-2}}\,\sigma_0 \qquad (16)$$

e

$$S_K = \frac{1}{2^H} S_{K-1} \qquad (17)$$

Definindo-se que:

$$S_0 = \sqrt{1 - 2^{2H-2}} \qquad (18)$$

Em termos computacionais, ao se gerar amostragens de um processo de Fractional Brownian Motion via RMD começa-se definindo-se o primeiro valor *Z(0) = 0* e por amostragem *Z(T) = GAUSS(0,T^2H)*. Em seguida, gera-se $Z\left(\frac{T}{2}\right) = \left(\frac{Z(0)+Z(T)}{2}\right) + OFFSET$ onde o OFFSET é uma variável aleatória Gaussiana com um desvio padrão associado a T^{2H} vezes o fator de escalonamento inicial $S_1 = 2^{-H} S_0 = 2^{-H}\sqrt{1 - 2^{2H-2}}$ multiplicado pelo fator de escalonamento. Reduz-se então o fator de escalonamento para $\frac{1}{2^H}$. À partir de

então, realiza-se o mesmo processo de divisão nos dois novos intervalos *0* à $\frac{T}{2}$ e $\frac{T}{2}$ à T e assim por diante. Abaixo encontra-se a exemplificação do algoritmo:

Passo inicial: $Z(0) = 0$

$Z(T) = GAUSS(0, T^{2H})$

Passo 1 $$Z\left(\frac{T}{2}\right) = \left(\frac{Z(0) + Z(T)}{2}\right) + (fe) * GAUSS(0,s_1) \quad //->offset!$$

Passo 2 $$Z\left(\frac{T}{4}\right) = \left(\frac{Z(0) + Z(\frac{T}{2})}{2}\right) + (fe^2) * GAUSS(0,s_1) \quad //->offset!$$

$$Z\left(\frac{3T}{4}\right) = \left(\frac{Z(\frac{T}{2}) + Z(T)}{2}\right) + (fe^2) * GAUSS(0,s_1) \quad //->offset!$$

Passo 3 $$Z\left(\frac{T}{8}\right) = \left(\frac{Z(0) + Z(\frac{T}{4})}{2}\right) + (fe^3) * GAUSS(0,s_1) \quad //->offset!$$

$$Z\left(\frac{3T}{8}\right) = \left(\frac{Z(\frac{T}{4}) + Z(\frac{T}{2})}{2}\right) + (fe^3) * GAUSS(0,s_1) \quad //->offset!$$

$$Z\left(\frac{5T}{8}\right) = \left(\frac{Z(\frac{T}{2}) + Z(\frac{3T}{4})}{2}\right) + (fe^3) * GAUSS(0,s_1) \quad //->offset!$$

$$Z\left(\frac{7T}{8}\right) = \left(\frac{Z(\frac{3T}{4}) + Z(T)}{2}\right) + (fe^3) * GAUSS(0,s_1) \quad //->offset!$$

e assim por diante...

onde *fe* é o fator de escalonamento dado por $fe = \frac{1}{2^H} * fe_{\_anterior}$.

O processo de Fractional Brownian Motion pode ser gerado pelo algoritmo RMD ao se considerar que:

$$A(t) = Mt + \sqrt{aM}\, Z(t) \qquad (19)$$

onde M é a taxa média, a é o fator de pico que é definido como a taxa de variância da média do número de células em uma unidade de intervalo de tempo.

O processo de incremento ou Fractional Gaussian Noise é dado por:

$$A(t) = Mt + \sqrt{aM}\, [Z(t+1) - Z(t)] \qquad (20)$$

4.2.2 Modelo de tráfego Auto-Semelhante II

Le Boudec e Droz[4] propuseram um gerador de Fractional Brownian Motion que também é baseado em Random Midpoint Displacement - RMD. Um processo Auto-Semelhante é um processo tal que: seja $X = (x_t : t=1,2,3,...)$ um processo estocástico estacionário de covariância com média $\mu = E[x_t]$, variância $\sigma^2 = E[(x_t - \mu)^2]$ e uma função de autocorrelação:

$$\tau(k) = \frac{E[(x_T - \mu)(x_{T+K} - \mu)]}{E[(x_t - \mu)^2]} \text{ para } k=0,1,2,3,... \qquad (21)$$

Assume-se uma função de autocorrelação da forma $\tau(k) \approx ak^{-\beta}$ com $k \to \infty$ e $0 < \beta < 2$ e o valor de "a" constante.

Assume-se que $X_m = (x_k^m : k = 1,2,3,...)$ é uma série formada de blocos de tamanho m tal que:

$$x_k^m = \frac{1}{m} \sum_{i=km-m+1}^{km} X_i \qquad (22)$$

onde $k = 1,2,3,...$ e $m = 1,2,3, ...$. Para cada m, X^m é um Processo estocástico de covariância. X é considerado Auto-Semelhante de segunda ordem e com parâmetro de Hurst $H = 1 - \beta/2$ e também Auto-Semelhante de segunda ordem assintótico desde que para todo $m = 1,2,3,...$ $var(X_m) = \sigma^2 m^{-\beta}$ e $\tau^m(k) = \tau(k)$, com $k >= 0$.

Existem três regiões para H. Para $0 < H < ½$, indica-se uma correlação negativa, para $½ < H < 1$, a correlação é positiva e para $H = ½$ tem-se correlação nula. Dado:

$$W = (w_t = \sum_{u=1}^{t} X_u) \qquad (23)$$

ser uma série de tempo consistindo de todas parciais somas da original série de tempo. Tem-se o desvio padrão:

$$S(t,s) = \sqrt{\frac{1}{s}\sum_{u=t+1}^{t+s} x_u^2 - \left(\frac{1}{s}\sum_{u=t+1}^{t+s} x_u\right)^2} \qquad (24)$$

Calcula-se o alcance $R(t,s)$:

$$R(t,s) = \max_{0<=u<=s}[w_{t+u} - w_t - \frac{u}{s}(w_{t+s} - w_t)] -$$

$$\min_{0<=u<=s}[w_{t+u} - w_t - \frac{u}{s}(w_{t+s} - w_t)] \qquad (25)$$

Chama-se de Alcance Rescaldai a taxa $R(s,t)/S(s,t)$. Gera-se então um gráfico assim chamado POX obtido pela relação:

$$log\left(\frac{R(s,t)}{S(s,t)}\right) \text{ contra } log(s) \qquad (26)$$

Para cada s calcula-se o valor médio para diferentes valores de k. Ao desenhar-se o gráfico, cruza-se uma linha onde a sua inclinação estabelece os parâmetros H.

O gráfico de Plotagem de Variância de tempo é obtido da relação $var(X^m) = \sigma^2 m^{-\beta}$ obtendo-se:

$$log\left(\frac{var(X^M)}{s^2}\right) \text{ contra } log(m) \qquad (27)$$

Como resultado, obtém-se uma curva onde a inclinação β sobre uma linha reta desta curva pode variar entre 0 e -2. O valor padrão da seguinte relação do parâmetro de Hurst está em $H = 1 - \beta/2$.

O algoritmo proposto por Droz e Le Boudec[4] para gerar um FBM pode ser descrito como: dado um intervalo $[0,T]$, tira-se a primeira amostra $x[T]$ através de um gerador de números aleatórios Gaussiano. Divide-se o intervalo ao meio gerando $x[T/2]$ que é determinado por $\frac{X[0] + X[T]}{2}$ mais um OFFSET determinado por τ que é o fator de escalonamento multiplicando um valor do gerador de números aleatórios Normal. O fator de escalonamento é dado por $1/2^H$ com $0 < H$

< 1. À partir de então são gerados novos pontos medianos "midpoints" de novas divisões.

O esquema é mostrado abaixo.

inicialização: $x[0] = 0$

$x[T] = G$

Passo 1 :
$$X\left[\frac{T}{2}\right] = \frac{X[0]+X[T]}{2} + \tau G$$

Passo 2 :
$$X\left[\frac{T}{4}\right] = \frac{X[0]+X\left[\frac{T}{2}\right]}{2} + \tau^2 G$$

$$X\left[\frac{3T}{4}\right] = \frac{X\left[\frac{T}{2}\right]+X[T]}{2} + \tau^2 G$$

Passo 3 :
$$X\left[\frac{T}{8}\right] = \frac{X[0]+\left[\frac{T}{4}\right]}{2} + \tau^3 G$$

$$X\left[\frac{3T}{8}\right] = \frac{X\left[\frac{T}{4}\right]+X\left[\frac{T}{2}\right]}{2} + \tau^3 G$$

$$X\left[\frac{5T}{8}\right] = \frac{X\left[\frac{T}{2}\right]+X\left[\frac{3T}{4}\right]}{2} + \tau^3 G$$

$$X\left[\frac{7T}{8}\right] = \frac{X\left[\frac{3T}{4}\right]+X[T]}{2} + \tau^3 G$$

e assim por diante. . . (28)

onde τ^k para $k = 1,2,3,...$ é o fator de escalonamento e k é a iteração. G é o gerador de números aleatórios Normal com média 0 e desvio padrão igual a 1.

Droz e Le Boudec[4] dividem o algoritmo em duas partes. A primeira, como já foi referenciado, gera-se traços FBM através das divisões "midpoints". Em seguida, gera-se amostras FBN da seguinte forma:

$$x[k] = x[k+1] - x[k] \qquad (29)$$

Objetiva-se enquadrar as amostras FBN de maneira a ficar confinado entre o intervalo 0 e o fator de escala para controle do traço. Este limitação do trace é baseada na multiplicação do valor de excesso (overbooking) pelo valor da velocidade de ligação (link_speed) e pela taxa de pico ρ. Isto permite então gerar o ruído escalado e transformado (a transformação não altera a estrutura da correlação, pois é linear). Com estes novos valores, é possível calcular-se a taxa de pico média das amostras do tráfego significando que o tráfego é controlado pela média m e a taxa de pico R. Tem-se o desvio padrão:

$$\sigma = \sqrt{m(R-m)} \qquad (30)$$

4.2.3 Modelo de tráfego Auto-Semelhante III

O modelo implementado por Mayor e Silvester[34] difere dos modelos propostos anteriormente por Lau et al[29] e Droz - Le Boudec[4]. Estes primeiros modelos foram baseados no algoritmo Random Midpoint Displacement, e, este último, na soma de processos de Markov de baixa e alta freqüências. Embora exposto por Mayor e Silvester[34], foi iniciado originalmente por Mandelbrot[42] e proposto por Chi[43].

Inicialmente, define-se um Fractional Brownian Motion, $B(t)$, como uma função aleatória real com incrementos Gaussianos independentes:

$$E[B(t+s) - B(t)] = 0$$

$$Var[B(t+s) - B(t)] = \sigma|s| \qquad (31)$$

Dado $dB(t)$ como sendo um movimento médio de um processo de Fractional Brownian Motion, com os incrementos anteriores de $B(t)$ associados a um padrão $(t - s)^{h-1/2}$, com um parâmetro H pertencendo ao intervalo $0 < H < 1$, tem-se:

$$B_H(t) = \frac{1}{\Gamma(H+1/2)}\int_{-\infty}^{0}((t-s)^{H-1/2} - (-s)^{H-1/2})dB(s) + \int_{0}^{t}(t-s)^{H-1/2}dB(s) \qquad (32)$$

Para que haja um Fractional Brownian Motion, o valor de H deve ser igual a ½. A Auto-Semelhança descrita é baseada no fato que a base da equação $B_H(\rho s)$ é a mesma em relação à distribuição $\rho^h * B_H(s)$, e, para os seus incrementos tal como o valor de Y_j, formando uma seqüência estacionária chamada de Fractional Gaussian Noise:

$$Y_j = B_H(j+1) - B_H(j),\ j = \ldots, -1, 0, 1, \ldots \qquad (33)$$

Estes incrementos somente serão considerados como Fractional Brownian Motion legitimo se o valor de H for igual a ½ como já fora dito, ou seja, apenas desta forma que estes incrementos se tornam independentes entre si, o que é uma característica crucial para gerar o processo corretamente. Geralmente isto é suposto. Além disso, a lei de Hurst afirma que:

$$Var[B_H(t+s) - B_H(t)] = \sigma s^{2H} \qquad (34)$$

E a variância de um processo Fractional Gaussian Noise não conduz-se de maneira linear se o valor de H for diferente de ½. Isto dificulta muito se acha soluções analíticas para retratar-se um determinado processo de Fractional Brownian Motion.

Um Fractional Gaussian Noise Discreto normalizado com média zero e variância unitária são considerados processos aleatórios Gaussianos e possuem uma função de autocovariância igual a:

$$C(s;H) = E[B_H(t + s + 1) - B_H(t + s)][B_H(t + 1) - B_H(t)]$$
$$= 2^{-1}[\,|\,s + 12\,|^{2H} - 2|\,s\,|^{2H} + |\,s - 1\,|^{2H}\,] \qquad (35)$$

Este método, uma contribuição de Chi[43], leva a um Fractional Gaussian Noise através da soma de processos de Markov de baixa e alta freqüências. O resultado do processo agregado tem uma função de autocorrelação muito parecida com as funções encontradas pelos dois primeiros métodos (Lau et al[29] e Droz - Le Boudec[4]). Contudo, a função de autocorrelação de um processo de Markov decai exponencialmente, enquanto que a função de autocorrelação de um Fractional Gaussian Noise decai linearmente. Para que haja uma utilização adequada dos processos de Markov no método, necessita-se da soma de vários processos dos mesmos desde que sejam independentes de maneira a se obter uma mesma estrutura de correlação. Considera-se:

$$X_f(t) = X_L(t) + X_H(t) \qquad (36)$$

Onde X_L e X_H são as freqüências baixa e alta respectivamente e X_f é um processo agregado. Considerando-se graficamente este processo, a curva associada ao mesmo, deverá ser larga, ou seja, a queda deve ser mais suave do que um curva de um gráfico exponencial. Para uma curva larga *(s)* a covariância de $X_L(t)$, $C_L(s;H)$, é aproximadamente:

$$C_L(s;H) = H(2H - 1)s^{2H-2} \qquad (37)$$

Para a representação do termo de baixa frequência, utiliza-se uma soma de *N* processos padronizados de Gauss-Markov.

$$X_L(t) = \sum_{n=0}^{N} (W_n)^{1/2} M^{(n)}(t) \qquad (38)$$

O processo de Gauss-Markov, $M^{(n)}(t)$, é suposto ser um par não correlacionado. O objetivo é calcular W_n de forma que a covariância de $X_L(t)$ case com $C_L(s;H)$. Obtém-se:

$$W_N = \frac{H(2H-1)}{\Gamma(3-2H)}(B^{1-H} - B^{H-1})B^{2(H-1)n} \qquad (39)$$

para $0 \leq n \leq N$, onde $\Gamma(3 - 2H)$ é a função Gamma.

Este casamento, tem uma diferença:

$$D(s;H) = C(s;H) - C_L(s;H) \qquad (40)$$

Para ajustar-se a diferença associada ao processo de frequência alta, o processo deve ter variância $D(0;H)$ e uma curva com uma covariância $D(1;H)$. Isto faz com que a covariância da soma dos processos de frequências baixa e alta tenham a mesma estrutura básica. A aproximação de um Fractional Gaussian Noise é dada por:

$$X_f(t) = X_L(t) + X_H(t)$$

$$C_f(s) = C_L(s) + C_H(s) \qquad (41)$$

A autocovariância do processo é dada por:

$$X(t) = aX(t - 1) + bG(t) \qquad (42)$$

onde $G(t)$ é a distribuição Normal padrão. Este processo tem média zero, com variância e autocovariância definida por:

$$Var[X] = \frac{b^2}{1 - a^2}$$

$$Covar(s) = E[X(s + t)X(t)] = \frac{b^2}{1 - a^2} a^s \qquad (43)$$

Capítulo 5

Procedimentos para a Simulação de Eventos Raros

O presente capítulo visa explicar a abordagem das técnicas de simulação de Eventos Raros evidenciando a Técnica Amostragem Importante e a Teoria dos Desvios Largos. Para tanto, duas aplicações foram estabelecidas, a primeira em um sistema simples, M/M/1 e a segunda, com um maior nível de complexidade, M/M/1 possuindo chegadas de células com característica Auto-Semelhante.

5.1 Procedimentos de Simulação de Eventos Raros baseado em um sistema de filas M/M/1

Deseja-se aplicar Amostragem Importante e a Teoria dos Desvios Largos para a estimação da taxa de probabilidade em uma fila M/M/1. Este interesse é devido à característica peculiar considerável bastante simples deste tipo de sistema[6],[18] o que permitiu algumas abordagens analíticas que estabelecem uma ligação eficaz com o modelo da simulação. como por exemplo, o valor da taxa de tempo médio entre estouros de uma fila *M/M/1*, que pode ser calculado

analiticamente e pode-se comparar tais resultados analíticos com o resultado da simulação.

Simular uma fila *M/M/1* utilizando o método de Monte Carlo é bem simples. A técnica utilizada baseia-se na chegada de pacotes e términos de serviços pelo servidor. A cada chegada de pacote, realiza-se uma escolha do tempo da próxima chegada como um evento aleatório. Para o término de serviço, o procedimento é o mesmo.

Uma fila *M/M/1* possui uma característica peculiar, taxa de chegada λ e com taxa de serviço μ. O caráter de seu tráfego é verificado de maneira Exponencial com fdp igual a $\alpha e^{-\alpha x}$, $x>0$.

Para a fila *M/M/1*, a rotina para a geração de números aleatórios, é obtida à partir da *F(x)* (FD) da Exponencial, descrita abaixo:

$$\tau(x) = P[X \le x] = \int_0^x \alpha e^{-\alpha t} dt = 1 - e^{-\alpha t}, x \ge 0 \qquad (1)$$

$$= 0 \quad , \text{para outros valores}$$

À partir deste ponto *(1)* obtém-se a nossa função para a geração de amostras exponenciais:

$$Y = 1 - e^{-\alpha t}$$

$$1 - Y = e^{-\alpha t}$$

$$LN(i - Y) = LN(e^{-\alpha t})$$

$$LN(i-Y) = -\alpha t$$

$$t = \left(\frac{-1}{\alpha}\right) LN(1-Y)$$

$$x = \left(\frac{-1}{\alpha}\right) LN(1-GNA(Y)) \qquad (2)$$

Onde *GNA* é a função geradora de números aleatórios entre *0* e *1*.

Deseja-se encontrar a probabilidade de acontecimento do Evento Raro, ou seja, a perda de um pacote. Uma fila M/M/1 possui um grande buffer, portanto, a probabilidade deverá ser da ordem de aproximadamente 10^{-9}. Subtende-se que ao estimar esta probabilidade com a simulação tradicional descrita acima (Monte Carlo) pode-se levar um tempo considerável de simulação.

Ao estimar tal taxa de probabilidade, tomar-se-á a técnica Amostragem Importante. Considera-se uma fila de tamanho finito *N* em um servidor determinístico com um uma taxa de chegada λ e uma taxa de serviço μ.

O objetivo é encontrar através da técnica Amostragem Importante a probabilidade de acontecimento de um evento raro (perda de célula) em uma fila *M/M/1* de uma maneira econômica já visto que para a simulação tradicional de Monte Carlo, chegaria a baixas taxas e representaria longos ciclos em vão. Para tal intuito, em vez de simular-se o sistema *M/M/1* desejado, simula-se um outro sistema *M/M/1'*, que é um sistema idêntico em suas propriedades ao *M/M/1*, só que agora modificado.

Dado *ER* ser o evento que deseja-se estimar a sua probabilidade, encontrar-se-á *ER'* onde o mesmo deverá possuir as seguintes características:

a) O evento *ER'* em *M/M/1'* deverá ser mais freqüente que *ER* em *M/M/1*.

b) A ligação entre *M/M/1* e *M/M/1'* permitirá se estimar *P(ER)* pelo conhecimento de *P(ER')*.

Como já descrito, um sistema *M/M/1* já é conhecido, e por conseguinte, a probabilidade de encontrar-se um erro (evento raro) é baseado na relação:

$$P(ER) = \frac{1}{n\sum_{i=1}^{n} 1_{ER}(w_i)} \qquad (3)$$

onde n é o número de ciclos, onde cada ciclo é determinado quando a fila está vazia (inicia o ciclo) até chegar-se ao seu limite acontecendo o erro (evento *ER*) ou esvaziando-se novamente (fim do ciclo). Também w_i é determinado pelo resultado dos experimentos e 1_{ER} fica igual a *1* quando o evento ocorre e *0* quando não.

Sabe-se que Amostragem Importante e todas as teorias associadas baseiam-se em reduzir-se a variância na simulação bem como a resolução de probabilidades em cima do sistema modificado. Destas teorias que fortalecem o uso de Amostragem Importante, um tratamento é realizado para se ter a associação entre os dois sistemas *M/M/1* e *M/M/1'* baseado na Teoria do Desvios Largos. Para o sistema *M/M/1*, tem-se uma variância:

$$E[\alpha - P(ER)_n]^2 = \frac{1}{n}(\alpha - P(ER)_n) \qquad (4)$$

Tem-se por associação o sistema *M/M/1'* com estimativa de probabilidade:

$$P'(ER) = \frac{1}{n\sum_{i=1}^{n} 1_{ER}(w_i)L(w_i)} \qquad (5)$$

onde $L = dP/dP'$ onde P é a medida de probabilidade de *M/M/1* e P' é a medida de probabilidade de *M/M/1'*,e w_i' é o resultado da simulação utilizando-se *M/M/1'* em n experimentos.

Tem-se uma nova variância de:

$$\frac{1}{n}\left(\int_{ER} L^2(w)dP'(w) - \alpha^2\right) \qquad (6)$$

e

$$(\sigma^*)^2 = \int_{ER} L^2(w)dP'(w) \qquad (7)$$

O interesse é que esta variância seja mínima e que haja um tempo de simulação também mínimo. O sistema *M/M/1'* deverá ser assintoticamente ótimo no limite que o tamanho do buffer tenda para o infinito[18].

Para encontrar-se uma otimização, utiliza-se o seguinte sequência de passos. Dado $x(k)$ como sendo um determinado estado de uma Cadeia de Markov formada pelas amostras da simulação de *M/M/1* onde a equação de transição entre cada estado é dada por:

$$x(k+1) = x(k) + w(k) \qquad (8)$$

onde $w(.)$ é um processo aleatório.

Dado que $F(.)$ seja a distribuição de salto da Cadeia de Markov $x(.)$ associada a um sistema S, onde este sistema possui a seguinte transformada de Cramer:

$$h(y) = \inf_{s \in \Re} \left[sy - \log \int_{-\infty}^{\infty} e^{sz} dF(z) \right] \qquad (9)$$

e dado

$$V(T, y_0, \ldots, y_{T-1}) = \sum_{k=0}^{T-1} h(y_k) \qquad (10)$$

onde y_k é o valor de y em qualquer tempo k. Deseja-se minimizar $V(.,.)$ em relação a T e y_k, sujeito à condição:

$$\sum_{k=0}^{T-1} h(y_k) = N \qquad (11)$$

Isto implica que, independente de quão longa seja a função de distribuição $F(.)$ o tempo e o estado não mudam, logo, o valor ótimo de y_k é uma constante.

A eq. de Lagrange é dado por:

$$\mathcal{L} = \sum_{k=0}^{T-1} h(y_k) - g \left[\sum_{k=0}^{T-1} y_k - N \right] \qquad (12)$$

onde g é o multiplicador de Lagrange.

Temporariamente fixa-se T diferenciando-se $\mathcal{L}$ em relação a y_k, e igualando-se a zero obtém-se:

$$\frac{\partial \mathcal{L}}{\partial y_k} = h'(y_k) - g \qquad (13)$$

$$\frac{\partial \mathcal{L}}{\partial y_k} = 0 \qquad (14)$$

Dado y^*_k ser o valor ótimo de y_k. Sujeito a $h'' > 0$. a equação *(13)* Implica:

$$y^*_k = y^* \qquad \forall k \in [0, T - 1] \qquad (15)$$

Para algum único y^*, poderá ser identificado e rescrevendo-se *(12)*:

$$\mathcal{L} = Th(y^*) - gTy^* + gN \qquad (16)$$

E a sua derivada em relação a T:

$$\frac{\partial \mathcal{L}}{\partial y_k} = h'(y^*) - gY^* \qquad (17)$$

$$\frac{\partial \mathcal{L}}{\partial y_k} = 0 \qquad (18)$$

Dado agora que y^* seja o valor ótimo de y_k, tem-se portanto como única solução positiva:

$$h(y^*) = y^* \frac{d}{dy} h(y^*) \qquad (19)$$

Exclusivamente para *M/M/1* com a seguinte Cadeia de Markov:

$$x(k+1) = x(k) + 1 \text{ com probabilidade } \lambda$$
$$+ -1 \text{ com probabilidade } \mu \qquad (20)$$

E transformada de Cramer de distribuição de salto de Bernoulli associado a *M/M/1*:

$$h(y) = \frac{1}{2}\left[(1+y)\log\left(\frac{(1+y)}{2\lambda}\right) + (1-y)\log\left(\frac{(1-y)}{2\mu}\right)\right] \qquad (21)$$

Substituindo-se por *h(.)* em *(19)* rejeitando-se a solução para $y < 0$ tem-se:

$$y^* = \mu - \lambda \qquad (22)$$

Existe uma conclusão onde a simulação ótima da fila possuirá uma taxa de chegada λ^* e taxa de serviço μ^*. Então a taxa média de aumento deste sistema será $y = \lambda^* - \mu^*$[18].

Assume-se portanto sem perda de generalidade, que $\lambda^* + \mu^* = 1$, logo percebe-se claramente a relação:

$$\lambda^* = \mu \qquad (23)$$

$$\mu^* = \lambda \qquad (24)$$

Conhecendo-se estes valores, basta realizar esta troca que o tempo de simulação será mais rápido, ou seja, fica evidenciado os parâmetros que aceleram a simulação onde os mesmos foram encontrados baseado na aplicação da Teoria dos Desvios Largos.

Agora, para chegar a probabilidade que se deseja encontrar, como parte da simulação aplicando-se Amostragem Importante, utiliza-se uma taxa modificadora ou "bias" aplicada ao sistema original para que possa estimar-se a probabilidade *ER'*. Esta taxa chamada "Likelihood Ratio" é uma derivada $L_k = dP/DP'$ durante um ciclo *k*, onde *P* é a medida em *M/M/1* tal que *P'* é a medida em *M/M/1'*, e, sabe-se que:

$$E[L_k ER_k] = E[ER_k] = P(ER') \qquad (25)$$

Simula-se então *M/M/1'* utilizando-se esta modificação: logo, a probabilidade de encontrar-se um evento raro (probabilidade que um ciclo termine em estouro) no sistema *M/M/1'* é:

$$P(ER') = \frac{1}{n}\left(L_1 ER_1 + LER_2 + L_3 ER_3 + \ldots + L_N ER_n\right) \qquad (26)$$

Sabendo-se como simular *M/M/1'* com a probabilidade *(26)* pode-se estipular-se o desejado. Contudo, existem duas dificuldades encontradas, a primeira refere-se em determinar-se o valor de *L* (no caso *M/M/1* é simples) e a segunda já demonstrada, o speed-up desejado para encurtar-se a simulação, pois, a modificação de parâmetros na simulação encurtando-a é compensado pela probabilidade modificada em *(26)*. O resultado encontrado é um valor real do sistema *M/M/1(4)*.

O valor de L é gerado da seguinte maneira: dado o sistema *M/M/1* com o tamanho do buffer N e com probabilidade de mudança de estado:

$$P\{w_k\} = \lambda^{N+l}\,\mu^{l} \qquad (27)$$

e o sistema *M/M/1'* com o mesmo tamanho de buffer N e com a probabilidade de mudança de estado:

$$P^*\{w_k\} = \mu^{N+l}\,\lambda^{l} \qquad (28)$$

onde l é o número de partidas geradas por um término de serviço e $N+l$ é o número de chegadas na fila.

L denominada de função "Peso", é encontrada através da seguinte relação[36]:

$$L = \frac{P(w_k)}{P^*(w_k)} \qquad (29)$$

Todavia, a maioria dos casos de Amostragem Importante, baseiam-se no tratamento Radon-Nikodym Derivative (RND).

Seja (Ω, E, P) um espaço de probabilidade. Onde Ω é o espaço amostral de interesse, E é o subconjunto dentro de Ω pela qual se mede a probabilidade e P a medida de probabilidade que pode ser associada a números entre *0* e *1* à eventos em E.

Seja P^* uma segunda probabilidade medida sobre eventos em E que possui a seguinte propriedade: para cada evento *ER* tal que $P^*(ER) = 0$, tem-se $P(ER) = 0$. Logo existe uma variável aleatória denotada por dP/dP^*.

No caso *M/M/1*, sejam λ e μ com $(0 < \lambda < \mu)$ respectivamente a chegada e serviço. Se *D* é uma diferencial exponencial com média *1/v*, então denota-se por M_v e h_v suas correspondentes transformadas de Laplace e Cramer respectivamente[6]. Dado:

$$M_v(s) = \frac{v}{v-s}, s < v$$

$$M_v(s) = \infty, \text{ caso contrário.} \qquad (30)$$

Dada a propriedade baseada no tratamento de uma fila *GI/GI/1*[6]:

$$\theta^* > 0$$

e

$$Mb(\theta^*)Ma(-\theta^*) = 1 \qquad (31)$$

onde *a* e *b* denotam tempos entre chegadas e serviço de uma fila *GI/GI/1*. Atribuindo-se a mesma propriedade para *M/M/1* tem-se:

$$\theta^* > 0$$

e

$$\frac{\lambda}{\lambda+\theta^*}\frac{\mu}{\mu-\theta^*}=1 \qquad (32)$$

Checando-se que a solução da equação é $\theta^* = \theta - \lambda$, L é derivado portanto:

$$L=\frac{dP}{dP^*}=\left(\frac{\lambda}{\mu}\right)^N \qquad (33)$$

5.2 Procedimentos de Simulação de Eventos Raros baseado em redes com Tráfego Auto-Semelhantes

Para a construção de um simulador adequado que possa representar com uma certa confiabilidade tráfegos que representem redes que apresentam características auto-semelhantes, como Ethernet[12] ou Variable bit Rate Vídeo (VBR)[16], deve-se primeiro determinar-se uma maneira de gerar tais amostras. O algoritmo a ser referenciado, é baseado na abordagem proposta por Huang et al[12],[16]. A Auto-Semelhança é apresentada por uma função de autocorrelação de longo alcance $\tau^{(k)} \sim k^{-\beta}$, com $k \to \infty$, para $0 < \beta \leq 1$ (A quantidade estimada $H = 1 - \beta/2$ é denominada parâmetro de Hurst) e pode ser representado por vários modelos estocásticos, dos quais os mais usados são o Fractional Gaussian Noise (FGN) e o Asymptotically Auto-Semelhante Fractional Autoregressive Integrated Moving-Average (F-ARIMA). F-ARIMA possui uma pequena vantagem relação ao FGN, pois com o mesmo pode obter-se tanto amostras de tráfegos apresentando dependência de longa duração (Large Range Dependence - LRD) como uma dependência de pequeno duração (Short Range Dependence), Contudo, esta abordagem é considerada inapropriada devido a dificuldade da estimativa de parâmetros envolvidos na mesma e para este enfoque (gerar tráfegos auto-semelhantes), somente eventos de longa duração são esperados.

Deseja-se criar um simulador que seja capaz de gerar tráfegos Auto-Semelhantes. Em seguida, será tratada a questão de como obter a probabilidade de perda de pacotes (Evento Raro) através da técnica de Amostragem Importante. O modelo a ser apresentado, é o comportamento de um servidor com fila única em estado constante submetido ao tráfego Auto-Semelhante utilizando-se FGN.

Em redes ATM reais, espera-se a explosividade do tráfego (Burstiness) em uma escala de tempo limitada. Logo, pode-se simular tal situação gerando-se um número considerável de replicações de longos sequências Auto-Semelhantes. O resultado estatístico é adequado já que eventos raros são freqüentes apenas na

ordem de probabilidade de 10^{-9}. Outra razão pela qual uma simulação normal não é eficiente.

Um Processo Fractional Gaussian Noise (FGN) pode ser descrito da seguinte maneira. Dado $X = \{X_k: k=1,2,...\}$ ser um processo Gaussiano estacionário (estado-constante) com média $m = E[X_k]$, e variância $\sigma^2 = E[(X_k - m)^2]$, este possui uma função de Autocorrelação:

$$\tau(k) = \frac{1}{2} (| k+1|^{2H} - 2|k|^{2H} + |k - 1|^{2H}), \; k = 1,2,3, \ldots \; e \; \frac{1}{2} < H < 1 \qquad (34)$$

Considera-se um servidor de fila única com uma taxa de serviço constante μ e uma taxa de chegada descrita de forma FGN. Dado X_k como sendo o número de células de chegada dentro do k-ésimo intervalo de tempo e Q_k ser o tamanho da fila para $k = 1,2,3,\ldots$. A característica da fila toma a forma de:

$$Q_k = (Q_{k-1} + Y_k) \text{ sendo que } Y_k = X_k - \mu \text{ , para } k = 1,2,3... \qquad (35)$$

Y_k é denominado aqui como sendo a carga de trabalho recebida pela fila e a carga total recebida pela fila será representada como a soma de todas as cargas Y_k acumuladas:

$$W_k = \sum_{i=1}^{k} Y_i \; , \quad k=1,2,3,... \qquad (36)$$

A probabilidade desejada é:

$$Pr(Q_k > b) = Pr (sup_{(0<=i<=k)} W_i > b) \text{ , para } k = 0,1,2,... \qquad (37)$$

que é a probabilidade de a capacidade da fila estar superior ao valor limite b onde esta probabilidade pode ser encontrada analiticamente da forma:

$$lim_{b->\infty}\ b^{-2(1-H)}\ log\ Pr(Q_\infty > b) = -c^{-2(1-H)}(c+\mu)^2\ /2 \qquad (38)$$

onde $c = \mu\ /H - \mu$ e $\mu > 0$. (39)

A geração de variadas amostras de processos Auto-Semelhantes é dificultado grandiosamente pela característica inerente de dependência de longa duração, e ainda, para a obtenção do mesmo, processos Auto-Semelhantes são possíveis quando faz-se uso de estruturas chamadas fractais, como o modelo Fractional Gaussian Noise FGN.

Para um processo X com média $m = 0$, a média condicional e a variância de X_k, dadas pelos valores passados $x_{k-1}, x_{k-2}, \ldots, x_0$ pode ser verificada da forma:

$$m_k = E(X_k \mid x_{k-1}, x_{k-2}, \ldots, x_0) = \sum_{j=1}^{k} \Phi_{kj} x_{k-j} \qquad (40)$$

e

$$v_k = Var(X_k \mid x_{k-1}, x_{k-2}, \ldots, x_0) = \sigma^2 \prod_{k=1}^{k} (1 - \Phi_{jj}^2) \qquad (41)$$

onde ϕ_{jj} é o j-ésimo coeficiente de correlação parcial de X_k e ϕ_{kj} são os coeficientes de regressão linear parcial.

O algoritmo proposto [12],[16] é descrito abaixo. Deve-se seguir estes passos para gerar-se $\{x_0, x_1, \ldots, x_{n-1}\}$ amostras de tamanho n de um processo Auto-Semelhante Gaussiano com correlação $\tau(k)$:

1. Gerar um valor inicial x_1 de uma distribuição Gaussiana com Gauss $(0, v_1)$ e atribuir $N_1 = 0$ e $D_1 = 1$.

2 . Para $k = 2,\ldots, n - 1$, calcular ϕ_{kj} , $j= 2, \ldots, k$ via as equações abaixo:

$$N_k = \tau(k-1) - \sum_{j=2}^{k-1} \Phi_{k-1,j}\tau(k-j)$$

$$D_k = D_{k-1} - \frac{N_{k-1}^2}{D_{k-1}}$$

$$\Phi_{kk} = \frac{N_k}{D_k}$$

$$\Phi_{kj} = \Phi_{k-1,j} - \Phi_{kk}\Phi_{k-1,k-j} \quad , j = 2,\ldots, k-1$$

3 . Calcule:

$$m_k = \sum_{j=2}^{k} \Phi_{kj} x_{k-j}$$

e

$$v_k = (1-\Phi_{kk}^2)v_{k-1}$$

4 . Gere x_k com Gauss (m_k , v_k).

Esta parte do simulador tem por objetivo gerar traços FGN. Chama-se a atenção quando em sua implementação, na inicialização de variáveis, pois não colocando-os de maneira adequada, o simulador não funcionará corretamente. O parâmetro de Hurst deve ser maior que *0,5* para que sua autocorrelação seja positiva. *i* corresponde o contador de amostras geradas e *n* e *d* são variáveis de apoio aos cálculos dos coeficientes ϕ's.

Com a obtenção da rotina que resolve a questão do tráfego Auto-Semelhante via amostra FGN, o próximo passo será a aplicação da mesma. Dado um servidor com uma fila, deseja-se calcular a probabilidade de perda de células

onde este possui uma taxa constante de serviço μ e seu estado único e também constante. a taxa de chegada está de acordo com o gerador de tráfego FGN.

Para estimar-se a probabilidade de haver um evento raro, utiliza-se a técnica Amostragem Importante adaptada para este tipo de tráfego, descrita à seguir.

Dado U como sendo uma variável aleatória de função de probabilidade $p(U)$, considera-se a procura da probabilidade P onde P é a probabilidade de acontecer o evento (no caso, perda de célula por estouro de fila), tem-se:

$$P = \int_{-\infty}^{\infty} I_A(t)p(t)dt = E_p[I_A(U)] \qquad (42)$$

Onde $I_A(t)$ é o a função indicadora do evento A no intervalo de t $(-\infty$ à $\infty)$.

Assume-se agora uma outra função densidade $p'(U)$ correspondente à $p(U)$. Então tem-se:

$$P = \int_{-\infty}^{\infty} I_A(t)\left[\frac{p(t)}{p'(t)}\right]p'(t)dt$$

$$= E_{p'}\left[I_A(U)\frac{p(U)}{p'(U)}\right]$$

$$= E_{p'}[I_A(U)L(U)]$$

$$= E_p[I_A(U)] \quad = \alpha \qquad (43)$$

Onde $L(U) = p(U)/p'(U)$, é a função peso.

Sabe-se porém que a probabilidade de acontecer o evento A' indicado por $I_A'(U) = I_A(U)L(U)$ é maior em $p'(U)$ do que A indicado por I_A em $p(U)$. Logo, simula-se a função $p'(u)$ e estima-se:

$$P' = \frac{(L_1(U)I_A(U)+L_2(U)I_A(U)+\ldots+L_n(U)I_A(U))}{N}$$

$$= \frac{1}{N\sum_{n=1}^{N} I_A(U_n)L(U_n)} \qquad (44)$$

Nota-se que $p'(U)$ presente em $L(U)$ é considerada densidade "twisted" , fator fundamental para a velocidade da ocorrência do evento. Existem diversas formas de sua otimização baseadas na teoria dos Desvios Largos.

A estimação do parâmetro para otimizar a simulação, é uma tarefa que pode ser tratada via Teoria dos Desvios Largos, onde os mesmos procuram achar o valor ótimo para aplicá-lo em Amostragem Importante chegando a melhores resultados na simulação e que estes apresentem-se de maneira confiável. Alguns autores utilizam métodos numéricos como interpolações para a otimização destes parâmetros. Huang[12] por sua vez, apresenta uma abordagem analítica.

Dado $Pr(Q_k > b) > max_{0<=i<=k}Pr(W_i > b) = P_{W,k}$. Verifica-se portanto que o valor é limitado por $Pr(Q_k > b)$ para qualquer valor referente à k desde que b seja grande. Como o valor de k cresce consideravelmente, pode ser mostrado que existe um valor de $k = k_s$ tal que $P_{W,\infty} = max_{i>=0} Pr(W_i > b) \cong Pr(W_{ks} > b)$, onde $ks = \lceil b/c \rceil$, e c foi definido em (5). Para $k > k_s$, $P_{W,k} \cong Pr(W_{ks} > b)$, sendo k_s o tempo quando a fila entra em estado constante, e $Pr(Q_\infty > b) \cong Pr(W_{ks} > b)$. Com esta abordagem, deseja-se procurar um valor "twisted" de média próxima ao ótimo para $Pr(W_{ks} > b)$ e aplicá-lo na simulação de $Pr(Q_\infty > b)$. Como W_{ks} é normalmente

$$= m^* + Ex(X_k \mid x_{k-1},\ldots,x_1)$$

$$= m^* + \sum_{j=2}^{k} \Phi_{kj} x_{k-j}$$

$$= m^* + \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{k-j} - m^*\right)$$

$$= m^* + m_{k,Y'} \qquad (47)$$

para $k = 2,3,\ldots$, onde

$$m_{k,Y'} = \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{k-j} - m^*\right)$$

e

$$Var_{Y'}(Y'_k|y'_{k-1}, \ldots, y'_1) = Var_X(Xk|x_{k-1}, \ldots, x_1) \qquad (48)$$

A carga de trabalho Y, será substituída por Y'. Logo, dado uma amostra $(y'_1,\ldots, y'_{k-1})$ de Y' e para todo $k = 2,3,\ldots$, têm-se os valores da média e variância:

$$E_Y\left(Y_k \mid y'_{k-1},\ldots,y'_1\right) = -\mu + \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{K-J} + \mu\right)$$

$$= -\mu + m_{k,Y} \qquad (49)$$

onde

$$m_{k,Y} = \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{k-j} + \mu\right)$$

e

$$Var_Y(Y_k|y'_{k-1},\ldots,y'_1) = Var_{Y'}(Y'_k|y'_{k-1},\ldots,y'_1) \qquad (50)$$

$$= m^{*} + Ex(X_k \mid x_{k-1},\ldots,x_1)$$

$$= m^{*} + \sum_{j=2}^{k} \Phi_{kj} x_{k-j}$$

$$= m^{*} + \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{k-j} - m^{*}\right)$$

$$= m^{*} + m_{k,Y'} \qquad (47)$$

para $k = 2,3,\ldots$, onde

$$m_{k,Y'} = \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{k-j} - m^{*}\right)$$

e

$$Var_{Y'}(Y'_k|y'_{k-1}, \ldots, y'_1) = Var_X(Xk|x_{k-1}, \ldots, x_1) \qquad (48)$$

A carga de trabalho Y, será substituída por Y'. Logo, dado uma amostra $(y'_1,\ldots, y'_{k-1})$ de Y' e para todo $k = 2,3,\ldots$, têm-se os valores da média e variância:

$$E_Y\left(Y_k \mid y'_{k-1},\ldots,y'_1\right) = -\mu + \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{K-J} + \mu\right)$$

$$= -\mu + m_{k,Y} \qquad (49)$$

onde

$$m_{k,Y} = \sum_{j=2}^{k} \Phi_{kj} \left(Y'_{k-j} + \mu\right)$$

e

$$Var_Y(Y_k|y'_{k-1},\ldots,y'_1) = Var_{Y'}(Y'_k|y'_{k-1},\ldots,y'_1) \qquad (50)$$

Pode-se estimar a taxa de probabilidade por:

$$L(k) = \frac{f_Y(Y_1', \ldots, Y_k')}{f_{Y'}(Y_1', \ldots, Y_k')}$$

$$= \frac{f_Y(Y_1')fY(y_2' \mid y_1') \ldots f_Y(y_K' \mid y_{k-1}', \ldots, y_1')}{f_{Y'}(Y_1')fY(y_2' \mid y_1') \ldots f_{Y'}(y_K' \mid y_{k-1}', \ldots, y_1')}$$

$$= \prod_{i=1}^{k} L_i \qquad (51)$$

Estas considerações matemáticas são abordadas em forma de algoritmo de maneira a obter em simulação, a taxa de probabilidade de acontecimento de um evento raro via Amostragem Importante. Este algoritmo descrito por Huang et al[12],[16] é detalhado à seguir.

Para chegar até $Pr(Q_k > b)$, o simulador deverá obedecer ao seguinte algoritmo:

1. O objetivo é observar N repetições de W (carga de trabalho total em um geração de k amostras de X_k) $w_{n1}, \ldots, w_{nN}$ onde $n = 1, \ldots, N$ obtendo a taxa de probabilidade de cada repetição.
2. Inicializar $i = 1$, $n = 1$;
3. Enquanto $W_i <= b$ e $i <= k$ continuar fazendo:
    3.1. Gerar amostra x_i pelo método de Hosking
    3.2. Gerar amostra Y'_i pela equação $Y'_i = x_i + m^*$
    3.3. Gerar W'_i
4. Se $i = k$ então
    4.1. $I_n <- 0$

senão

4.2. fazer I_n <- *1*

4.3. fazer o Cálculo de $L^{(n)} = L_{(i)}$ pelas equações abaixo descritas.Utilizar a equação *(18)* levando-se em consideração as equações abaixo sugeridas por Huang et al[12],[16]:

$$L_i = \frac{e^{\theta i y i}}{M_i} \qquad \text{para } i = 2,3,...$$

onde

$$\theta_i = -\frac{\left(\mu - m_{i,j} + m^{*} + m_{i,j}\right)}{\left(\sigma^2 \prod_{j=2}^{i}\left(1-\Phi_{jj}^2\right)\right)}$$

e

$$M_i = e^{\frac{-\theta i}{2(\mu - mi,y + m^{*} + mi,y')}} \qquad (52)$$

5. se $n = N$ então

5.1. avaliar $$P' = \frac{1}{N\sum_{n=1}^{N} I_n L^{(n)}}$$

senão

5.2. fazer $i = 0$ e $n = n + 1$ e volte ao passo 2.

Capítulo 6

Conclusão

Este Capítulo apresenta as contribuições deste trabalho, apontando propostas e considerações sobre o aperfeiçoamento dos sistemas envolvidos. Por fim será apresentado uma avaliação do trabalho, situando-o dentro do contexto e realidade em que foi desenvolvido.

6.1 Contribuições

Este trabalho apresenta diversas contribuições. Como principais contribuições pode-se mencionar:

i) A criação de um sistema de simuladores para o ambiente Windows das técnicas apresentadas neste trabalho: GTRAF - Geradores de Tráfegos e GEVR - Geradores de Eventos Raros.

ii) A criação de um conjunto de simuladores para o ambiente Unix das técnicas apresentadas neste trabalho.

6.2 Trabalhos Futuros

Podemos citar como trabalhos futuros:

i) A ampliação das bibliotecas de rotinas criadas neste trabalho.

ii) A integração com a ferramenta TANGRAN do projeto CNPq Protem Almadem.

iii) Estender a técnica de Amostragem Importante para a simulação Amostragem Importante Dinâmica através da criação de novos simuladores com a orientação para sistemas que apresentam grandes dimensões observando a viabilidade de simulações regenerativas.

iv) Comparar Amostragem Importante, Amostragem Importante Dinâmica e a Teoria dos Valores Extremos verificando suas adequabilidades e a quais sistemas se aplicariam.

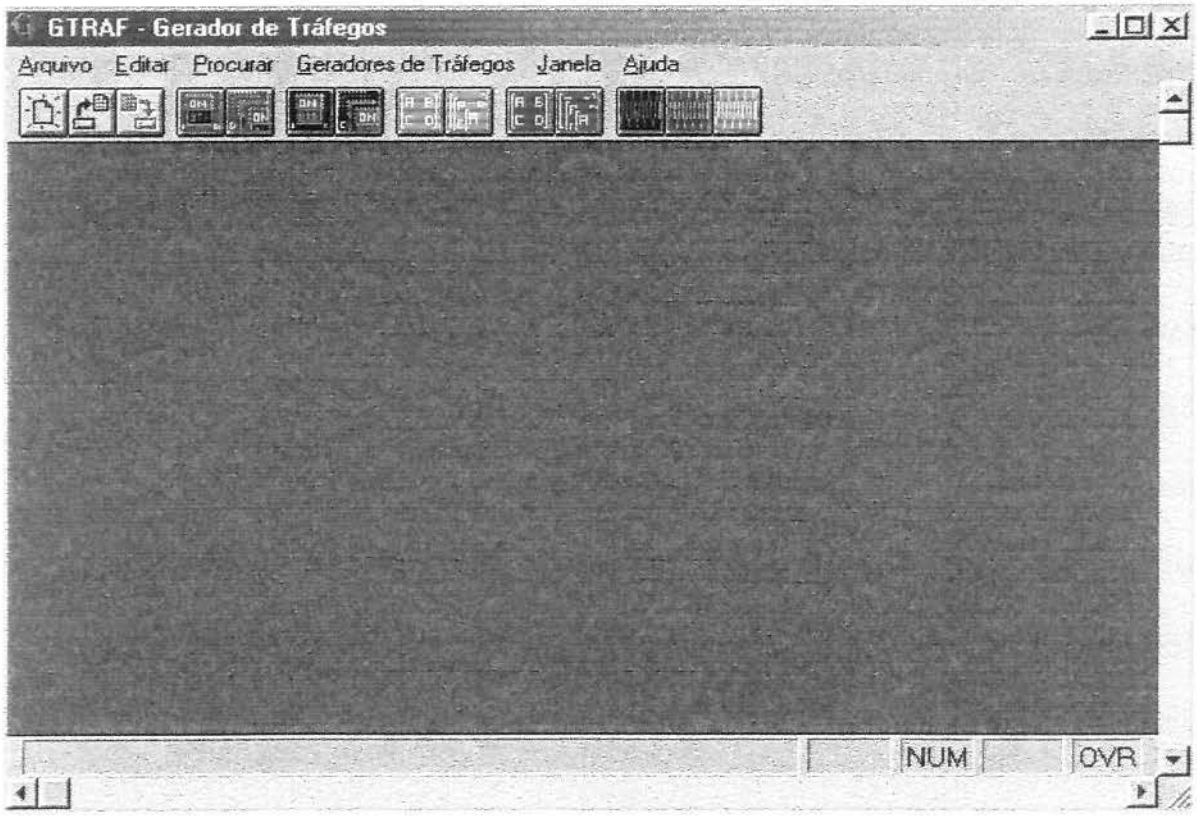
APÊNDICE A

Simuladores

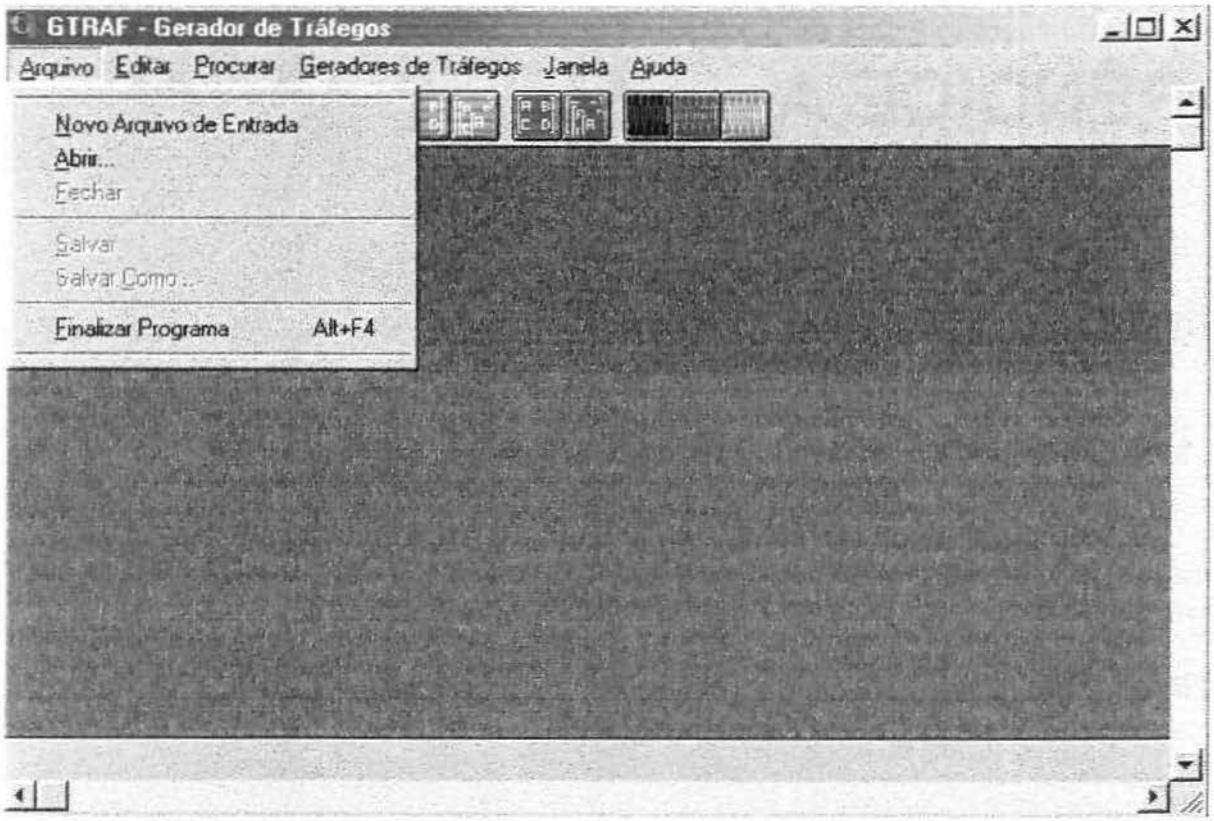
A.1 GTRAF - O Gerador de Tráfegos

Este programa apresenta tem como intuito, simular diversos tipos de tráfegos que podem ser utilizados ao gerar traços que possam se adequar a situações reais de redes que atuam com a tecnologia ATM.

GTRAF.



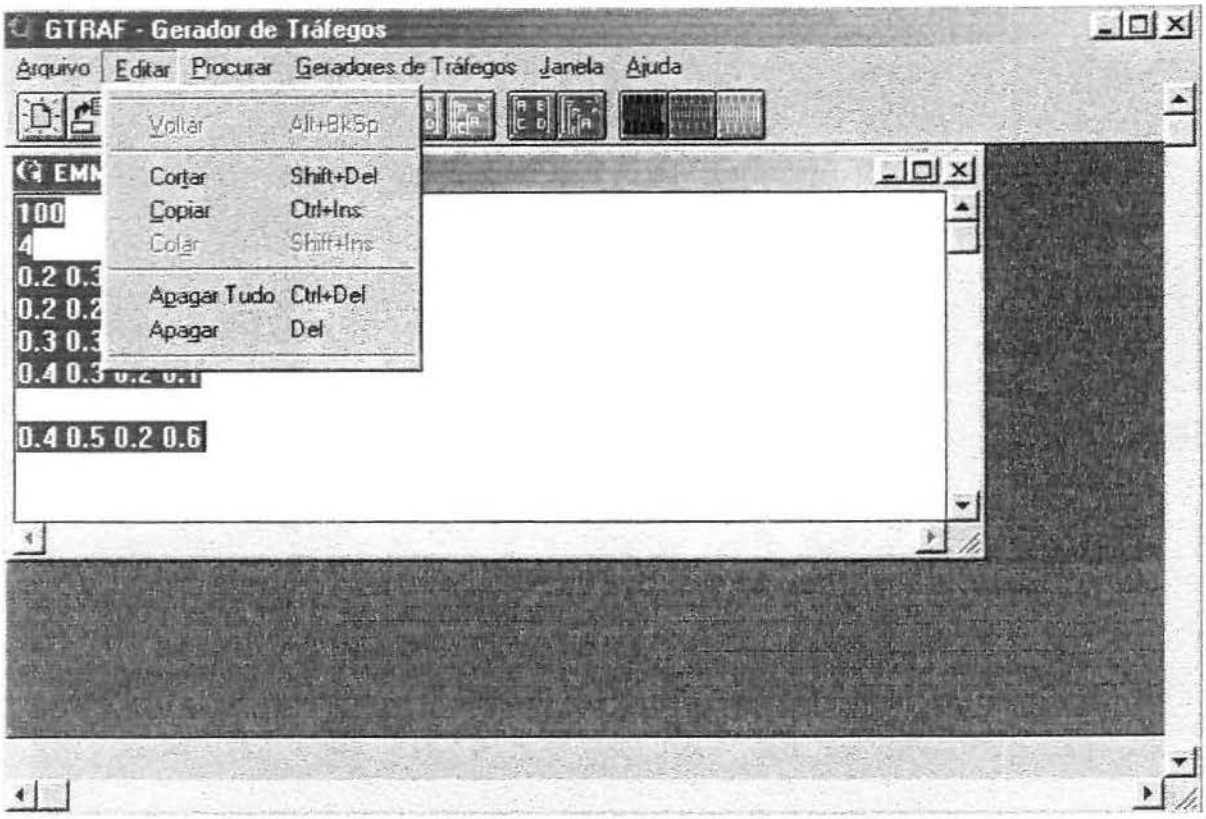
A tela principal do GTRAF é composta de um menu principal pelo qual, apresenta-se um submenu de acesso aos programas e uma barra de ferramentas para acesso mais rápido.



Elementos do Menu Arquivo:

O Menu Arquivo fornece comandos para a criação de novos arquivos, abertura de arquivos, gravação de arquivos e outros.

Novo	Cria um novo documento, sem nome.
Abrir	Abre um arquivo existente.
Fechar	Fecha o documento atual.
Salvar	Salva o documento se seu conteúdo foi alterado.
Salvar Como	Salva o documento atual com um outro nome.
Finalizar Programa	Sai do GTRAF.



Elementos do Menu Editar:

O Menu Editar fornece comandos para copiar, mover, apagar objetos ou textos.

Voltar Elimina a última ação.

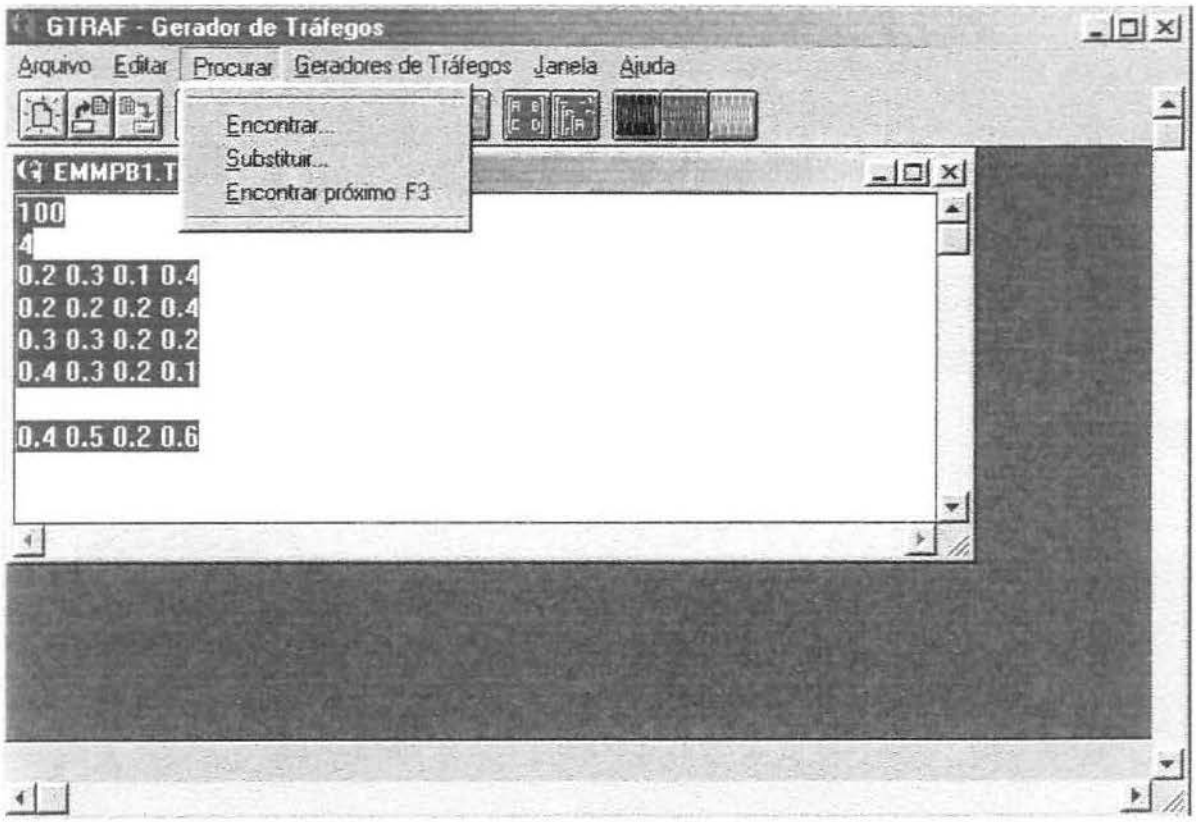
Cortar Recorta do texto um objeto selecionado.

Copiar Copia do texto um objeto selecionado.

Colar Cola no texto um objeto copiado ou cortado.

Apagar tudo Limpa todo o texto.

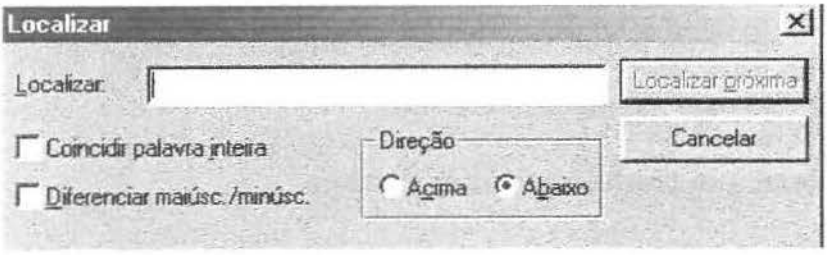
Apagar Limpa o texto selecionado.



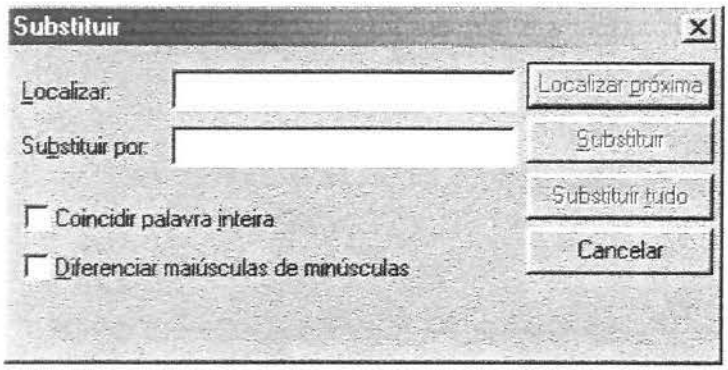
Elementos do Menu Procurar:

O Menu Procurar fornece comandos para procurar e substituir textos.

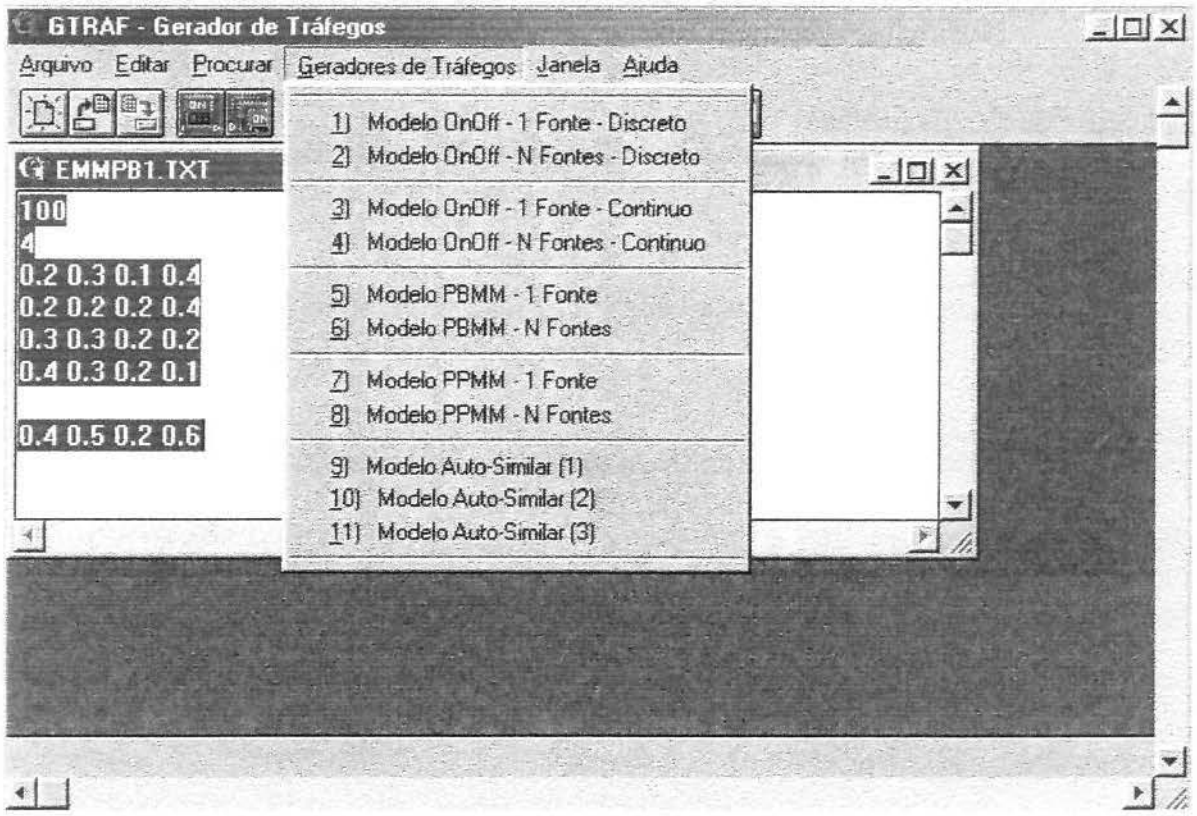
Localizar Procura um determinado texto.



Substituir Procura um determinado texto.



Encontrar Próximo Permite refazer a ação de Encontrar ou Substituir.



Elementos do Menu Geradores de Tráfegos:

O Menu Geradores de Tráfegos apresenta rotinas que simulam variados tipos de tráfegos de redes de computadores.

1) Modelo OnOff - 1 Fonte - Discreto
Simula o tráfego de uma única fonte OnOff apresentando suas chegadas em tempo discreto.

2) Modelo OnOff - N Fontes - Discreto

Simula o tráfego de várias fontes OnOffs apresentando suas chegadas em tempo discreto.

3) Modelo OnOff - 1 Fonte - Continuo
Simula o tráfego de uma única fonte OnOff apresentando suas chegadas em tempo continuo.

4) Modelo OnOff - N Fontes - Continuo
Simula o tráfego de várias fontes OnOffs apresentando suas chegadas em tempo continuo.

5) Modelo PBMM - 1 Fonte
Simula o tráfego de uma única fonte com um Processo Bernoulli Modulado de Markov.

6) Modelo PBMM - N Fontes
Simula o tráfego de várias fontes com um Processo Bernoulli Modulado de Markov.

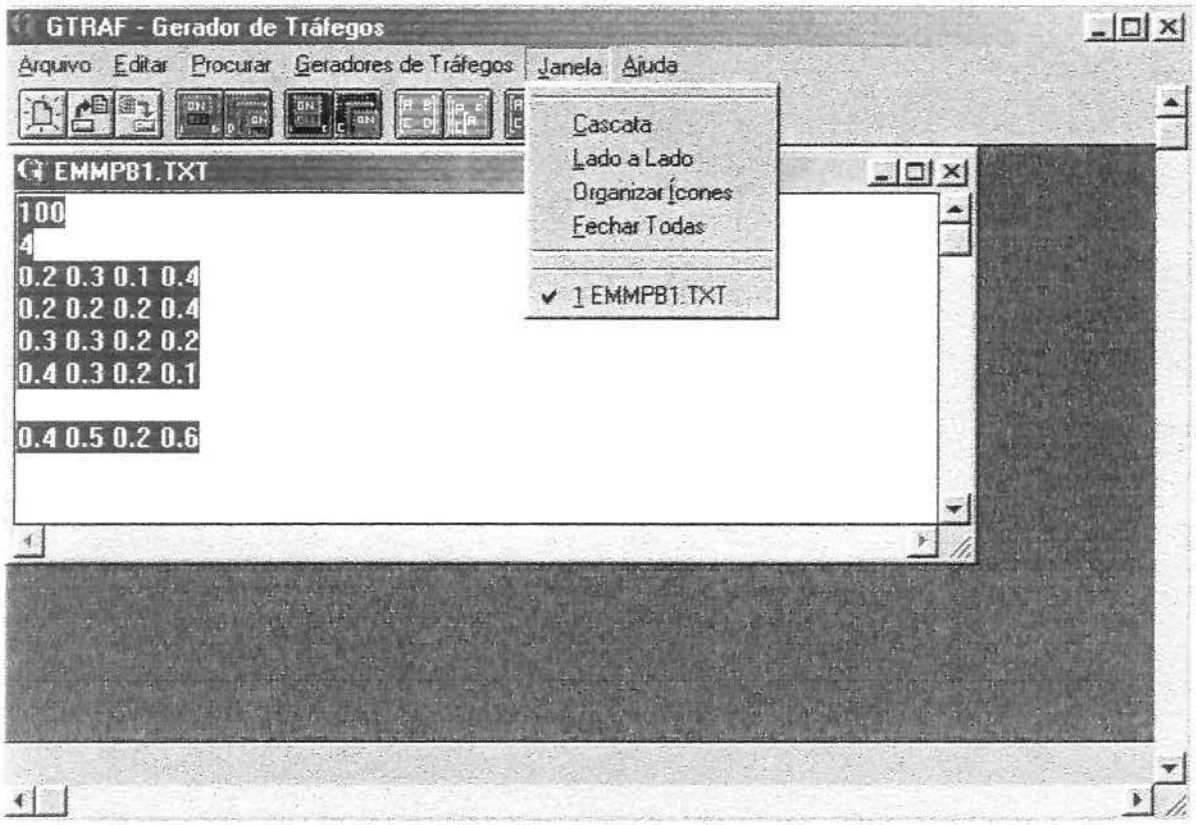
7) Modelo PPMM - 1 Fonte
Simula o tráfego de uma única fonte com um Processo Poisson Modulado de Markov.

8) Modelo PBMM - N Fontes
Simula o tráfego de várias fontes com um Processo Poisson Modulado de Markov.

9) Modelo Auto-Semelhante (1)
Simula o tráfego Auto-Semelhante com uma abordagem baseada em Ponto-Médio (Mid-Point).

10) Modelo Auto-Semelhante (2)
Simula o tráfego Auto-Semelhante com uma outra abordagem baseada também em Ponto-Médio (Mid-Point).

11) Modelo Auto-Semelhante (3)
Simula o tráfego Auto-Semelhante com uma abordagem baseada no algoritmo de Hosking.



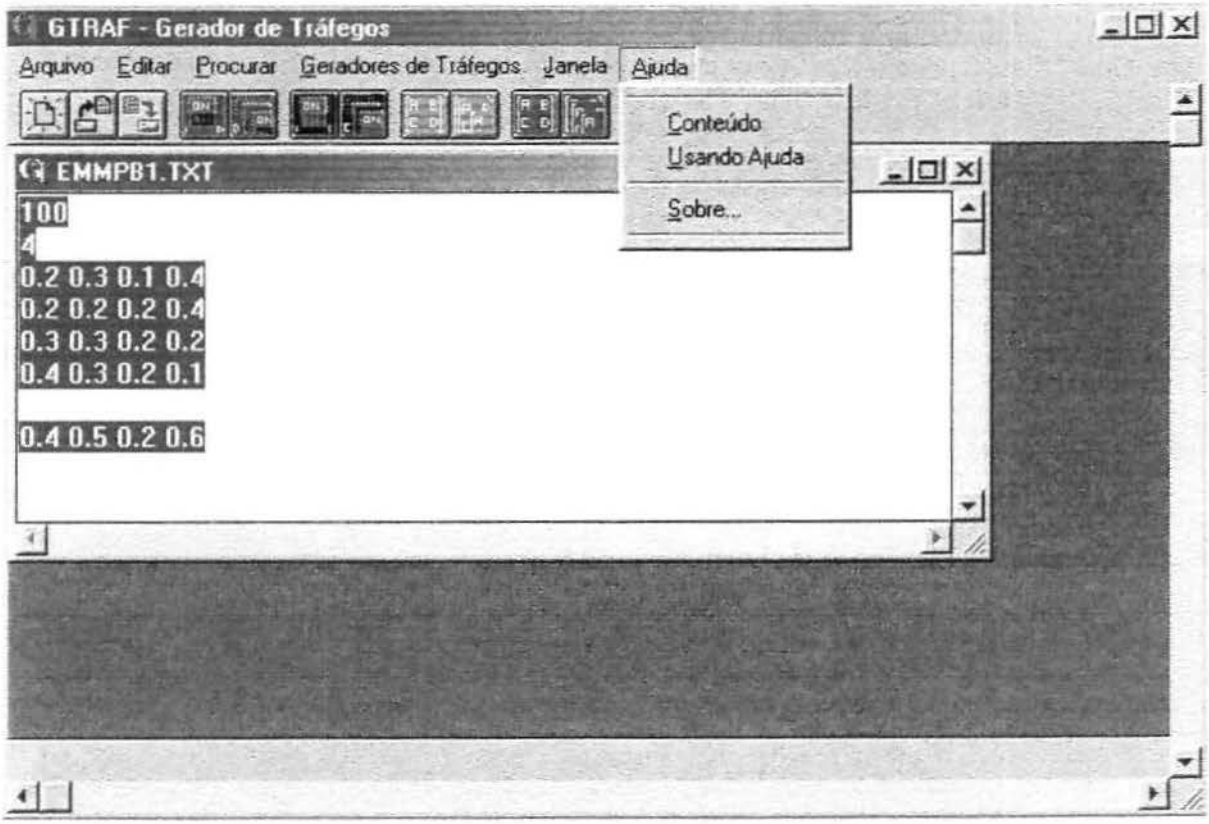
Elementos do Menu Janela:

O Menu Janela fornece comandos ao controlar o posicionamento e a organização das janelas do programa.

Cascata Coloca as janelas sobrepostas.

Lado à Lado Coloca as janelas Lado à Lado

Fechar Todas Fecha todas as janelas abertas.



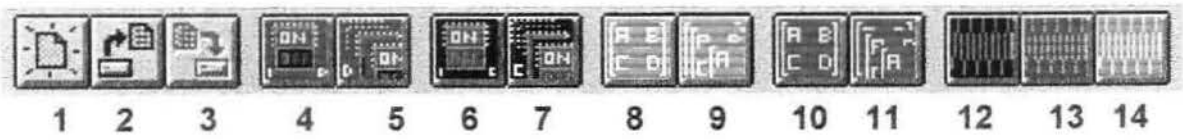
Elementos do Menu Ajuda:

Conteúdo Apresenta um Help à respeito do programa.

Usando Ajuda Apresenta uma explicação de como utilizar o Help.

Sobre Apresenta uma identificação do programa.

A Barra de Ferramentas



1: Novo Arquivo.

2: Abre Arquivo.

3: Salva Arquivo.

4: Simula OnOff - 1 Fonte - Caso Discreto.

5: Simula OnOff - N Fontes - Caso Discreto.

6: Simula OnOff - 1 Fonte - Caso Contínuo.

7: Simula OnOff - N Fontes - Caso Contínuo.

8: Simula PBMM - 1 Fonte.

9: Simula PBMM - N Fontes.

10: Simula PPMM - 1 Fonte.

11: Simula PPMM - N Fontes.

12: Simula Auto-Semelhante I.

13: Simula Auto-Semelhante II.

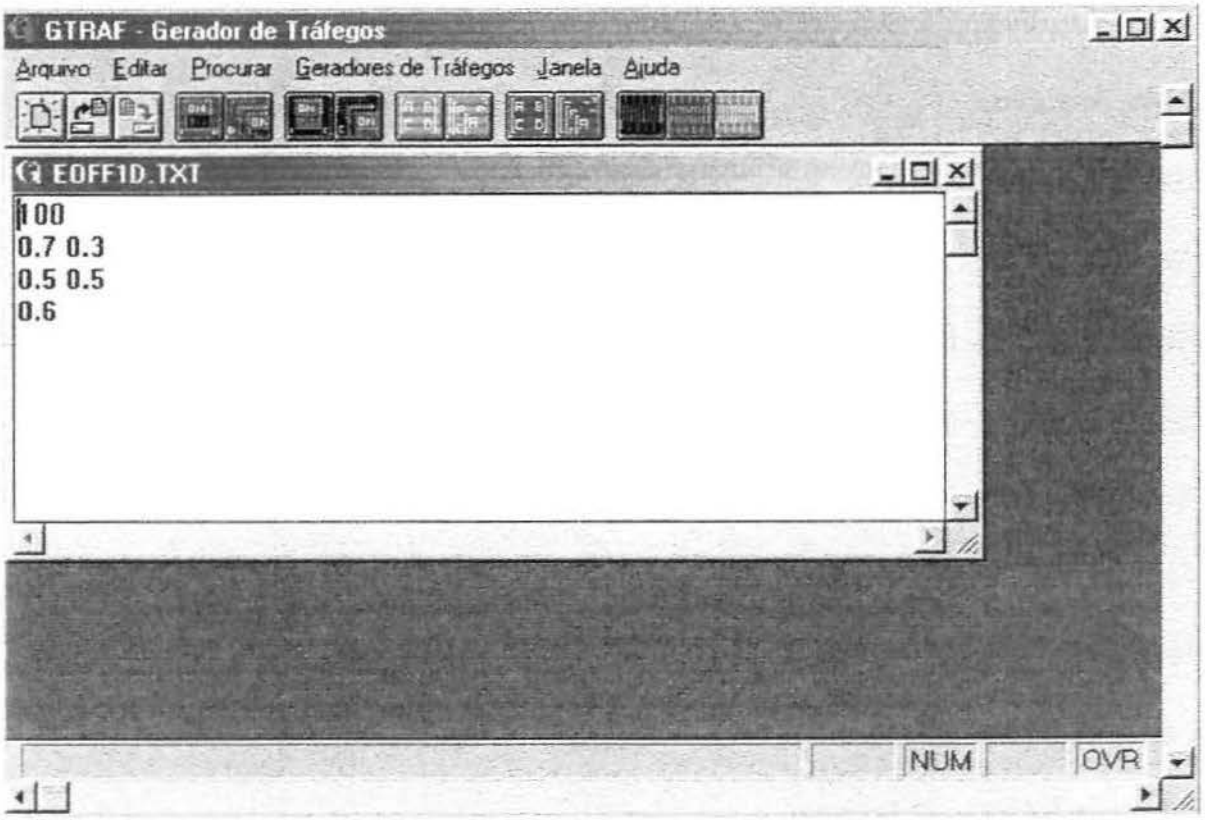
14: Simula Auto-Semelhante III.

Os Simuladores.

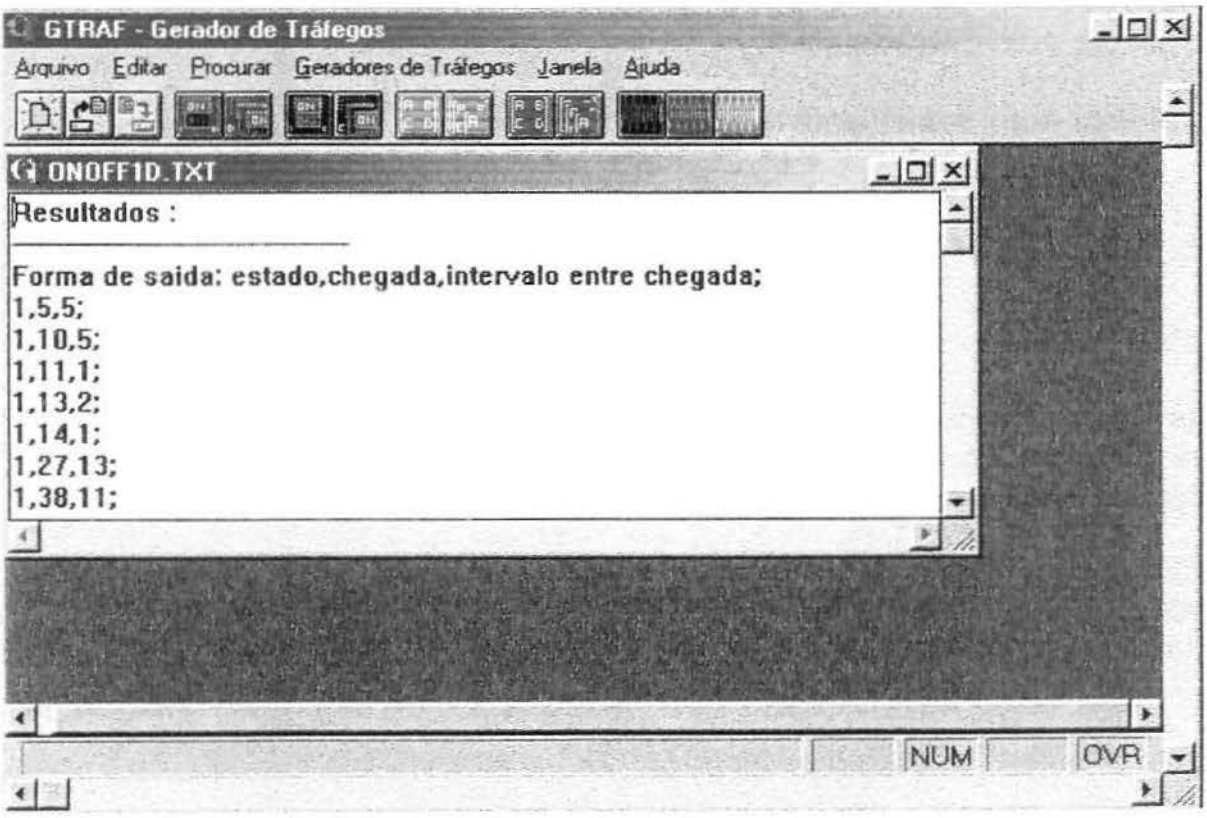
1) Modelo OnOff - 1 Fonte - Discreto

Este programa consiste de uma entrada de dados via arquivo bem simples. Basta abrir o documento chamado **eoff1d.txt** ou criar um novo com este nome. Este programa precisa das seguintes informações como entrada de dados:

{ Número de Ciclos}
{Matriz 2x2 de probabilidades}
{A taxa de probabilidade de chegada }



O resultado final é apresentado através do arquivo **onoff1d.txt**.



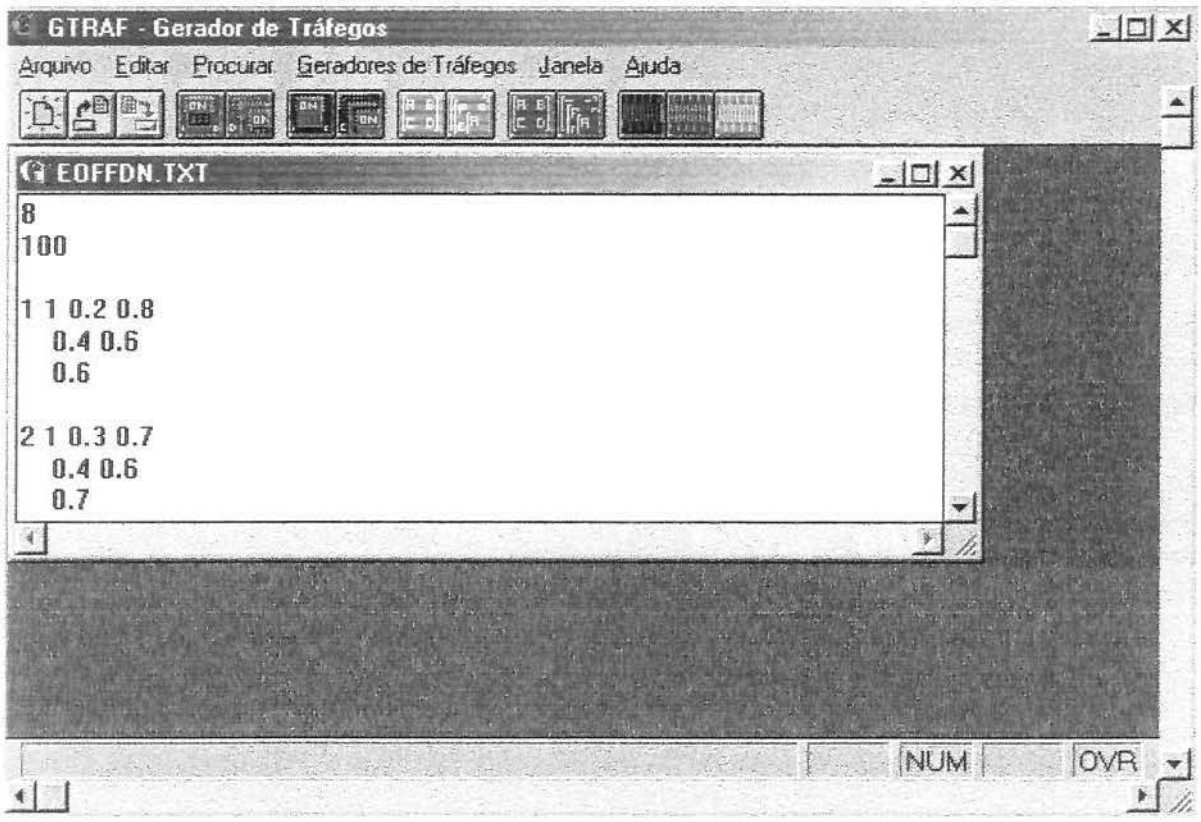
2) Modelo OnOff - N Fontes - Discreto

Este programa exige uma entrada de dados maior. O tamanho da entrada dependerá de quantas fontes o usuário desejar que o programa simule. Os dados de entrada são colocados no arquivo de entrada chamado **eoffdn.txt**. Pode-se também criá-lo. A entrada de dados consiste de :

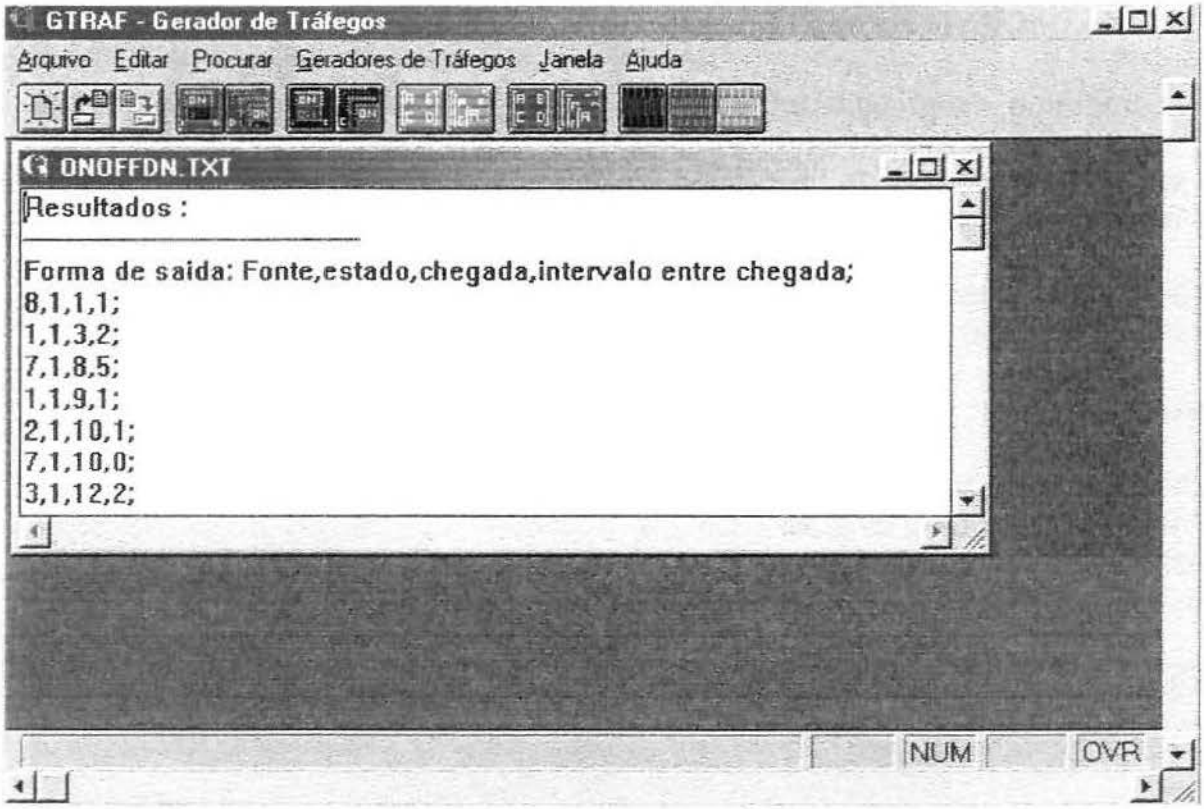
{ Número de Fontes}
{Número de ciclos}

Para cada fonte:

{Um número que descreve a fonte}
{O estado inicial que a fonte se encontra}
{Matriz 2x2 de probabilidade}
{Taxa de probabilidade de chegada de dados da fonte}



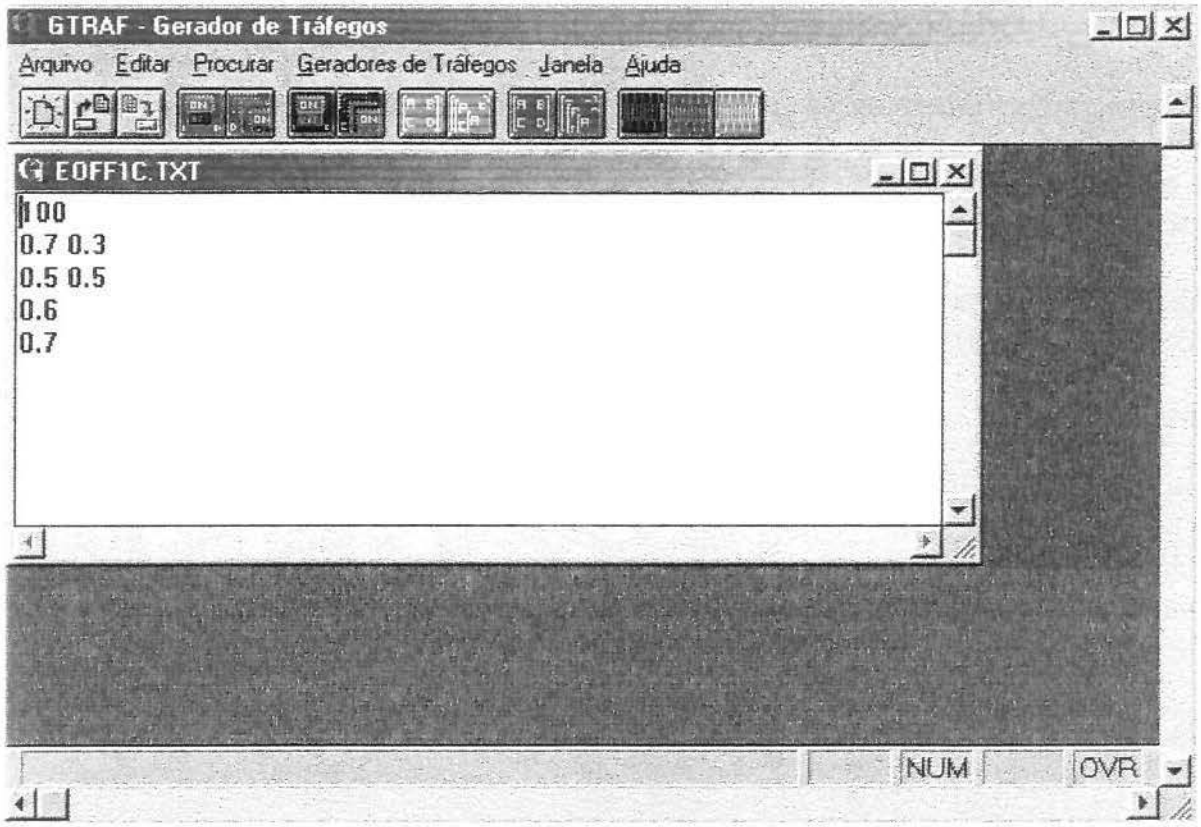
O resultado final é apresentado através do arquivo **onoffdn.txt**.



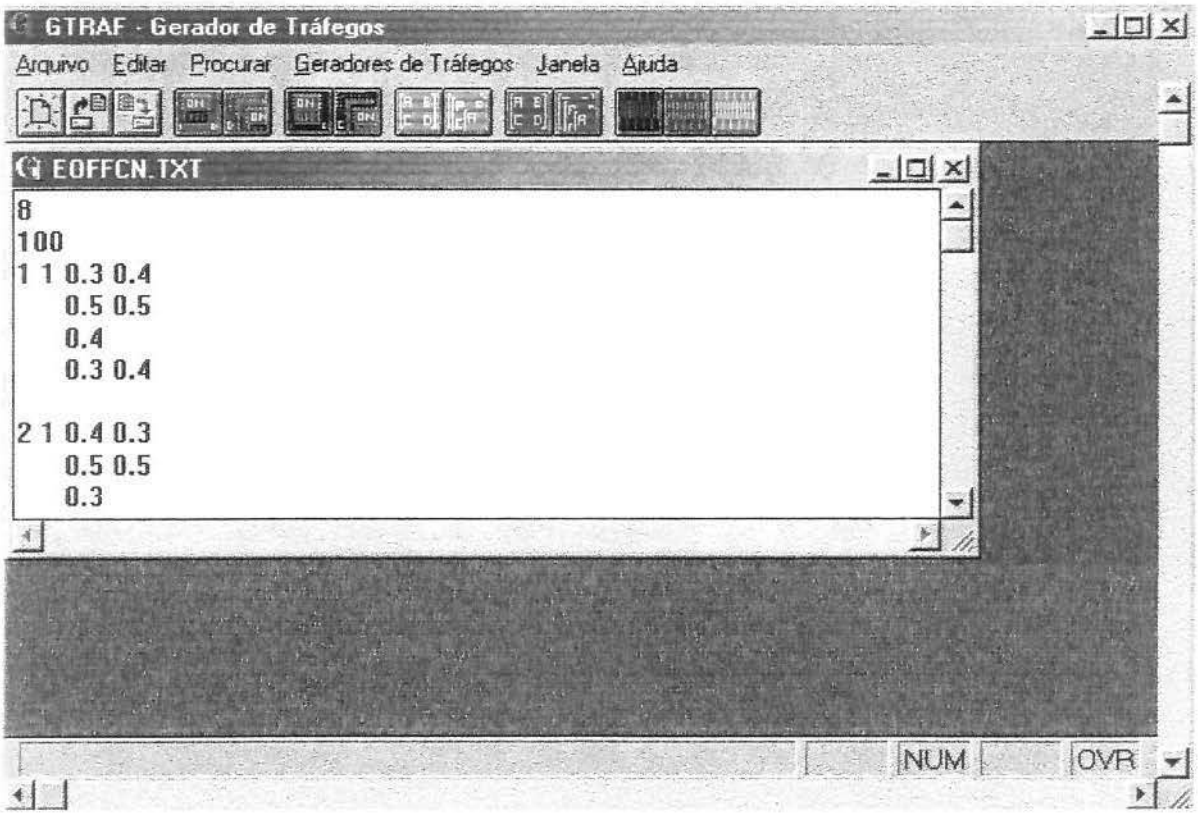
3) Modelo OnOff - 1 Fonte - Continuo

Este programa consiste também de uma entrada de dados via arquivo bem simples. Basta abrir o documento chamado **eoff1c.txt** ou criar um novo com este nome. Este programa precisa das seguintes informações como entrada de dados:

{Número de Ciclos}
{Matriz 2x2 de transição de estados}
{Taxa média de chegada para a função exponencial}
{Taxa média de mudança para a função exponencial}



Os resultados são verificados no arquivo de saída **onoff1c.txt**.



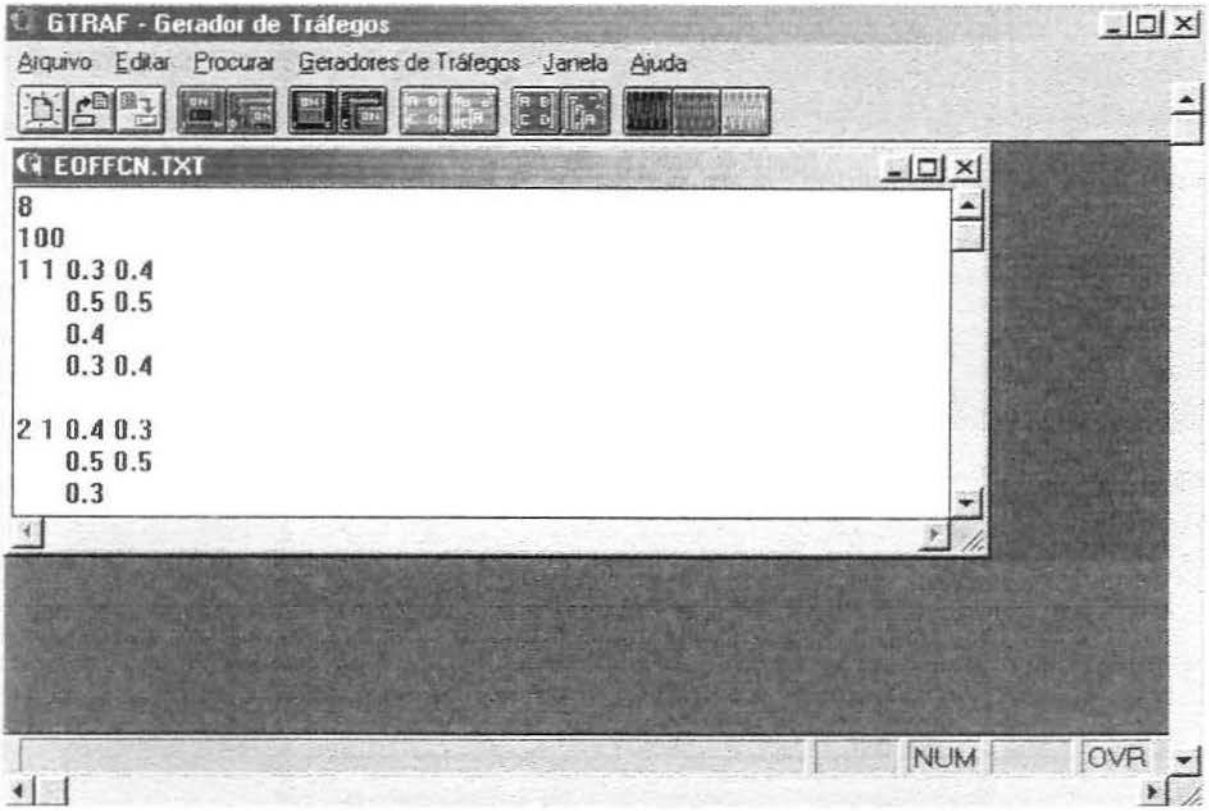
4) Modelo OnOff - N Fontes - Continuo

Este programa também exige uma entrada de dados maior. O tamanho da entrada dependerá de quantas fontes o usuário desejar que o programa simule. Os dados de entrada são colocados no arquivo de entrada chamado **eoffcn.txt**. Pode-se também criá-lo. A entrada de dados consiste de :

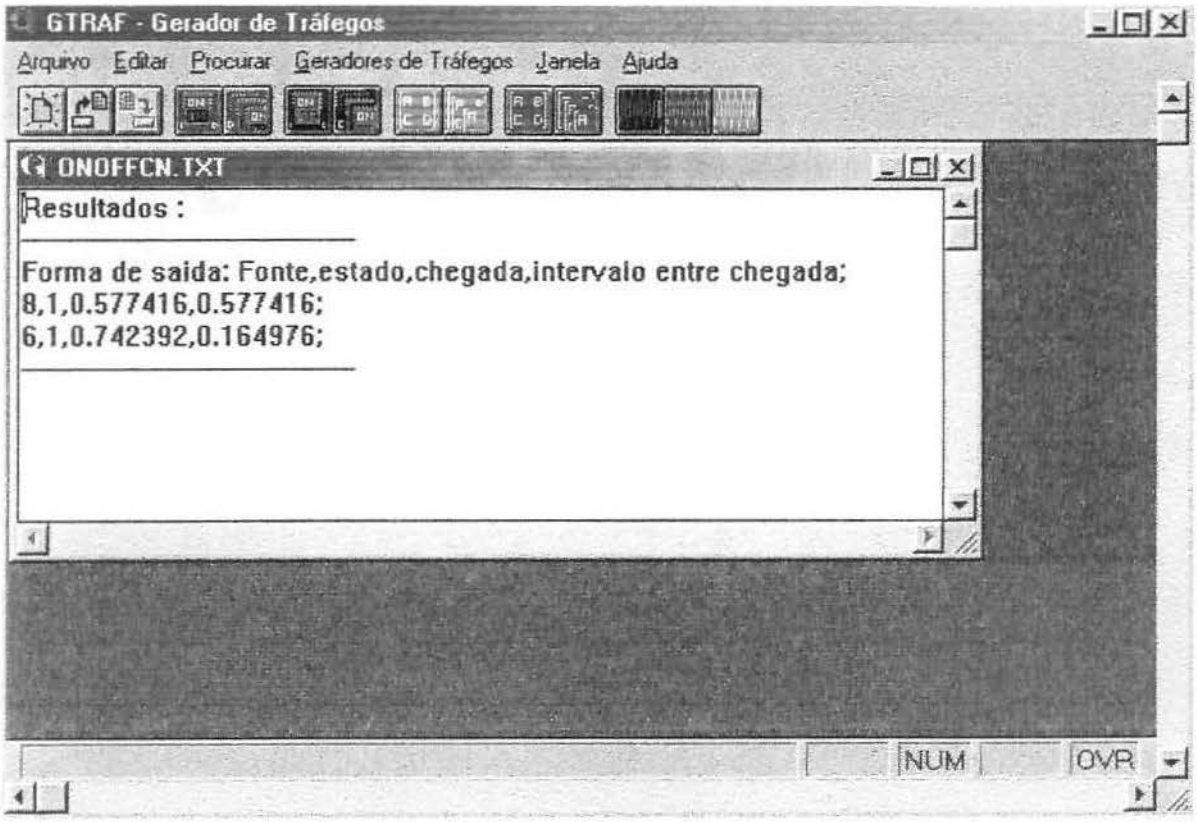
{Número de Fontes}
{Número de ciclos}

Para cada fonte:

{Um número identificando a fonte}
{O estado inicial da fonte}
{Matriz 2x2 da fonte}
{Taxa média de chegada para a função exponencial}
{Vetor de médias de mudança para cada estado para a função exponencial}



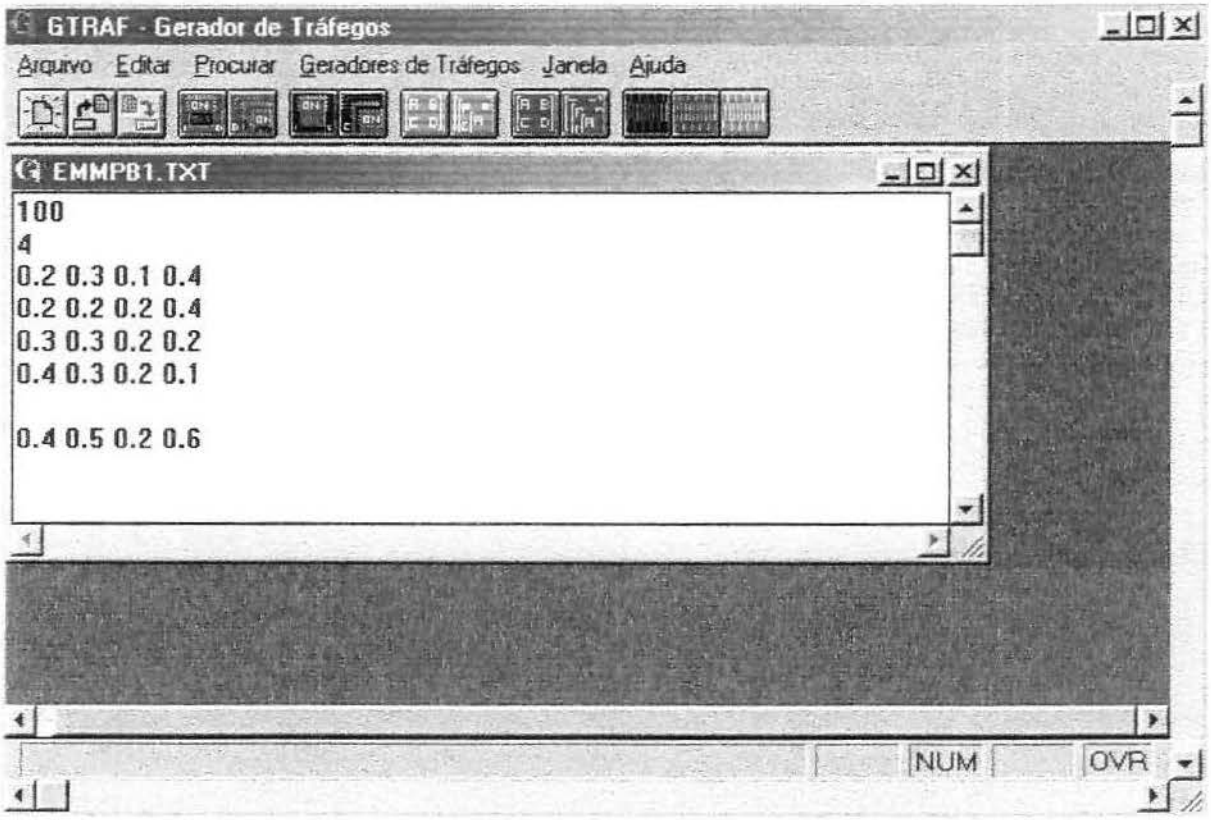
Os resultados são verificados no arquivo de saída **onoffcn.txt**.



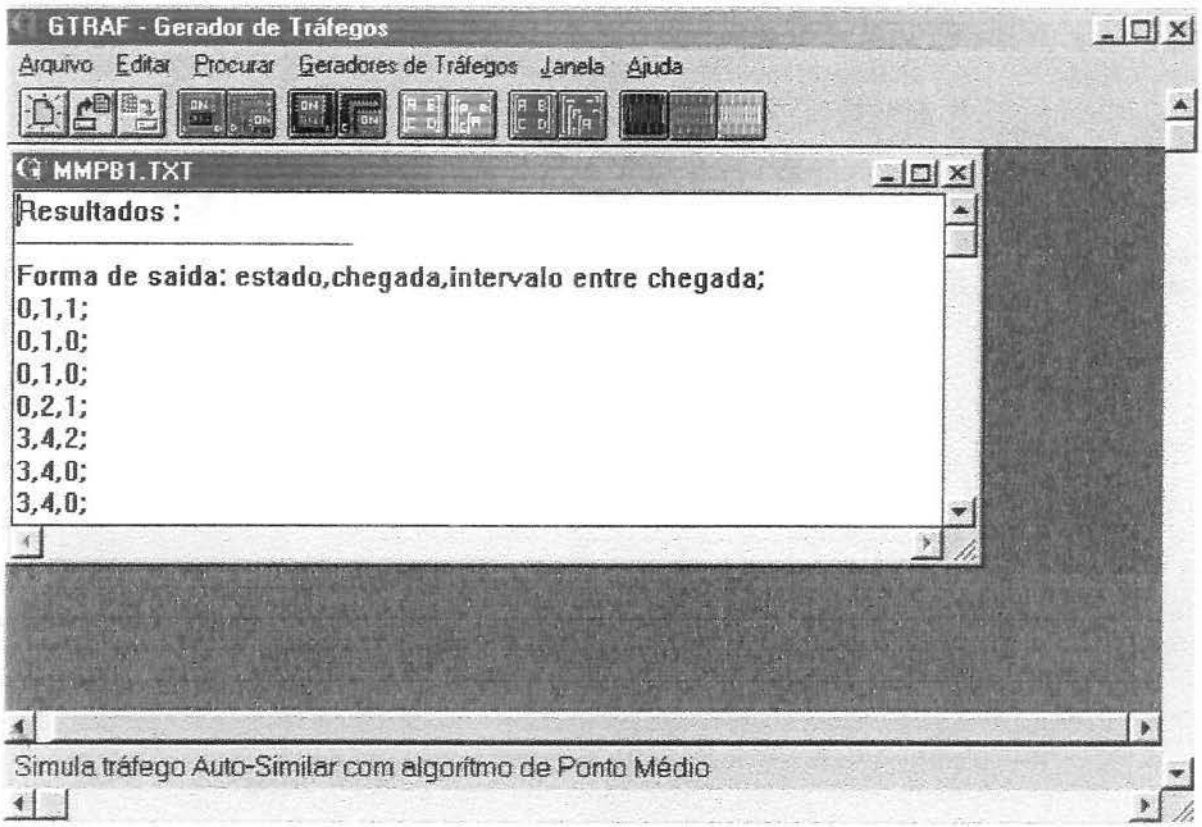
5) Modelo PBMM - 1 Fonte

Este programa possui um arquivo de entrada onde o tamanho dependerá de quantos estados a fonte terá. O arquivo de entrada chama-se **emmpb1.txt**. Este arquivo pode ser também criado. A entrada de dados é da forma:

{Número de ciclos}
{Número de estados}
{Matriz de transição de cada estado}
{Vetor de probabilidade de transição de cada estado}



Os resultados podem ser encontrados no arquivo **mmpb1.txt**.



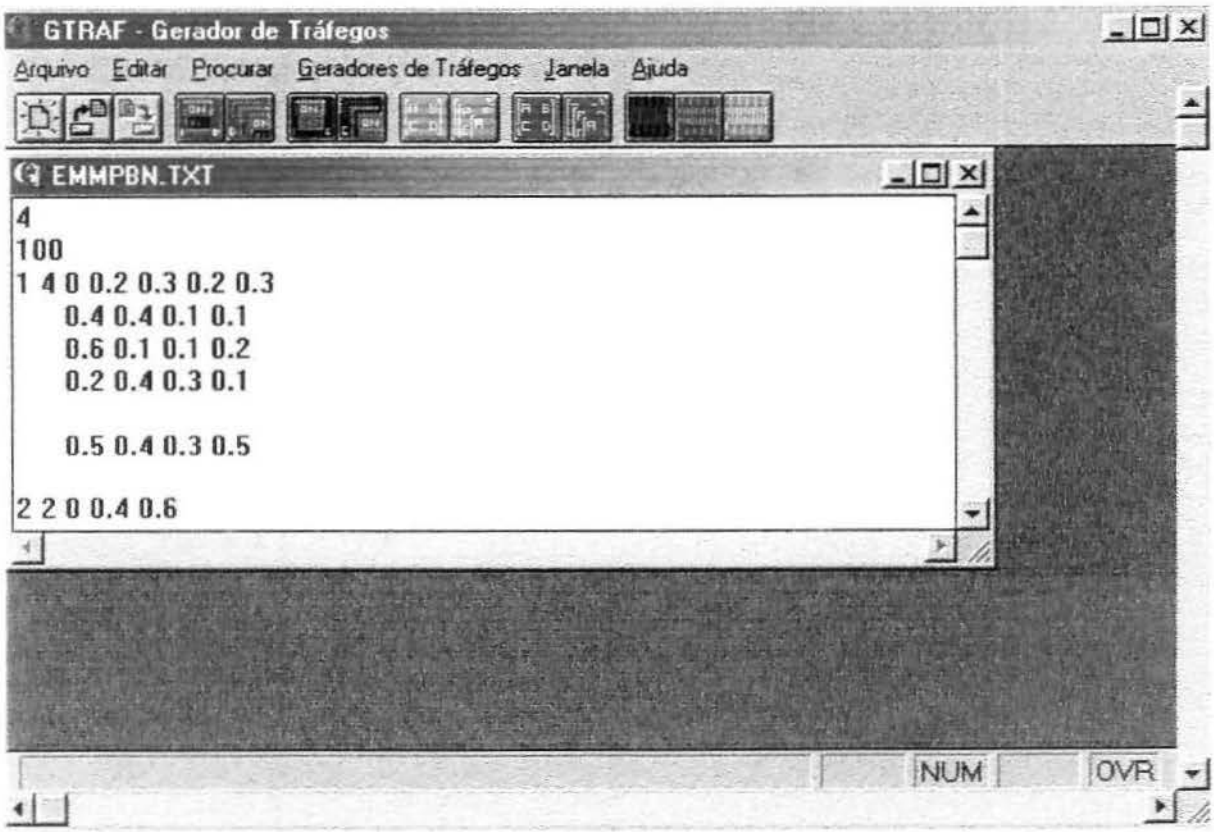
6) Modelo PBMM - N Fontes

Este programa possui um arquivo de entrada onde o tamanho dependerá do número de fontes presentes bem como quantos estados cada fonte terá. O arquivo de entrada chama-se **emmpbn.txt**. Este arquivo pode ser também criado. A entrada de dados é da forma:

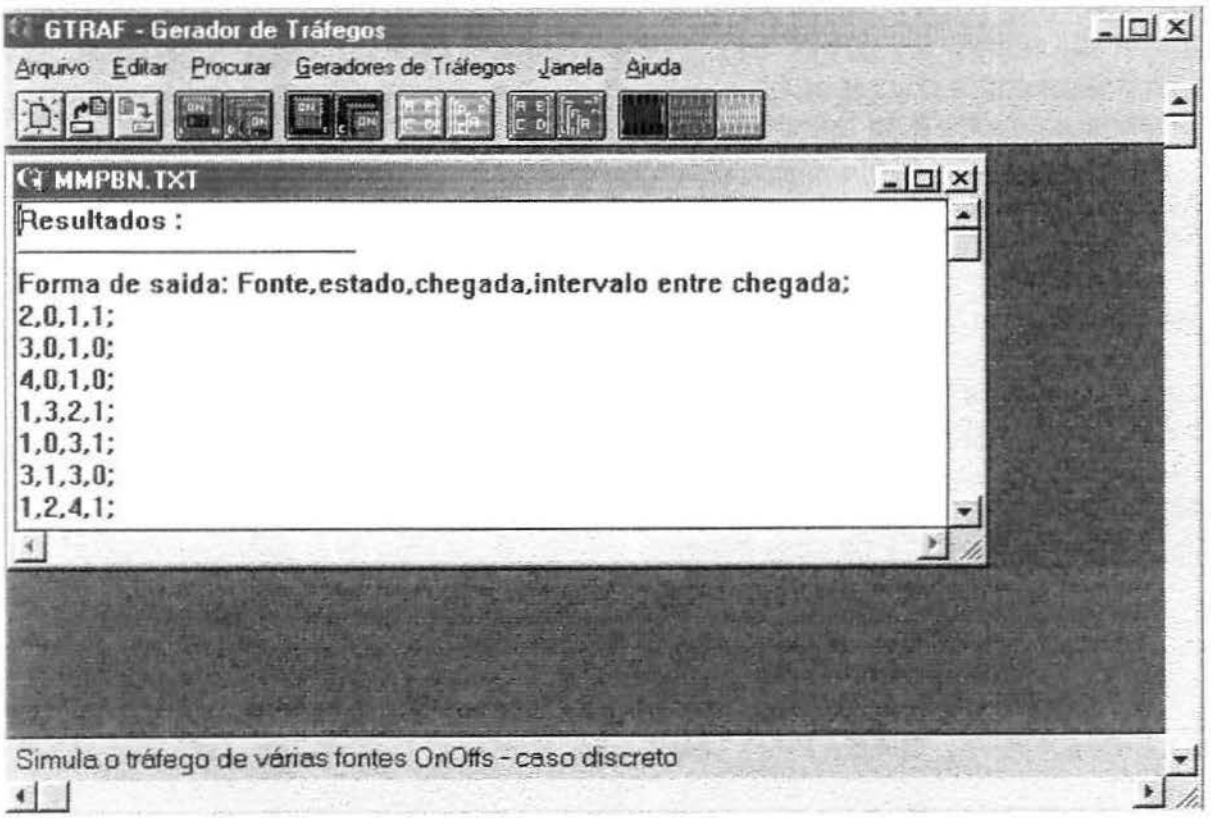
{Número de Fontes}
{Número de Ciclos}

Para cada fonte:

{Número da Fonte}
{Número de estados}
{Estado atual}
{Matriz de Transição}
{Vetor de probabilidade de chegadas}



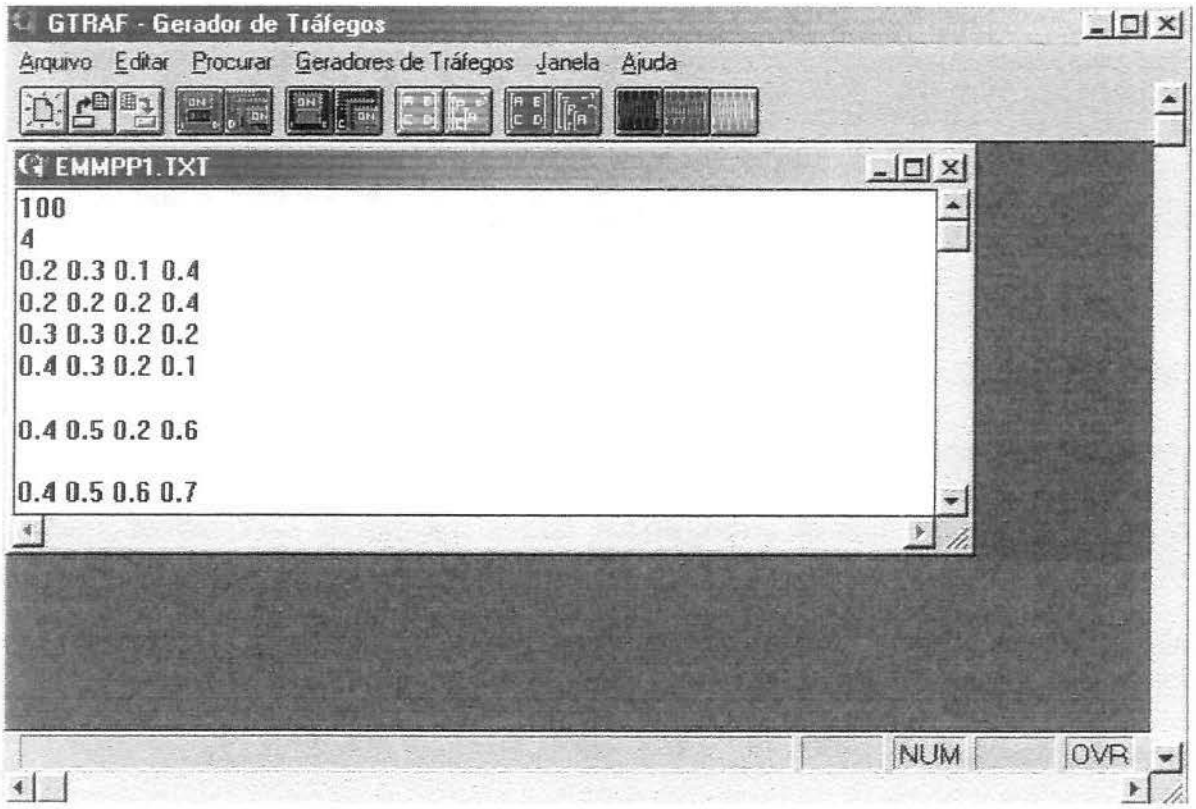
Os resultados podem ser encontrados no arquivo **mmpbn.txt**.



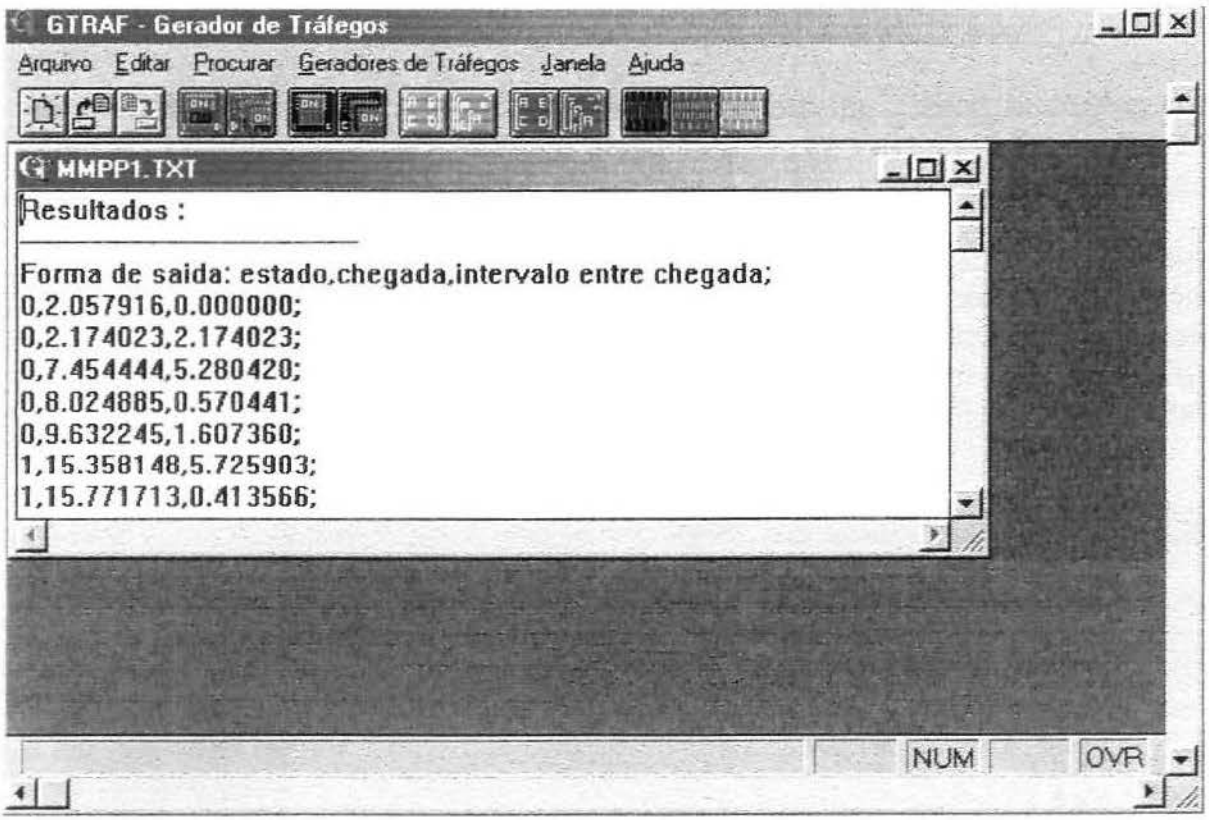
7) Modelo PPMM - 1 Fonte

Este programa possui um arquivo de entrada onde o tamanho dependerá de quantos estados a fonte terá. O arquivo de entrada chama-se **emmpp1.txt**. Este arquivo pode ser também criado. A entrada de dados é da forma:

{Número de Ciclos}
{Número de estados}
{Matriz de Transição}
{Vetor com taxas médias de chegadas de cada estado para a função exponencial}
{Vetor com taxas médias de mudanças de cada estado para a função exponencial
}



Os resultados podem ser encontrados no arquivo **mmpp1.txt**.



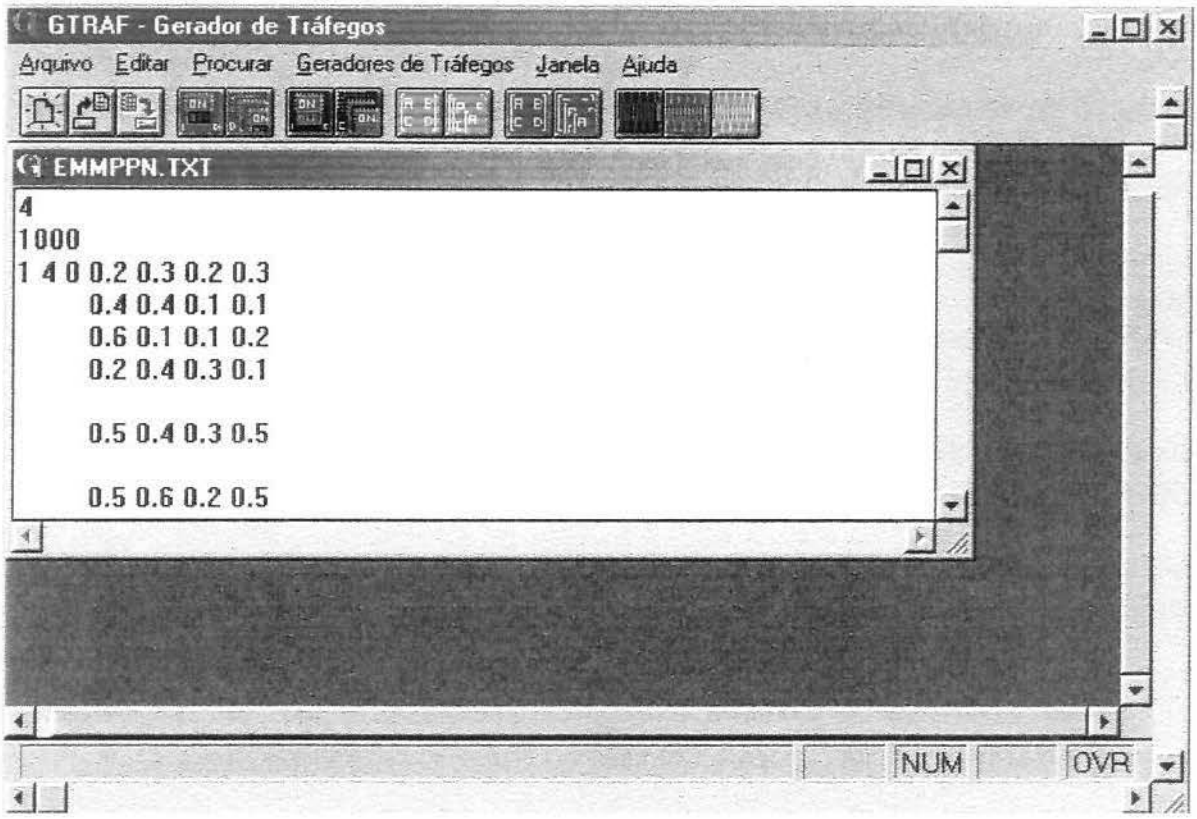
8) Modelo PPMM - N Fontes

Este programa possui um arquivo de entrada onde o tamanho dependerá do número de fontes presentes bem como quantos estados cada fonte terá. O arquivo de entrada chama-se **emmppn.txt**. Este arquivo pode ser também criado. A entrada de dados é da forma:

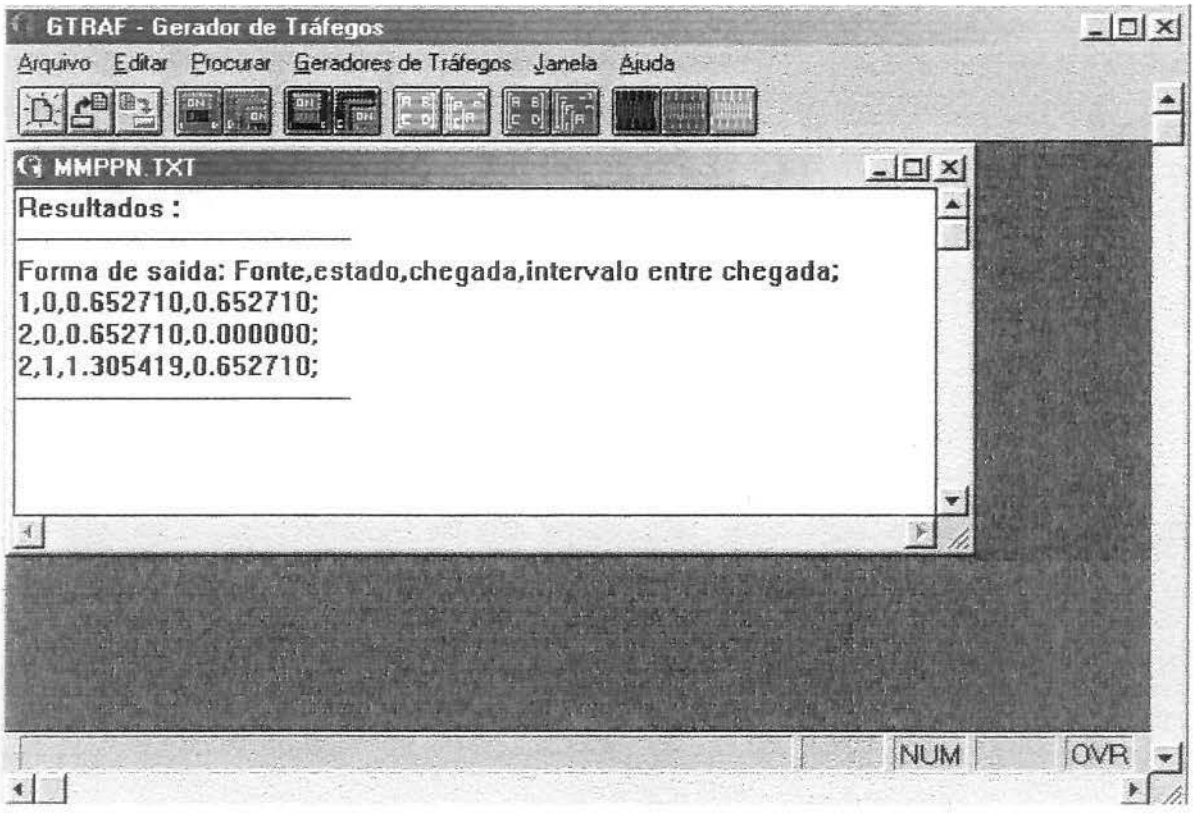
{Número de Fontes}
{Número de Ciclos}

Para cada fonte:

{Número da fonte}
{Número de estados}
{Estado atual da fonte}
{Matriz de Transição}
{Vetor com taxas médias de chegadas de cada estado para a função exponencial}
{Vetor com taxas médias de mudanças de cada estado para a função exponencial}



Os resultados podem ser encontrados no arquivo **mmppn.txt**.



9) Modelo Auto-Semelhante (1)

O arquivo de entrada para este simulador é bastante simples. O nome do arquivo é **elau.txt**, e pode também ser criado de um documento novo. Os valores de entrada deste simulador são:

{Número de chegada por ciclo}
{Taxa de pico}
{Parâmetro de Hurst}
{Número de ciclos}

Estes valores de entrada são importantes para a geração de traços auto_semelhantes baseados nas equações:

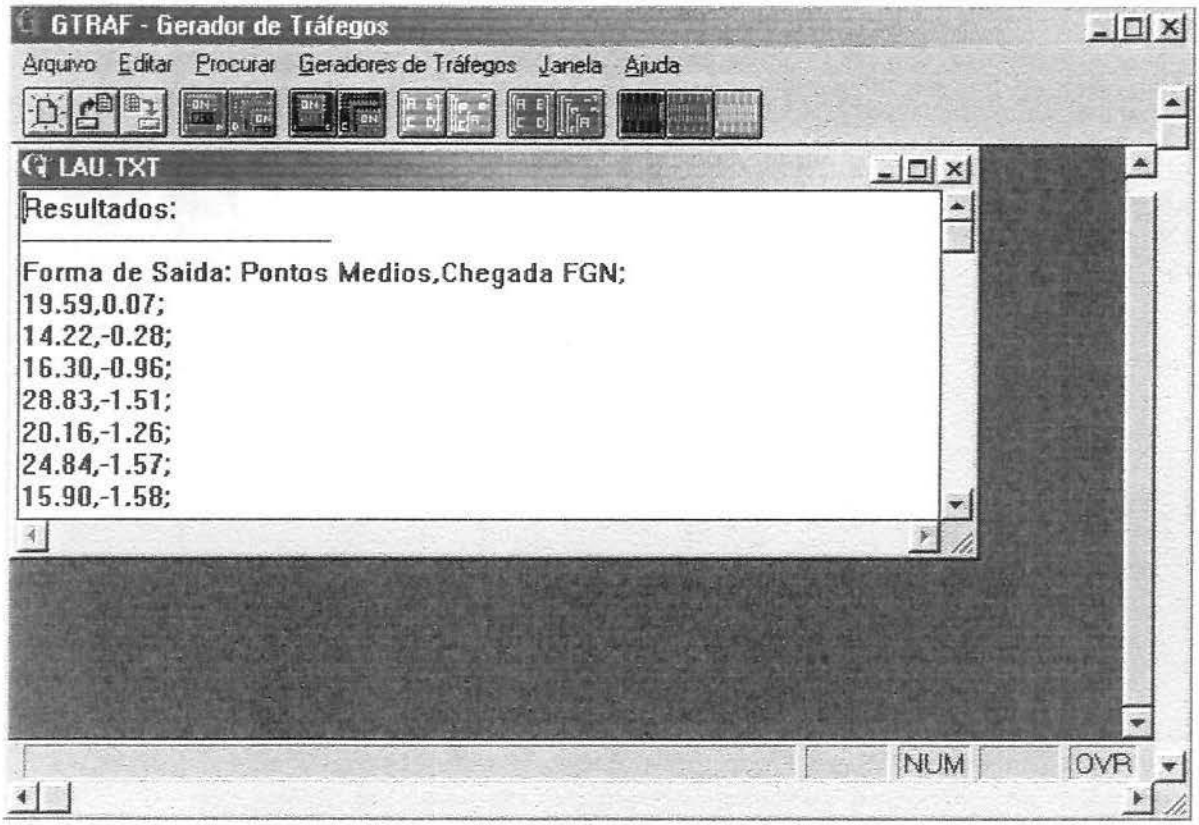
RMD (Random Midpoint Displacement) ou Deslocamento de Ponto Médio Aleatório:

$$A(t) = Mt + Raiz(aMZ(t))$$

E o traço FGN (Fractional Gaussian Noise) ou Ruído Gaussiano Fracional:

$$A'(t) = M + Raiz(aM)[Z(t+1) - Z(t)]$$

O resultado são traços Auto-Semelhantes apresentados no arquivo **lau.txt**.



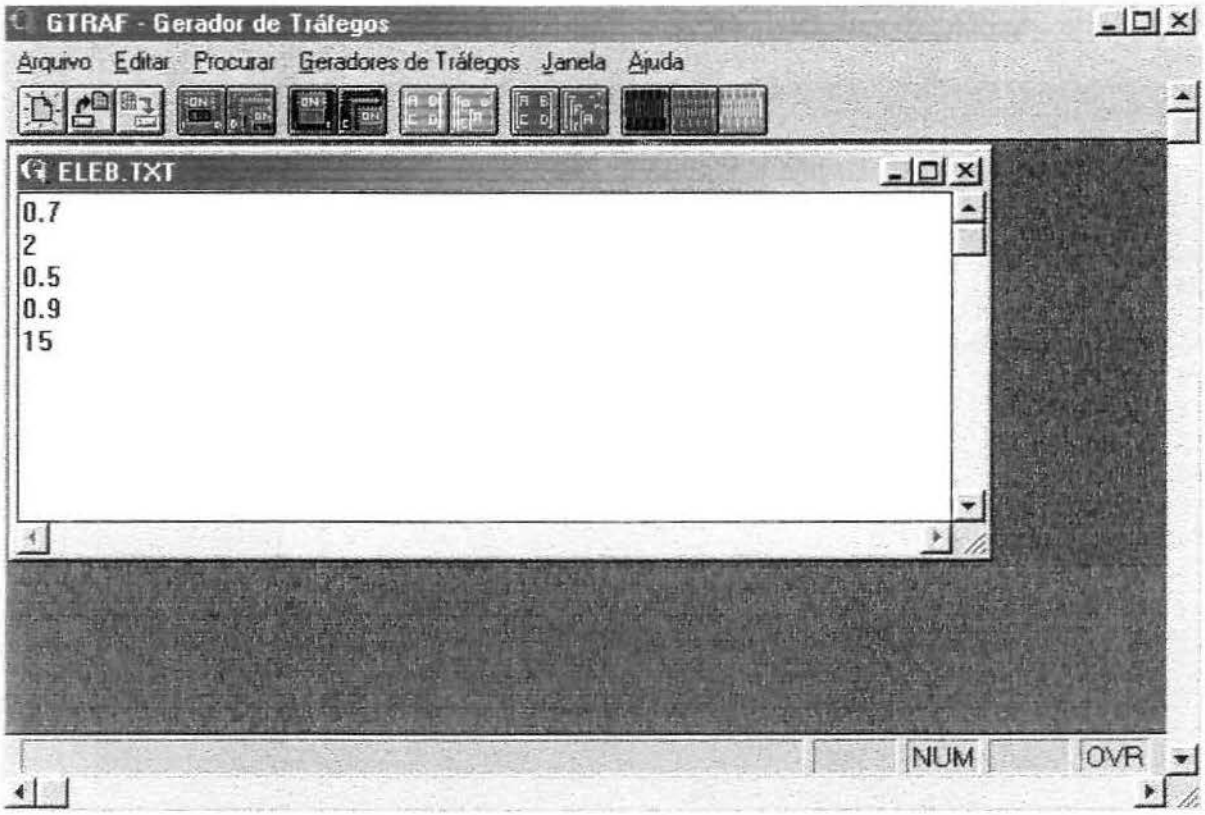
10) Modelo Auto-Semelhante (2)

O arquivo de entrada para este simulador também é bastante simples. O nome do arquivo é **eleb.txt**, e pode também ser criado de um documento novo. Os valores de entrada deste simulador são:

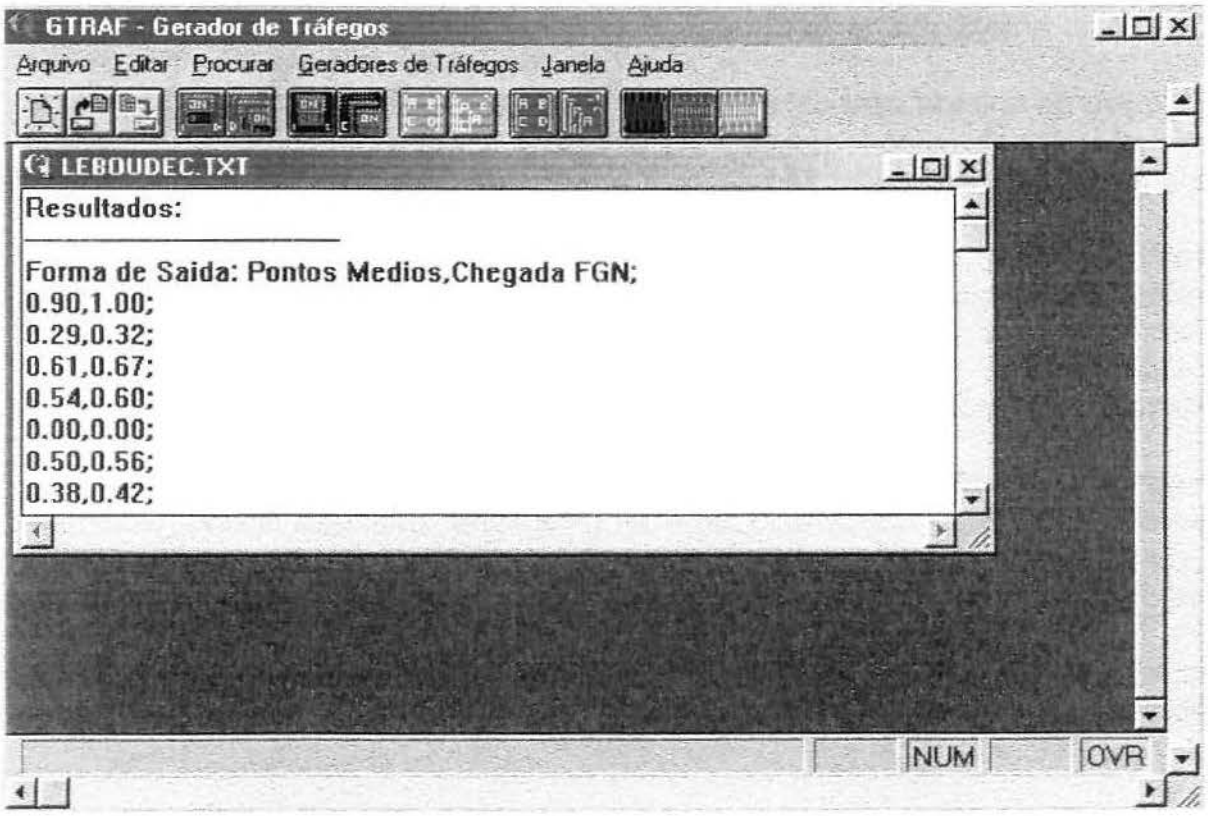
{Parâmetro de Hurst}
{Velocidade de Ligação}
{Fator de excesso de carga}
{Taxa de pico}
{Número de passos por ciclo }

Segundo Le Boudec et al, depois do traço FBM (Fractional Brownian Motion) é gerado deverá ser transformado em FGN (Fractional Gaussian Noise). Obtém-se o ruído (noise) pela escala e traduzindo ao FGN baseado nos valores de Velocidade de ligação, Fator de excesso e a taxa de pico.

Os autores consideram a escala e a tradução como uma transformação linear, não alterando a correlação da estrutura.



O resultado encontrado é verificado no arquivo de saída **leboudec.txt**.



11) Modelo Auto-Semelhante (3)

O arquivo de entrada para este simulador também é bastante simples. O nome do arquivo é **efgn.txt**, e pode também ser criado de um documento novo. Os valores de entrada deste simulador são:

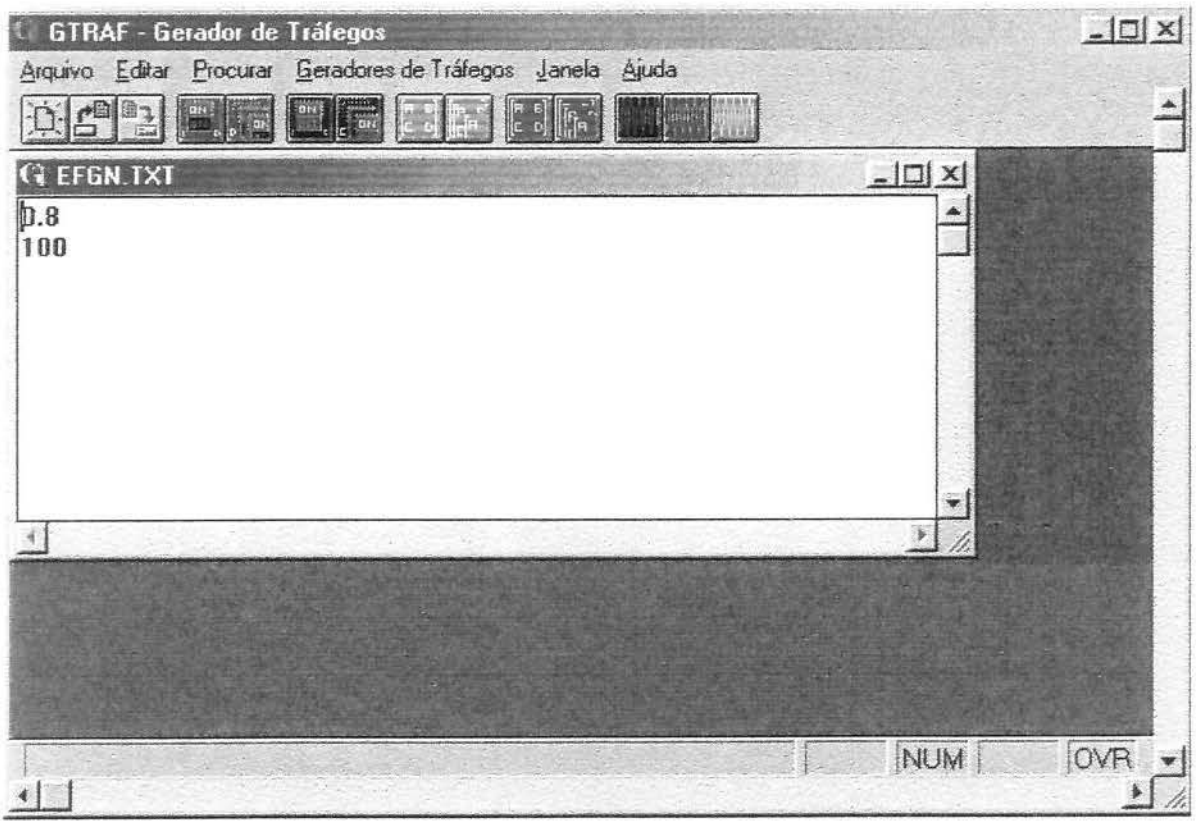
{Parâmetro de Hurst}
{Número de Amostras}

Baseado no algoritmo de Hosking mostrado na dissertação, traço FGN (Fractional Gaussian Noise) podem ser gerados . Para isto, utilizou-se a equação de Autocorrelação:

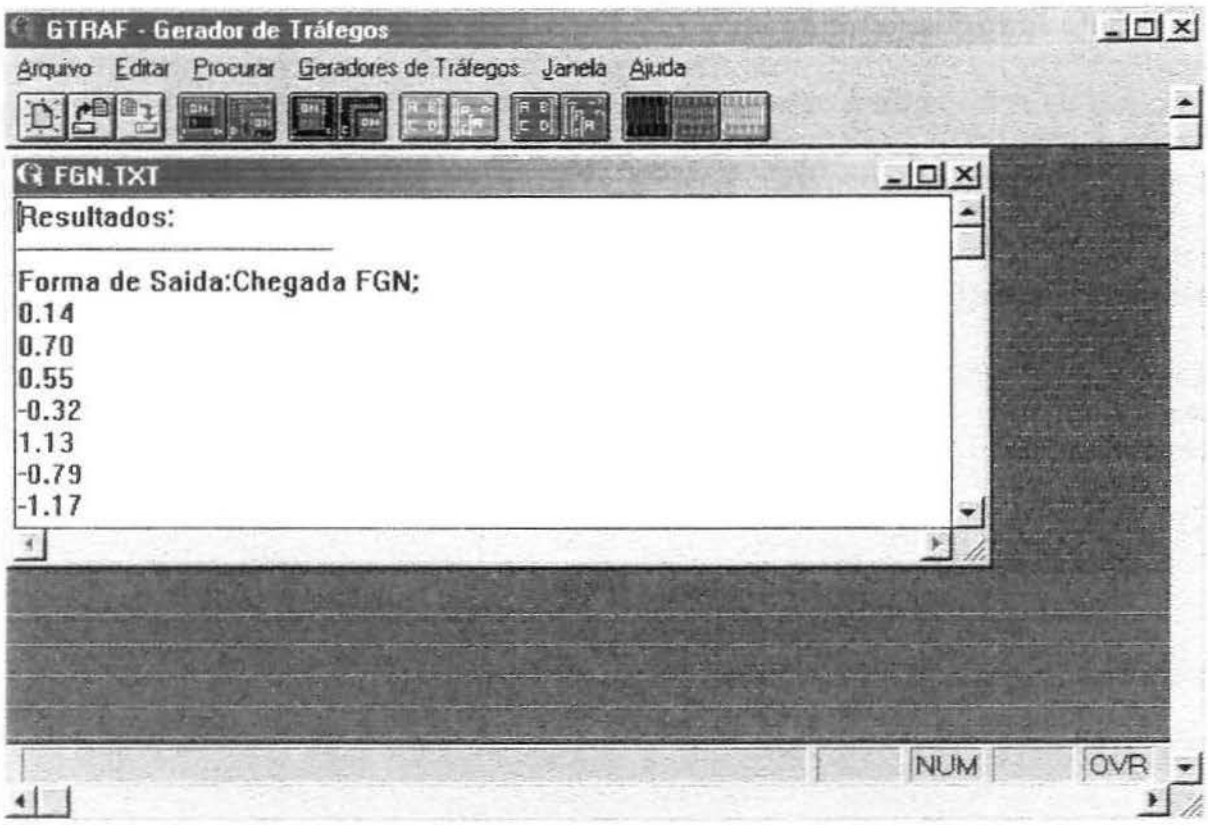
$\tau(k) = \frac{1}{2}(|k+1|2H - 2[k]2H + |k-1|2H)$, k=1,2,3,...

e

valores de uma distribuição Normal (m_k,v_k).



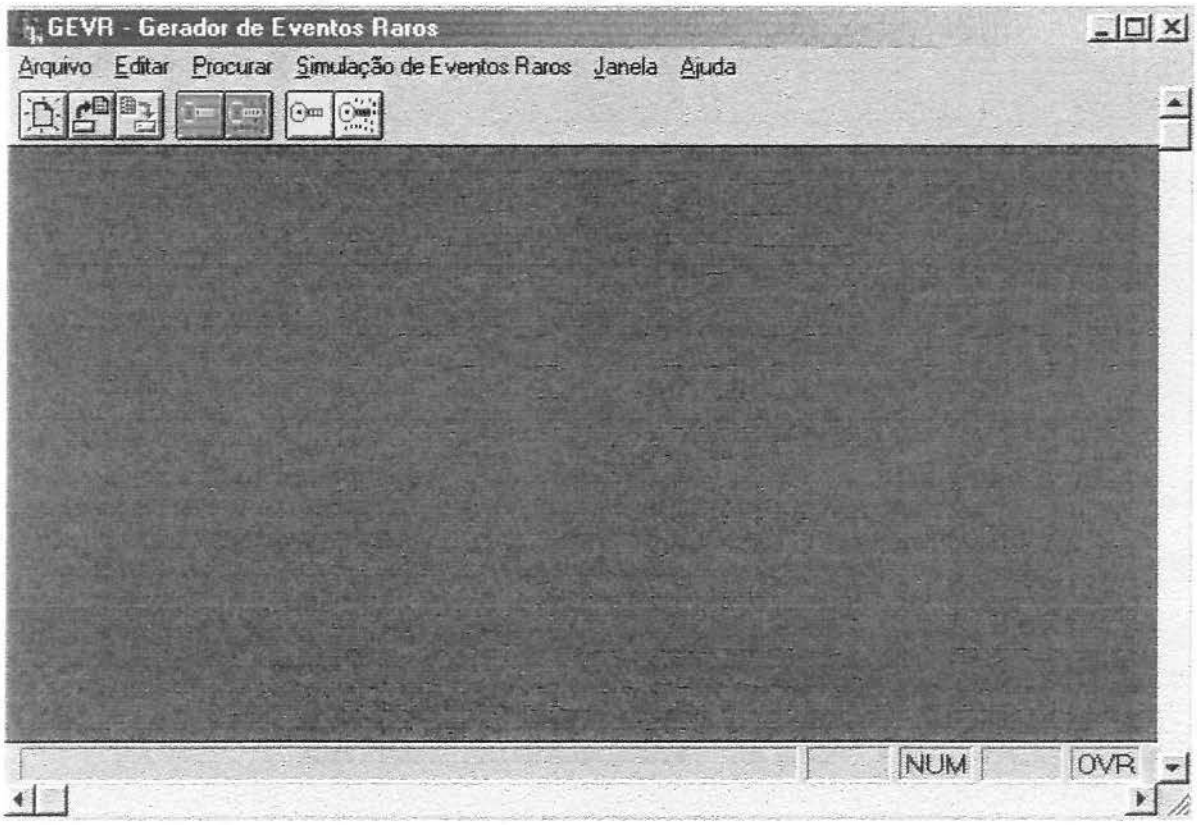
Os resultados são mostrados no arquivo **fgn.txt**.


GTRAF - Gerador de Tráfegos
Arquivo Editar Procurar Geradores de Tráfegos Janela Ajuda
FGN.TXT
Resultados:
Forma de Saida:Chegada FGN;
0.14
0.70
0.55
-0.32
1.13
-0.79
-1.17
NUM
OVR

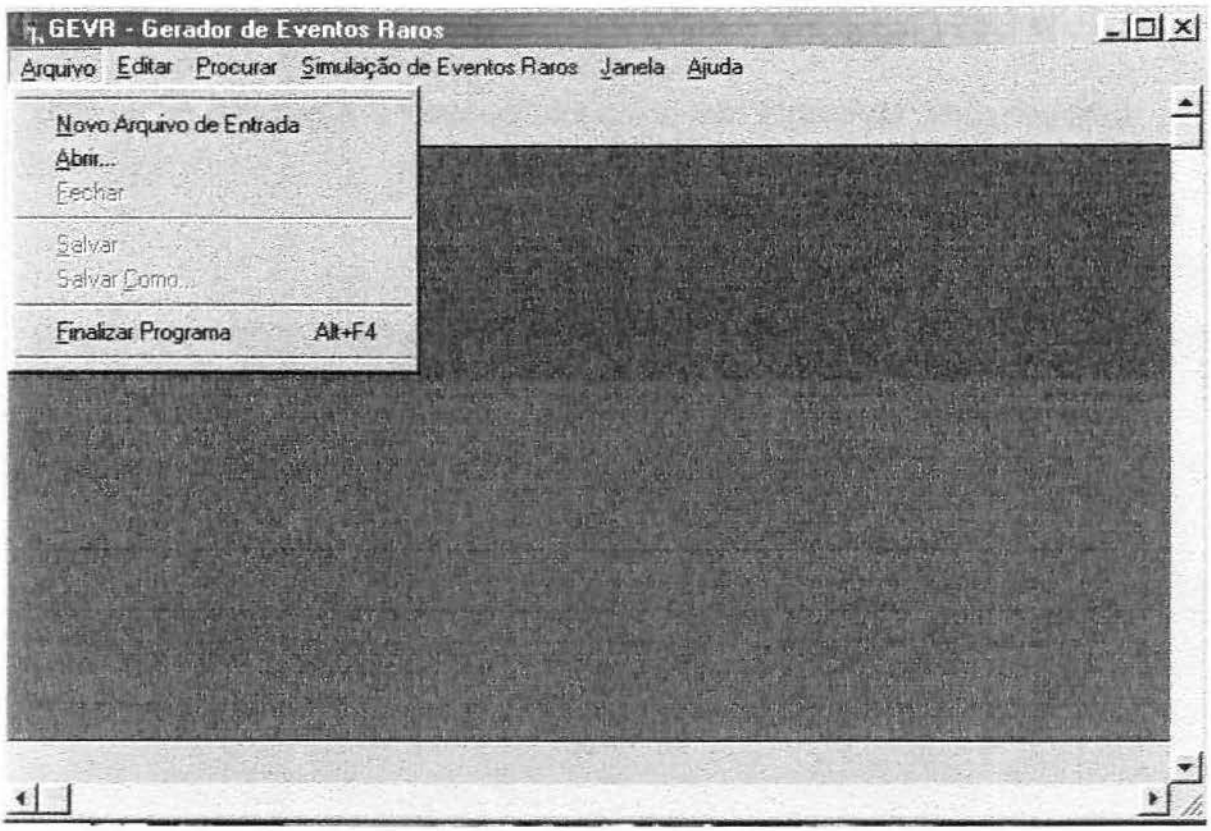
A.1 GEVR - O Gerador de Eventos Raros

Este programa apresenta tem como intuito, simular a geração de eventos raros.

GEVR.



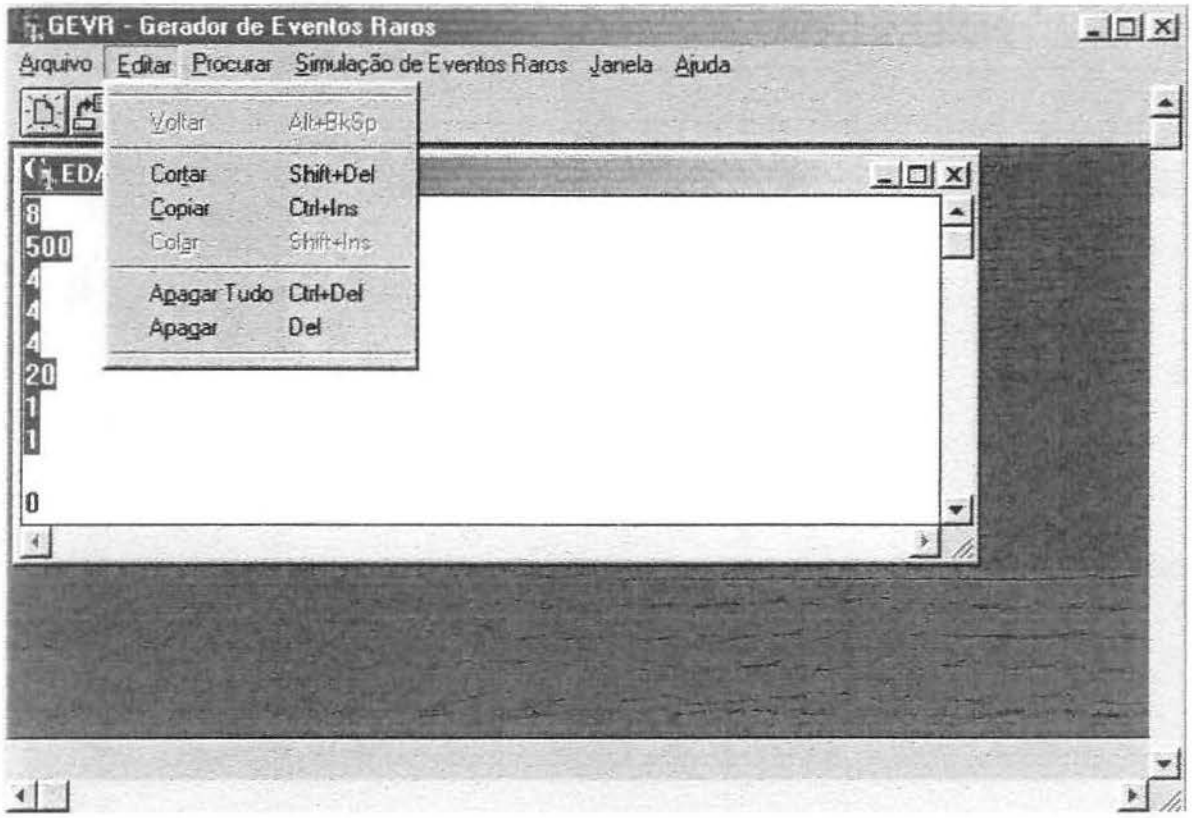
A tela principal do GEVR é composta de um menu principal pelo qual, apresenta-se um submenu de acesso aos programas e uma barra de ferramentas para acesso mais rápido.



Elementos do Menu Arquivo:

O Menu Arquivo fornece comandos para a criação de novos arquivos, abertura de arquivos, gravação de arquivos e outros.

Novo	Cria um novo documento, sem nome.
Abrir	Abre um arquivo existente.
Fechar	Fecha o documento atual.
Salvar	Salva o documento se seu conteúdo foi alterado.
Salvar Como	Salva o documento atual com um outro nome.
Finalizar Programa	Sai do GTRAF.



Elementos do Menu Editar:

O Menu Editar fornece comandos para copiar, mover, apagar objetos ou textos.

Voltar Elimina a última ação.

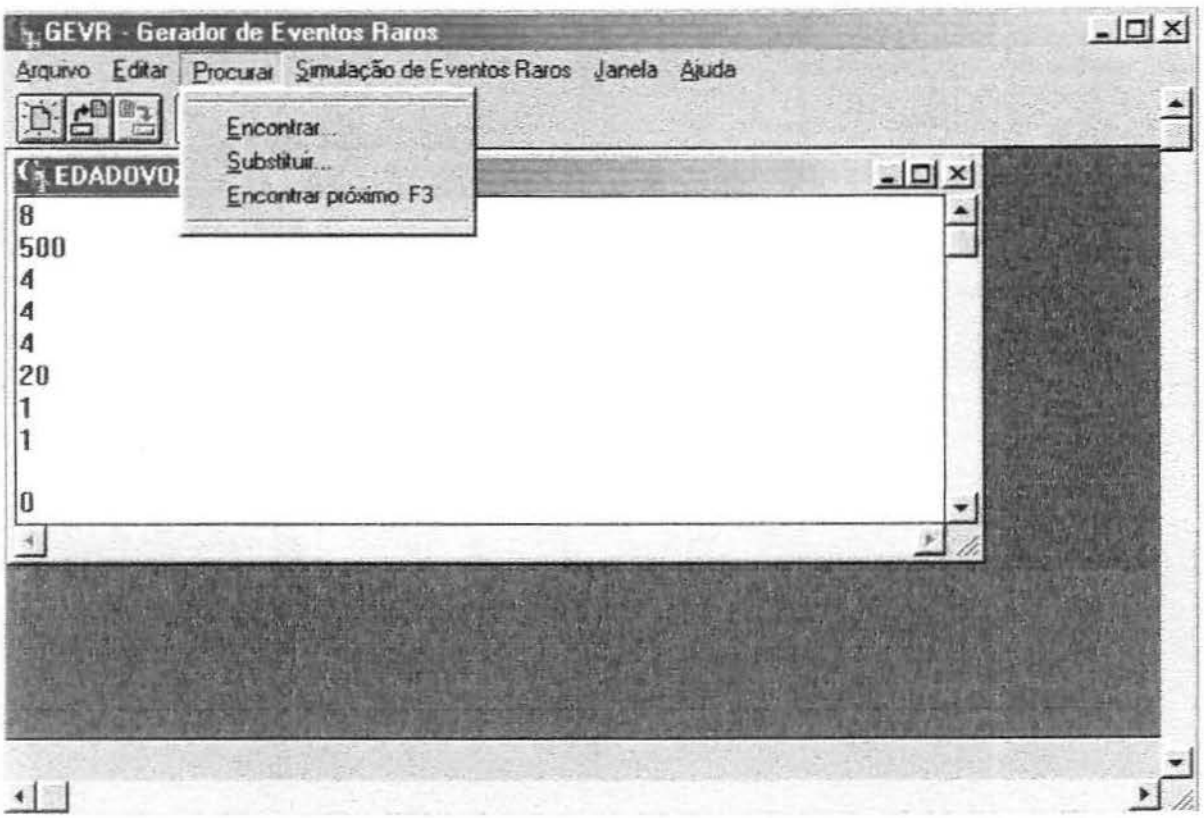
Cortar Recorta do texto um objeto selecionado.

Copiar Copia do texto um objeto selecionado.

Colar Cola no texto um objeto copiado ou cortado.

Apagar tudo Limpa todo o texto.

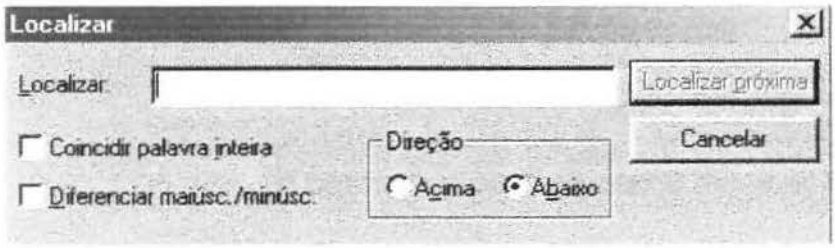
Apagar Limpa o texto selecionado.



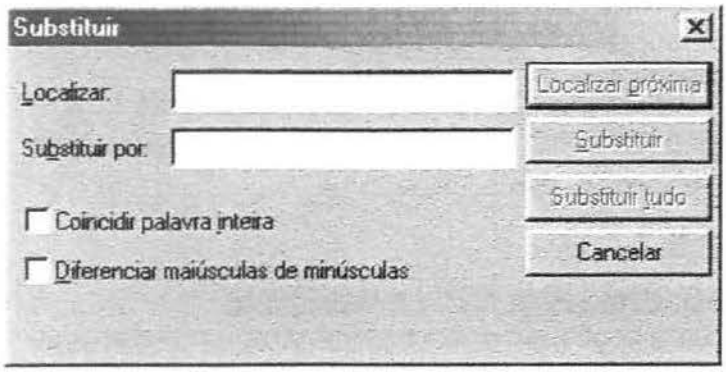
Elementos do Menu Procurar:

O Menu Procurar fornece comandos para procurar e substituir textos.

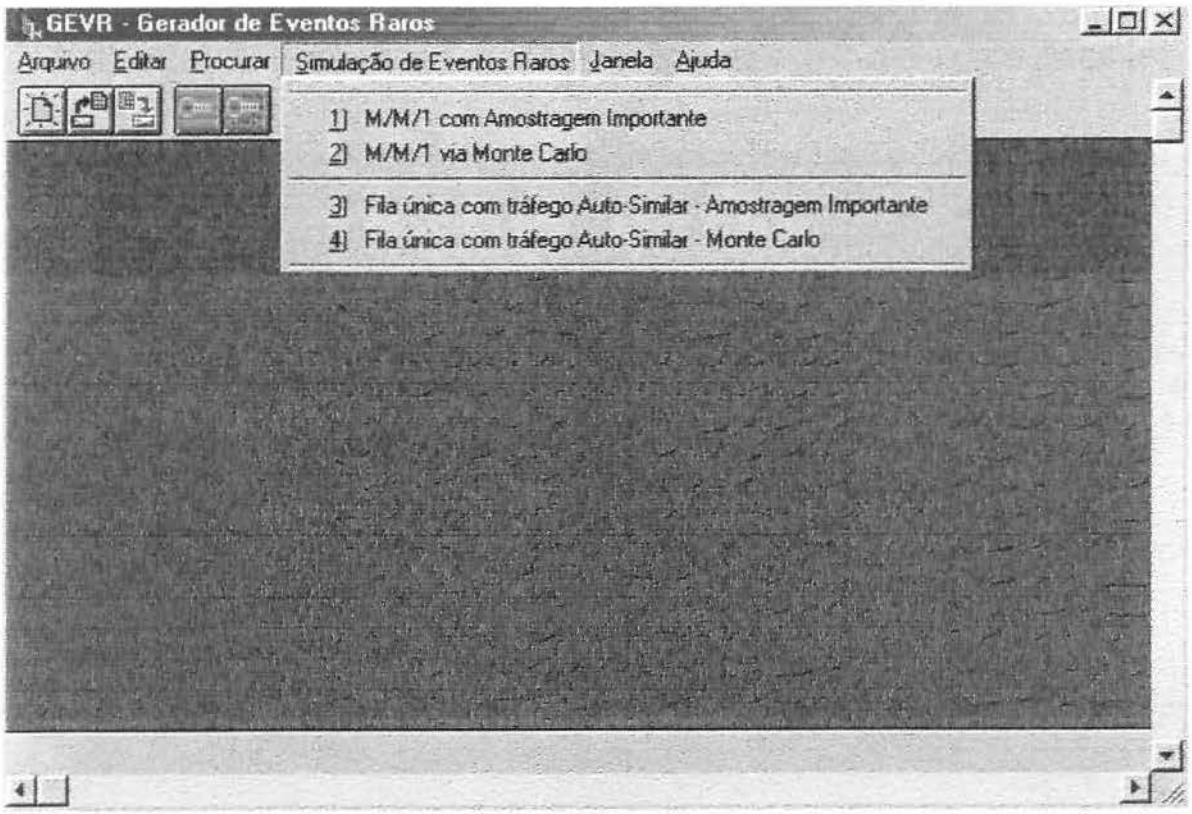
Localizar Procura um determinado texto.



Substituir Procura um determinado texto.



Encontrar Próximo Permite refazer a ação de Encontrar ou Substituir.



Elementos do Menu Geradores de Eventos Raros

O Menu Geradores de Eventos Raros apresenta rotinas que simulam situações em redes de computadores baseadas em ATM que hajam perda de pacotes "Eventos Raros"

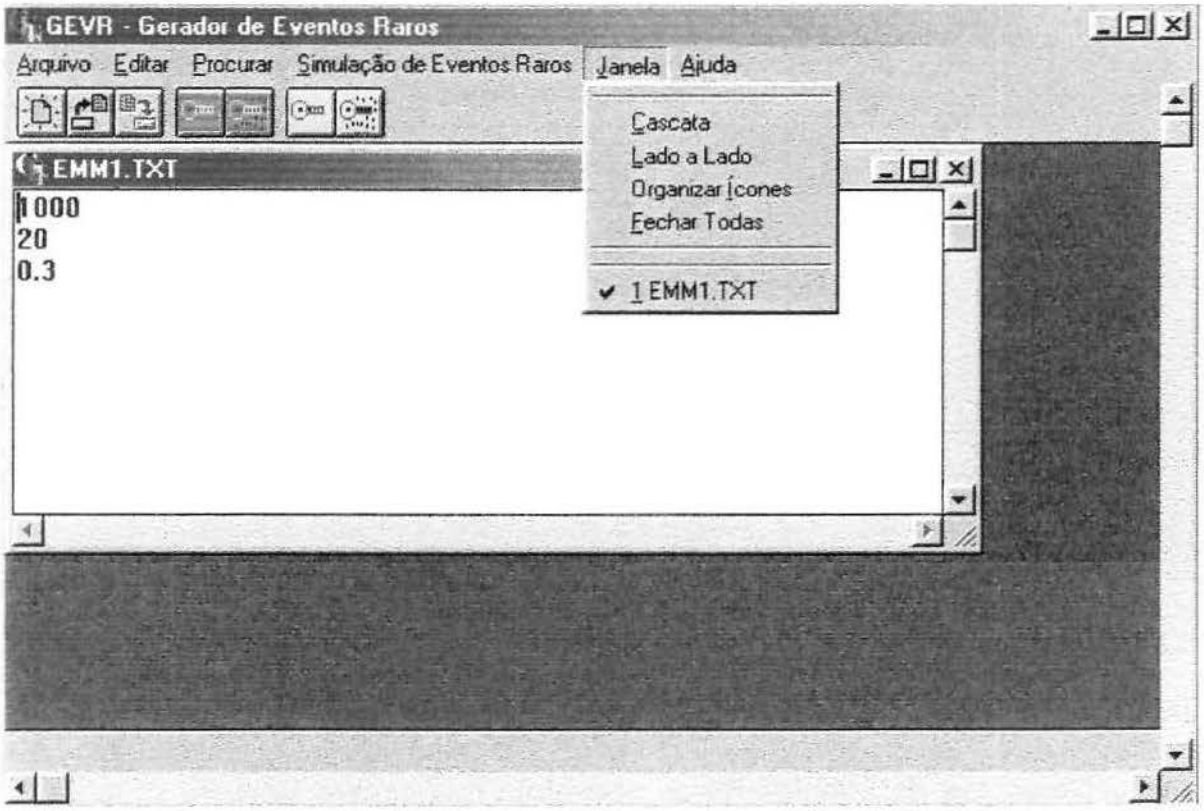
1) M/M/1 via Monte Carlo

Simula uma fila M/M/1 tradicionalmente tentando-se estimar também a taxa de perda de células. Serve como um comparador de resultados com o método da simulação M/M/1 com Amostragem Importante.

2) M/M/1 com amostragem Importante M/M/1 via Monte Carlo
Simula uma fila M/M/1 aplicando a teoria dos Desvios Largos juntamente com a técnica de Amostragem Importante para encontrar a estimativa de evento raro neste tipo de fila.

3) Fila única com tráfego Auto-Semelhante - Amostragem Importante
Aplica a simulação proposta por Huang et al aplicando-se Amostragem Importante em cima de um processo Auto-Semelhante baseado em traços FGN através do algoritmo de Hosking.

4) Fila única com tráfego Auto-Semelhante - Monte Carlo
Uma adaptação da simulação anterior com uma diferença, sem a aplicação de Amostragem Importante. Serve para comparação de resultados.



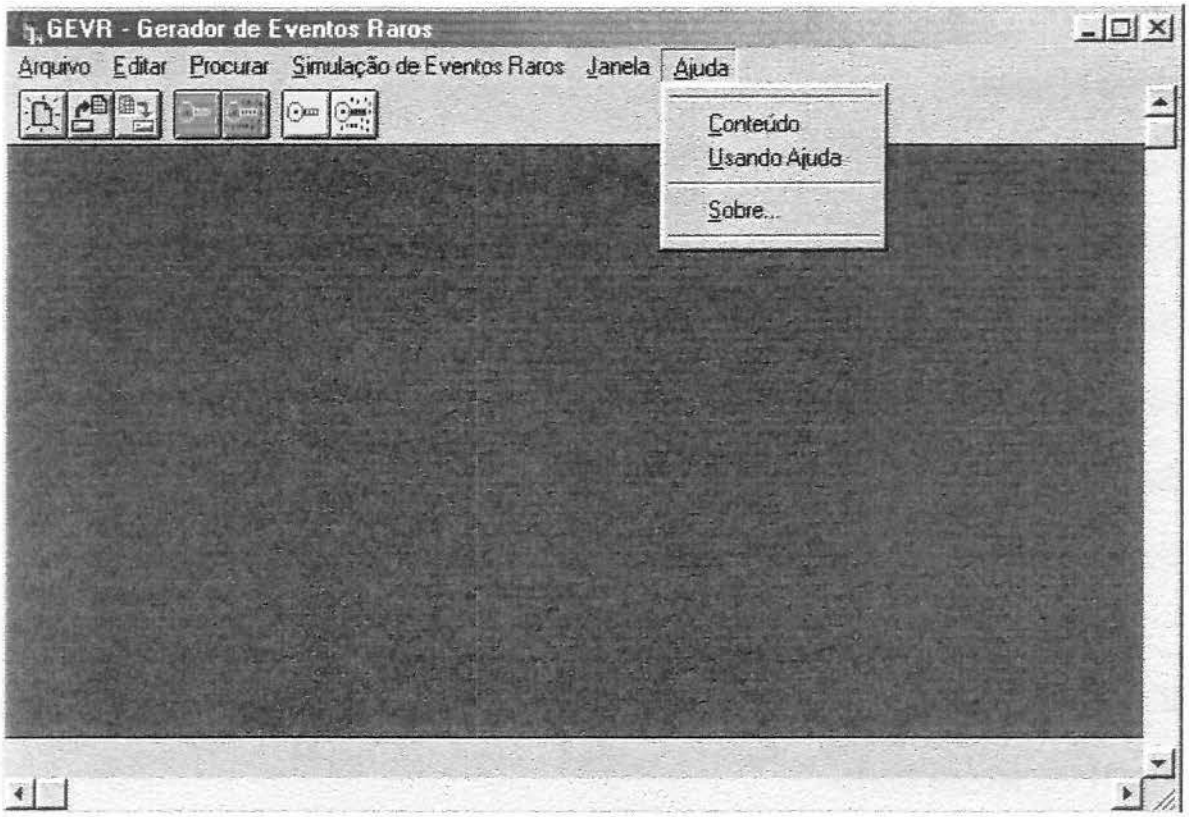
Elementos do Menu Janela:

O Menu Janela fornece comandos ao controlar o posicionamento e a organização das janelas do programa.

Cascata Coloca as janelas sobrepostas.

Lado à Lado Coloca as janelas Lado à Lado

Fechar Todas Fecha todas as janelas abertas.



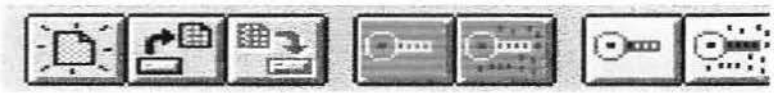
Elementos do Menu Ajuda:

Conteúdo Apresenta um Help à respeito do programa.

Usando Ajuda Apresenta uma explicação de como utilizar o Help.

Sobre Apresenta uma identificação do programa.

A Barra de Ferramentas



1 2 3 4 5 6 7

1: Novo Arquivo.

2: Abre Arquivo.

3: Salva Arquivo.

4: Simula M/M/1 - Monte Carlo.

5: Simula M/M/1 - Amostragem Importante.

6: Simula M/M/1 - Tráfego Auto-Semelhante - Monte Carlo.

7: Simula M/M/1 - Tráfego Auto-Semelhante - Amostragem Importante.

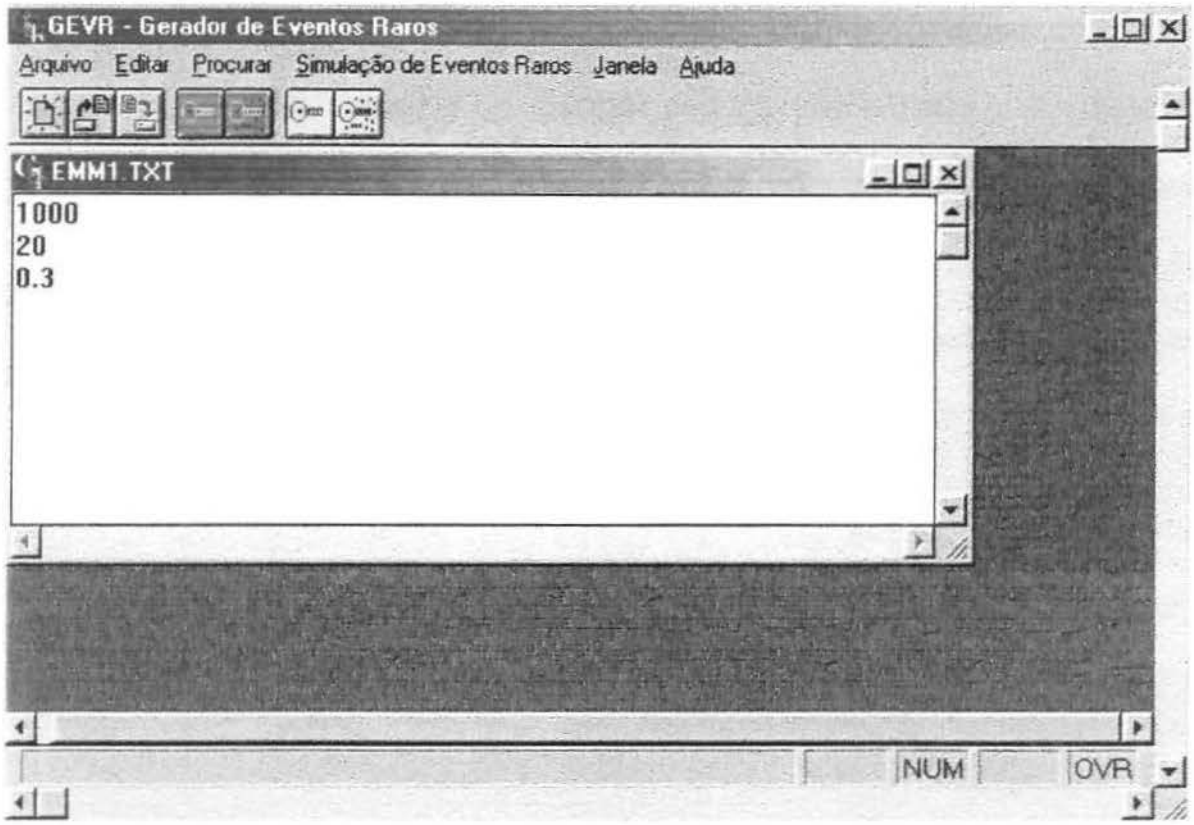
Os Simuladores.

1) M/M/1 via Monte Carlo

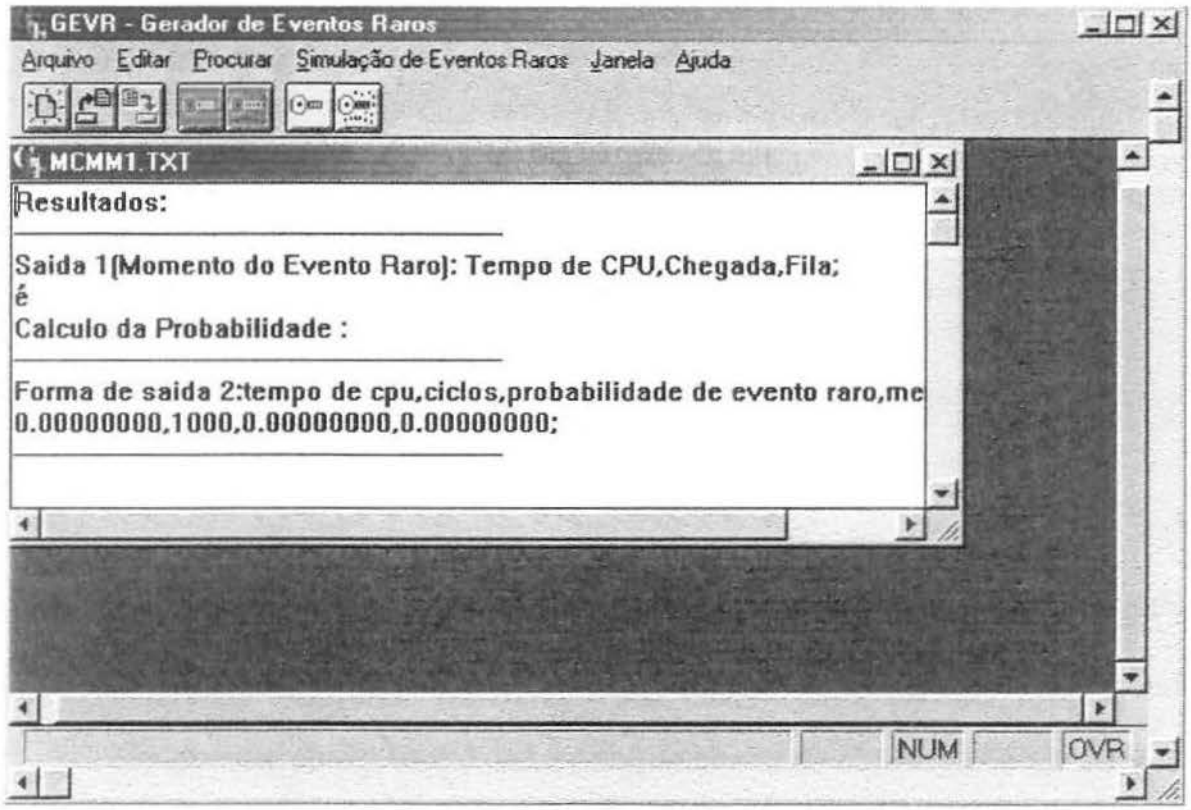
A entrada de dados deste programa é muito simples. Necessita apenas de três informações incluídas em um arquivo de entrada denominado **emm1.txt**. É possível também criar um novo com as mesmas entradas. As entradas são:

{Número de amostras}
{Tamanho da fila}
{Taxa média de chegada}

Não é necessário entrar com a taxa média de serviço porque foi determinado que a soma das taxas de chegada e serviço resulta 1. Logo, o programa realiza o tratamento das respectivas taxas à partir da taxa de chegada.



O resultado esperado é colocado em um arquivo chamado **mcmm1.txt**.



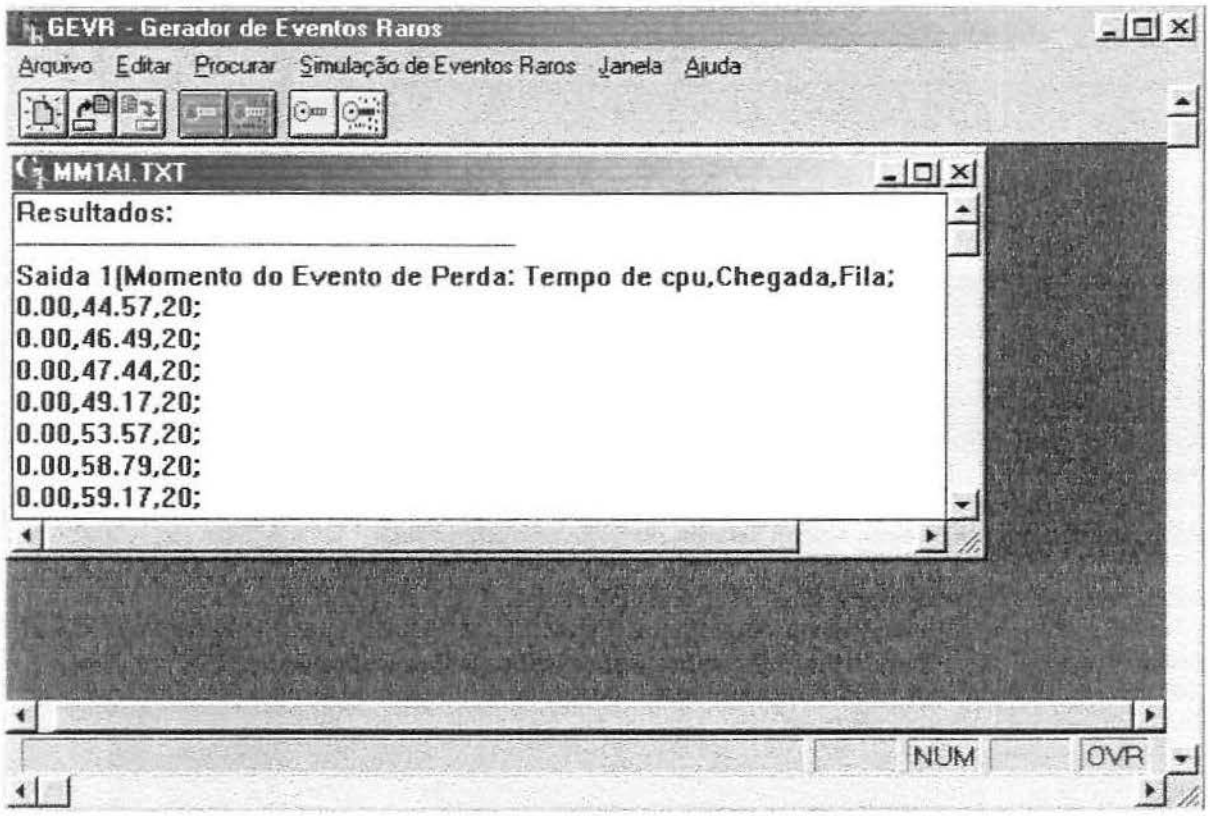
2) M/M/1 com amostragem Importante

A entrada de dados deste programa também é simples. Utiliza-se o mesmo arquivo de entrada do simulador anterior, **emm1.txt**. É possível também criar um novo com as mesmas entradas. As entradas são:

{Número de amostras}
{Tamanho da fila}
{Taxa média de chegada}

Mais uma vez, chama-se a atenção de que não é necessário entrar com a taxa média de serviço porque foi determinado que a soma das taxas de chegada e serviço resulta 1. Logo, o programa realiza o tratamento das respectivas taxas à partir da taxa de chegada.

O resultado esperado é colocado em um arquivo chamado **mm1ai.txt**.

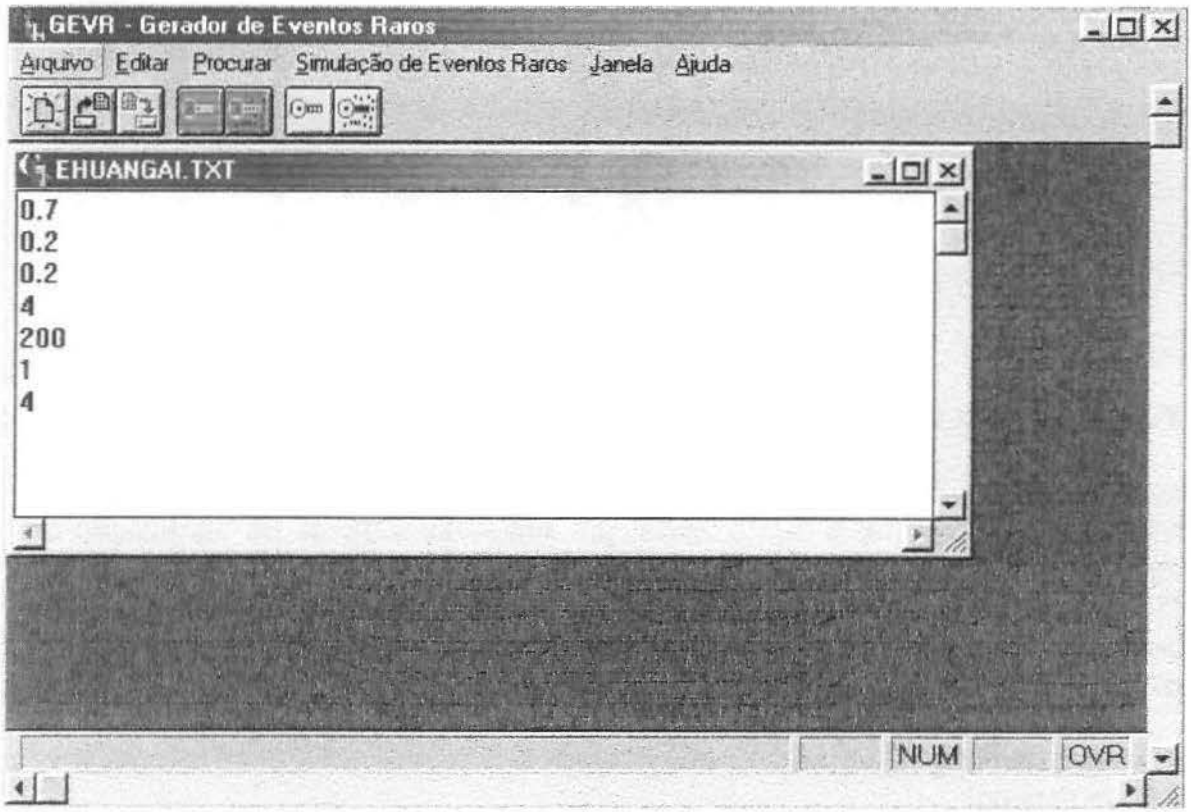


3) Fila única com tráfego Auto-Semelhante - Amostragem Importante

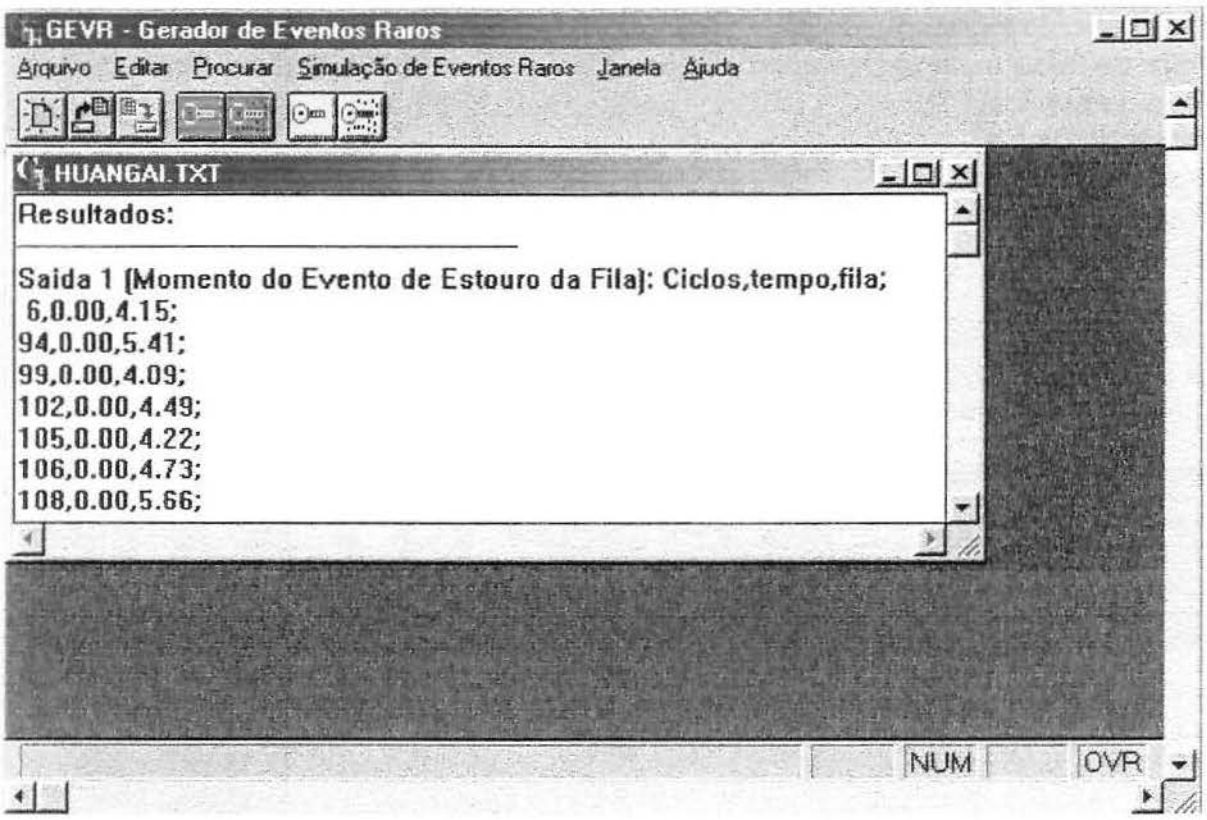
Este simulador é baseado no simulador descrito por Huang. A entrada de dados é um pouco maior que os dois primeiros simuladores, contudo, também é simples. O arquivo de entrada é denominado **ehuangai.txt**. Os dados de entrada são:

{Parâmetro de Hurst}
{Valor de "bias – Large Deviation"}
{Taxa de serviço}
{Tamanho da fila}
{Número de ciclos}
{Valor de incremento ao número de ciclos}
{Número de repetições da simulação}

O simulador comporta-se da seguinte forma. Repete-se a simulação várias vezes e a cada vez, o número de ciclos é incrementado por um valor, no final, verifica-se a probabilidade para variados ciclos.



O resultado esperado é colocado em um arquivo chamado **huangai.txt**.



4) Fila única com tráfego Auto-Semelhante - Monte Carlo

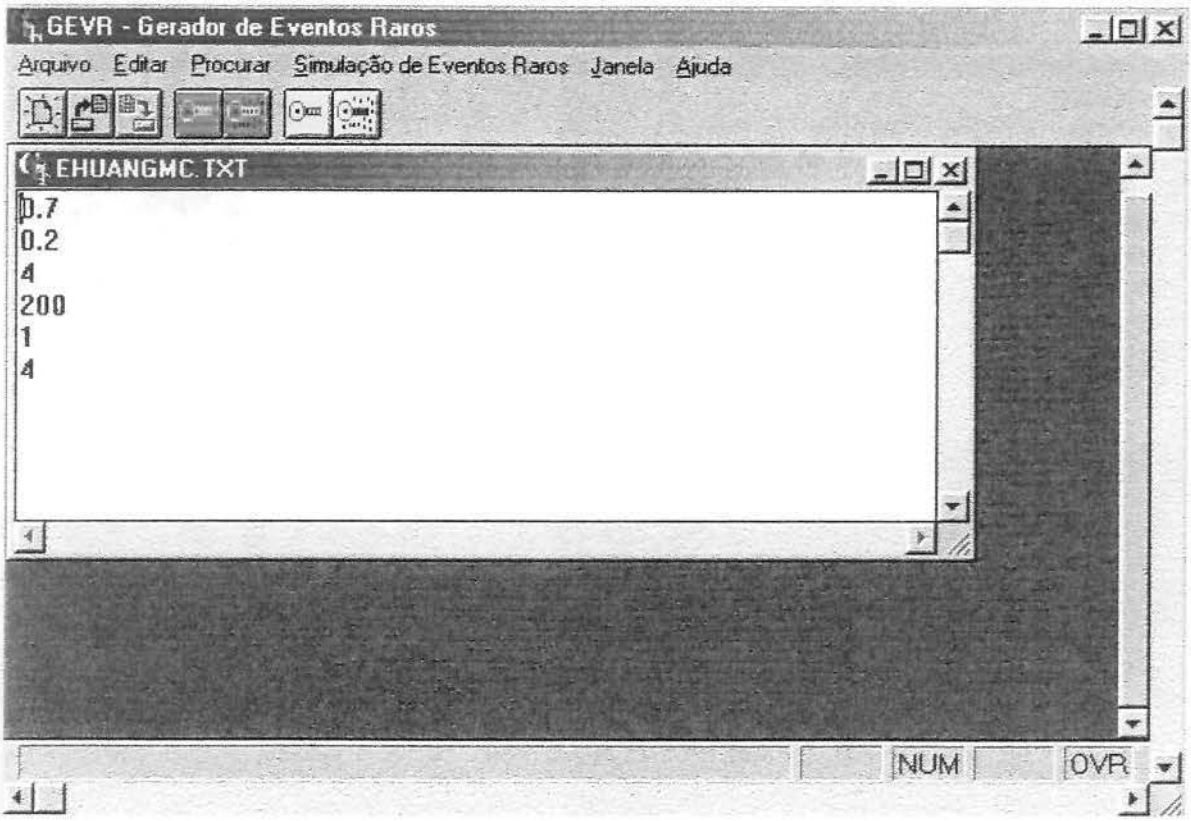
Este simulador é uma réplica do anterior. O detalhe está na ausência da técnica Amostragem Importante, o que o torna, um simulador simples de verificação de chegadas e fila. A entrada de dados é idêntica à do simulador anterior, porém, com uma diferença, não coloca-se o valor "Large Deviation" para a simulação rápida. O arquivo de entrada é denominado **ehuangmc.txt**. Os dados de entrada são:

{Parâmetro de Hurst}
{Taxa de serviço}
{Tamanho da fila}
{Número de ciclos}
{Valor de incremento ao número de ciclos}
{Número de repetições da simulação}

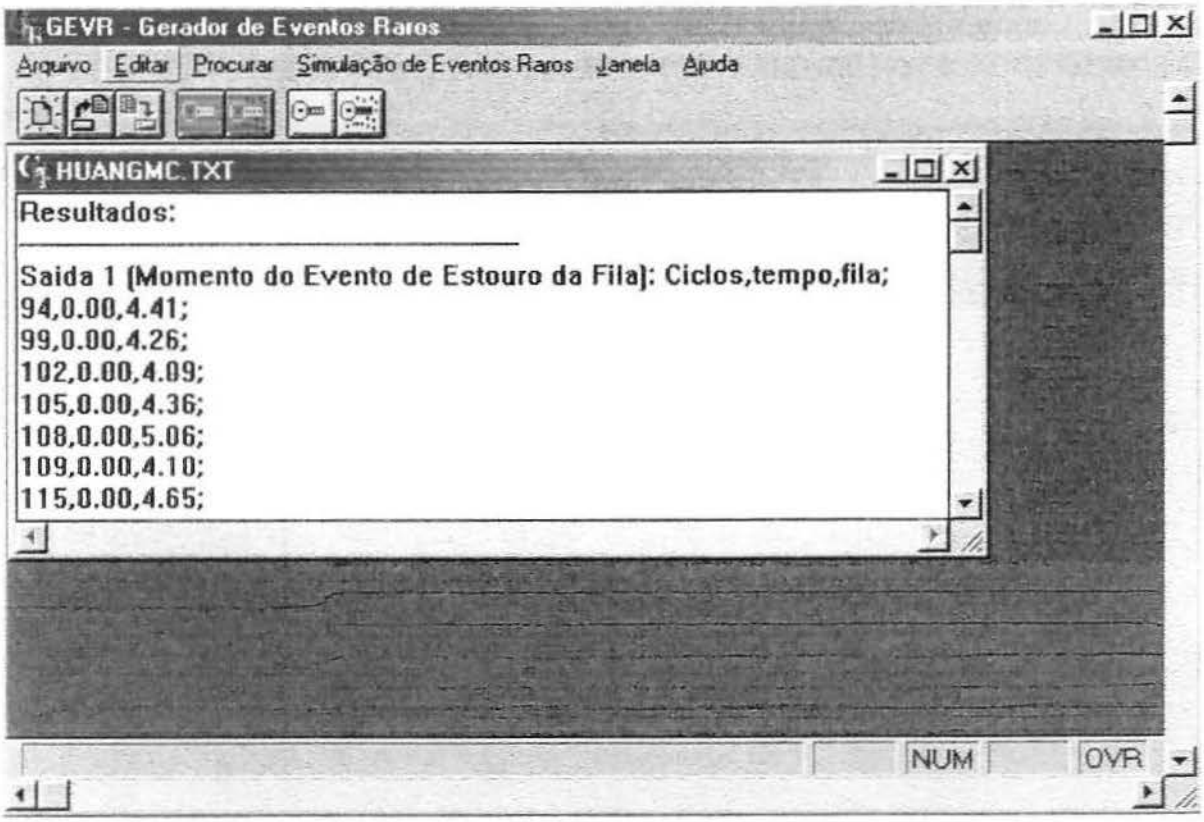
O simulador comporta-se da mesma maneira do caso anterior. Repete-se a simulação várias vezes e a cada vez, o número de ciclos é incrementado por um valor, no final, verifica-se a probabilidade para variados ciclos.

A idéia da presença deste simulador serve para comparação dos resultados que podem ser encontrados pelo simulador anterior com os resultados encontrados

por este, onde pode-se verificar uma sensível diferença nos valores, principalmente no tempo em que ocorrem os eventos.



O resultado esperado é colocado em um arquivo chamado **huangmc.txt**.


GEVR - Gerador de Eventos Raros
Arquivo Editar Procurar Simulação de Eventos Raros Janela Ajuda
HUANGMC.TXT
Resultados:
Saida 1 (Momento do Evento de Estouro da Fila): Ciclos,tempo,fila;
94,0.00,4.41;
99,0.00,4.26;
102,0.00,4.09;
105,0.00,4.36;
108,0.00,5.06;
109,0.00,4.10;
115,0.00,4.65;
NUM
OVR

APÊNDICE B

Código fontes dos simuladores

1) Simulador On-Off - 1 Fonte - Caso Contínuo:

```
// Simulacao de uma fonte On/Off - Caso Continuo - 1 Fonte
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define Off 0
#define On 1

double GNR(double fmean);
double GNR2(void);
void trata_estado(void);
void trata_chegada(void);
void grava_dados(int vestado,float vtempo,float vintervalo);
void trata_probabilidade(void);
void obtem_dados_iniciais(void);

/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
   int estado;
   char virgula0;
   float tempo;
   char virgula1;
   float intervalo;
   char ponto_virgula;
   } registro;

// Define Matriz de estado e vetor de taxa de chegadas

float prob [2][2]; //Matriz de transicao da fonte (2 estados:On - Off)
```

```
float lambda;   //Taxa de chegada do Estado ON

float lambda2; //Taxa de mudanca em cada estado


// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

float   chegada_de_pacote,
        mudanca_estado,
        relogio,
        ref1=0,ref2=0,
        prob_mudanca;

int num_ciclos,
   estado,
   i,
   contnc = 0;

void main()
{
obtem_dados_iniciais();
fclose(arq_entra);
arq_sai = fopen("onoff1c.txt", "w");
trata_probabilidade();

relogio = 0;
estado = On;
mudanca_estado = 0;
chegada_de_pacote = 0;

contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  if (chegada_de_pacote <= mudanca_estado)
     relogio = chegada_de_pacote;
  else
     relogio = mudanca_estado;

  if ((relogio == chegada_de_pacote)&&(estado==On))
    {
          grava_dados(estado,relogio,ref2-ref1);
          ref1 = ref2;
          trata_chegada();
    }
  else
    {
          trata_estado();
    }
 contnc=contnc+1;
} /* termino do laco while */
```

```
  printf("Fim da simulacao!!!\n");
  getch();

  fclose(arq_sai);
}

double GNR(double fmean)

{
        double result,rannd;
        rannd = rand();
        result = rannd/32768;
        result = -(1/fmean) * log(result);
        return(result);
}

double GNR2(void)

{
        double result1,rannd1;
        rannd1 = rand();
        result1 = rannd1/32768;
        return(result1);
}
    void grava_dados(int vestado,float vtempo,float vintervalo)
    {
          registro reg_dados;
          reg_dados.estado = vestado;
          reg_dados.virgula0 = ',';
          reg_dados.tempo = vtempo;
          reg_dados.virgula1 = ',';
          reg_dados.intervalo = vintervalo;
          reg_dados.ponto_virgula = ';';
          fprintf(arq_sai, "%d",reg_dados.estado);
          fprintf(arq_sai, "%c",reg_dados.virgula0);
          fprintf(arq_sai, "%-4.2lf",reg_dados.tempo);
          fprintf(arq_sai, "%c",reg_dados.virgula1);
       fprintf(arq_sai, "%-4.2lf",reg_dados.intervalo);
          fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(void)
{ //a
chegada_de_pacote = chegada_de_pacote + GNR(lambda);
ref2 = chegada_de_pacote;
} //a

void trata_estado(void)
{ //a
prob_mudanca = GNR2();
switch(estado)
{ //b
case Off:
```

```
 if ((prob_mudanca >= prob[0][0])&&
   (prob_mudanca < 1)){ //1
          estado = On;
          mudanca_estado = mudanca_estado + GNR(lambda2);
          trata_chegada();
                    } //1
    else
                    { //2
         estado = Off;
         mudanca_estado = mudanca_estado + GNR(lambda2);
                    } //2
break;

case On:
 if ((prob_mudanca >= prob[1][0])&&
   (prob_mudanca < 1)){ //1
          estado = Off;
          mudanca_estado = mudanca_estado + GNR(lambda2);
                    } //1
    else
                    { //2
         estado = On;
         mudanca_estado = mudanca_estado + GNR(lambda2);
         trata_chegada();
                    } //2
break;

} //b
} //a

void trata_probabilidade(void)
{
int i,j;

for(j=0;j<=1;j++)
  {
  prob [j][1] = prob [j][0];
  prob [j][0] = 0;
  }

}

// Rotina para inicializar vetores e matrizes

void obtem_dados_iniciais(void)
{ //a
 if(( arq_entra = fopen("eoff1c.txt", "r")) == NULL)
   printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
 else
  {
   fscanf(arq_entra,"%d %f %f %f %f %f %f",&num_ciclos,&prob[0][0],&prob[0][1],
          &prob [1][0],&prob[1][1],&lambda,&lambda2);
  }
return;
} //a
```

2) Simulador On-Off - 1 Fonte - Caso Discreto:

```
// Simulacao de uma fonte On/Off - Caso Discreto - 1 Fonte
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define Off 0
#define On 1

float GNR(void);
void trata_estado(void);
void trata_chegada(void);
void grava_dados(int vestado,int vtempo,int vintervalo);
void trata_probabilidade(void);
void obtem_dados_iniciais(void);

/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
  int estado;
  char virgula0;
  int tempo;
  char virgula1;
  int intervalo;
  char ponto_virgula;
  } registro;

// Define Matriz de estado e vetor de taxa de chegadas

float prob [2][2]; //Matriz de transicao da fonte (2 estados:On - Off)
float prob_chegada; //Probabilidade de Chegada no estado ON
float prob_mudanca; //Probabilidade de Mudanca de Estados

// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

float  chegada_de_pacote;

int  ref1=0,
     ref2=0,
     num_ciclos,
     estado,
     i,
     contnc = 0;
```

```
void main()
{
obtem_dados_iniciais();
fclose(arq_entra);
arq_sai = fopen("onoff1d.txt", "w");
trata_probabilidade();

estado = On;
chegada_de_pacote = 0;

contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
    if (estado == On)
    {
          trata_chegada();
          trata_estado();
    }
    else
    {
         trata_estado();
    }
    contnc=contnc+1;
} /* termino do laco while */

printf("Fim da simulacao!!!\n");
getch();

fclose(arq_sai);
}

float GNR(void)

{
         double result,rannd;
         rannd = rand();
         result = rannd/32768;
         return(result);
}
    void grava_dados(int vestado,int vtempo,int vintervalo)
    {
           registro reg_dados;
           reg_dados.estado = vestado;
           reg_dados.virgula0 = ',';
           reg_dados.tempo = vtempo;
           reg_dados.virgula1 = ',';
           reg_dados.intervalo = vintervalo;
           reg_dados.ponto_virgula = ';';
           fprintf(arq_sai, "%d",reg_dados.estado);
           fprintf(arq_sai, "%c",reg_dados.virgula0);
           fprintf(arq_sai, "%d",reg_dados.tempo);
           fprintf(arq_sai, "%c",reg_dados.virgula1);
           fprintf(arq_sai, "%d",reg_dados.intervalo);
```

```
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(void)
{ //a
chegada_de_pacote = GNR();
if (chegada_de_pacote > prob_chegada)
  {
   ref2 = contnc;
   grava_dados(estado,contnc,ref2-ref1);
   ref1 = ref2;
  }

} //a

void trata_estado(void)
{ //a
prob_mudanca = GNR();
switch(estado)
{ //b
case Off:
 if ((prob_mudanca >= prob[0][0])&&
   (prob_mudanca < 1)){ //1
            estado = On;
            trata_chegada();
                        } //1
    else
                        { //2
          estado = Off;
                        } //2
break;

case On:
 if ((prob_mudanca >= prob[1][0])&&
   (prob_mudanca < 1)){ //1
            estado = Off;
                        } //1
    else
                        { //2
          estado = On;
          trata_chegada();
                        } //2
break;

} //b
} //a

void trata_probabilidade(void)
{
int i,j;

for(j=0;j<=1;j++)
  {
  prob [j][1] = prob [j][0];
```

```
   prob [j][0] = 0;
   }

}

// Rotina para inicializar vetores e matrizes

void obtem_dados_iniciais(void)
{ //a
 if(( arq_entra = fopen("eoff1d.txt", "r")) == NULL)
    printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
 else
  {
    fscanf(arq_entra,"%d %f %f %f %f %f %f",&num_ciclos,&prob[0][0],&prob[0][1],
           &prob [1][0],&prob[1][1],&prob_chegada,&prob_mudanca);
   }
return;
} //a
```

3) Simulador On-Off - N Fontes - Caso Contínuo

```
// Simulacao de uma fonte On/Off - Caso Continuo - N Fontes
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define Off 0
#define On 1

float GNR(float fmean);
float GNR2(void);
void trata_estado(struct lista *no);
void trata_chegada(struct lista *no);
void grava_dados(int vfonte,int vestado,float vtempo,float vintervalo);
void trata_probabilidade(struct lista *no);
int  obtem_dados_iniciais(void);
float menor_chegada(struct lista *no);
float menor_mudanca(struct lista *no);
void obtem_dados(struct lista *no);


/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
   int fonte;
   char virgula0;
   int estado;
```

```
  char virgula1;
  float tempo;
  char virgula2;
  float intervalo;
  char ponto_virgula;
  } registro;

// Define Alocacao Dinamica da Matriz de estado e vetor de taxa de chegadas

struct lista {
  int  num_fonte;
  int estado_atual_da_fonte; //guarda o estado atual da fonte
  float chegdepac,
        muddeest;          //guarda valores de chegadas e mudancas
                           //de estado em um ciclo
  float matriz_estado [2][2]; //Matriz de transicao da fonte
  float taxa_chegada;
  float matriz_mudanca[2];
  struct lista *proxfonte;
  } lista1,*primfonte,*newfonte;

// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

 float  prob,
        chegada_de_pacote,
        mudanca_estado,
        relogio,
        ref1,
        ref2,
        media_chegada,
        prob_mudanca,
        media_mudanca;

int fonte_mudanca,
   fonte_chegada,
   fonte_atual,
   num_ciclos,
   num_fontes,
   num_estados = 2,
   estado,
   estado_chegada,
   estado_mudanca,
   i,
   f,
   ok = 0,
   contnc = 0;

void main()
{
ok = obtem_dados_iniciais();
fclose(arq_entra);
if (ok == 0)
{ //ok
trata_probabilidade(&lista1);
```

```
arq_sai = fopen("onoffcn.txt", "w");
mudanca_estado = 0;
chegada_de_pacote = 0;
contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  chegada_de_pacote = menor_chegada(&lista1);
  mudanca_estado = menor_mudanca(&lista1);

  if (chegada_de_pacote <= mudanca_estado)
    {
     relogio = chegada_de_pacote;
     estado = estado_chegada;
    }
  else
    {
     relogio = mudanca_estado;
     estado = estado_mudanca;
    }

  if ((relogio == chegada_de_pacote)&&(estado == On))
    {
          ref2 = relogio;
          grava_dados(fonte_chegada,estado,relogio,ref2-ref1);
          ref1 = ref2;
          trata_chegada(&lista1);
    }
  else
    {
        trata_estado(&lista1);
    }
 contnc=contnc+1;
} /* termino do laco while */


 printf("Fim da simulacao!!!\n");
 getch();

 fclose(arq_sai);
} //ok
else
{
  printf("Programa terminado de maneira anormal!!!\n");
}
}

float GNR(float fmean)

{
        float result,rannd;
        rannd = rand();
        result = rannd/32768;
        result = -(1/fmean) * log(result);
```

```
        return(result);
}

float GNR2(void)

{
        float result1,rannd1;
        rannd1 = rand();
        result1 = rannd1/32768;
        return(result1);
}
    void grava_dados(int vfonte,int vestado,float vtempo,float vintervalo)
    {
            registro reg_dados;
            reg_dados.fonte = vfonte;
            reg_dados.virgula0 = ',';
            reg_dados.estado = vestado;
            reg_dados.virgula1 = ',';
            reg_dados.tempo = vtempo;
            reg_dados.virgula2 = ',';
            reg_dados.intervalo = vintervalo;
            reg_dados.ponto_virgula = ';';
            fprintf(arq_sai, "%d",reg_dados.fonte);
            fprintf(arq_sai, "%c",reg_dados.virgula0);
            fprintf(arq_sai, "%d",reg_dados.estado);
            fprintf(arq_sai, "%c",reg_dados.virgula1);
            fprintf(arq_sai, "%f",reg_dados.tempo);
            fprintf(arq_sai, "%c",reg_dados.virgula2);
            fprintf(arq_sai, "%f",reg_dados.intervalo);
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(struct lista *no)
{ //a
no = primfonte;
media_chegada = no->taxa_chegada;
while(no->num_fonte != fonte_chegada)
{
    no=no->proxfonte;
}
    media_chegada = no->taxa_chegada;
    chegada_de_pacote = chegada_de_pacote + GNR(media_chegada);
    no->chegdepac = chegada_de_pacote;
    estado_chegada = no->estado_atual_da_fonte;
} //a

void trata_estado(struct lista *no)
{ //a
int i,j;
float VRP1,VRP2;
no = primfonte;
while (no->num_fonte != fonte_mudanca)
{ //b
 no=no->proxfonte;
```

```
} //b
prob_mudanca = GNR2();
i=no->estado_atual_da_fonte;
for (j = 0;j < num_estados; j++)
{ //c
 VRP1 = no->matriz_estado[i][j];
 VRP2 = no->matriz_estado[i][j+1];

 if ((prob_mudanca >= VRP1)&&(prob_mudanca < VRP2))
  { //d
    no->estado_atual_da_fonte = j;
    estado_mudanca = no->estado_atual_da_fonte;
    media_mudanca = no->matriz_mudanca[j];
    mudanca_estado = mudanca_estado + GNR(media_mudanca);
    no->muddeest = mudanca_estado;
    trata_chegada(&lista1);
   } //d
} //c

if ((prob_mudanca >= no->matriz_estado[i][num_estados-1])&&
  (prob_mudanca < 1))
{ //e
    no->estado_atual_da_fonte = j-1;
    estado_mudanca = no->estado_atual_da_fonte;
    media_mudanca = no->matriz_mudanca [j-1];
    mudanca_estado = mudanca_estado + GNR(media_mudanca);
    no->muddeest = mudanca_estado;
    trata_chegada(&lista1);
} //e
return;
} //a

void trata_probabilidade(struct lista *no)
{ //a
int i,j;
float matriz_aux[10];
no=primfonte;
do
{ //b
 for (i = 0;i<= num_estados - 1;i++)
            { //c
                for (j = 0;j <= num_estados - 1;j++)
                 { //d
                     matriz_aux[j]=no->matriz_estado[i][j];
                 } //d
                 no->matriz_estado[i][0] = 0;
                for (j = 1;j <= num_estados - 1;j++)
                 { //e
                   no->matriz_estado[i][j] = matriz_aux[j-1]
                   + no->matriz_estado[i][j-1];
                 } //e
            } //c
}while ((no=no->proxfonte)!=NULL);//b
return;
} //a
```

```
// Rotina para inicializar vetores e matrizes

int obtem_dados_iniciais(void)
{ //a
  if(( arq_entra = fopen("eoffcn.txt", "r")) == NULL)
    {
    printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
    exit (1);
    }
 else
  { //b
   fscanf(arq_entra,"%d %d",&num_fontes,&num_ciclos);
   if (num_fontes <= 0) {   num_fontes = 1;}
   for(f=1;f<=num_fontes;f++)
   { //c
    // Aloca memoria para a fonte
     if (primfonte == NULL)
          { //d
            primfonte = (struct lista*) malloc(num_fontes*sizeof(struct lista));
            if (primfonte==NULL) exit (1);
            obtem_dados(&lista1);
            *primfonte=lista1;
            newfonte=primfonte;
          } //d
     else
          { //e
             obtem_dados(&lista1);
             newfonte->proxfonte=(struct lista*) malloc(num_fontes*sizeof(struct lista));
             if (newfonte->proxfonte==NULL) exit (1);
             newfonte=newfonte->proxfonte;
             *newfonte=lista1;
          } //e
   }//c
             newfonte->proxfonte=NULL;
  }//b
return(0);
} //a

float menor_chegada(struct lista *no)
{
int i,j;
float menor_chegada;
no=primfonte;
menor_chegada = no->chegdepac;
fonte_chegada = no->num_fonte;
estado_chegada = no->estado_atual_da_fonte;
do
{
  if(menor_chegada > no->chegdepac)
    {
      menor_chegada = no->chegdepac;
      fonte_chegada = no->num_fonte;
      estado_chegada = no->estado_atual_da_fonte;
    }
} while((no=no->proxfonte)!=NULL);
return(menor_chegada);
```

```
}

float menor_mudanca(struct lista *no)
{
int i;
float menor_mudanca;
no=primfonte;
menor_mudanca = no->muddeest;
fonte_mudanca = no->num_fonte;
estado_mudanca = no->estado_atual_da_fonte;
do
{
 if(menor_mudanca > no->muddeest)
   {
    menor_mudanca = no->muddeest;
    fonte_mudanca = no->num_fonte;
    estado_mudanca = no->estado_atual_da_fonte;
   }
} while((no=no->proxfonte)!=NULL);
return(menor_mudanca);
}

void obtem_dados(struct lista *no)
{
 int i,j;

 fscanf(arq_entra,"%d %d %f %f",
          &no->num_fonte,//1
          &no->estado_atual_da_fonte, //2
          &no->chegdepac, //3
          &no->muddeest); //4

for (i = 0;i<=num_estados-1;i++)
 {
  for (j = 0;j<=num_estados-1;j++)
    {
       fscanf(arq_entra,"%f",&no->matriz_estado[i][j]);
    }
 }
fscanf(arq_entra,"%f",&no->taxa_chegada);

for (i = 0;i<=num_estados-1;i++)
 {
  fscanf(arq_entra,"%f",&no->matriz_mudanca[i]);
 }
 no->chegdepac = GNR(no->taxa_chegada);
 no->muddeest  = GNR(no->matriz_mudanca[0]);
return;
}
```

4) Simulador On-Off - N Fontes - Caso Discreto

```
// Simulacao de uma fonte On/Off - Caso Discreto - N Fontes
// Autor: Andre Luiz Garcia Pereira
```

```
#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define Off 0
#define On  1

float GNR(void);
void trata_estado(struct lista *no);
void trata_chegada(struct lista *no);
void grava_dados(int vfonte,int vestado,int vtempo,int vintervalo);
void trata_probabilidade(struct lista *no);
int  obtem_dados_iniciais(void);
float menor_chegada(struct lista *no);
float menor_mudanca(struct lista *no);
void obtem_dados(struct lista *no);


/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
  int fonte;
  char virgula0;
  int estado;
  char virgula1;
  int tempo;
  char virgula2;
  int intervalo;
  char ponto_virgula;
  } registro;

// Define Alocacao Dinamica da Matriz de estado e vetor de taxa de chegadas

struct lista {
  int  num_fonte;
  int estado_atual_da_fonte; //guarda o estado atual da fonte
  float matriz_estado [2][2]; //Matriz de transicao da fonte k=1..10
  float taxa_chegada;
  struct lista *proxfonte;
  } lista1,*primfonte,*newfonte;

// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

 float  prob,
        chegada_de_pacote,
        mudanca_estado,
        relogio,
        ref1,
        ref2,
```

```
        media_chegada,
        prob_mudanca,
        media_mudanca;

int fonte_mudanca,
    fonte_chegada,
    fonte_atual,
    num_ciclos,
    num_fontes,
    num_estados = 2,
    estado,
    estado_chegada,
    estado_mudanca,
    i,
    f,
    ok = 0,
    contnc = 0;

void main()
{
ok = obtem_dados_iniciais();
fclose(arq_entra);
if (ok == 0)
{ //ok
trata_probabilidade(&lista1);
arq_sai = fopen("onoffdn.txt", "w");
mudanca_estado = 0;
chegada_de_pacote = 0;
contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  trata_chegada(&lista1);
  trata_estado(&lista1);
  contnc=contnc+1;
} /* termino do laco while */
 printf("Fim da simulacao!!!\n");
 getch();

 fclose(arq_sai);
 } //ok
else
{
  printf("Programa terminado de maneira anormal!!!\n");
 }
}

float GNR(void)
{
        float result,rannd;
        rannd = rand();
        result = rannd/32768;
        return(result);
}
```

```
void grava_dados(int vfonte,int vestado,int vtempo,int vintervalo)
   {
         registro reg_dados;
         reg_dados.fonte = vfonte;
         reg_dados.virgula0 = ',';
         reg_dados.estado = vestado;
         reg_dados.virgula1 = ',';
         reg_dados.tempo = vtempo;
         reg_dados.virgula2 = ',';
         reg_dados.intervalo = vintervalo;
         reg_dados.ponto_virgula = ';';
         fprintf(arq_sai, "%d",reg_dados.fonte);
         fprintf(arq_sai, "%c",reg_dados.virgula0);
         fprintf(arq_sai, "%d",reg_dados.estado);
         fprintf(arq_sai, "%c",reg_dados.virgula1);
         fprintf(arq_sai, "%d",reg_dados.tempo);
         fprintf(arq_sai, "%c",reg_dados.virgula2);
         fprintf(arq_sai, "%d",reg_dados.intervalo);
         fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

   }


void trata_chegada(struct lista *no)
{ //a
no = primfonte;
do
{
  chegada_de_pacote = GNR();
  estado = no->estado_atual_da_fonte;
  fonte_chegada = no->num_fonte;
  if ((chegada_de_pacote > no->taxa_chegada)&&(estado == On))
    {
     ref2 = contnc;
     grava_dados(fonte_chegada,estado,contnc,ref2-ref1);
     ref1 = ref2;
    }
}while((no=no->proxfonte)!=NULL);
return;
} //a

void trata_estado(struct lista *no)
{ //a
int i,j;
float VRP1,VRP2;
no = primfonte;
do
{
mudanca_estado = GNR();
i = no->estado_atual_da_fonte;
for (j = 0;j < num_estados; j++)
{  //c
 VRP1 = no->matriz_estado[i][j];
 VRP2 = no->matriz_estado[i][j+1];
```

```
 if ((mudanca_estado >= VRP1)&&(mudanca_estado < VRP2))
  { //d
    no->estado_atual_da_fonte = j;
  } //d
} //c

if ((mudanca_estado >= no->matriz_estado[i][num_estados-1])&&
  (mudanca_estado < 1))
 { //e
    no->estado_atual_da_fonte = j-1;
 } //e
} while ((no=no->proxfonte)!=NULL);
return;
} //a

void trata_probabilidade(struct lista *no)
{ //a
int i,j;
float matriz_aux[10];
no=primfonte;
do
{ //b
 for (i = 0;i<= num_estados - 1;i++)
             { //c
                  for (j = 0;j <= num_estados - 1;j++)
                   { //d
                       matriz_aux[j]=no->matriz_estado[i][j];
                   } //d
                   no->matriz_estado[i][0] = 0;
                  for (j = 1;j <= num_estados - 1;j++)
                   { //e
                     no->matriz_estado[i][j] = matriz_aux[j-1]
                     + no->matriz_estado[i][j-1];
                   } //e
             } //c
}while ((no=no->proxfonte)!=NULL);//b
return;
} //a

// Rotina para inicializar vetores e matrizes

int obtem_dados_iniciais(void)
{ //a
  if(( arq_entra = fopen("eoffdn.txt", "r")) == NULL)
    {
    printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
    exit (1);
    }
 else
 { //b
  fscanf(arq_entra,"%d %d",&num_fontes,&num_ciclos);
  if (num_fontes <= 0) {  num_fontes = 1;}
  for(f=1;f<=num_fontes;f++)
  { //c
   // Aloca memoria para a fonte
    if (primfonte == NULL)
```

```
        { //d
          primfonte = (struct lista*) malloc(num_fontes*sizeof(struct lista));
          if (primfonte==NULL) exit (1);
          obtem_dados(&lista1);
          *primfonte=lista1;
          newfonte=primfonte;
        } //d
    else
        { //e
          obtem_dados(&lista1);
          newfonte->proxfonte=(struct lista*) malloc(num_fontes*sizeof(struct lista));
          if (newfonte->proxfonte==NULL) exit (1);
          newfonte=newfonte->proxfonte;
          *newfonte=lista1;
        } //e
  }//c
          newfonte->proxfonte=NULL;
 }//b
return(0);
} //a

void obtem_dados(struct lista *no)
{
 int i,j;

 fscanf(arq_entra,"%d %d",
        &no->num_fonte,//1
        &no->estado_atual_da_fonte); //2
        num_estados = num_estados;
for (i = 0;i<=num_estados-1;i++)
 {
   for (j = 0;j<=num_estados-1;j++)
     {
        fscanf(arq_entra,"%f",&no->matriz_estado[i][j]);
     }
 }
fscanf(arq_entra,"%f",&no->taxa_chegada);

return;
}
```

5) Processo Markoviano Modulado de Poisson - 1 Fonte

```
// Simulacao MMPP - 1 Fonte
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define MAX 4
```

```
float GNR(float fmean);
float GNR2(void);
void trata_estado(void);
void trata_chegada(void);
void grava_dados(int vestado,float vtempo,float vintervalo);
void trata_probabilidade(void);
int  obtem_dados_iniciais(void);

/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
  int estado;
  char virgula0;
  float tempo;
  char virgula1;
  float intervalo;
  char ponto_virgula;
  } registro;

// Definicao de matrizes para as taxas de probabilidades
  float matriz_estado [MAX][MAX]; //Matriz de transicao
  float matriz_chegada[MAX];    //Vetor de taxas de chegada
  float matriz_mudanca[MAX];    //Vetor de taxas de mudancas

// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

 float  prob,
        chegada_de_pacote,
        mudanca_estado,
        relogio,
        ref1,
        ref2,
        media_chegada,
        prob_mudanca,
        media_mudanca;

int num_ciclos,
   num_estados,
   estado,
   i,
   f,
   ok = 0,
   contnc = 0;

void main()
{
ok = obtem_dados_iniciais();
fclose(arq_entra);
if (ok == 0)
{ //ok
 trata_probabilidade();
 arq_sai = fopen("mmpp1.txt", "w");
```

```
 relogio = 0;
 estado = 0;
 media_chegada = matriz_chegada[estado];
 chegada_de_pacote = GNR(media_chegada);
 media_mudanca = matriz_mudanca[estado];
 mudanca_estado = GNR(media_mudanca);
 contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  if (chegada_de_pacote <= mudanca_estado)
          relogio = chegada_de_pacote;
  else
          relogio = mudanca_estado;

  if (relogio == chegada_de_pacote)
    {
          grava_dados(estado,relogio,ref2-ref1);
          ref1 = ref2;
          trata_chegada();
    }
  else
    {
        trata_estado();
    }
 contnc=contnc+1;
} /* termino do laco while */


 printf("Fim da simulacao!!!\n");
 getch();

 fclose(arq_sai);
} //ok
else
{
  printf("Programa terminado de maneira anormal!!!\n");
}
}

float GNR(float fmean)

{
        float result,rannd;
        rannd = rand();
        result = rannd/32768;
        result = -(1/fmean) * log(result);
        return(result);
}

float GNR2(void)

{
        float result1,rannd1;
```

```
        rannd1 = rand();
        result1 = rannd1/32768;
        return(result1);
}
    void grava_dados(int vestado,float vtempo,float vintervalo)
    {
            registro reg_dados;
            reg_dados.estado = vestado;
            reg_dados.virgula0 = ',';
            reg_dados.tempo = vtempo;
            reg_dados.virgula1 = ',';
            reg_dados.intervalo = vintervalo;
            reg_dados.ponto_virgula = ';';
            fprintf(arq_sai, "%d",reg_dados.estado);
            fprintf(arq_sai, "%c",reg_dados.virgula0);
            fprintf(arq_sai, "%f",reg_dados.tempo);
            fprintf(arq_sai, "%c",reg_dados.virgula1);
            fprintf(arq_sai, "%f",reg_dados.intervalo);
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(void)
{ //a
    media_chegada = matriz_chegada[estado];
    chegada_de_pacote = chegada_de_pacote + GNR(media_chegada);
    ref2 = chegada_de_pacote;
} //a

void trata_estado(void)
{ //a
int i,j;
float VRP1,VRP2;
i = estado;
j = 0;
prob_mudanca = GNR2();

for (j = 0;j < num_estados; j++)
{  //c
 VRP1 = matriz_estado[i][j];
 VRP2 = matriz_estado[i][j+1];

if ((prob_mudanca >= VRP1)&&(prob_mudanca < VRP2))
  {  //d
    estado = j;
    media_mudanca = matriz_mudanca[j];
    mudanca_estado = mudanca_estado + GNR(media_mudanca);
  }  //d
} //c

if ((prob_mudanca >= matriz_estado[i][num_estados-1])&&
  (prob_mudanca < 1))
 { //e
    estado = j-1;
    media_mudanca = matriz_mudanca [j-1];
```

```
    mudanca_estado = mudanca_estado + GNR(media_mudanca);
 } //e
return;
} //a

void trata_probabilidade()
{ //a
int i,j;
float matriz_aux[10];

 for (i = 0;i<= num_estados - 1;i++)
            { //c
                for (j = 0;j <= num_estados - 1;j++)
                 { //d
                     matriz_aux[j]=matriz_estado[i][j];
                 } //d
                 matriz_estado[i][0] = 0;
                for (j = 1;j <= num_estados - 1;j++)
                 { //e
                   matriz_estado[i][j] = matriz_aux[j-1]
                   + matriz_estado[i][j-1];
                 } //e
            } //c

return;
} //a

// Rotina para inicializar vetores e matrizes

int obtem_dados_iniciais(void)
{ //a
int i,j;

  if(( arq_entra = fopen("emmpp1.txt", "r")) == NULL)
    {
    printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
    exit (1);
    }
 else
 {
  fscanf(arq_entra,"%d %d",&num_ciclos,&num_estados);
  for (i = 0;i<=num_estados-1;i++)
     {
          for (j = 0;j<=num_estados-1;j++)
             {
              fscanf(arq_entra,"%f",&matriz_estado[i][j]);
             }
     }
          for (i = 0;i<=num_estados-1;i++)
             {
              fscanf(arq_entra,"%f",&matriz_chegada[i]);
             }
          for (i = 0;i<=num_estados-1;i++)
             {
              fscanf(arq_entra,"%f",&matriz_mudanca[i]);
             }
```

```
 }
return(0);
}
```

6) Processo Markoviano Modulado de Bernoulli - 1 Fonte

```
// Simulacao MMPB - 1 Fonte
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define MAX 4

float GNR(void);
void trata_estado(void);
void trata_chegada(void);
void grava_dados(int vestado,int vtempo,int vintervalo);
void trata_probabilidade(void);
int  obtem_dados_iniciais(void);

/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
   int estado;
   char virgula0;
   int tempo;
   char virgula1;
   int intervalo;
   char ponto_virgula;
   } registro;

// Definicao de matrizes para as taxas de probabilidades
   float matriz_estado [MAX][MAX]; //Matriz de transicao
   float matriz_chegada[MAX];    //Vetor de taxas de chegada

// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

 float  chegada_de_pacote,
          mudanca_estado,
          ref1,
          ref2;


int num_ciclos,
    num_estados,
    estado,
```

```
    i,
    f,
    ok = 0,
    contnc = 0;

void main()
{
ok = obtem_dados_iniciais();
fclose(arq_entra);
if (ok == 0)
{ //ok
 trata_probabilidade();
 arq_sai = fopen("mmpb1.txt", "w");
 estado = 0;
 chegada_de_pacote = 0;
 mudanca_estado = 0;
 contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  trata_chegada();
  trata_estado();
  contnc=contnc+1;

} /* termino do laco while */

  printf("Fim da simulacao!!!\n");
  getch();

  fclose(arq_sai);
 } //ok
else
 {
   printf("Programa terminado de maneira anormal!!!\n");
 }
}

float GNR(void)

{
        float result,rannd;
        rannd = rand();
        result = rannd/32768;
        return(result);
}

  void grava_dados(int vestado,int vtempo,int vintervalo)
    {
          registro reg_dados;
          reg_dados.estado = vestado;
          reg_dados.virgula0 = ',';
          reg_dados.tempo = vtempo;
          reg_dados.virgula1 = ',';
          reg_dados.intervalo = vintervalo;
```

```
            reg_dados.ponto_virgula = ';';
            fprintf(arq_sai, "%d",reg_dados.estado);
            fprintf(arq_sai, "%c",reg_dados.virgula0);
            fprintf(arq_sai, "%d",reg_dados.tempo);
            fprintf(arq_sai, "%c",reg_dados.virgula1);
            fprintf(arq_sai, "%d",reg_dados.intervalo);
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(void)
{ //a
    chegada_de_pacote = GNR();
for (i=0;i<=num_estados-1;i++)
  {
   if (chegada_de_pacote >= matriz_chegada[i])
    {
     ref2 = contnc;
     grava_dados(estado,contnc,ref2-ref1);
     ref1 = ref2;
    }
  }
} //a

void trata_estado(void)
{ //a
int i,j;
float VRP1,VRP2;
i = estado;
j = 0;
mudanca_estado = GNR();

for (j = 0;j < num_estados; j++)
{  //c
 VRP1 = matriz_estado[i][j];
 VRP2 = matriz_estado[i][j+1];

if ((mudanca_estado >= VRP1)&&(mudanca_estado < VRP2))
  {  //d
    estado = j;
  }  //d
}  //c

if ((mudanca_estado >= matriz_estado[i][num_estados-1])&&
  (mudanca_estado < 1))
 { //e
    estado = j-1;
 } //e
return;
} //a

void trata_probabilidade()
{ //a
int i,j;
float matriz_aux[10];
```

```
 for (i = 0;i<= num_estados - 1;i++)
              { //c
                    for (j = 0;j <= num_estados - 1;j++)
                     { //d
                         matriz_aux[j]=matriz_estado[i][j];
                     } //d
                     matriz_estado[i][0] = 0;
                    for (j = 1;j <= num_estados - 1;j++)
                     { //e
                       matriz_estado[i][j] = matriz_aux[j-1]
                       + matriz_estado[i][j-1];
                     } //e
              } //c

return;
} //a

// Rotina para inicializar vetores e matrizes

int obtem_dados_iniciais(void)
{ //a
int i,j;

  if(( arq_entra = fopen("emmpb1.txt", "r")) == NULL)
    {
    printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
    exit (1);
    }
 else
 {
  fscanf(arq_entra,"%d %d",&num_ciclos,&num_estados);
  for (i = 0;i<=num_estados-1;i++)
     {
          for (j = 0;j<=num_estados-1;j++)
             {
              fscanf(arq_entra,"%f",&matriz_estado[i][j]);
             }
     }
          for (i = 0;i<=num_estados-1;i++)
             {
              fscanf(arq_entra,"%f",&matriz_chegada[i]);
             }
 }
return(0);
}
```

7) Processo Markoviano Modulado de Poisson - N Fontes

```
// Simulacao MMPP - N Fontes
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define MAX 4

float GNR(float fmean);
float GNR2(void);
void trata_estado(struct lista *no);
void trata_chegada(struct lista *no);
void grava_dados(int vfonte,int vestado,float vtempo,float vintervalo);
void trata_probabilidade(struct lista *no);
int  obtem_dados_iniciais(void);
float menor_chegada(struct lista *no);
float menor_mudanca(struct lista *no);
void obtem_dados(struct lista *no);


/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
  int fonte;
  char virgula0;
  int estado;
  char virgula1;
  float tempo;
  char virgula2;
  float intervalo;
  char ponto_virgula;
  } registro;

// Define Alocacao Dinamica da Matriz de estado e vetor de taxa de chegadas

struct lista {
  int  num_fonte;
  int  num_estados_fonte;
  int estado_atual_da_fonte; //guarda o estado atual da fonte
  float chegdepac,
        muddeest;          //guarda valores de chegadas e mudancas
                           //de estado em um ciclo
  float matriz_estado [MAX][MAX]; //Matriz de transicao da fonte k=1..10
  float matriz_chegada[MAX];
  float matriz_mudanca[MAX];
  struct lista *proxfonte;
  } lista1,*primfonte,*newfonte;
```

```
// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

 float  prob,
          chegada_de_pacote,
          mudanca_estado,
          relogio,
          ref1,
          ref2,
          media_chegada,
          prob_mudanca,
          media_mudanca;

int fonte_mudanca,
   fonte_chegada,
   fonte_atual,
   num_ciclos,
   num_fontes,
   num_estados,
   estado,
   estado_chegada,
   estado_mudanca,
   i,
   f,
   ok = 0,
   contnc = 0;

void main()
{
ok = obtem_dados_iniciais();
fclose(arq_entra);
if (ok == 0)
{ //ok
trata_probabilidade(&lista1);
arq_sai = fopen("mmpbn.txt", "w");
mudanca_estado = 0;
chegada_de_pacote = 0;
contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  chegada_de_pacote = menor_chegada(&lista1);
  mudanca_estado = menor_mudanca(&lista1);

  if (chegada_de_pacote <= mudanca_estado)
    {
     relogio = chegada_de_pacote;
     estado = estado_chegada;
    }
  else
    {
     relogio = mudanca_estado;
     estado = estado_mudanca;
```

```
    }

  if (relogio == chegada_de_pacote)
    {
          ref2 = relogio;
          grava_dados(fonte_chegada,estado,relogio,ref2-ref1);
          ref1 = ref2;
          trata_chegada(&lista1);
    }
   else
    {
        trata_estado(&lista1);
    }
  contnc=contnc+1;
 } /* termino do laco while */


  printf("Fim da simulacao!!!\n");
  getch();

  fclose(arq_sai);
 } //ok
else
{
  printf("Programa terminado de maneira anormal!!!\n");
}
}

float GNR(float fmean)

{
        float result,rannd;
        rannd = rand();
        result = rannd/32768;
        result = -(1/fmean) * log(result);
        return(result);
}

float GNR2(void)

{
        float result1,rannd1;
        rannd1 = rand();
        result1 = rannd1/32768;
        return(result1);
}
    void grava_dados(int vfonte,int vestado,float vtempo,float vintervalo)
    {
          registro reg_dados;
          reg_dados.fonte = vfonte;
          reg_dados.virgula0 = ',';
          reg_dados.estado = vestado;
          reg_dados.virgula1 = ',';
          reg_dados.tempo = vtempo;
          reg_dados.virgula2 = ',';
          reg_dados.intervalo = vintervalo;
```

```
            reg_dados.ponto_virgula = ';';
            fprintf(arq_sai, "%d",reg_dados.fonte);
            fprintf(arq_sai, "%c",reg_dados.virgula0);
            fprintf(arq_sai, "%d",reg_dados.estado);
            fprintf(arq_sai, "%c",reg_dados.virgula1);
            fprintf(arq_sai, "%f",reg_dados.tempo);
            fprintf(arq_sai, "%c",reg_dados.virgula2);
            fprintf(arq_sai, "%f",reg_dados.intervalo);
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(struct lista *no)
{ //a
no = primfonte;
media_chegada = no->matriz_chegada[no->estado_atual_da_fonte];
while(no->num_fonte != fonte_chegada)
{
    no=no->proxfonte;
}
    media_chegada = no->matriz_chegada[no->estado_atual_da_fonte];
    chegada_de_pacote = chegada_de_pacote + GNR(media_chegada);
    no->chegdepac = chegada_de_pacote;
    estado_chegada = no->estado_atual_da_fonte;
} //a

void trata_estado(struct lista *no)
{ //a
int i,j;
float VRP1,VRP2;
no = primfonte;
while (no->num_fonte != fonte_mudanca)
{ //b
 no=no->proxfonte;
} //b
prob_mudanca = GNR2();
i=no->estado_atual_da_fonte;
for (j = 0;j < no->num_estados_fonte; j++)
{  //c
 VRP1 = no->matriz_estado[i][j];
 VRP2 = no->matriz_estado[i][j+1];

 if ((prob_mudanca >= VRP1)&&(prob_mudanca < VRP2))
   {  //d
    no->estado_atual_da_fonte = j;
    estado_mudanca = no->estado_atual_da_fonte;
    media_mudanca = no->matriz_mudanca[j];
    mudanca_estado = mudanca_estado + GNR(media_mudanca);
    no->muddeest = mudanca_estado;
    trata_chegada(&lista1);
   } //d
} //c

if ((prob_mudanca >= no->matriz_estado[i][no->num_estados_fonte-1])&&
  (prob_mudanca < 1))
```

```
 { //e
    no->estado_atual_da_fonte = j-1;
    estado_mudanca = no->estado_atual_da_fonte;
    media_mudanca = no->matriz_mudanca [j-1];
    mudanca_estado = mudanca_estado + GNR(media_mudanca);
    no->muddeest = mudanca_estado;
    trata_chegada(&lista1);
 } //e
return;
} //a

void trata_probabilidade(struct lista *no)
{ //a
int i,j;
float matriz_aux[10];
no=primfonte;
do
{  //b
 for (i = 0;i<= num_estados - 1;i++)
              { //c
                    for (j = 0;j <= num_estados - 1;j++)
                     { //d
                         matriz_aux[j]=no->matriz_estado[i][j];
                     } //d
                     no->matriz_estado[i][0] = 0;
                    for (j = 1;j <= num_estados - 1;j++)
                     { //e
                       no->matriz_estado[i][j] = matriz_aux[j-1]
                       + no->matriz_estado[i][j-1];
                     } //e
              }  //c
}while ((no=no->proxfonte)!=NULL);//b
return;
} //a

// Rotina para inicializar vetores e matrizes

int obtem_dados_iniciais(void)
{ //a
  if(( arq_entra = fopen("emmppn.txt", "r")) == NULL)
    {
    printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
    exit (1);
    }
 else
  { //b
   fscanf(arq_entra,"%d %d",&num_fontes,&num_ciclos);
   if (num_fontes <= 0) {   num_fontes = 1;}
   for(f=1;f<=num_fontes;f++)
   { //c
    // Aloca memoria para a fonte
     if (primfonte == NULL)
           { //d
             primfonte = (struct lista*) malloc(num_fontes*sizeof(struct lista));
             if (primfonte==NULL) exit (1);
             obtem_dados(&lista1);
```

```
            *primfonte=lista1;
            newfonte=primfonte;
          } //d
    else
          { //e
            obtem_dados(&lista1);
            newfonte->proxfonte=(struct lista*) malloc(num_fontes*sizeof(struct lista));
            if (newfonte->proxfonte==NULL) exit (1);
            newfonte=newfonte->proxfonte;
            *newfonte=lista1;
          } //e
  }//c
            newfonte->proxfonte=NULL;
 }//b
return(0);
} //a

float menor_chegada(struct lista *no)
{
int i,j;
float menor_chegada;
no=primfonte;
menor_chegada = no->chegdepac;
fonte_chegada = no->num_fonte;
estado_chegada = no->estado_atual_da_fonte;
do
{
  if(menor_chegada > no->chegdepac)
    {
      menor_chegada = no->chegdepac;
      fonte_chegada = no->num_fonte;
      estado_chegada = no->estado_atual_da_fonte;
    }
} while((no=no->proxfonte)!=NULL);
return(menor_chegada);
}

float menor_mudanca(struct lista *no)
{
int i;
float menor_mudanca;
no=primfonte;
menor_mudanca = no->muddeest;
fonte_mudanca = no->num_fonte;
estado_mudanca = no->estado_atual_da_fonte;
do
{
  if(menor_mudanca > no->muddeest)
    {
      menor_mudanca = no->muddeest;
      fonte_mudanca = no->num_fonte;
      estado_mudanca = no->estado_atual_da_fonte;
    }
} while((no=no->proxfonte)!=NULL);
return(menor_mudanca);
}
```

```
void obtem_dados(struct lista *no)
{
  int i,j;

 fscanf(arq_entra,"%d %d %d %f %f",
            &no->num_fonte,//1
            &no->num_estados_fonte,    //2
            &no->estado_atual_da_fonte,
            &no->chegdepac,
            &no->muddeest); //3
            num_estados = no->num_estados_fonte;
for (i = 0;i<=num_estados-1;i++)
 {
   for (j = 0;j<=num_estados-1;j++)
     {
         fscanf(arq_entra,"%f",&no->matriz_estado[i][j]);
     }
 }
for (i = 0;i<=num_estados-1;i++)
 {
  fscanf(arq_entra,"%f",&no->matriz_chegada[i]);
 }
for (i = 0;i<=num_estados-1;i++)
 {
  fscanf(arq_entra,"%f",&no->matriz_mudanca[i]);
 }
 no->chegdepac = GNR(no->matriz_chegada[0]);
 no->muddeest  = GNR(no->matriz_mudanca[0]);
return;
}
```

8) Processo Markoviano Modulado de Bernoulli - N Nontes

```
// Simulacao MMPB -  N Fontes
// Autor: Andre Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#define MAX 4

float GNR(void);
void trata_estado(struct lista *no);
void trata_chegada(struct lista *no);
void grava_dados(int vfonte,int vestado,int vtempo,int vintervalo);
void trata_probabilidade(struct lista *no);
int  obtem_dados_iniciais(void);
float menor_chegada(struct lista *no);
float menor_mudanca(struct lista *no);
void obtem_dados(struct lista *no);
```

```
/* Comeca a funcao principal do Simulador */

// Define Registro de saida

typedef struct {
   int fonte;
   char virgula0;
   int estado;
   char virgula1;
   int tempo;
   char virgula2;
   int intervalo;
   char ponto_virgula;
   } registro;

// Define Alocacao Dinamica da Matriz de estado e vetor de taxa de chegadas

struct lista {
   int   num_fonte;
   int   num_estados_fonte;
   int estado_atual_da_fonte; //guarda o estado atual da fonte
   float matriz_estado [MAX][MAX]; //Matriz de transicao da fonte k=1..10
   float matriz_chegada[MAX];
   struct lista *proxfonte;
   } lista1,*primfonte,*newfonte;

// Define Arquivo de saida

FILE *arq_sai,*arq_entra;

 float  prob,
          chegada_de_pacote,
          mudanca_estado,
          relogio,
          ref1,
          ref2,
          media_chegada,
          prob_mudanca,
          media_mudanca;

int fonte_mudanca,
    fonte_chegada,
    fonte_atual,
    num_ciclos,
    num_fontes,
    num_estados,
    estado,
    estado_chegada,
    estado_mudanca,
    i,
    f,
    ok = 0,
    contnc = 0;

void main()
```

```
{
ok = obtem_dados_iniciais();
fclose(arq_entra);
if (ok == 0)
{ //ok
trata_probabilidade(&lista1);
arq_sai = fopen("mmpbn.txt", "w");
mudanca_estado = 0;
chegada_de_pacote = 0;
contnc=contnc+1;

/* Comeca o laco principal do simulador */

while (contnc != num_ciclos)
{
  trata_chegada(&lista1);
  trata_estado(&lista1);
  contnc=contnc+1;
} /* termino do laco while */


 printf("Fim da simulacao!!!\n");
 getch();

 fclose(arq_sai);
} //ok
else
{
  printf("Programa terminado de maneira anormal!!!\n");
}
}

float GNR(void)
{
        float result,rannd;
        rannd = rand();
        result = rannd/32768;
        return(result);
}

void grava_dados(int vfonte,int vestado,int vtempo,int vintervalo)
   {
          registro reg_dados;
          reg_dados.fonte = vfonte;
          reg_dados.virgula0 = ',';
          reg_dados.estado = vestado;
          reg_dados.virgula1 = ',';
          reg_dados.tempo = vtempo;
          reg_dados.virgula2 = ',';
          reg_dados.intervalo = vintervalo;
          reg_dados.ponto_virgula = ';';
          fprintf(arq_sai, "%d",reg_dados.fonte);
          fprintf(arq_sai, "%c",reg_dados.virgula0);
          fprintf(arq_sai, "%d",reg_dados.estado);
          fprintf(arq_sai, "%c",reg_dados.virgula1);
          fprintf(arq_sai, "%d",reg_dados.tempo);
```

```
            fprintf(arq_sai, "%c",reg_dados.virgula2);
            fprintf(arq_sai, "%d",reg_dados.intervalo);
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);

    }


void trata_chegada(struct lista *no)
{ //a
no = primfonte;
do
{
  chegada_de_pacote = GNR();
  if (chegada_de_pacote > no->matriz_chegada[no->estado_atual_da_fonte])
    {
     fonte_chegada = no->num_fonte;
     estado = no->estado_atual_da_fonte;
     ref2 = contnc;
     grava_dados(fonte_chegada,estado,contnc,ref2-ref1);
     ref1 = ref2;
    }
}while((no=no->proxfonte)!=NULL);
return;
} //a

void trata_estado(struct lista *no)
{ //a
int i,j;
float VRP1,VRP2;
no = primfonte;
do
{
mudanca_estado = GNR();
i = no->estado_atual_da_fonte;
for (j = 0;j < no->num_estados_fonte; j++)
{  //c
 VRP1 = no->matriz_estado[i][j];
 VRP2 = no->matriz_estado[i][j+1];

 if ((mudanca_estado >= VRP1)&&(mudanca_estado < VRP2))
   {  //d
     no->estado_atual_da_fonte = j;
   }  //d
}  //c

if ((mudanca_estado >= no->matriz_estado[i][no->num_estados_fonte-1])&&
   (mudanca_estado < 1))
 { //e
     no->estado_atual_da_fonte = j-1;
 } //e
} while ((no=no->proxfonte)!=NULL);
return;
} //a

void trata_probabilidade(struct lista *no)
{ //a
```

```
int i,j;
float matriz_aux[10];
no=primfonte;
do
{  //b
 for (i = 0;i<= num_estados - 1;i++)
             { //c
                  for (j = 0;j <= num_estados - 1;j++)
                   { //d
                       matriz_aux[j]=no->matriz_estado[i][j];
                   } //d
                   no->matriz_estado[i][0] = 0;
                  for (j = 1;j <= num_estados - 1;j++)
                   { //e
                     no->matriz_estado[i][j] = matriz_aux[j-1]
                     + no->matriz_estado[i][j-1];
                   } //e
             } //c
}while ((no=no->proxfonte)!=NULL);//b
return;
} //a

// Rotina para inicializar vetores e matrizes

int obtem_dados_iniciais(void)
{ //a
 if(( arq_entra = fopen("emmpbn.txt", "r")) == NULL)
   {
   printf("\nErro - Arquivo de Entrada Danificado ou Nao Encontrado\n");
   exit (1);
   }
 else
 { //b
  fscanf(arq_entra,"%d %d",&num_fontes,&num_ciclos);
  if (num_fontes <= 0) {   num_fontes = 1;}
  for(f=1;f<=num_fontes;f++)
  { //c
   // Aloca memoria para a fonte
    if (primfonte == NULL)
          { //d
            primfonte = (struct lista*) malloc(num_fontes*sizeof(struct lista));
            if (primfonte==NULL) exit (1);
            obtem_dados(&lista1);
            *primfonte=lista1;
            newfonte=primfonte;
          } //d
    else
          { //e
             obtem_dados(&lista1);
             newfonte->proxfonte=(struct lista*) malloc(num_fontes*sizeof(struct lista));
             if (newfonte->proxfonte==NULL) exit (1);
             newfonte=newfonte->proxfonte;
             *newfonte=lista1;
          } //e
  }//c
             newfonte->proxfonte=NULL;
```

```
 }//b
return(0);
} //a

void obtem_dados(struct lista *no)
{
  int i,j;

 fscanf(arq_entra,"%d %d %d",
            &no->num_fonte,//1
            &no->num_estados_fonte,    //2
            &no->estado_atual_da_fonte); //3
            num_estados = no->num_estados_fonte;
for (i = 0;i<=num_estados-1;i++)
  {
    for (j = 0;j<=num_estados-1;j++)
      {
          fscanf(arq_entra,"%f",&no->matriz_estado[i][j]);
      }
  }
for (i = 0;i<=num_estados-1;i++)
  {
   fscanf(arq_entra,"%f",&no->matriz_chegada[i]);
  }
return;
}
```

9) Modelo de Tráfego Auto-Semelhante I

```
// Programa que gera trafego Auto-Similar com RMD - Lau et al
// Gerar tra‡os em um intervalo de 0 a T

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>
#include <bios.h>
#include <conio.h>

// Funcao Gaussiana
double gauss(double media, double var);
// Fim

//Definindo Vari veis
typedef struct {
  double ponto;
  char virgula;
  double z;
  char ponto_virgula;
  } registro;

FILE *arq_sai;

registro reg_dados;

int min,
   max,
   meio,
   i,
   j,
   T,
   n,
   d;

double z [2000],
     A [2000],
     M,
     a,
     H,
     s [2000],
     desv_desloc,
     v,
     rv;

void main()
{

// Inicializando vari veis:
// T = 2 elevado a n
M = 30;
a = 5;
n = 10;
```

```
T = pow(2,n);
H = 0.5;
max = T;
min = 1;
meio = T/2;
desv_desloc = pow(T,2*H);

z [min] = 0;
v = pow(T,2*H);
rv = sqrt(v);
z [max] = gauss (0,v);
s [1] = pow(2,-H)*sqrt(1 - pow(2,2*H - 2));
v = desv_desloc*s [1];
rv = sqrt(v);
z [meio] = (z [max] + z [min])/2 + gauss(0,rv);

max = meio;
meio = meio/2;

for (i = 1;i < n;i++)
{
  for (d=1;d <= pow(2,i);d++)
    {
     s[i+1] = pow(2,-H)*s[i];
     v = desv_desloc*s[1];
     rv = sqrt(v);
     z [meio] =  (z [max] + z [min])/2 + s[i+1]*gauss(0,rv);
     min = max;
     max = max + (T/pow(2,i));
     meio = (max + min)/2;
    }
    min = 0;
    max = (T/pow(2,i+1));
    meio = (max + min)/2;
    }

// Geracao de tr fego FGN
// M -> TAXA M□DIA DE CHEGADA
// a -> FATOR DE PICO

arq_sai = fopen("LAU.txt", "w");

for (j=1;j <= T;j++)
{
 A [j] = M + sqrt(a*M)*(z [j+1] - z [j]);
 reg_dados.ponto = A[j];
 reg_dados.virgula = ',';
 reg_dados.z = z [j];
 reg_dados.ponto_virgula = ';';
 fprintf(arq_sai, "%-4.2lf",reg_dados.ponto);
 fprintf(arq_sai, "%c",reg_dados.virgula);
 fprintf(arq_sai, "%-4.2lf",reg_dados.z);
 fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);
 //printf("%-4.2lf\n",z[j]);
}
 fclose(arq_sai);
```

```
 printf("Fim da Simula‡Æo");
 getch();
 }

double gauss (double media, double var)
// Retorna Amostra de distribuicao gaussiana
{
        double resultado,num,num1;
        num1 = rand();
        num =num1/32768;
        resultado=sqrt(-2*log(num))*sin(2*3.1415926*num);
        resultado=resultado*var+media;
        return(resultado);
}
```

10) Modelo de Tráfego Auto-Semelhante II

```
//
Gerador dr tráfego - Algoritmo RMD - leboudec
André Luiz Garcia Pereira
//
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <string.h>


//Funcoes para Calculo de min e max
double max_function(void);
double min_function(void);
//Fim

//Funcao para gravar em arquivo
void grava_dados(double m,double p);
//Fim

// Funcao Gaussiana
double gauss(double media, double var);
// Fim

// Define estrutura
typedef struct {
  double mean;
  char virgula;
  double peak;
  char ponto_virgula;
  } registro;
//Fim

//Define arquivo de saida
FILE *arq_sai;
//Fim
```

```
double reduction,
       H = 0.7,
       scale_mean,
       max = 1,
       min = 1,
       mean = 0,
       peak= 0,
       factor = 2,
       link_speed = 2,
       over_booking = 0.5,
       rho = 0.9,
       translation = 0,
       vmean [2000],
       vpeak [2000];

int    nr_of_reservation_steps = 100,
       step = 0,
       left = 0,
       right = 0,
       i = 0;

void main()
{
 /* Gera o MEAN e o PEAK RATE */

 arq_sai = fopen("leboudec.txt", "w");

 reduction = 1.0/pow(2.0,H);
 scale_mean = 1.0;
 step = nr_of_reservation_steps-1;
 while (step > 1)
  {
    left = 0;
    right = step;
    while (right < nr_of_reservation_steps)
          {
           vmean[(left+right)/2] = 0.5*(vmean[right]+vmean[left])
           + scale_mean*gauss(0.0,1.0);
           left = right;
           right+=step;
          }
     scale_mean *= reduction;
     step/=2;
  }

 /* Gera o Ruido (noise) */

 for (i=0;i<nr_of_reservation_steps-1;i++)
  {
    vmean[i] = vmean[i+1] - vmean[i];
  }

 max = max_function();
 min = min_function();
 factor = link_speed*over_booking*rho / (max - min);
 translation = factor * min * -1.0;
```

```
  for(i=0;i<nr_of_reservation_steps;i++)
   {
     vmean[i] = vmean[i] * factor + translation;
     mean = vmean [i];
     vpeak[i] = vmean[i] / rho;
     peak = vpeak [i];
     grava_dados(mean,peak);
   }

 printf("Fim da simulacao!!!\n");
 getch();

 fclose(arq_sai);
 }

double max_function()
{
 int i;double valor;
 valor = vmean [0];

 for (i=0;i<nr_of_reservation_steps-1;i++)
  {
   if (valor < vmean [i])
      valor = vmean [i];
  }
return(valor);
}

double min_function()
{
 int i;double valor;
 valor = vmean [0];

 for (i=0;i<nr_of_reservation_steps-1;i++)
  {
   if (valor >  vmean [i])
      valor = vmean [i];
  }
return(valor);
}

void grava_dados(double m,double p)
   {
          registro reg_dados;
          reg_dados.mean = m;
          reg_dados.virgula = ',';
          reg_dados.peak = p;
          reg_dados.ponto_virgula = ';';
          fprintf(arq_sai, "%-4.2lf",reg_dados.mean);
          fprintf(arq_sai, "%c",reg_dados.virgula);
          fprintf(arq_sai, "%-4.2lf",reg_dados.peak);
          fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);
   }

double gauss (double media, double var)
// Retorna Amostra de distribuicao gaussiana
```

```
{
        double resultado,num,num1;
        num1 = rand();
        num =num1/32768;
        resultado=sqrt(-2*log(num))*sin(2*3.1415926*num);
        resultado=resultado*var+media;
        return(resultado);
}
```

11)Eventos Raros baseado em um sistema de filas M/M/1

```
// Simulador MM1 com Importance Sampling e Large Deviation
// Autor: Andr, Luiz Garcia Pereira

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <math.h>
#include <conio.h>
#include <time.h>

#define IS 1
#define MC 2
#define DESOCUPADO 0
#define OCUPADO 1

// Define funcoes a serem utilizadas pelo programa MM1 com IS
double GNR(double amostra);
void inicializa(void);
void grava_dados(double ct,int nc,double pe,double tmf,double em,double ed);
void Calculo_da_Probabilidade_MC(void);
void Calculo_da_Probabilidade_IS(void);
// Fim

// Define Registro de Saida

typedef struct {
  double cpu_time;
  char virgula0;
  int n_ciclos;
  char virgula1;
  double prob_evento;
  char virgula2;
  double t_medio_fila;
  char virgula3;
  double empirical_mean;
  char virgula4;
  double empirical_stddv;
  char ponto_virgula;
  } registro;

//Fim de Definicao de Registro
```

```
//Define Arquivo

FILE *arq_sai;

//Fim

// Define variaveis Globais
double chegada_de_pacote,
            termino_servico,
            lambda,
            lambda_LD,
            mi,
            mi_LD,
            relogio,
            media,
            desvio_padrao,
            tinicial,
            tfinal,
            tempo,
            L,
            tamanho_medio_fila,
            soma_evento,
            prob_de_evento;

int flag ,
   repeticao = 1 ,
   Num_repeticoes = 0,
   n_ciclo = 0,
   fim_simulacao = 0,
   livre = DESOCUPADO,
   fila = 0,
   Num_chegadas = 0,
   tam_max,
   cont = 0;
// Fim

// Comeca a funcao principal do Simulador

void main()
{
inicializa();
if (flag == IS)
            chegada_de_pacote = GNR(lambda_LD);
     else
            chegada_de_pacote = GNR(lambda);
termino_servico = chegada_de_pacote;
printf(" inicio do simulador \n");

// Define L ->TAXA DE PROBABILIDADE
L = pow(lambda/mi,tam_max-1);
//printf("L : %-4.20lf \n",L);
//getch();
// Fim
if (flag == IS)
arq_sai = fopen("ismm1.txt", "w");
else
```

```
arq_sai = fopen("mcmm1.txt", "w");


// Comeca o laco que define o numero de repeticoes que o
// simulador vai ter referente a simulacao em varios ciclos
// diferentes

tinicial = time(NULL);
while (repeticao <= Num_repeticoes)
{

while (cont <= fim_simulacao*repeticao)
{  // inicio do laco principal

if (chegada_de_pacote <= termino_servico)
        relogio = chegada_de_pacote;
    else
        relogio = termino_servico;

if (relogio == chegada_de_pacote)  //Houve Chegada
  { // inicio do primeiro if
    if (flag == IS)
        chegada_de_pacote = relogio + GNR(lambda_LD);
    else
        chegada_de_pacote = relogio + GNR(lambda);

    Num_chegadas = Num_chegadas + 1;
    if (livre == DESOCUPADO) // Servidor Desocupado
        { // inicio do segundo if
            livre = OCUPADO;    // Ocupa Servidor
            if (flag == IS)
               termino_servico = relogio + GNR(mi_LD);
            else
               termino_servico = relogio + GNR(mi);
        } // termino do segundo if

    else  // Servidor Ocupado
        { // inicio do primeiro else
          fila = fila + 1;
          //printf("fila: %d \n",fila);
          tamanho_medio_fila = tamanho_medio_fila + fila;
          if (fila >= tam_max)
                {
                    fila = tam_max;
                    // AQUI HOUVE ESTOURO
                    n_ciclo = n_ciclo + 1;
                    if (flag == MC)
                       soma_evento = soma_evento + 1;
                    else
                       soma_evento = soma_evento + L;
                //    printf("ESTOURO!!!");
                }
        } // termino do segundo else

  } // termino do primeiro if
  else     //Houve um fim de Servico
```

```
  { // inicio do segundo else
    if ( fila > 0)
          { // inicio do terceiro if
                fila = fila - 1;
                livre = OCUPADO;
                if (fila == 0) n_ciclo = n_ciclo + 1;
                if (flag == IS)
                      termino_servico = relogio + GNR(mi_LD);
                else
                      termino_servico = relogio + GNR(mi);
           } // termino do terceiro if
          else
           { // inicio do terceiro else
                livre = DESOCUPADO;
                fila = 0;
                n_ciclo = n_ciclo + 1;
            } // termino do terceiro else
  } // termino do segundo else
  cont = cont + 1;
 } // termino do laco while -> ciclos
tfinal = time(NULL);
// Calcula o valor da probabilidade
if (flag == MC)
  Calculo_da_Probabilidade_MC();
else
  Calculo_da_Probabilidade_IS();
tempo = tfinal - tinicial;
// Grava Resultados
grava_dados(tempo,n_ciclo,prob_de_evento,tamanho_medio_fila,media,desvio_padrao);
repeticao = repeticao + 1;
cont = 0;
} // Termino do laco while -> repeticoes

 printf("Fim da simulacao!!!\n");
 getch();
 fclose(arq_sai);

}

// Funcao geradora de numeros aleatorios

double GNR (double amostra)
{
  double z,g;
        z=rand();
        g=z/32768;
        //g=-amostra*log(g);
        g=-(1/(amostra))*log(1-g);
  return (g);
}
// Fim da funcao GNR

// Funcao inicial de obtencao de dados

void inicializa(void)
{
```

```
int resp = 0;

printf("Simulador MM1 com Importance Sampling \n");
printf("\n");

printf("Quantos simulacoes deseja fazer? \n");
scanf("%d",&Num_repeticoes);
if (Num_repeticoes < 1) Num_repeticoes = 1;
printf("\n");

printf("Digite o n£mero de repeticoes da simulacao:\n");
scanf("%d",&fim_simulacao);
printf("\n");
lambda = 2;
while ((lambda < 0)||(lambda > 1))
{
  printf("Digite taxa de chegada:\n");
  scanf("%lf",&lambda);
  mi = 1 - lambda;
}

printf("Digite o tamanho da fila: \n");
scanf("%d",&tam_max);
printf("\n");
//if (tam_max < 20) tam_max = 20;

while ((resp != 1)&&(resp != 2))
{
printf("Deseja simular via IS(Digite 1) ou Normal(Digite 2)?\n");
scanf("%d",&resp);
}
switch (resp)
{
 case 1:
 flag = IS;
 // Large Deviation
 lambda_LD = mi;
 mi_LD = lambda;
// printf("lambda: %-4.2lf lambda_LD: %-4.2lf\n",lambda,lambda_LD);
// printf("mi: %-4.2lf mi_LD: %-4.2lf\n",mi,mi_LD);
// getch();
 //Fim
 break;
 case 2:
 flag = MC;
 break;
}
return;
}

// Fim da funcao inicial

// Funcao que gera resultados em arquivo

void grava_dados(double ct,int nc,double pe,double tmf,double em,double ed)
{
```

```
        registro reg_dados;
        reg_dados.cpu_time = ct;
        reg_dados.virgula0 = ',';
        reg_dados.n_ciclos = nc;
        reg_dados.virgula1 = ',';
        reg_dados.prob_evento = pe;
        reg_dados.virgula2 = ',';
        reg_dados.t_medio_fila = tmf;
        reg_dados.virgula3 = ',';
        reg_dados.empirical_mean = em;
        reg_dados.virgula4 = ',';
        reg_dados.empirical_stddv = ed;
        reg_dados.ponto_virgula = ';';
        fprintf(arq_sai, "%-4.2lf",reg_dados.cpu_time);
        fprintf(arq_sai, "%c",reg_dados.virgula0);
        fprintf(arq_sai, "%d",reg_dados.n_ciclos);
        fprintf(arq_sai, "%c",reg_dados.virgula1);
        fprintf(arq_sai, "%-4.20lf",reg_dados.prob_evento);
        fprintf(arq_sai, "%c",reg_dados.virgula2);
        fprintf(arq_sai, "%-4.8lf",reg_dados.t_medio_fila);
        fprintf(arq_sai, "%c",reg_dados.virgula3);
        fprintf(arq_sai, "%-4.8lf",reg_dados.empirical_mean);
        fprintf(arq_sai, "%c",reg_dados.virgula4);
        fprintf(arq_sai, "%-4.8lf",reg_dados.empirical_stddv);
        fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);
}

//Funcao que calcula a probabilidade de perda via MC

void Calculo_da_Probabilidade_MC(void)
{
tamanho_medio_fila = tamanho_medio_fila/(double)cont;
prob_de_evento = soma_evento/(double)n_ciclo;
media = 0;
desvio_padrao = 0;
}
// Fim

//Funcao que calcula a probabilidade de perda via IS

void Calculo_da_Probabilidade_IS(void)
{
tamanho_medio_fila = tamanho_medio_fila/(double)cont;
prob_de_evento = soma_evento/(double)n_ciclo;
media = 0;
desvio_padrao = L*prob_de_evento - prob_de_evento*prob_de_evento;
}
//Fim
```

12)Eventos Raros baseado em em redes de Tráfego Auto-Similares

```
// Versao - HUANG
// modificada para unix em 26/02/97 -  Andre

#include <stdio.h>
```

```
#include <math.h>
#include <stdlib.h>
#include <curses.h>
#include <string.h>
#include <time.h>
#define number_of_sample 2000

// Funcao para gravar dados finais
void grava_dados(int sTime1,double tQlength1,double qVar1,int count1);
//Fim

// Define arquivo de saida
FILE *arq_sai;
//Fim

//Define Registro de saida

typedef struct {
  int    sTime;
  char   virgula0;
  double log10tQLength;
  char   virgula1;
  double log10qVar;
  char   virgula2;
  double qVar;
  char   virgula3;
  int    count;
  char   ponto_virgula;
} registro;
//Fim

// Funcao que mostra resultados
void resultados(void);
//Fim

// Funcao para obter o valor inicial de mu
double putInit(double mu1);
//Fim

// New function - ultima parte do programa do HUANG
void ultima(void);
//Fim

// Funcao para calcular o valor twisted
double getTwist(void);
//Fim

// Funcao para a proxima taxa de chegada
double getNValue(void);
//Fim

// Funcao Hosking
double getAValue(void);
//Fim

// Funcao  Gaussiana
```

```
//double rannd(int);
double gauss(double gmean, double gdev);
// Fim

// Definicoes do Programa Principal
int    sNumber,     //Sample number
          repNum,  // Replication number
          step,
          sTime,
          count,
          index_step,
          i1,i2,i3;

double qLength, // queue length
      mQLength,
      qProb [2000],
      qVar,
      maxQ,
      mu,
      hpara,
      tQLength,
      wLoad; // work load
float rao1,
     th;
//Fim

// Definicoes  Hosking
double phi [number_of_sample] [number_of_sample], // partial correlation
m [number_of_sample],                    //conditional mean value
rao [number_of_sample],                  //service rate
d,
d1,
n,
n1,
v [number_of_sample],                    // conditional variance
rv [number_of_sample],                   //square root of conditional variance
x [number_of_sample],                    //simulated sample point
co [number_of_sample],                   //intermidiate value for twistion
mu,                                 // twisted service rate
wload,                               //service rate
inter1,
inter2,
inter3,
inter4,
hpara;                               // Hurst parameter

int   ii,iii;

// Fim


void main()
{

// Inicializa  valores para Hosking
i1=1;
```

```
inter1=0.0;
inter2=0.0;
v[0]=1.0;
n=0.0;
d=1.0;
hpara=0.7;
// Fim

   /*printf("Input: \n");
printf("Input shift mean: ");
scanf("%f",&rao1);
printf("Input the threshold: ")
scanf("%f",&th);
printf("Input replic number: ");
scanf("%d",&repNum);
printf("Input stop time: ");
scanf("%d",&sTime);
*/

arq_sai = fopen("huang.txt", "w");

rao1=2.2;
th=20.0;
repNum=1000;
sTime=50;
mu=rao1;
step=50;

for (index_step=0;index_step<2;index_step++)
 {
   sNumber=0;
   tQLength=0.0;
   mQLength=0.0;
   hpara=0.7;
   count=0;
   sTime=(index_step+1)*step;


   for (ii=0;ii<repNum+1;ii++)
        {
       //printf("Replica : %i\n" , ii);
       qProb[ii]=0.0;
          srand(ii+1);
          qLength=putInit(mu);
          maxQ=qLength;
          sNumber=0;
          if (ii==0)
           {
             while (sNumber < sTime)
          // Obtem-se sTime's valores de X e phi
                {
                  sNumber++;
                  wLoad=getAValue();
                  qLength=qLength+wLoad;
                  if (qLength>maxQ){ maxQ=qLength;}
              }
```

```
            }
          else
          {
            while (maxQ<th && sNumber<sTime)
                // Retira valores para fila e verifica se
           // ha estouro
           {
               sNumber++;
                  wLoad=getNValue();
                  qLength=qLength+wLoad;

             // impressao do tam. da fila
            //printf("queue size: %9.9lf\n",qLength);

            if (qLength>maxQ) {maxQ=qLength;}
                }
          if (maxQ>=th)
                {
                  count++;
                  qProb[ii-1]=getTwist();
                  mQLength+=qProb[ii-1];
            //printf(" Num. de estouros: %i\n",count);
            //printf(" Probabilidade Twist: %9.9lf\n",qProb[ii-1]);
                }
           }
         }
ultima();
 }
  printf("Fim da simulacao!!!\n");
  getchar();

  fclose(arq_sai);
}


// FUNCOES GAUSSIANAS
double rannd(double number)
/* return a random value between 0 and 1 */
{
        double  number1;
        number1 = rand();
        //printf("Number1 %4.9lf\n",number1);
    //  getch();
        //number =(double) number1/(double)2147483647;
     number = number1/32768;
        //printf("Number1/32768: %4.9lf\n",number);
    //  getch();
        return (number);
/*      return((double)number/(double)2147483647);*/
}

double expon(double fmean)
/* returns sample from exponential distribuition with mean of fmean */
{
        double result;
        result=rannd(0);
```

```
    //  printf("result %4.2lf\n",result);
    //  getch();
        result = -fmean * log(result);
    //  printf("-fmean * log(result): %4.2lf\n",result);
    //  getch();
        return(result);
}

double gauss (double gmean, double gdev)
/* returns sample from gaussian distribuition */
{
        double gresult;
        gresult=sqrt(-2*log(rannd(0)))*sin(2*3.1415926*rannd(0));
    //  printf("gresult %4.2lf\n",gresult);
    //  getch();
        gresult=gresult*gdev+gmean;
    //  printf("gresult*gdev+gmean: %4.2lf\n",gresult);
    //  getch();
        return(gresult);
}




// FIM FUNCOES GAUSSIANS

// FUNCAO HOSKING

double getAValue(void){
 /* calculate Hosking's parameters */

 m[i1]=0.0;
 rao[i1]=(pow(i1+1,2*hpara)-2*pow(i1,2*hpara)+
                pow(i1-1,2*hpara))/2.0;

 n1=rao[i1];

 for (i2=1;i2<i1;i2++)
        {
        n1+=-phi[i1-1][i2]*rao[i1-i2];
         }

 d1=d-pow(n,2)/d;
 phi[i1][i1]=n1/d1;
 n=n1;
 d=d1;

 for (i2=1;i2<i1;i2++)
        {
        phi[i1][i2]=phi[i1-1][i2]-
           phi[i1][i1]*phi[i1-1][i1-i2];
        }

 for (i2=1;i2<=i1;i2++)
        {
```

```
            m[i1]=m[i1]+phi[i1][i2]*x[i1-i2];
             }

 co[i1]=1.0;
 for (i2=1;i2<=i1;i2++)
{
  co[i1]+=-phi[i1][i2];
}

 v [i1] = (1-phi[i1][i1]*phi[i1][i1])*v[i1-1];
 rv[i1]=sqrt(v[i1]);

 x[i1]=gauss(m[i1],rv[i1]);

 i1++;

 return(x[i1-1]-wload);

 }

// Fim Hosking

// Gera a proxima taxa de chegada

double getNValue(void)
{
  /* generate next arrival rate */

m[i1]=0.0;

for (i3=1;i3<=i1;i3++)
 {
   m[i1]=m[i1]+phi[i1][i3]*x[i1-i3];
 }
   x[i1]=gauss(m[i1],rv[i1]);
  i1++;
 return(x[i1-1]-wload);
}

// Fim taxa de chegada

double getTwist(void)
{
/* calculate twisted value */

inter4=0.0;
for (i2=1;i2<i1;i2++)
  {
   inter1=1;
   inter1=co[i2];
   inter2=-mu*inter1/v[i2];/* O(i) value */
   inter3=-inter2*(-2*m[i2]+mu*(inter1-2.0)+4.0)/2.0;
   inter4+=inter2*(x[i2]-wload)-inter3;
   }
inter4=inter4-mu*x[0]-mu*mu/2;
// printf("inter4: %4.2lf\n",inter4);
```

```
inter4=exp(inter4);
return (inter4);

}

void ultima(void)
// Ultima Funcao  criada para entender a ultima parte do prg. Huang
{

 tQLength=mQLength/(float)repNum;
 qVar=0.0;
 for (iii=0;iii<repNum;iii++)
  {
    qVar+=(qProb[iii]-tQLength)*(qProb[iii]-tQLength);
  }

  qVar=qVar/(float)(repNum-1);
  qVar=qVar/tQLength/tQLength;

   grava_dados(sTime,tQLength,qVar,count);

//  printf(" %10d  %10.4f %10.4f %10.4f %10d\n",sTime,log10(tQLength),log10(qVar),qVar,count);
  // getch();
  count=0;
}

double putInit(double mu1)
/* Obtencao do valor de mu*/
{
 i1=1;
 inter1=0.0;
 inter2=0.0;
 v[0]=1.0;
 x[0]=gauss(0.0,v[0]);
 mu=mu1;
 wload=2.0-mu;
 return(x[0]-wload);
}

void resultados(void)
{
 printf("Resultados da simulacao...\n");
}

// Funcao para gravar resultados em disco

  void grava_dados(int sTime1,double tQLength1,double qVar1,int count1)
    {
          registro reg_dados;
          reg_dados.sTime = sTime1;
          reg_dados.virgula0 = ',';
          reg_dados.log10tQLength = log10(tQLength1);
       reg_dados.virgula1 = ',';
       reg_dados.log10qVar = log10(qVar1);
       reg_dados.virgula2 = ',';
       reg_dados.qVar = qVar1;
```

```
        reg_dados.virgula3 = ',';
        reg_dados.count = count1;
            reg_dados.ponto_virgula = ';';
        fprintf(arq_sai, "%10d",reg_dados.sTime);
        fprintf(arq_sai, "%c",reg_dados.virgula0);
            fprintf(arq_sai, "%-4.2lf",reg_dados.log10tQLength);
            fprintf(arq_sai, "%c",reg_dados.virgula1);
        fprintf(arq_sai, "%-4.2lf",reg_dados.log10qVar);
        fprintf(arq_sai, "%c",reg_dados.virgula2);
            fprintf(arq_sai, "%-4.2lf",reg_dados.qVar);
        fprintf(arq_sai, "%c",reg_dados.virgula3);
        fprintf(arq_sai, "%10d",reg_dados.count);
            fprintf(arq_sai, "%c\n",reg_dados.ponto_virgula);
}
```

Bibliografia

[1] Bernabei et ali: "ATM system buffer design under very low cell loss probability constraints". IEEE Transactions on Communications,paper 8C.3.1,1991.

[2] Glasserman, Paul e Kou, Shing-Gang: "Analysis of an Importance Sampling Estimator for Tandem Queues". ACM Transactions on Modeling and Computer Simulation, EUA, janeiro,Vol.5,No. 1,páginas 22-42.,1995.

[3] Helvik, Bjarne e Heegaard, Poul: "A Technique for Measuring Rare Cell Losses in ATM Systems". ITC 14, Trondheim (Noruega),páginas 917-1021,1994.

[4] Droz, Patrick e Le Boudec, Jean- Yves: "A High-Speed Self-Similar ATM Traffic Generator Including Traffic Parameters".IBM Research Division,Zurich (Suiça), maio,RZ 2854,1996.

[5] Shanmugan, K. e Balaban, P.: "A Modified Monte-Carlo Simulation Technique for the Evaluation of Error Rate in Digital Communication Systems". IEEE Transactions on Communications, Washington D.C. (EUA), novembro,Vol. COM-28,No 11, páginas 1917-1923,1980.

[6] Parekh, Shyam e Walrand, Jean: "A Quick Simulation Method for Excessive Backlogs in Networks of Queues". IEEE Transactions on Automatic Control, Berkeley (EUA), janeiro,Vol. 34,No 1, páginas 55-65,1989.

[7] Devetsikiotis, Michael e Townsend, Keith: "A Dynamic Importance Sampling Methodology for the Efficient Estimation of Rare Event Probabilities in Regenerative Simulations of Queueing Systems". IEEE Transactions on Communications,páginas 1290-1296,1992.

[8] Weiss, Alan: "An Introduction to Large Deviations for Communication Networks". IEEE Journal on Selected Areas in Communications, EUA, abril,Vol. 13, No. 6, páginas 938-952,1995.

[9] Freebersyser et ali: "Efficient Simulation of High Speed Tandem Networks Using Importance Sampling and Stochastic Gradient Techniques". Center for Communications & Signal Processing, North Carloina State University, Telecomunications Research Institute of Ontario, Raleigh CA(EUA), Ontario(Canada),páginas 1095-1099,1995.

[10] Wang, Qinglin e Frost, Victor: "Efficient Estimation of Cell Blocking Probability for ATM Systems". EUA, abril,Vol.1,No 2, páginas 230-235,1993.

[11] Heidelberger, Philip: "Fast Simulation of Rare Events in Queueing and Reliability Models". ACM Transactions on Modeling and Computer Simulation,IBM T.J. Watson Research Center, Hawthorne, Nova York(EUA),janeiro,Vol. 5, No. 1, páginas 43-85,1995.

[12] Huang et ali: "Fast Simulation for Self-Similar Traffic in ATM Networks". IEEE ICC'95, Seattle(EUA), junho,7 páginas,1995.

[13] Mitchell,R.L.: "Importance Sampling Applied to Simulation of False Alarm Statistics". IEEE Transactions on Aerospace and Electronic Systems, Hanscom AFB,MA (EUA),janeiro,Vol.Aes-17,No.1,páginas 15-24,1981.

[14] Cottrell et ali: "Large Deviation and Rare Events in the Study of Stochastic Algorithms".IEEE Transactions on Automatic Control, Orsay (França),setembro,Vol. AC-28,No. 9, páginas 907-920,1983.

[15] Simonian, Alain e Guilbert, Jacky: "Large Deviations Approximation for Fluid Queues Fed by a Large Number of On/Off Sources". IEEE Journal on Selected Areas in Communications, agosto,Vol. 13, No. 6, páginas 1017-1027,1995.

[16] Huang et ali: "Modeling and Simulation of Self-Similar Variable Bit Rate Compressed Video: A Unified Approach".ACM, SIGCOMM'95,Cambridge MA(USA),páginas 114-125,1995.

[17] Devetsikiotis, Michael e Townsend, Keith: "On The Efficient Simulation of Large Communication Networks Using Importance Sampling". IEEE Transactions on Communications, North Carolina State University, Raleigh, North Carolina(EUA),páginas 1455-1459,1992.

[18] Frater, M.R. et ali: "Optimally Efficient Simulation of Buffer Overflows in Queues with Deterministic Service Times via Importnace Sampling", Australian Telecomunications and Electronics Research Board (ATERB) e Centre for Information Science Research (CISR), maio,Vol.24,No.1, 10 páginas,1990.

[19] Milsten, Laurence B.: "Performance Monitoring of Digital Communication Systems Using Extreme-Value Theory".IEEE Transactions on Communications,

University of California at San Diego, La Jolla, CA (EUA), setembro,páginas 1032-1036,1976.

[20] Nicol, David e Heidelberger, Philip: "Parallel Execution for Serial Simulators". NSF,NASA,Institute for Computer Application in Science and Engineering,NASA Langley Research Center, Hampton,VA (EUA),setembro,24 páginas,1995.

[21] Kesidis, G. e Walrand, J.: "Quick Simulation of ATM Buffers with On-Off Multiclass Markov Fluid Sources". ACM Transactions on Modeling and Computer Simulation, University of California, Berkeley CA (EUA), junho,Vol.2,No.3,páginas 269-276,1993.

[22] Kovalenko, Igor N.: "Rare events in queueing systems-A survey". J.C.Baltzar AG, Science Publishers, V.M.Glushkov Institute of Cybernetics, Academy of Sciences of Ukraine, Kiev (Ukrania), julho,50 páginas,1993.

[23] Devetsikiotis, Michael e Towsend, Keith: "Statistical Optimization of Dynamic Importance Sampling Parameters for Efficient Simulation -of Communication Networks". IEEE/ACM Transactions on Networking, Center for Communications and Signal Processing, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh CA (EUA), junho,Vol.1,No.3,páginas 293-305,1993.

[24] Devetsikiotis et ali: "Stochastic Gradient Optimization of Importance Sampling for the Efficient Simulation of Digital Communication Systems". IEEE Transactions on Communications,26 páginas,1994.

[25] Devetsikiotis et ali: "Stochastic Gradient Techniques for The Efficient Simulation of High-Speed Networks Using Importance Sampling", IEEE Transactions on Communications,páginas 751-755,1993.

[26] Jeruchim, Michael C.: "Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems", IEEE Journal on Selected Areas in Communications, Philadelphia PA (EUA), janeiro,Vol. SCA-2,No. 1, páginas 153-170,1984.

[27] Frost, Victor S. e Melemed, Benjamin: "Traffic Modeling for Telecomunications Networks". IEEE Communications Magazine, Laboratory at the University of Kansas, Kansas (EUA), março,páginas 70-81,1994.

[28] Lundquist et ali: "Transmission of 40-Phase-Shift Keyed Time-Division Multiple Access over Satellite Channels". IEEE Transactions on Communications, setembro,Vol. com-22,No.9, páginas 1354,1974.

[29] Lau et ali: "Self-Similar Traffic Generation: The Random Midpoint Displacement Algorithm and Its Properties". IEEE Transactions on Communications,páginas 466-472,1995.

[30] Lank, Gerald W.: "Theorical Aspects of Importance Sampling Applied to False Alarms". IEEE Tranasactions on Information Theory,janeiro,Vol.IT-29,No.1, páginas 73-82,1983.

[31] Weistein,Stephen B.:"Theory and Application of Some Classical and Generalized Asymptotic Distributions of Extreme Values". IEEE Transactions on Information Theory,março,Vol. IT-19,No.2, páginas 148-154,1973.

[32] Monteiro, José Augusto Suruagy: "Rede Digital de Serviços Integrados de Faixa Larga(RDSI-FL)".IX Escola de Computação, Recife (Brasil),207 páginas,1994.

[33] Carmo et ali: "Cálculo de Descritores de Tráfego em Modelos Markovianos de Fontes Multimídia". XV Simpósio Brasileiro de Redes de Computadores, páginas 189-205,1995.

[34] Mayor, Gilberto e Silvester, John: "Simulation of an ATM Queueing System with a Fractional Brownian Noise Arrival Process". IEEE,1996.

[35] Parekh, Shyam P.: "Quick Simulation of Stationary Tail Probabilities at a Packet Switch". ITC 14, Trondheim (Noruega),páginas 887-895,1994.

[36] Smith et ali: "Quick Simulation: A Review of Importance Sampling Techniques in Communications Systems". IEEE Journal on Selected Areas in Communications, maio,Vol.15,No.4.páginas 597-613,1997.

[37] Heidelberger, Philip e Simha, Rahul: "Fast Simulation of a Voice-Data Multiplexer". IEEE, 1995.

[38] Kleinrock, Leonard: "Queueing Systems-Volume 1: theory". A Wiley-Interscience publication, John Wiley & Sons. Inc., 1975.

[39] Kovács,Z.L.: "Teoria da Probabilidade e Processos Estocásticos- com Aplicações em Engenharia de Sistemas e Processamento de Sinais".Edição Acadêmica São Paulo,OESP Gráfica SA,120 páginas,1996.

[40] Tanenbaum, Andrew S.: "Computer Networks".Prentice-Hall International Inc,660 páginas,1989.

[41] Soares et ali: "Redes de Computadores: das LANs,MANs e WANs às redes ATM". Rio de Janeiro,Editora Campus, 1995.

[42] Mandelbrot, B.:"A Fast Fractional Gaussian Noise Generator", Water Resources Research, Vol. 7, 543-553, 1971.

[43] Chi et ali: "Pratical Application of Fractional Brownian Motion and Noise to Synthetic Hydrology". Water Resources Research, Vol. 9, 1523-1533, 1973.