Ministério da
Ciência e Tecnologia

INPE-16698-TDI/1642

CONTROLE DE ADMISSÕES DE PACIENTES ELETIVOS: UMA APLICAÇÃO DE PROCESSOS MARKOVIANOS DE DECISÃO

Luiz Guilherme Nadal Nunes

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Solon Venâncio de Carvalho, e Rita de Cássia Meneses Rodrigues, aprovada em 09 de abril de 2010.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/03.10.15.17>

INPE
São José dos Campos
2010

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

INPE-16698-TDI/1642

CONTROLE DE ADMISSÕES DE PACIENTES ELETIVOS: UMA APLICAÇÃO DE PROCESSOS MARKOVIANOS DE DECISÃO

Luiz Guilherme Nadal Nunes

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Solon Venâncio de Carvalho, e Rita de Cássia Meneses Rodrigues, aprovada em 09 de abril de 2010.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/03.10.15.17>

INPE
São José dos Campos
2010

Dados Internacionais de Catalogação na Publicação (CIP)

Nunes, Luiz Guilherme Nadal.
N922c Controle de admissões de pacientes eletivos: uma aplicação de processos markovianos de decisão / Luiz Guilherme Nadal Nunes. – São José dos Campos : INPE, 2010.
285 p. ; (INPE-16698-TDI/1642)

Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2010.
Orientadores : Drs. Solon Venâncio de Carvalho, e Rita de Cássia Meneses Rodrigues.

1. Controle de admissões. 2. Fluxo de pacientes. 3. Processo markoviano de decisão. 4. Método evolutivo. 5. Método simulado. 6. Horizonte de planejamento reduzido. I.Título.

CDU 523.62-726

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Doutor(a) em

Computação Aplicada

Dr. **Horacio Hideki Yanasse**

Presidente / INPE / SJCampos - SP

Dra. **Rita de Cássia Meneses Rodrigues**

Orientador(a) / INPE / SJCampos - SP

Dr. **Soion Venâncio de Carvalho**

Orientador(a) / INPE / SJCampos - SP

Dr. **José Demisio Simões da Silva**

Membro da Banca / INPE / SJCampos - SP

Dr. **Armando Zeferino Milioni**

Convidado(a) / ITA / São José dos Campos - SP

Dr. **Paulo Renato de Morais**

Convidado(a) / UNIVAP / São José dos Campos - SP

Dr. **Takashi Yoneyama**

Convidado(a) / ITA / SJCampos - SP

Aluno (a): Luiz Guilherme Nadal Nunes

São José dos Campos, 09 de abril de 2010

AGRADECIMENTOS

Agradeço a segura orientação dos Doutores Solon e Rita.

Agradeço o apoio constante da Rede Sarah de Hospitais de Reabilitação.

Agradeço, especialmente, o esteio proporcionado pela minha companheira de todos os momentos, Fabiana Todesco.

RESUMO

Os principais objetivos do presente trabalho são: (1) modelar o controle de admissões de pacientes eletivos como um Processo Markoviano de Decisão (PMD), (2) desenvolver uma meta-heurística capaz de solucionar PMDs com grandes dimensões, e (3) testar a aplicabilidade do modelo proposto e o desempenho do método desenvolvido. O propósito de controlar a admissão de pacientes é promover uma utilização mais eficiente dos recursos hospitalares, prevenindo-se a ociosidade ou o uso excessivo dos recursos e considerando-se as suas importâncias relativas. O modelo de controle das admissões como um PMD atinge grandes dimensões, relativas aos espaços de estados e de ações, à medida que se consideram mais especialidades de pacientes, capacidade de admissões, disponibilidade e tipos distintos de recursos. Propõe-se, para a solução de PMDs com grandes dimensões, uma meta-heurística evolutiva em horizonte de planejamento reduzido e amostrado. Este método é implementado e testado para diversas instâncias do problema de controle de admissões modelado. Aplicando-se o tradicional Algoritmo de Iteração de Valores (AIV), foi possível determinarem-se políticas de decisão ótimas (POs) através das quais se pôde manter, considerando-se os resultados experimentais teóricos relativos ao modelo em proposição, o consumo dos recursos próximo aos níveis desejados de utilização. Empregando-se a meta-heurística desenvolvida neste estudo: (1) foram obtidas políticas de decisão para o modelo em proposição, cujos desempenhos se aproximaram dos desempenhos das POs e (2) foi possível obterem-se soluções para instâncias do problema modelado maiores do que as que o AIV, por falta de espaço de memória computacional para a execução do algoritmo e pelo excessivo tempo de processamento, poderia suportar. Contudo, para instâncias do problema modelado, compatíveis com aquelas que se podem verificar habitualmente em um hospital (instâncias que atingem milhões de possíveis estados e ações), o tempo de processamento para obtenção de políticas de decisão pela meta-heurística proposta foi grande (da ordem de dias). O modelo de decisão proposto é complexo devido à sua dinâmica estocástica e à sua dimensionalidade, mas possui grande potencial de aplicação, requerendo, para o seu aproveitamento prático, o desenvolvimento de novos métodos.

CONTROL OF ELECTIVE PATIENTS ADMISSIONS: A MARKOV DECISION PROCESSES APPLICATION

ABSTRACT

The main objectives of this work are: (1) to model the control of elective patients' admissions as a Markov decision process, (2) to develop a heuristic method capable of finding solutions for Markov decision processes with large dimensions, and (3) to evaluate the applicability of the proposed model and the performance of the developed method. The purpose of controlling patient admission is to promote a more efficient utilization of hospital resources, thereby preventing idleness or excessive use of these resources, while considering their relative importance. When a greater number of patient specialties, more resource types and greater availability, and additional admission capacity are considered, the Markov decision process for patient admission control reaches large dimensions concerning the state and action spaces. An evolutionary meta-heuristic approach in a sampled reduced planning horizon is presented here to solve large dimension Markov decision processes. The proposed method is implemented and tested for several instances of the admission control problem modeled. Using the traditional value iteration algorithm, it was possible to generate optimal admission control policies, which made it possible to maintain resources consumption close to the desired levels of utilization, considering the theoretical experimental results obtained for the control model in proposition. Using the meta-heuristics developed in this study: (1) decisions policies for the proposed model were obtained and their performances approached the performances of the optimal policies; and (2) it was possible to obtain decision policies for instances of the modeled problem wider than the ones obtained with the value iteration algorithm, because of the lack of computational memory and the excessive computing time to be dealt with. However, for modeled problem instances compatible with the instances that usually occur in a hospital setting (instances that reach millions of possible states and actions), the meta-heuristics running time for a decision policy was large (measured in days). The proposed decision model is complex due to its stochastic dynamics and dimensionality, but it has great potential for application, and, for its practical exploitation, the development of new methods is necessary.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

PMD	Processo markoviano de decisão.
AIP	Algoritmo de iteração de políticas.
AIV	Algoritmo de iteração de valores.
EPI	*Evolutionary policy iteration.*
ERPS	*Evolutionary random policy search.*
KMN	Kearns, Mansour e Ng.
PIRS	*Policy improvement with reward swapping.*
EHRA	Evolutivo em horizonte de planejamento reduzido e amostrado.
PO	Política ótima.
G1	Política gulosa 1.
G2	Política gulosa 2.
EHRA1	Política obtida pelo método EHRA, tendo como política de base a G1.
EHRA2	Política obtida pelo método EHRA, tendo como política de base a G2.
EHRA12	Política obtida pelo método EHRA, tendo como políticas de base a G1 e a G2.
Espec.	Especialidade.
Adm.	Admissão ou admissões.
Pad.	Padrão.
Max.	Máximo.
Esp.	Esperança.
Rec.	Recurso.
Exces.	Excesso.
Process.	Processamento.
Reg.	Registro.
Probab.	Probabilidade.
Reentr.	Reentrada.
Tp. proc.	Tempo de processamento.
Itera.	Iterações.
Tp. itera.	Tempo médio de processamento para uma iteração.
Repet.	Número de vezes que uma ação é selecionada em 10 simulações.

LISTA DE SÍMBOLOS

t	Instante de decisão.
X	Espaço de estados.
x	Estado pertencente a X.
x_t	Estado observado no instante de decisão t.
A	Espaço de ações.
$A(x)$	Ações de A admissíveis para o estado x.
a_t	Ação adotada no instante de decisão t.
P	Distribuições de probabilidades de transições entre estados.
$P(x,a)$	Distribuição de probabilidades de se atingir outros estados a partir do par (x,a).
$P(x_{t+1} \mid (x_t, a_t))$	Probabilidade de se atingir o estado x_{t+1} dado o par (x_t, a_t).
P^{π}	Matriz de probabilidades associada à cadeia de Markov embutida em um PMD sob controle de uma política de decisão estacionária $\pi \in \Pi_S$.
P_x	Distribuição de probabilidades para seleção de ações de $A(x)$.
R	Função de recompensa esperada.
$R(x_t, a_t)$	Recompensa esperada dado o par (x_t, a_t).
$R_{\max}$	Limite superior de R.
H	Horizonte de planejamento (número de períodos).
π ou $\hat{\pi}$	Política de decisão que determina para cada estado uma ação.
π_k	Política de decisão para a $k-ésima$ iteração.
π^*	Política de decisão ótima.
π_H^*	Política de decisão ótima no horizonte de planejamento H.
π_H^{re}	Política de decisão para o horizonte de planejamento reduzido de tamanho H.
π_H^{kmn}	Política de decisão obtida pelo método KMN para o horizonte de planejamento reduzido de tamanho H.
π_H^{ro}	Política de decisão obtida pelo método *rollout* para o horizonte de planejamento reduzido de tamanho H.
π_H^{pr}	Política de decisão obtida pelo método *parallel rollout* para o horizonte de planejamento reduzido de tamanho H.

π_H^{hd}	Política de decisão obtida pelo método *hindsight* para o horizonte de planejamento reduzido de tamanho H.
π_k^e	Política de decisão de elite da $k-ésima$ população de políticas $\Lambda_k \subset \Pi_s$.
π_H^{ehra}	Política de decisão obtida pelo método EHRA para o horizonte de planejamento reduzido de tamanho H.
π^s	Vetor linha que contém as probabilidades limites dos estados de um PMD sob controle de uma política de decisão estacionária $\pi \in \Pi_s$.
Π	Conjunto de todas as políticas de decisão possíveis.
Π_s	Conjunto das políticas de decisão estacionárias.
Λ	Conjunto de políticas de decisão, tal que $\Lambda \subset \Pi_s$.
γ	Fator de desconto, $\gamma \in (0,1]$.
V_H^*	Função de recompensa total descontada esperada ótima para um dado H.
$V_H^*(x)$	Recompensa total descontada esperada ótima dados $x_0 = x$ e H.
$\hat{V}_H^*(x)$	Estimador simulado de $V_H^*(x)$ através de uma árvore do tipo *look-ahead* com profundidade H e abertura C.
$V_H^\pi(x)$	Recompensa total descontada esperada sob controle de π dados $x_0 = x$ e H.
$\hat{V}_{H-1}^\pi(x,a)$	Valor simulado da recompensa descontada esperada considerando o controle da política de base π no horizonte de planejamento $H-1$, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada.
$Q_H^*(x,a)$	Função de utilidade máxima da ação a dado um estado x em um horizonte H, equivalente à soma entre $R(x,a)$ e a esperança descontada, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada, da "recompensa total descontada esperada ótima a partir do segundo instante de decisão".
$\hat{Q}_H^*(x,a)$	Estimador para $Q_H^*(x,a)$, equivalente à soma entre $R(x,a)$ e a estimativa simulada para a "recompensa total descontada esperada ótima a partir do segundo instante de decisão".

$\hat{Q}_H^{hd}(x,a)$ Estimador de um limitante superior para $Q_H^*(x,a)$, equivalente à soma entre $R(x,a)$ e a estimativa simulada para a "recompensa total descontada esperada ótima a partir do segundo instante de decisão".

$Q_H^{\pi}(x,a)$ Função de utilidade da ação a dado um estado x em um horizonte de planejamento H, equivalente à soma entre $R(x,a)$ e a esperança descontada, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada, da "recompensa total descontada esperada sob controle da política π a partir do segundo instante de decisão".

$\hat{Q}_H^{\pi}(x,a)$ Estimador para $Q_H^{\pi}(x,a)$, equivalente à soma entre $R(x,a)$ e a estimativa simulada para a esperança descontada da "recompensa total descontada esperada sob controle da política π a partir do segundo instante de decisão".

$Q_H^{\Lambda}(x,a)$ Função de utilidade máxima da ação a em relação ao conjunto de políticas de decisão Λ dado um estado x em um horizonte H, equivalente à soma entre $R(x,a)$ e a esperança descontada, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada, da "recompensa total descontada esperada ótima em relação ao conjunto Λ a partir do segundo instante de decisão".

$\hat{Q}_H^{\Lambda}(x,a)$ Estimador para $Q_H^{\Lambda}(x,a)$, equivalente à soma entre $R(x,a)$ e a estimativa simulada para a esperança descontada da "recompensa total descontada esperada ótima em relação ao conjunto Λ a partir do segundo instante de decisão".

$B(X)$ Espaço de funções em X que geram valores reais não negativos e limitados.

$J_{\infty}^{\pi}(x)$ Recompensa média esperada por período para uma política π em um horizonte de planejamento infinito.

J_{∞}^{*} Máxima recompensa média esperada por período considerando todas as políticas possíveis em um horizonte de planejamento infinito.

$E[\ ]$ Valor esperado da expressão entre colchetes.

E_i Padrão de atendimento i.

$E_{i,t}^{d}$ Número de pacientes da especialidade d que estiveram no padrão de atendimento E_i durante o último período, entre os instantes de decisão $t-1$ e t.

S^d	Número de pacientes da especialidade d a ser admitido no próximo período.
$\max S^d$	Capacidade máxima de admissões de pacientes do tipo d em um período.
L_j	Recurso hospitalar j.
L_{ij}	Quantidade média esperada de recursos do tipo L_j utilizada em um período por um paciente atendido sob o padrão de atendimento E_i.
$\max L_j$	Capacidade disponível de recursos do tipo L_j em um período.
p_{ij}^d	Probabilidade de um paciente da especialidade d passar do padrão de atendimento E_i para o padrão de atendimento E_j em um período.
p_i^d	Probabilidade de um paciente da especialidade d gastar o seu primeiro período no hospital sob o padrão de atendimento E_i.
N_j	Nível desejado de utilização do recurso L_j.
O_j	Custo de ociosidade de uma unidade a menos de L_j em relação à N_j
B_j	Custo de excesso de uma unidade a mais de L_j em relação à N_j.
C_j	Custo de sobreutilização de uma unidade a mais de L_j em relação à $\max L_j$.
$f(x,a,\omega)$	Função que gera o próximo estado a partir do par (x,a) e do número (ou vetor de números) aleatório ω.
$O(f(n))$	A função $f(n)$ retorna o maior tempo de processamento (número de passos) gasto por um algoritmo para resolver um problema. O argumento n está associado a algum parâmetro de dimensionalidade do problema. Conhecida como complexidade computacional no pior caso.
$d(.,.)$	Distância métrica.
$\|.\|$	Cardinalidade (número de elementos) de um conjunto.
$\|\|.\|\|_{\sup}$	Norma do supremo de um vetor.
$U[a,b]$	Distribuição uniforme de probabilidades no intervalo entre a e b, incluindo os limites do intervalo.

CAPÍTULO 1

INTRODUÇÃO

Existem muitos aspectos envolvidos quando se reflete sobre controlar a admissão de pacientes em um hospital. As admissões podem ser programadas para satisfazer objetivos diferentes e, algumas vezes, contraditórios, tais como manter elevada a utilização da capacidade hospitalar instalada (possivelmente resultando em alguns gargalos internos) ou proporcionar prontamente os recursos necessários para o tratamento dos pacientes, minimizando o tempo de espera e aumentando a quantidade de pacientes atendidos (possivelmente resultando em alguma ociosidade de recursos). Como mencionado por Kusters e Groot (1996), controlar a admissão de pacientes é uma atividade-chave que permite ao gestor do hospital balancear a demanda dos pacientes por atendimento e a disponibilidade dos recursos hospitalares. Escolher os pacientes “certos” de uma lista de espera, com o propósito de alcançar esse balanço, não é uma tarefa simples. É pouco provável que exista um modelo único capaz de gerar uma solução ótima que envolva toda a complexidade desse problema. Uma vez que a decisão final deve levar em consideração uma grande variedade de fatores, ela só pode ser tomada por um gestor humano. Porém, é possível prover ferramentas de apoio à decisão para esse gestor. Com a finalidade de prover esse tipo de suporte, neste estudo será proposto um modelo de controle de admissões e serão apresentados alguns métodos computacionais para a determinação de soluções para o modelo sugerido.

Ao contrário de Adan e Vissers (2002), não se pretende controlar as admissões diariamente. Antes disso, considera-se o planejamento do número de admissões em períodos fixos de tempo (por exemplo, uma semana ou quinze dias). O objetivo de se controlar o número de pacientes admitidos ao longo

desses períodos fixos de tempo é manter estabilizado, em níveis desejados, o consumo médio de recursos-chave do hospital (tais como consultas médicas, fisioterapia, exames, salas cirúrgicas, leitos), prevenindo-se a ociosidade ou o uso excessivo dos mesmos. Em outras palavras, será desenvolvido um modelo para auxiliar o gestor do hospital em um nível de decisão estratégico e tático, sendo as orientações geradas pelo modelo dirigidas para o nível operacional. No nível estratégico (*input*), o gestor do hospital deve estabelecer quais recursos considerar, bem como o nível desejado de utilização e a importância relativa de cada recurso. No nível tático (*output*), o modelo fornece uma política de decisão que, dependendo do número de pacientes que estiveram em atendimento no período anterior, das especialidades desses pacientes e do padrão de consumo de recursos registrado, recomenda o número de admissões de pacientes de cada especialidade a ser realizado durante o próximo período. No nível operacional, se o número recomendado de admissões for sempre observado em cada período, o hospital manterá estabilizada a utilização dos recursos hospitalares conforme os níveis desejados e a importância relativa de cada recurso.

Antes de um aprofundamento dos assuntos abordados, citam-se, a seguir, quatro estudos que em conjunto proporcionaram as ideias essenciais para o desenvolvimento do modelo que será apresentado nesta pesquisa: (1º) o estudo de Gemmel e Van Dierdonck (1999), no qual se encontra uma ampla compilação de modelos teóricos para o planejamento de admissões em hospitais, a qual inclui, de forma detalhada, os estudos de Groot (1993) e de Roth e Van Dierdonck (1995); (2º) o estudo original de Kapadia *et al.* (1985), no qual se encontra o conceito de “padrão de atendimento”, bem como a sua aplicação em um modelo markoviano para estimar a utilização de recursos hospitalares; (3º) o estudo de Adan e Vissers (2002), autores que foram os primeiros a elaborar uma função de custo com o objetivo de estabilizar a utilização dos recursos hospitalares em níveis desejados de consumo e que em seu estudo modelaram o controle de admissões como um problema de

programação linear inteira; e (4º) o abrangente estudo de Puterman (2005) sobre os processos markovianos, no qual se encontram os fundamentos utilizados para se elaborar um modelo de decisão a partir das características estocásticas da dinâmica do atendimento dos pacientes. Em relação à determinação de políticas de decisão para o modelo proposto, ainda deve ser mencionado o não menos importante trabalho de Chang *et al.* (2007), no qual se encontra uma coleção de métodos simulados para solução de processos markovianos de decisão, dos quais dois formaram a base para o método de solução desenvolvido no presente estudo.

Em suma, neste trabalho de pesquisa o controle da admissão de pacientes é modelado como um processo markoviano de decisão a tempo discreto e serão avaliados métodos computacionais para determinar políticas de decisão para o modelo proposto.

No restante deste capítulo, primeiramente a motivação para a escolha do tema de pesquisa é apresentada. A seguir, há a definição do escopo do estudo. Apresenta-se, então, uma introdução ao conceito de “padrão de atendimento”. Na sequência, expõe-se a motivação para a modelagem do problema como um processo markoviano de decisão. Em seguida, mostra-se uma breve discussão sobre os métodos para solução de processos markovianos de decisão. Finalmente, os objetivos da pesquisa são enumerados, e a estrutura subsequente do texto é descrita.

1.1 Motivação circunstancial

O ambiente de trabalho no qual os hospitais estão inseridos tem experimentado tremendas mudanças ao longo das últimas décadas. No Brasil, assim como em outros países, é sabido que a área de saúde pública encontra dificuldades para angariar, disponibilizar e, principalmente, administrar os recursos financeiros.

Os hospitais são forçados, por um ou mais destes motivos, a conviver com orçamentos limitados. Esse fato, por si só, já encorajaria o desenvolvimento de um trabalho com a intenção de utilizar os recursos dos hospitais de forma mais eficiente e controlada. A eficiência se refere à obtenção dos melhores resultados em relação ao pleno emprego dos recursos disponibilizados. O controle se refere à coordenação conjunta do emprego de cada um dos recursos considerando-se as necessidades dos pacientes.

A necessidade do uso mais eficiente e controlado dos recursos hospitalares não é resultado apenas da escassez de recursos; é devida, também, à crescente complexidade envolvida nos atendimentos aos pacientes. As tecnologias empregadas na área de saúde avançam em passo acelerado. Novos equipamentos, exames, técnicas cirúrgicas e tratamentos clínicos e de reabilitação ficam disponíveis rapidamente e continuam a surgir frequentemente. O atendimento dos pacientes requer cada vez mais a combinação e o desencadeamento da utilização de recursos dos mais diversos tipos.

A utilização eficiente e controlada dos recursos de um hospital também atende a necessidade de se melhorar a qualidade do atendimento prestado aos pacientes, reduzindo o tempo de espera e garantindo a continuidade do atendimento até o momento em que os pacientes recebam alta de seus tratamentos. Sem uma boa coordenação da utilização dos recursos, transformam-se filas externas de espera por admissão em filas internas de pacientes já admitidos aguardando por recursos para dar continuidade a seus tratamentos.

Concluindo, as duas principais motivações circunstanciais para se realizar um planejamento do fluxo de pacientes através do controle das admissões são: (1) procurar atingir a utilização plena e equilibrada dos recursos de um hospital, de forma a (2) garantir a continuidade do tratamento de cada paciente,

melhorando-se, assim, a qualidade dos serviços prestados. Entende-se que ao se controlar a quantidade de pacientes admitidos é possível equilibrar de um lado as necessidades dos que estarão em tratamento e de outro a disponibilidade dos recursos oferecidos.

1.2 Escopo da pesquisa

De acordo com Adan e Vissers (2002), existem dois tipos de admissão de pacientes: programadas e não programadas. Admissões não programadas, também chamadas de atendimentos de emergência, referem-se a pacientes que são imediatamente admitidos como consequência das condições médicas que apresentam. As admissões programadas, também denominadas atendimentos eletivos, são selecionadas de uma lista de espera para o atendimento em uma data prefixada. Nesta pesquisa estuda-se exclusivamente as admissões programadas de pacientes. Em particular, neste estudo analisam-se os procedimentos para admissões eletivas em um hospital – denominado por isso – eletivo, que não oferece serviços de emergência. Hospitais terciários, tais como os que oferecem serviços de reabilitação, são exemplos típicos de hospitais eletivos. Porém, assegura-se que o conceito do modelo desenvolvido aqui é flexível, de forma que, com ajustes, também poderia ser aplicado a hospitais que oferecem serviços de emergência.

Neste estudo focaliza-se um único hospital e sua capacidade de atendimento. Como mencionado, analisa-se especificamente um hospital com atendimento eletivo. Mas, mesmo considerando as restrições já colocadas, o fluxo de pacientes e a consequente relação de consumo de recursos hospitalares podem ser entendidos de diversas maneiras. Assim, dedica-se o restante da seção à apresentação de um esquema de atendimento, que retrata o fluxo do atendimento de pacientes em um hospital eletivo, como será considerado no escopo desta pesquisa.

A Figura 1.1 representa o fluxo dos pacientes, subdividido em quatro passagens. Parte-se das listas de espera por admissão das especialidades oferecidas pelo hospital. Presume-se que o hospital possui alguma sistematização capaz de registrar a solicitação de atendimento de um cidadão e que possa direcioná-lo a uma das especialidades disponíveis. Nesta pesquisa não se estudam as taxas de chegada de solicitações de atendimento, nem a elaboração das listas de espera. Considera-se que sempre existirão candidatos nas listas de espera aguardando para serem admitidos, passando o foco de interesse do estudo para a segunda passagem da figura.

A segunda passagem da Figura 1.1 representa o processo de controle do número de candidatos de cada especialidade que devem ser admitidos para iniciar o tratamento no hospital em cada período do horizonte de planejamento. Estabelecer esses quantitativos é o foco de interesse deste estudo. A admissão dos pacientes é a porta de entrada para os atendimentos subsequentes.

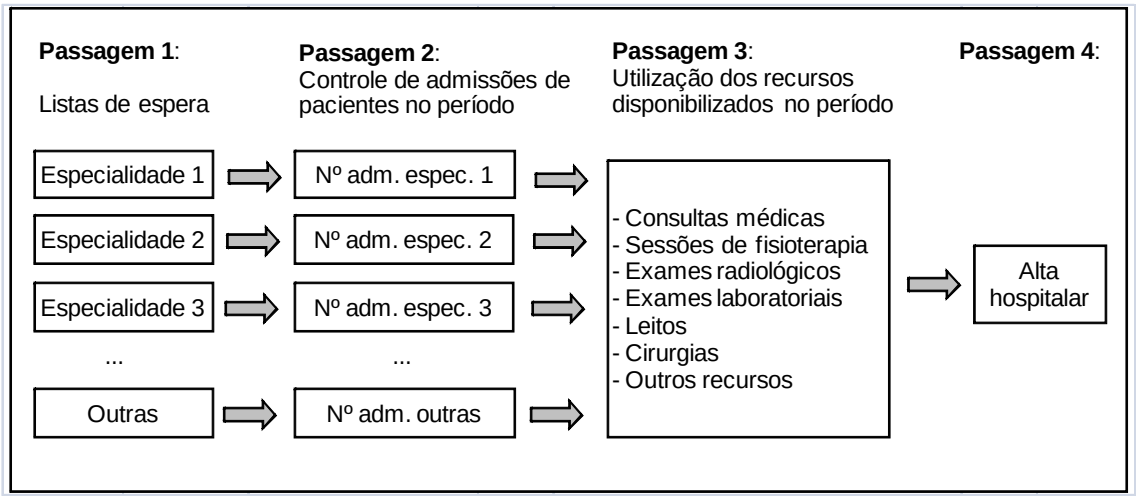


Figura 1.1 – Representação esquemática do fluxo de pacientes em um hospital que oferece apenas atendimento eletivo.

Uma vez admitidos para o atendimento, os pacientes dão prosseguimento ao tratamento fazendo uso dos recursos do hospital (Figura 1.1, Passagem 3), até

finalizarem o atendimento recebendo alta hospitalar (Figura 1.1, Passagem 4). As passagens 3 e 4 introduzem o elemento estocástico no sistema, pois não se podem determinar previamente de forma exata quais recursos e em que quantidade serão utilizados pelos pacientes em cada período de atendimento, nem qual será o período de alta de cada paciente.

Assim, o escopo desta pesquisa fica delimitado pela análise de um sistema composto por uma porta de entrada (Figura 1.1, Passagem 2), através da qual se deve determinar a quantidade de admissões de pacientes de cada especialidade em função do consumo esperado de recursos limitados (Figura 1.1, Passagem 3) e da expectativa de alta hospitalar (Figura 1.1, Passagem 4).

1.3 Conceito de padrão de atendimento

Para se controlar a admissão de pacientes, com o propósito de estabilizar em níveis preestabelecidos a utilização dos recursos hospitalares, primeiramente é necessário determinar uma forma de se estimar o consumo esperado dos recursos em um período do horizonte de planejamento. Com essa finalidade, emprega-se nesta pesquisa o conceito de “padrão de atendimento” desenvolvido por Kapadia *et al.* (1985, 2000).

Padrão de atendimento de um paciente é uma configuração indicativa do seu consumo médio esperado de recursos ao longo de um período de atendimento do horizonte de planejamento. O período de atendimento considerado pode ser de um dia, de uma semana, de um mês, devendo ser definido em função do planejamento estratégico/tático do gestor do hospital. Igualmente, os recursos a serem considerados devem ser predefinidos em função do interesse estratégico/tático do gestor.

Acompanhando os registros em prontuários do tratamento de uma amostra considerável de pacientes é possível, através da aplicação de técnicas de análise de *clusters*, determinar um número adequado de padrões de atendimento com características distintas.

Uma vez determinados os padrões de atendimento, analisando-se o consumo de recursos de um paciente em um período, é possível classificar este período de consumo em um dos padrões de atendimento definidos. Acompanhando todo o tempo de tratamento de um paciente, dividido em períodos com o mesmo tamanho definido para os padrões de atendimento, é possível identificar as passagens de um padrão de atendimento a outro.

Analisando-se as sequências de mudanças de padrões de atendimento de muitos pacientes, pode-se construir uma matriz de probabilidades de transições entre padrões de atendimento. Como observam Kapadia *et al.* (1985, 2000), as probabilidades de transições entre os padrões de atendimento podem ser distintas para as especialidades consideradas, ou seja, cada especialidade pode apresentar a sua própria matriz de transições entre padrões de atendimento. Nos capítulos 2 e 4 deste trabalho, são apresentados mais detalhes sobre os padrões de atendimento.

1.4 Modelagem markoviana

Não é muito extensa a literatura específica sobre modelos markovianos aplicados à descrição da dinâmica estocástica do atendimento de pacientes em hospitais. A seguir, apresentam-se, resumidamente, alguns estudos que, embora com foco distinto, aplicaram a modelagem markoviana nesse contexto.

Smallwood *et al.* (1969) e Kao (1973) consideraram modelos em que se formam grupos de pacientes com semelhantes taxas de chegadas ao hospital,

admitindo que esses grupos seguiam dinâmicas de movimentação governadas por processos semi-markovianos distintos e independentes. Os autores desenvolveram as formulações necessárias para se estimar, dentre outros parâmetros de desempenho, o número esperado de pacientes presentes e o tempo de permanência dos pacientes em atendimento.

Hershey *et al.* (1981) e Côté & Stein (2000) também empregaram processos semi-markovianos, mas consideraram um único grupo de chegada, sendo que Hershey e seus colaboradores concentraram seu estudo no tratamento de um sistema formado por estados transitórios como representações de unidades de serviço capacitadas, enquanto Côté & Stein introduziram a distribuição de probabilidade Erlang para determinar os tempos entre transições.

Navarro (1969), através de estimativas de máxima verossimilhança, obteve as matrizes de probabilidades de transições das cadeias de Markov a tempo discreto componentes do sistema em estudo e, através dessas cadeias, estimou parâmetros de desempenho para este sistema. Navarro considerou sistemas fechados, sendo cada sistema composto por estados recorrentes e um único grupo de pacientes com características semelhantes.

Hincapié *et al.* (2004) acompanharam coortes de pacientes em séries históricas. Através dessas coortes, eles estimaram as probabilidades de transições de uma cadeia de Markov a tempo discreto e a utilizaram para fazer estimativas sobre parâmetros de desempenho do sistema.

Weiss *et al.* (1982) adotaram uma modelagem semelhante à de Smallwood *et al.* (1969) e Kao (1973); no entanto, aqueles inovaram ao desenvolver um interessante método iterativo para testar a validade das hipóteses dos modelos markovianos.

Belderrain (1998) estudou o controle ótimo de um sistema de programação de admissões hospitalares, considerando a ocupação de leitos para dois tipos de pacientes, graves e não graves, cada tipo com sua própria taxa de chegada e de serviço. Belderrain, empregando um modelo markoviano de decisão a tempo contínuo, determinou uma política ótima de controle para a programação de admissões, levando em conta os valores das taxas de chegada e de serviço de cada tipo de paciente, que minimizou o custo médio esperado do sistema em um horizonte infinito de planejamento.

No presente estudo, propõe-se uma nova abordagem através da modelagem do controle de admissões de pacientes como um processo markoviano de decisão (PMD). Os PMDs têm provado sua utilidade como modelo para problemas de decisão sequenciais com características estocásticas e propriedades markovianas, isto é, problemas nos quais as decisões e os estados futuros independem das decisões e dos estados passados, dado que o estado presente é conhecido (PUTERMAN, 2005). Em cada instante de decisão de um PMD, observa-se o estado do sistema e adota-se uma ação. A partir desse par de informações (estado; ação) definem-se as probabilidades de se atingir qualquer um dos estados possíveis no próximo instante de decisão e determina-se o custo esperado incorrido no período até a próxima decisão.

A motivação para se modelar o sistema de admissões em um hospital eletivo como um PMD surgiu da identificação de características deste sistema que se ajustam bem aos pressupostos dos PMDs, quais sejam: (1) a existência de um processo de tomada de decisão sequencial sob incerteza, que gera uma dinâmica estocástica; (2) a possibilidade de se observar o estado do sistema em instantes de decisão igualmente espaçados no tempo (decisão a tempo discreto); (3) as características markovianas podem ser presumidas e modeladas (o futuro depende apenas do estado presente e da decisão tomada).

O processo decisório envolvido no planejamento das admissões é repetido enquanto o hospital estiver em atividade. No nível gerencial tático, a decisão sobre o número de pacientes a serem admitidos em cada especialidade pode ser realizada em intervalos de tempo igualmente espaçados (por exemplo, uma semana ou quinze dias). Presume-se que é sempre possível contar o número de pacientes presentes no último período e classificar o padrão de atendimento dispensado a cada paciente. Kapadia *et al.* (1985, 2000) mostraram que se pode estimar a probabilidade de um novo paciente ingressar em cada um dos distintos padrões de atendimento e que, no início de um período, é possível determinar as probabilidades de transições entre os padrões de atendimento para os pacientes que já estão em tratamento. O conjunto dessas informações permite que seja estimada a utilização de cada recurso durante o próximo período. Dessa forma, obtêm-se os elementos da dinâmica estocástica necessários à modelagem do controle de admissões como um PMD. O modelo obtido encontra-se detalhadamente descrito no Capítulo 2.

1.5 Métodos para solução do modelo

As metodologias empregadas para solucionar os PMDs são estudadas há mais de cinquenta anos. Existem algoritmos bem conhecidos através dos quais políticas de decisão ótimas para os PMDs podem ser determinadas. Entre estes, destacam-se o algoritmo de iteração de políticas (AIP) e o algoritmo de iteração de valores (AIV) (PUTERMAN, 2005). Os métodos tradicionais de solução dos PMDs usualmente requerem abordagens *off-line* (elaboração prévia de uma política de decisão, com ações predeterminadas para cada estado, para posterior aplicação no processo decisório ao longo do horizonte de planejamento), principalmente quando aplicados à solução de modelos com maior grau de complexidade. Uma das limitações das metodologias tradicionais é o tamanho dos espaços de estados e de ações dos sistemas em estudo

(LITTMAN *et al.*, 1995). Quando os espaços de estados e/ou de ações de um PMD apresentam grandes dimensões, como ocorre não raramente em instâncias realistas dos problemas modelados, os métodos tradicionais são computacionalmente onerosos, o que muitas vezes inviabiliza a obtenção de uma política de decisão ótima.

Diversas abordagens têm sido estudadas nas últimas décadas para contornar as dificuldades envolvidas no tratamento desses modelos vultosos, incluindo métodos de decomposição dos espaços de estados e ações e métodos que empregam técnicas de inteligência artificial (BERTSEKAS; CASTAÑON, 1989; BOUTILIER *et al.*, 1995; DEAN; GIVAN, 1997; DEAN; LIN, 1995; DEAN *et al.*, 1997, 1998; GIVAN *et al.*, 2000, 2003; HAUSKRECHT, 1998; KIM; DEAN, 2003; KOLLER; PARR, 1999, 2000; KUSHNER; CHEN, 1974; LANE; KAELBLING, 2002; LIN, 1997; MEULEAU *et al.*, 1998; PARR, 1998a, 1998b; PARR; RUSSELL, 1998; SALLANS, 2002; SAUL; SINGH, 1996; SINGH; COHN, 1998). No entanto, essas técnicas não são eficientes em muitas situações práticas, obtendo bons resultados apenas em situações bastante específicas (BERNSTEIN *et al.*, 2002; BLONDEL; TSITSIKLIS, 2000).

A partir da segunda metade da década de 90, foram desenvolvidas novas linhas de pesquisa promissoras em relação à viabilização de soluções de PMDs com grandes dimensões, menos dependentes da estrutura dos espaços de estados e de ações. Nessas novas linhas, os métodos tradicionais que empregam cálculos exaustivos sobre os espaços de estados e ações a cada iteração são substituídos por métodos amostrados e métodos de buscas especializadas que evitam esses cálculos, sendo capazes de gerar boas soluções, convergentes para a solução ótima.

Uma vertente de pesquisa desenvolvida nessa linha aborda a questão específica de PMDs compostos por espaço de estados extremamente grande e espaço de ações pequeno. Um dos métodos propostos, desenvolvido por

Kearns *et al.* (2002), parte de um estado observado e determina uma árvore de busca com profundidade limitada (horizonte reduzido) formada por estados aleatoriamente amostrados do espaço de estados original. Explorando-se a árvore com uma adequada profundidade “H” e largura “C” (tamanho de amostra para cada ação em cada nível da árvore), desenvolve-se uma metodologia capaz de determinar uma política de decisão $\varepsilon - \acute{o}tima$. O erro ε em relação à aproximação do valor ótimo de recompensa é determinado em função dos valores de H e C. Os autores demonstraram que o valor de recompensa obtido através desse método converge para o valor ótimo com probabilidade igual a 1 à medida que se ampliam a largura e a profundidade da árvore.

Outro método também aplicado aos PMDs com espaço de ações grande e espaço de estados pequeno, denominado *rollout*, foi desenvolvido por Bertsekas e Castañon (1999). No método *rollout* parte-se de um estado observado e constrói-se, para cada ação possível, uma quantidade C de caminhos contendo H estados subsequentes, que simulam o funcionamento do sistema. Os “caminhos de estados subsequentes” são amostrados do espaço de estados do modelo original. Considera-se que no instante de decisão inicial adota-se a ação que está sendo avaliada, e a partir do segundo instante de decisão as ações são escolhidas de acordo com uma política de decisão de base. A política de base deve ser determinada previamente através da utilização de uma “boa” heurística. A melhor ação a ser adotada no estado observado é determinada através da avaliação das estimativas de recompensas médias de cada possível ação inicial. Os autores demonstraram que a política de decisão resultante da aplicação sucessiva do método a cada época de decisão melhora a política de base e que, escolhendo-se adequadamente C e H, obtém-se uma boa aproximação para o valor verdadeiro da função de recompensa da nova política construída sob a política de base. Os autores demonstraram ainda que esse valor é um limitante inferior para o valor ótimo de controle do sistema. A qualidade da solução desse método é dependente de uma boa política de base.

Um método derivado do método *rollout* foi proposto por Chang (2001), que o denominou *parallel rollout*. Incrementando o método *rollout*, que emprega apenas uma política de base, o método *parallel rollout* emprega um conjunto de políticas de base. Dessa forma, espera-se que o método de Chang melhore a qualidade da solução em relação ao método *rollout*. O autor demonstrou que o desempenho alcançado em cada estado não é pior do que o melhor desempenho determinado pela aplicação de qualquer uma das políticas de decisão pertencentes ao conjunto de políticas de base.

Ainda para os PMDs com espaço de estados grande e espaço de ações pequeno, Chong *et al.* (2001) propuseram um método denominado *hindsight optimization*. Nesse método, assim como nos métodos *rollout* e *parallel rollout*, são empregados caminhos compostos por H estados subsequentes do sistema, para se avaliar as ações de controle disponíveis. Entretanto, o método *hindsight* determina para cada caminho o valor estimado da função recompensa para a ação avaliada, através da soma entre “o valor esperado até a próxima época de decisão, dado que a ação em questão é adotada no estado presente” e “o valor ótimo da função recompensa nos H-1 estados subsequentes”. Para tanto, presume-se a hipótese de que os estados que compõem o caminho simulado são conhecidos de antemão pelo controlador, e, assim, as ações adotadas por este são ótimas no horizonte de planejamento H-1. Sendo conhecidos os caminhos, a determinação de uma solução ótima para os H-1 estados subsequentes torna-se um problema determinístico. Obtendo-se a média da avaliação de muitos caminhos amostrados, determina-se o valor *hindsight* da função recompensa para cada ação avaliada. Os autores do método demonstraram que, empregando-se um número adequado de caminhos de funcionamento do sistema, o valor *hindsight* fornece um limitante superior para o melhor valor da função recompensa da ação avaliada no horizonte de planejamento H.

Existem outros estudos com abordagens semelhantes para solução de PMDs com espaços de estados grandes e espaços de ações pequenos, os quais, embora não descritos aqui, merecem citação. Esse é o caso dos trabalhos de Chang *et al*. (2005a), Kearns *et al.* (2000), Marbach e Tsitsiklis (2001), Ng e Jordan (2000) e Peret e Garcia (2004).

Considerando a solução para PMDs com espaço de estados pequeno e espaço de ações grande, Chang *et al.* (2005b) desenvolveram uma abordagem denominada *evolutionary policy iteration* (EPI). O método EPI aplica conceitos dos algoritmos genéticos para gerar populações de políticas, que representam pequenos subconjuntos do conjunto de todas as políticas possíveis. As iterações sucessivas do EPI geram novas populações de políticas, cujas políticas de elite (a melhor política construída a partir de uma população de políticas), com probabilidade igual a 1, convergem monotonicamente para a política ótima de controle do sistema.

Seguindo a mesma linha do EPI, outro método que emprega técnicas de algoritmos genéticos para escapar à necessidade de explorar exaustivamente o espaço de ações é o método denominado *evolutionary random policy search* (ERPS), desenvolvido por Hu *et al.* (2007). Assim como no método EPI, o método ERPS prevê a geração iterativa de uma sequência de populações de políticas, cujas políticas de elite convergem para a política ótima com probabilidade um. A principal diferença entre os dois métodos é a introdução, no método ERPS, de uma busca local viesada, baseada em uma heurística denominada *nearest neighbor*. Essa busca local tende a acelerar a convergência das políticas de elite para a solução ótima. (Análises preliminares sobre a eficiência da utilização de algoritmos genéticos para solucionar PMDs podem ser encontradas no estudo de Barash (1999).)

A modelagem do controle de admissões em um hospital eletivo como um PMD, evidentemente, à medida que se avaliam sistemas com maior número de

especialidades e recursos hospitalares, envolve o tratamento de modelos com grandes dimensões em relação aos espaços de estados e de ações. Para enfrentar esse problema de dimensionalidade, nessa pesquisa desenvolve-se um método meta-heurístico que combina os métodos *parallel rollout* e ERPS. O método desenvolvido, bem como uma avaliação dos resultados obtidos através da sua implementação, serão detalhadamente apresentados adiante nos capítulos 3 e 4.

1.6 Objetivos

O problema que motivou esta pesquisa, como descrito nas seções 1.1 e 1.2, foi desenvolver uma ferramenta de apoio ao planejamento do fluxo de pacientes em um hospital eletivo, com o objetivo de manter a utilização dos recursos hospitalares em patamares desejados, através do controle das admissões, esperando-se garantir, com isso, um equilíbrio entre a necessidade dos pacientes em tratamento e a disponibilidade de recursos.

O conceito de “padrão de atendimento”, introduzido na Seção 1.3, permite estimar conjuntamente o consumo de diversos recursos hospitalares em um período definido de tempo e possibilita, também, acompanhar a trajetória estocástica dos pacientes ao longo de seus tratamentos, passando de um a outro padrão de atendimento.

As características dos PMDs, como observado na Seção 1.4, se adequam bem ao problema em questão. Espera-se que a solução de um PMD modelado para o controle de admissões de pacientes seja capaz de estabelecer os quantitativos de admissões, de forma a atingir o objetivo de estabilizar a utilização dos recursos.

Conforme comentado na Seção 1.5, o problema em questão formulado com um PMD pode gerar grandes espaços de estados e ações. Sendo assim, surge a necessidade de se desenvolverem métodos alternativos aos métodos tradicionais de solução dos PMDs.

Feitas as considerações acima, enunciam-se, a seguir, os principais objetivos desta pesquisa:

1. Modelar o controle de admissões de pacientes em um hospital eletivo como um PMD.
2. Desenvolver um método combinado (evolutivo em horizonte de planejamento reduzido e amostrado) capaz de solucionar PMDs com espaços de estados e de ações grandes.
3. Testar a aplicabilidade do modelo proposto e o desempenho do método desenvolvido.

As contribuições esperadas ao se desenvolverem os objetivos 1 e 2, até onde se sabe, são inovadoras. Particularmente em relação ao objetivo 1, o primeiro estudo publicado com essa abordagem é o artigo de Nunes *et al.* (2009), originado a partir desta pesquisa.

Os objetivos propostos são desenvolvidos dentro desse contexto em cinco capítulos. Neste primeiro capítulo, uma visão geral e os objetivos do estudo foram apresentados. No Capítulo 2, apresenta-se a modelagem de decisão markoviana desenvolvida para o controle de admissões em um hospital eletivo. No Capítulo 3, descreve-se o método proposto para solucionar PMDs com espaços de estados e de ações grandes. O Capítulo 4 traz os resultados das implementações computacionais do método proposto para o modelo formulado. Finalmente, no Capítulo 5, as conclusões finais sobre esta pesquisa são descritas, e sugestões para pesquisas futuras são apresentadas.

Não seria justo deixar de enunciar nesta introdução algumas palavras sobre o material em que se baseou a presente pesquisa. Como mencionou o admirável historiador Hobsbawm (2009), todos os pesquisadores são mais versados (ou, colocando-se o fato de outra maneira, mais ignorantes) em alguns campos do que em outros. Fora de uma área relativamente estreita eles necessitam contar em grande parte com o trabalho de outros pesquisadores. A literatura existente sobre os assuntos abordados nesta pesquisa constitui, por si só, material abrangente cujo conhecimento completo e profundo está além da capacidade de qualquer individuo. Muito do que se encontra neste texto é, portanto, de segunda ou mesmo de terceira mão. Além disso, inevitavelmente, o texto pode conter erros inadvertidos, bem como simplificações de que se ressentirá o estudioso perspicaz. Na tentativa de minimizar essa última deficiência, ao longo de todo o texto procurou-se oferecer uma bibliografia complementar abrangente que pode servir como base para estudos mais detalhados.

CAPÍTULO 2

CONTROLE DE ADMISSÕES DE PACIENTES ELETIVOS MODELADO COMO UM PROCESSO MARKOVIANO DE DECISÃO

Apresenta-se, a seguir, o modelo de decisão proposto neste estudo para o controle de admissões de pacientes de especialidades distintas (por exemplo, pacientes portadores de complicações ortopédicas ou neurológicas) em um hospital eletivo (sem serviço de emergência). O propósito imediato de se controlar a admissão de pacientes é promover uma utilização estabilizada dos recursos hospitalares, evitando-se ociosidade ou consumo excessivo, considerando-se, também, a importância relativa dos recursos. No contexto que foi apresentado mais detalhadamente ao longo do Capítulo 1, o controle de admissões de pacientes será modelado como um Processo Markoviano de Decisão – PMD.

O restante do capítulo se divide como segue. Apresenta-se, na Seção 2.1, uma breve definição dos PMDs e da notação que será empregada neste estudo. Na Seção 2.2, são descritos os elementos do sistema hospitalar que serão considerados na modelagem proposta: a característica do fluxo de pacientes, os padrões de atendimento e os recursos hospitalares. Na Seção 2.3, determinam-se os elementos do PMD: espaço de estados, espaço de ações, dinâmica probabilística e função do custo esperado. Encerra-se o capítulo na Seção 2.4, na qual se apresentam as considerações finais sobre o modelo proposto.

2.1 Introdução aos processos markovianos de decisão

Nesta seção apresentam-se, concisamente, os conceitos relativos aos PMDs, os quais são necessários à compreensão da modelagem proposta. Recomenda-se, para um estudo aprofundado sobre o assunto, o abrangente livro de Puterman (2005).

Um PMD pode ser definido de forma genérica por uma *n-upla* (X, A, P, R), em que: X representa o conjunto de possíveis estados observáveis nos instantes de decisão; A representa o conjunto das possíveis ações, sendo que em um instante de decisão as ações disponíveis ficam limitadas em função do estado observado; P representa as distribuições de probabilidade de transição entre estados, sendo dependente do estado observado e da ação adotada em um instante de decisão; e R representa a função que determina o custo imediato esperado associado ao estado observado e à ação adotada. Em um instante de decisão qualquer t, observa-se o estado do sistema $x_t \in X$ e adota-se uma ação $a_t \in A(x_t)$, gerando-se, assim, o custo imediato esperado $R(x_t, a_t)$ e a probabilidade de o sistema passar a um estado qualquer $x_{t+1} \in X$, representada por $P(x_{t+1} \mid (x_t, a_t))$. Atendendo-se às características da modelagem pretendida, neste estudo considera-se um tipo particular de PMD, com espaços de estados e de ações finitos e enumeráveis, tempo discreto entre instantes de decisão, função de custo esperado limitada e horizonte de planejamento infinito.

Uma política de decisão π é uma sequência de funções $\pi = \{\pi_0, \pi_1, ...\}$, em que $\pi_t : X \to A$ de forma que $\pi_t(x_t) = a_t \in A(x_t)$. Seja $\prod$ o conjunto de todas as políticas de decisão possíveis. O problema que se deseja resolver ao se formular um sistema como um PMD pode ser definido da seguinte forma: dado um estado inicial x_0 em um instante de decisão $t = 0$, deseja-se encontrar uma

política de decisão $\pi^* \in \Pi$ que otimize a função de custo esperado associada ao problema. No Capítulo 3 serão apresentados alguns métodos que podem ser empregados com essa finalidade.

2.2 Elementos do sistema hospitalar

Antes de se modelar o controle de admissões de pacientes eletivos como um PMD, os elementos componentes do hospital a serem representados devem ser definidos de forma sistematizada. A seguir, apresentam-se as definições dos três elementos do sistema hospitalar considerados no modelo: pacientes, padrões de atendimento e recursos.

2.2.1 Pacientes

Como mencionado, o modelo em proposição aplica-se a um sistema de admissões eletivas, ou seja, sem tratamentos de emergências. Considera-se que os candidatos devem solicitar sua admissão para tratamento no hospital, sendo, então, inscritos em uma lista de espera correspondente à sua especialidade. Os candidatos aguardam até o momento de sua convocação para comparecerem ao hospital e darem início ao seu processo de tratamento, o qual se encerra no momento da alta hospitalar.

Considera-se que o hospital realiza admissões em m tipos de especialidades. Representam-se as especialidades pelo índice d; assim, $d \in \{1,...,m\}$. Presume-se que sempre existem candidatos de todas as especialidades aguardando a convocação para o tratamento.

2.2.2 Padrões de atendimento

Com o propósito de se atingir o objetivo do modelo, ou seja, estabilizar a utilização dos recursos em níveis desejados, faz-se necessário empregar uma metodologia para mensurar o consumo dos recursos. Neste estudo adota-se a metodologia proposta por Kapadia *et al.* (1985, 2000). Ao contrário de Fetter e Thompson (1969) e Kao (1974), que empregaram mudanças de locais de atendimento; de Kao (1972, 1973), que empregou mudanças do estado de saúde do paciente; e de Thomas (1968), que empregou mudanças do nível de recuperação do paciente, Kapadia e seus colaboradores descreveram o curso de tratamento de um paciente em um hospital como uma sequência de mudanças de "padrões de atendimento". Um padrão de atendimento, como já definido na Seção 1.3 do Capítulo 3, é uma configuração indicativa da utilização média esperada de cada recurso realizada por um paciente em um período de tempo fixo preestabelecido. Esse período deve ser escolhido de acordo com o propósito do planejamento, podendo ser de quinze dias, uma semana ou mesmo um dia.

Como demonstraram Kapadia *et al.* (1985, 2000), é possível determinar um número discreto de padrões de atendimento para um hospital através da análise amostrada dos cursos de tratamento de seus pacientes. Dividem-se os cursos de tratamento amostrados em períodos de tempo constantes e aplicam-se técnicas de agrupamento para se identificar os padrões de atendimento.

Para a modelagem em proposição, considera-se a existência de n padrões de atendimento, representados por E_i, $i = 1,...,n$. Assim, por exemplo, o curso do tratamento de um paciente que gastou um total de 8 períodos de tempo no hospital pode ser descrito como uma sequência de padrões de atendimento, tal como $E_2E_1E_1E_4E_2E_2E_3E_5E_n$. Nessa sequência exemplificada, o período inicial do paciente no hospital foi classificado como padrão de atendimento E_2;

então, o paciente gastou os períodos 2 e 3 no padrão de atendimento E_1, o período 4 no padrão de atendimento E_4, os períodos 5 e 6 no padrão de atendimento E_2, os períodos 7 e 8 nos padrões E_3 e E_5 e finalmente recebeu alta. Define-se o padrão E_n como representante da alta hospitalar. Esse padrão deve aparecer encerrando a representação de todos os cursos de tratamento, e nele não há consumo de recursos.

2.2.3 Recursos

Ao se exercer o controle de admissões de pacientes em um longo horizonte de planejamento, pretende-se estabilizar a utilização média dos recursos hospitalares. Portanto, nessa modelagem serão consideradas utilizações médias dos recursos em períodos fixos de tempo, em vez de se considerarem quantidades exatas de consumo.

Um padrão de atendimento determina a quantidade média esperada de utilização de cada recurso hospitalar considerado, realizada por um paciente em um período. Por exemplo, um padrão de atendimento qualquer E_i pode determinar o número médio de consultas médicas, de sessões de fisioterapia, de dias de internação e de exames radiológicos que se espera que sejam utilizados por um paciente em um período de atendimento.

No modelo em proposição consideram-se k recursos hospitalares denotados pelo índice L_j, $j \in \{1,...,k\}$. Define-se L_{ij}, $i \in \{1,...,n\}$ e $j \in \{1,...,k\}$ como a quantidade média esperada de recursos do tipo L_j utilizada em um período por um paciente atendido sob o padrão de atendimento E_i. Define-se, também, $\max L_j$ como a capacidade disponível de recursos do tipo L_j em um período.

Admite-se que se um recurso – L_j – for utilizado além da capacidade disponibilizada – $\max L_j$ –, o hospital sempre o proverá de alguma forma, mas a um custo mais elevado. Esse custo extra será imputado na função de custo do modelo (ver Subseção 2.3.5).

2.3 Elementos do PMD

Considerando-se as definições apresentadas nas seções 2.1 e 2.2, os correspondentes elementos do PMD para o modelo em proposição serão apresentados nesta seção.

2.3.1 Espaço de estados

A cada instante de decisão t, presume-se que o hospital pode ser observado em um estado x_t,

$$x_t = \{E_{1,t}^1,...,E_{1,t}^m,...,E_{n,t}^1,...,E_{n,t}^m\},$$

em que $E_{i,t}^d$ representa o número de pacientes da especialidade d, $d \in \{1,...,m\}$, que estiveram no padrão de atendimento E_i, $i \in \{1,...,n\}$, durante o último período, entre os instantes de decisão $t-1$ e t. Assim, o espaço de estados X é o conjunto ao qual pertencem todos os possíveis estados do tipo x_t.

2.3.2 Espaço de ações

Nos instantes de decisão igualmente espaçados ao longo do horizonte de planejamento, o número de pacientes de cada especialidade a ser admitido no próximo período deve ser determinado. Ou seja, a cada instante de decisão,

uma ação do tipo a deve ser selecionada, tal que

$$a = \{S^1, ..., S^m\},$$

em que S^d representa o número de pacientes da especialidade d, $d \in \{1, ..., m\}$, a ser admitido no próximo período. Presume-se que S^d é limitado, variando no intervalo entre 0 e $\max S^d$, sendo $\max S^d$ a capacidade máxima de admissões de pacientes do tipo d em um período. Assim, o espaço de ações A é definido como o conjunto finito ao qual pertencem todas as possíveis ações do tipo a.

2.3.3 Dinâmica estocástica

Presume-se que cada especialidade, durante o curso do tratamento de um paciente, possui uma dinâmica estocástica distinta e independente em relação às transições entre padrões de atendimento.

Analisando-se um número suficiente de cursos de tratamento, a contagem das transições de um padrão de atendimento a outro permite que sejam calculadas estimativas de máxima verossimilhança para as probabilidades de transições entre padrões de atendimento (KAPADIA *et al.*, 1985, 2000). A matriz de probabilidades de transições entre padrões de atendimento para um paciente da especialidade d, $d \in \{1, ..., m\}$, pode ser representada por

$$\begin{bmatrix} p_{11}^d & p_{12}^d & p_{13}^d & \cdots & p_{1n}^d \\ p_{21}^d & p_{22}^d & p_{23}^d & \cdots & p_{2n}^d \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{(n-1)1}^d & p_{(n-1)2}^d & p_{(n-1)3}^d & \cdots & p_{(n-1)n}^d \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix},$$

em que p_{ij}^d representa a probabilidade de um paciente da especialidade d passar do padrão de atendimento E_i para o padrão de atendimento E_j em um

período. Essa matriz representa uma cadeia de Markov, em que os estados correspondem aos padrões de atendimento e existe um único estado absorvente E_n correspondente à alta hospitalar.

Além da matriz de probabilidades de transições entre padrões de atendimento, é possível determinar, para cada especialidade, as probabilidades de um paciente ingressar no sistema em cada um dos padrões de atendimento. A probabilidade de um paciente da especialidade d gastar o seu primeiro período no hospital sob o padrão de atendimento E_i é representada por p_i^d. Considera-se que os pacientes não podem ingressar diretamente no padrão de alta – E_n –, ou seja, $p_n^d = 0$ para $d = 1,...,m$.

Antes de se estabelecerem as probabilidades de transições entre estados pertencentes a X, é necessário que se defina $P(\{E_{1,t+1}^d,...,E_{n,t+1}^d\}|(x_t,a))$, a probabilidade de se atingir a quantidade especificada de pacientes da especialidade d em cada padrão de atendimento no instante de decisão $t+1$, dado que no instante de decisão t o hospital foi observado no estado x_t e a ação a foi adotada. Essa probabilidade pode ser obtida através da soma das probabilidades de todas as combinações possíveis para se atingir $\{E_{1,t+1}^d,...,E_{n,t+1}^d\}$ a partir de (x_t,a); e a probabilidade de uma combinação possível pode ser obtida através da convolução de distribuições multinomiais. A formulação para o cálculo dessa probabilidade pode ser representada como segue:

$P(\{E_{1,t+1}^d,...,E_{n,t+1}^d\}|(x_t,a)) =$

$$\sum_{y_1=\max(0;E^d_{1,t+1}-\sum_{i=1}^{n-1}E^d_{i,t}+E^d_{n,t+1})}^{\min(E^d_{1,t+1};S^d)} \sum_{x_{11}=\max(0;E^d_{1,t+1}-(\sum_{i=2}^{n-1}E^d_{i,t}+y_1))}^{\min(E^d_{1,t+1}-y_1;E^d_{1,t})} \cdots \sum_{x_{(n-2)1}=\max(0;E^d_{1,t+1}-(E^d_{n-1,t}+\sum_{i=1}^{n-3}x_{i1}+y_1))}^{\min(E^d_{1,t+1}-(\sum_{i=1}^{n-3}x_{i1}+y_1);E^d_{n-2,t})}$$

$$\sum_{y_2=\max(0;E^d_{2,t+1}-\sum_{i=1}^{n-1}(E^d_{i,t}-x_{i1})+E^d_{n,t+1})}^{\min(E^d_{2,t+1};S^d-y_1)} \sum_{x_{12}=\max(0;E^d_{2,t+1}-(\sum_{i=2}^{n-1}(E^d_{i,t}-x_{i1})+y_2))}^{\min(E^d_{2,t+1}-y_2;E^d_{1,t}-x_{11})} \cdots$$

$$\sum_{x_{(n-2)2}=\max(0;E^d_{2,t+1}-(E^d_{n-1,t}-x_{(n-1)1}+\sum_{i=1}^{n-3}x_{i2}+y_2))}^{\min(E^d_{2,t+1}-(\sum_{i=1}^{n-3}x_{i2}+y_2);E^d_{n-2,t}-x_{(n-2)1})} \cdots$$

$$\sum_{y_{n-2}=\max(0;E^d_{n-2,t+1}-\sum_{i=1}^{n-1}(E^d_{i,t}-\sum_{j=1}^{n-2}x_{ij})+E^d_{n,t+1})}^{\min(E^d_{n-2,t+1};S^d-\sum_{j=1}^{n-3}y_j)} \sum_{x_{1(n-2)}=\max(0;E^d_{n-2,t+1}-(\sum_{i=2}^{n-1}(E^d_{i,t}-\sum_{j=1}^{n-3}x_{ij})+y_{n-2}))}^{\min(E^d_{n-2,t+1}-y_{n-2};E^d_{1,t}-\sum_{j=1}^{n-3}x_{1j})} \cdots$$

$$\sum_{x_{(n-2)(n-2)}=\max(0;E^d_{n-2,t+1}-(E^d_{n-1,t}-\sum_{j=1}^{n-3}x_{(n-1)j}+\sum_{i=1}^{n-3}x_{i(n-2)}+y_{n-2}))}^{\min(E^d_{n-2,t+1}-(\sum_{i=1}^{n-3}x_{i(n-2)}+y_{n-2});E^d_{n-2,t}-\sum_{j=1}^{n-3}x_{(n-2)j})}$$

$$\sum_{x_{1(n-1)}=\max(0;E^d_{n-1,t+1}-(\sum_{i=2}^{n-1}(E^d_{i,t}-\sum_{j=1}^{n-2}x_{ij})+(S^d-\sum_{i=1}^{n-2}y_i)))}^{\min(E^d_{n-1,t+1}-(S^d-\sum_{i=1}^{n-2}y_i);E^d_{1,t}-\sum_{j=1}^{n-2}x_{1j})} \cdots$$

$$\sum_{x_{(n-2)(n-1)}=\max(0;E^d_{n-1,t+1}-(E^d_{n-1,t}-\sum_{j=1}^{n-2}x_{(n-1)j}+\sum_{i=1}^{n-3}x_{i(n-1)}+(S^d-\sum_{i=1}^{n-2}y_i)))}^{\min(E^d_{n-1,t+1}-(\sum_{i=1}^{n-3}x_{i(n-1)}+(S^d-\sum_{i=1}^{n-2}y_i));E^d_{n-2,t}-\sum_{j=1}^{n-2}x_{(n-2)j})}$$

$$\sum_{x_{1n}=\max(0;E^d_{n,t+1}-(\sum_{i=2}^{n-1}(E^d_{i,t}-\sum_{j=1}^{n-1}x_{ij})))}^{\min(E^d_{n,t+1};E^d_{1,t}-\sum_{j=1}^{n-1}x_{1j})} \cdots \sum_{x_{(n-2)n}=\max(0;E^d_{n,t+1}-(E^d_{n-1,t}+\sum_{i=1}^{n-3}x_{in}))}^{\min(E^d_{n,t+1}-(\sum_{i=1}^{n-3}x_{in});E^d_{n-2,t}-\sum_{j=1}^{n-1}x_{(n-2)j})}$$

$$\left\{ \begin{array}{l} \dfrac{S^d!}{y_1!y_2!\ldots y_{n-2}!(S^d-\sum_{i=1}^{n-2}y_i)!}(p^d_1)^{y_1}(p^d_2)^{y_2}\ldots(p^d_{(n-2)})^{y_{n-2}}(p^d_{(n-1)})^{S^d-\sum_{i=1}^{n-2}y_i} \\ \dfrac{E^d_{1,t}!}{x_{11}!x_{12}!\ldots x_{1n}!}(p^d_{11})^{x_{11}}(p^d_{12})^{x_{12}}\ldots(p^d_{1n})^{x_{1n}} \\ \dfrac{E^d_{2,t}!}{x_{21}!x_{22}!\ldots x_{2n}!}(p^d_{21})^{x_{21}}(p^d_{22})^{x_{22}}\ldots(p^d_{2n})^{x_{2n}} \quad \ldots \\ \dfrac{E^d_{n-1,t}!}{(E^d_{1,t+1}-\sum_{i=1}^{n-2}x_{i1}-y_1)!\ldots(E^d_{n,t+1}-\sum_{i=1}^{n-2}x_{in})!}(p^d_{(n-1)1})^{E^d_{1,t+1}-\sum_{i=1}^{n-2}x_{i1}-y_1}\ldots(p^d_{(n-1)n})^{E^d_{n,t+1}-\sum_{i=1}^{n-2}x_{in}} \end{array} \right\}$$

em que y_i representa o número de pacientes da especialidade d admitidos durante o período entre t e $t+1$ classificados no padrão de atendimento E_i no instante de decisão $t+1$, e x_{ij} representa o número de pacientes da especialidade d que passam do padrão de atendimento E_i no instante de decisão t para o padrão de atendimento E_j no instante de decisão $t+1$. Duas características importantes da complexa formulação enunciada são: (1) uma vez que se definiu não ser possível um paciente receber alta no mesmo período de admissão, essa possibilidade foi excluída da formulação (y_n foi excluído); (2) como o padrão E_n representa a alta hospitalar e definiu-se que não ocorrem transições partindo do padrão de alta, exclui-se da formulação essa possibilidade (x_{ni} foi excluído para $i=1...n$). A formulação traz subentendidas quatro restrições:

(1) somando-se os y_i, $i \in \{1,..,n-1\}$, obtém-se a quantidade de pacientes da especialidade d que devem ser admitidos durante o período entre t e $t+1$, conforme determinado pela ação a, ou seja, $S^d = \sum_{i=1}^{n-1} y_i$;

(2) o número de transições de pacientes da especialidade d durante o período entre t e $t+1$ partindo do padrão de atendimento E_i deve ser igual à quantidade de pacientes da especialidade d presentes no padrão de atendimento E_i no instante de decisão t, ou seja, $\sum_{j=1}^{n} x_{ij} = E_{i,t}^d$, para $i = 1,..., n-1$;

(3) o número de pacientes da especialidade d no instante de decisão $t+1$ deve ser igual ao número de pacientes da especialidade d no instante de decisão t somado ao número de pacientes da especialidade d admitidos durante o período entre t e $t+1$, excluindo-se o número de pacientes presentes no padrão de alta no instante de decisão t, ou seja, $\sum_{i=1}^{n} E_{i,t+1}^d = \sum_{i=1}^{n-1} E_{i,t}^d + S^d$;

(4) Qualquer combinação que não satisfaça as três restrições anteriores tem probabilidade igual a zero.

Considerando-se que a dinâmica estocástica é independente para cada especialidade, então $P(x_{t+1} \mid (x_t, a))$, a probabilidade de que o sistema passe do estado $x_t \in X$ para o estado $x_{t+1} \in X$, dado que a ação a seja adotada no instante de decisão t, pode ser obtida através do seguinte produto:

$$P(x_{t+1} \mid (x_t, a)) = \prod_{d=1}^{m} P(\{E_{1,t+1}^{d}, ..., E_{n,t+1}^{d}\} \mid (x_t, a))$$

2.3.4 Limites dos espaços de estados e de ações

Presume-se que nem todas as ações podem ser escolhidas em todos os estados. Em estados nos quais a utilização média esperada no próximo período de pelo menos um dos k recursos estiver acima da capacidade disponibilizada, não se admitem novos pacientes no próximo período. Ou seja, nesses estados a única ação permitida é aquela em que $S^d = 0$ para todo $d \in \{1, ..., m\}$. Portanto, as ações possíveis são dependentes do estado observado. Formalmente, dado um estado $x_t \in X$ em um instante de decisão qualquer t, o conjunto de possíveis ações $A(x_t)$ é composto pelas ações do tipo $a = \{S^1, ..., S^m\}$, tal que para todo $d \in (1, ..., m)$,

$S^d \in \{0, ..., \max S^d\}$ *se* $\sum_{i=1}^{n} L_{ij} \sum_{d=1}^{m} \sum_{l=1}^{n} E_{l,t}^{d} p_{li}^{d} \leq \max L_j$ *para todo* $j \in (1, ..., k)$, *e*

$S^d = 0$ *se* $\sum_{i=1}^{n} L_{ij} \sum_{d=1}^{m} \sum_{l=1}^{n} E_{l,t}^{d} p_{li}^{d} > \max L_j$ *para pelo menos um* $j \in (1, ..., k)$.

Devido à restrição de dependência imposta ao espaço de ações, garante-se que o espaço de estados X é enumerável e finito, pois: (1) as admissões são cessadas quando pelo menos um dos recursos apresenta expectativa de ser utilizado acima da capacidade disponibilizada; (2) a capacidade disponibilizada

de todos os recursos considerados, por definição, é limitada; (3) existe um estado absorvente para todas as especialidades que representa a alta hospitalar; (4) os pacientes que recebem alta em um instante de decisão t não são considerados nos quantitativos dos próximos estados a partir do instante de decisão $t+1$; e (5) no estado inicial, impõe-se que $E_{i,0}^{d}$ seja limitado para todo $d \in \{1,...,m\}$ e $i \in \{1,...,n\}$.

2.3.5 Função de custos

O estudo de Adan e Vissers (2002) motivou o desenvolvimento de uma função de custos considerando o objetivo do modelo em proposição: determinar o número de pacientes de cada especialidade a ser admitido durante períodos de planejamento fixos, com o propósito de estabilizar a utilização dos recursos do hospital em níveis desejados preestabelecidos, procurando-se prevenir a ocorrência de ociosidade ou excesso no consumo dos recursos, levando-se em conta as importâncias relativas dos recursos.

Define-se N_j, $j \in \{1,...,k\}$, como o nível desejado de utilização do recurso L_j. Ou seja, deseja-se que a utilização de L_j em um período permaneça próxima de N_j.

Com o intuito de manter a utilização dos recursos próxima do nível desejado, estabelecem-se os custos de desvio em relação aos N_j. Quando o uso do recurso L_j for inferior em uma unidade do nível desejado de utilização, imputa-se uma unidade do custo de ociosidade O_j. Quando o uso do recurso L_j supera N_j em uma unidade, imputa-se uma unidade do custo de excesso B_j. Para se penalizar a utilização dos recursos acima da capacidade

disponibilizada, imputa-se também um custo de sobreutilização. Ou seja, caso ocorra a utilização de uma unidade do recurso L_j acima da capacidade disponibilizada – $\max L_j$ –, imputa-se uma unidade de C_j, o custo de sobreutilização do recurso. Os valores para os parâmetros N_j, O_j, B_j e C_j devem ser definidos pelos gestores do hospital levando-se em consideração objetivos estratégicos e táticos, o que certamente envolve a importância relativa de cada recurso.

Considerando que no estado observado x_t a ação a seja adotada, para se estimar o desempenho do sistema durante o período subsequente, entre t e $t+1$, define-se a função de custo esperado $R(x_t, a)$ como:

$$R(x_t, a) = \sum_{x_{t+1} \in X} P\left(x_{t+1} \mid (x_t, a)\right) \sum_{j=1}^{k} \left\{ \begin{array}{l} O_j \times \max(N_j - \sum_{i=1}^{n} L_{ij} \sum_{d=1}^{m} E_{i,t+1}^{d}; 0) + \\ B_j \times \max(\sum_{i=1}^{n} L_{ij} \sum_{d=1}^{m} E_{i,t+1}^{d} - N_j; 0) + \\ C_j \times \max(\sum_{i=1}^{n} L_{ij} \sum_{d=1}^{m} E_{i,t+1}^{d} - \max L_j; 0) \end{array} \right\}$$

em que, como definido na Subseção 2.2.3, L_{ij} representa a quantidade média de recursos do tipo L_j gastos em um período por um paciente sob o padrão de atendimento E_i.

2.3.6 Representação esquemática

A dinâmica do modelo proposto está sumarizada na Figura 2.1. Considera-se que existem candidatos de m especialidades distintas aguardando para iniciarem seus tratamentos e k recursos hospitalares cuja utilização se deseja manter estabilizada em níveis de consumo preestabelecidos. No início de cada período de planejamento, o estado do hospital pode ser observado, referente ao padrão de atendimento dos pacientes que estiveram em tratamento durante o último período; então, uma decisão deve ser tomada sobre o número de

pacientes de cada especialidade a ser admitido no próximo período. Partindo-se dessa decisão, torna-se possível determinar as probabilidades dos possíveis próximos estados e o custo esperado para o próximo período.

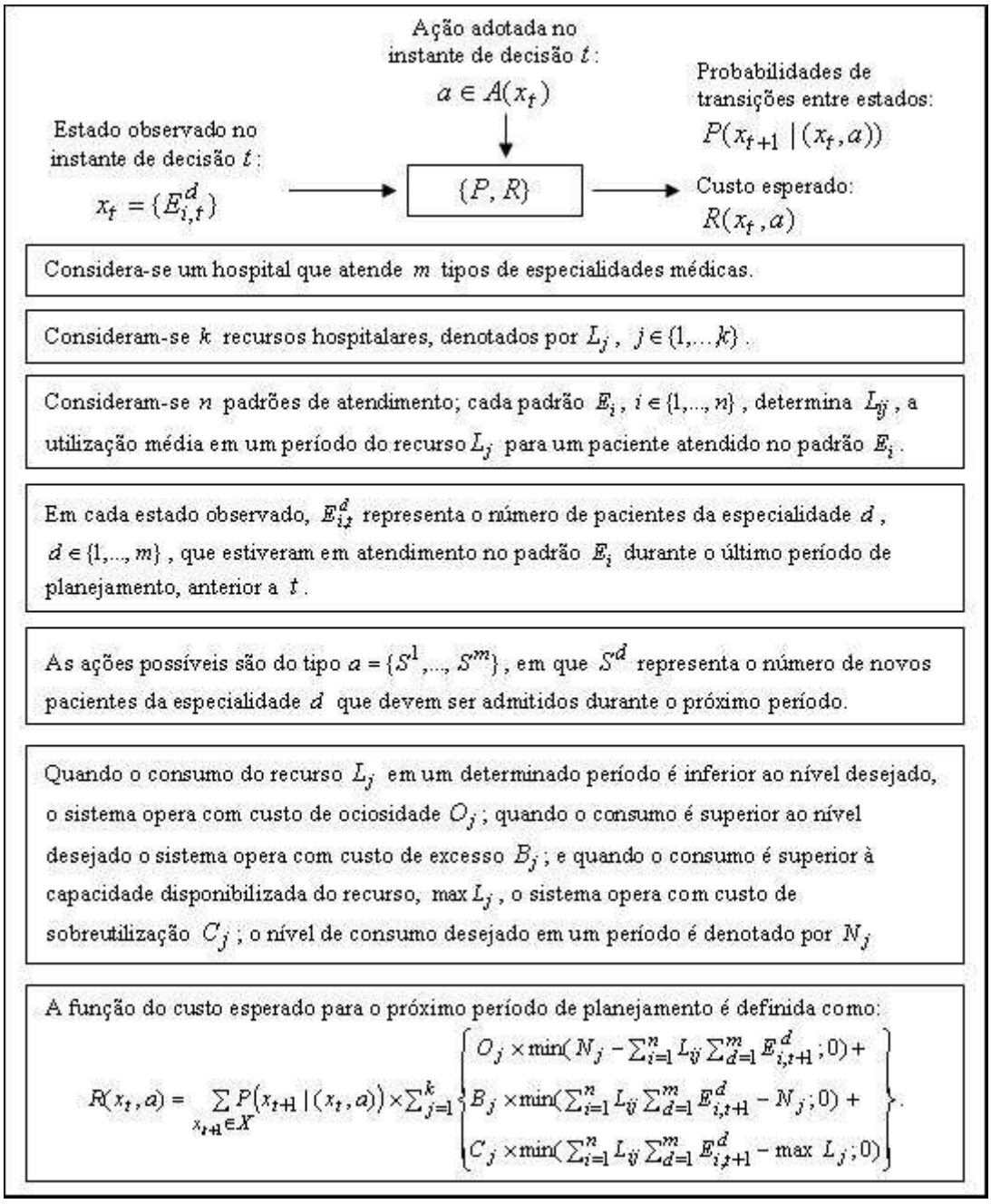


Figura 2.1 – Processo markoviano de decisão aplicado ao controle de admissões eletivas de pacientes.

2.4 Considerações sobre o modelo

Em lugar de considerar o número diário de admissões de pacientes, considerou-se a possibilidade de se controlar o número de admissões em períodos de tempo sucessivos mais extensos, em um longo horizonte de planejamento, possivelmente infinito. Nesse contexto, com o objetivo de estabilizar em níveis desejados a utilização média dos recursos do hospital, prevenindo-se a ociosidade ou o uso excessivo e considerando-se a importância relativa dos recursos, modelou-se o controle de admissões de pacientes como um PMD.

Aplicando-se a tradicional teoria para solução dos PMDs (ver Seção 3.1 do Capítulo 3 e Seção 4.2 do Capítulo 4), o modelo proposto é capaz de gerar uma política de controle ótima que mantém a utilização dos recursos próxima dos níveis desejados, penalizando os desvios em relação a esses níveis. Além disso, o modelo permite a análise de parâmetros de desempenho de políticas de controle distintas, como realizado na Seção 4.2 do Capítulo 4 com as políticas gulosas, as políticas evolutivas em horizonte de planejamento reduzido e amostrado e a política ótima.

No modelo proposto considerou-se o controle de admissões eletivas. Porém, ressalta-se que a estrutura do modelo apresentado é generalista e flexível. Espera-se que, com modificações simples, o modelo possa ser empregado também às admissões de emergência ou, como realizado na Seção 4.3 do Capítulo 4, possa ser adequado às características particulares de cada hospital considerado.

Uma limitação grave para a aplicação prática do modelo proposto está relacionada ao problema da dimensionalidade. Observa-se que (ver Subseção 4.3.5 do Capítulo 4), à medida que se avaliam sistemas mais próximos da realidade, o PMD modelado para o controle de admissões assume dimensões

extremamente grandes em relação aos espaços de estados e de ações. Por isso, há o interesse, neste estudo, de se pesquisar os métodos para soluções de PMDs com grandes dimensões. No Capítulo 3, apresentam-se alguns métodos desenvolvidos com essa finalidade, bem como propõe-se uma nova combinação de métodos para a abordagem de PMDs com espaços de estados e de ações simultaneamente grandes.

Os padrões de atendimento e as probabilidades associadas às transições entre eles são parâmetros importantes que podem ser estimados (ver subseções 4.3.3 e 4.3.4 do Capítulo 4) a partir dos registros de atendimentos dos pacientes, os quais, em geral, ficam bem guardados e acessíveis através dos sistemas de informação dos hospitais. Com a finalidade de gerar esses parâmetros devem ser empregadas técnicas estatísticas de análise de agrupamentos (KOGAN *et al.*, 2006).

CAPÍTULO 3

MÉTODOS PARA SOLUÇÃO DE PROCESSOS MARKOVIANOS DE DECISÃO COM GRANDES DIMENSÕES

Formas de modelar matematicamente problemas que envolvem tomadas de decisão sob incerteza são de interesse de diversas áreas de estudo e encontram ampla abrangência de aplicações. Dentre os modelos explorados nesse contexto destacam-se os processos markovianos de decisão. Os PMDs são empregados como modelo para problemas de decisões sequenciais com característica estocástica, que apresentam a propriedade markoviana, qual seja, os estados e as decisões futuros são independentes dos estados e das decisões passados, dado o conhecimento do estado presente do sistema considerado.

Como mencionado no Capítulo 1, as metodologias empregadas para solucionar os PMDs são estudadas há mais de cinquenta anos, existindo algoritmos bem conhecidos capazes de determinar políticas de decisão ótimas. Dentre eles destacam-se o algoritmo de iteração de políticas – AIP – e o algoritmo de iteração de valores – AIV – (PUTERMAN, 2005). A maior limitação para a aplicação prática das metodologias tradicionais é o tamanho dos espaços de estados e de ações. Quando estes são muito grandes, métodos como o AIV e o AIP se tornam computacionalmente onerosos ou impraticáveis (PAPADIMITRIOU; TSITSIKLIS, 1987). Diversas abordagens têm sido estudadas nas últimas décadas para contornar as dificuldades envolvidas no tratamento de modelos com grandes dimensões, incluindo métodos de decomposição dos espaços de estados e de ações (BERTSEKAS; CASTAÑON, 1989; DEAN *et al.*, 1997; GIVAN *et al.*, 2000, 2003; PARR, 1998a; KOLLER; PARR, 2000; SINGH; COHN, 1998; CAO *et al.*, 2002), métodos que exploram estruturas específicas do modelo (BOUTILIER *et al.*,

1995; PORTEUS, 1982; SMITH; McCARDLE, 2002) e métodos que empregam técnicas de inteligência artificial (BERTSEKAS; TSITSIKLIS, 1996; KAELBLING *et al.*, 1996; WATKINS, 1992). No entanto, mesmo essas técnicas são ineficientes ou inviáveis em muitas situações práticas (BERNSTEIN *et al.*, 2002; BLONDEL; TSITSIKLIS, 2000), principalmente quando nenhuma representação favorável da estrutura do sistema modelado é conhecida.

Novas linhas de pesquisa promissoras em relação à viabilização de soluções para PMDs de grandes dimensões, dependendo menos da estrutura do sistema modelado, foram desenvolvidas a partir da segunda metade da década de 90 (CHANG *et al.*, 2007; POWELL, 2007). Os métodos destas novas linhas visam a evitar a necessidade de se avaliar exaustivamente os espaços de estados e de ações a cada iteração do processo de solução dos PMDs, procurando obter uma solução, se não ótima, próxima à solução ótima. Essa aproximação é realizada através de métodos de buscas especializadas para espaços de ações grandes e de técnicas de simulação que, reduzindo o horizonte de planejamento, permitem o tratamento de espaços de estados grandes.

Neste capítulo propõe-se uma nova combinação de dois métodos, um método amostrado em horizonte de planejamento reduzido e um método evolutivo de iteração de políticas, para aplicação na obtenção de soluções para PMDs com espaço de estados e de ações simultaneamente grandes. Ao invés de definir uma política ótima com ações predeterminadas para cada estado em cada instante de decisão através de um processo de otimização *off-line*, o objetivo do método combinado proposto é, através de uma busca evolutiva no espaço de ações, determinar uma “boa” ação, possivelmente ótima, a cada instante de decisão, dado o estado observado do sistema, sendo a determinação desta ação realizada através de um método amostrado em um horizonte de planejamento reduzido.

O restante do capítulo está organizado desta forma: na Seção 3.1, descrevem-se os conceitos e as técnicas da teoria dos PMDs, os quais serão necessários à contextualização e à compreensão das seções subsequentes. Na Seção 3.2 e 3.3, respectivamente, apresentam-se os conceitos de política $\varepsilon - \acute{o}tima$ e de controle *on-line* em horizonte de planejamento reduzido. Uma revisão sobre os métodos amostrados em horizonte de planejamento reduzido é apresentada na Seção 3.4. Na Seção 3.5, apresenta-se uma revisão sobre os métodos evolutivos de iteração de políticas. Na Seção 3.6, apresenta-se o método combinado proposto neste estudo, evolutivo em horizonte de planejamento reduzido e amostrado. Finalmente, apresenta-se, na Seção 3.7, uma breve discussão sobre o método proposto e as considerações finais sobre este capítulo.

3.1 Processos markovianos de decisão

Como mencionado no Capítulo 2 e seguindo a revisão teórica apresentada por Chang (2001), um PMD pode ser definido de forma geral por uma *n-upla* (X, A, P, R), em que: X é o conjunto de possíveis estados; A é o conjunto de possíveis ações aplicáveis em função dos estados; P representa as distribuições de probabilidades de transições entre estados em função do estado observado e da ação adotada em um instante de decisão; e R determina a recompensa ou o custo associado ao estado observado e à ação adotada. No contexto desta pesquisa, presume-se que R é uma função de recompensa esperada, sob as distribuições de probabilidades P, dependente do estado observado e da ação adotada, cujos valores de retorno são não negativos e limitados. Presume-se, também, que o espaço de estados X é um conjunto enumerável. Aos leitores interessados em um desenvolvimento teórico que inclua PMDs com espaço de estados e função de recompensa menos restritos, recomendam-se os textos de Bertsekas e Shreve (1978) e de Hernández-Lerma e Lasserre (1996).

Em um PMD, em um instante de decisão qualquer t, observa-se o estado do sistema $x_t \in X$ e adota-se uma ação $a_t \in A(x_t)$, gerando-se, assim, uma recompensa $0 \leq R(x_t, a_t) \leq R_{\max}$ e estabelecendo-se a probabilidade de o sistema passar a um estado x_{t+1}, representada por $P(x_{t+1} \mid (x_t, a_t))$, para todo $x_{t+1} \in X$.

Pode-se definir uma política de decisão π como uma sequência de funções $\pi = \{\pi_0, \pi_1, ...\}$, em que cada π_t é uma função de X em A, ou seja, $\pi_t : X \rightarrow A$, tal que $\pi_t(x_t) = a_t \in A(x_t)$.

Se π_t é invariante em relação à t, então π é dita uma política de decisão estacionária.

Seja $\prod$ o conjunto de todas as políticas de decisão possíveis, o problema que se deseja resolver ao se formular um sistema como um PMD é, dado um estado inicial x_0 em um instante de decisão $t = 0$, encontrar uma política de decisão ótima $\pi^* \in \prod$ que maximize uma função objetiva pertinente ao problema.

Os critérios mais utilizados para otimização dos PMDs são a maximização da recompensa total descontada (com horizonte de planejamento finito ou infinito) e a maximização da recompensa média (com horizonte de planejamento infinito) (WHITE, 1993).

Seja $V_H^{\pi}(x)$ a recompensa total descontada esperada, dado um estado inicial $x_0 = x$, $x \in X$, sob as probabilidades $P(x_{t+1} \mid (x_t, a_t))$ definidas para o PMD,

em um horizonte de planejamento H e sob o controle de uma política $\pi \in \Pi$, segue que

$$V_H^{\pi}(x) = E\,[\{\sum_{t=0}^{H-1} \gamma^t R(x_t, \pi_t(x_t))\} \mid x_0 = x],$$

em que $0 < \gamma \leq 1$ é o "fator de desconto". O horizonte de planejamento H pode ser infinito, sendo requerido, nesse caso, que $\gamma < 1$.

A função de recompensa total descontada esperada ótima em um horizonte de planejamento H é denotada por $V_H^* : X \to \Re^+$, em que o valor ótimo para um dado estado $x \in X$ é obtido por

$$V_H^*(x) = \sup_{\pi \in \Pi} (V_H^{\pi}(x)),$$

e uma política ótima correspondente π^* atinge esse valor se

$$V_H^*(x) = V_H^{\pi^*}(x), \text{ para todo } x \in X.$$

No caso de um horizonte de planejamento finito ($H < \infty$), dado um estado x em um instante de decisão qualquer i, pode-se obter a máxima recompensa esperada alcançável sobre o horizonte de planejamento restante $H - i$ pela expressão

$$V_{H-i}^*(x) = \sup_{\pi \in \Pi} (V_{H-i}^{\pi}(x)), \text{ em que}$$

$$V_{H-i}^{\pi}(x) = E\,[\{\sum_{t=i}^{H-1} \gamma^{(t-i)} R(x_t, \pi_t(x_t))\} \mid x_i = x].$$

O valor de $V_{H-i}^*(x)$ pode ser obtido através da solução da equação recursiva apresentada a seguir

$$V_{H-i}^*(x) = \sup_{a \in A(x)} \{R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V_{H-(i+1)}^*(y)\},$$

em que $i \in \{0,..., H-1\}$, presumindo que $V_0^*(x) = 0$ para todo $x \in X$.

Para um horizonte de planejamento $H - i$, pode-se definir a função de utilidade máxima da ação $a \in A(x)$, dado o estado $x \in X$ em um instante de decisão i, representada por $Q^*_{H-i}(x,a)$ (CHANG *et al.*, 2007), através da expressão

$$Q^*_{H-i}(x,a) = R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V^*_{H-(i+1)}(y),$$

de forma que

$$V^*_{H-i}(x) = \sup_{a \in A(x)} Q^*_{H-i}(x,a).$$

Pode-se definir, também, a função $Q^{\pi}_{H-i}(x,a)$ de utilidade da ação a dado um estado x, sob controle de uma política $\pi \in \prod$, substituindo-se, na expressão anterior, o valor da recompensa máxima esperada alcançável no horizonte de planejamento $H-(i+1)$, $V^*_{H-(i+1)}(y)$, pelo valor de $V^{\pi}_{H-(i+1)}(y)$ para todo $y \in X$.

A política definida por

$$\pi^*_i(x) = \arg\sup_{a \in A(x)} Q^*_{H-i}(x,a),$$

para todo $x \in X$ e para todo $i \in \{0,...,H-1\}$ é uma política ótima que atinge $V^*_H(x)$ e pode ser obtida aplicando-se um algoritmo recursivo, partindo-se de $i = H-1$ até atingir $i = 0$. Pode-se dizer que, para um horizonte de planejamento finito de tamanho H, a ação $\pi^*_0(x)$ é uma "ação ótima corrente", dado o estado observado $x \in X$.

O caso em que se utiliza como critério de otimização a maximização de uma função de recompensa total descontada em um horizonte de planejamento infinito é uma extensão do caso a horizonte finito, fazendo-se $H \to \infty$ e utilizando-se um fator de desconto $0 < \gamma < 1$. Define-se a função de recompensa

total descontada esperada ótima para o horizonte infinito $V_\infty^* : X \to \Re^+$, dado o estado observado $x \in X$, como

$$V_\infty^*(x) = \sup_{\pi \in \Pi} (V_\infty^\pi(x)) \text{, em que}$$

$$V_\infty^\pi(x) = E\,[\{\sum_{t=0}^{\infty} \gamma^t R(x_t, \pi_t(x_t))\} \mid x_0 = x].$$

Sabe-se da teoria dos PMDs que, para o caso de otimização de sistemas utilizando-se a função de recompensa total descontada em um horizonte de planejamento infinito, considerando-se as condições de regularidade descritas por Puterman (2005), existe uma política ótima que é estacionária. Denota-se o conjunto das políticas estacionárias possíveis por Π_S. Além disso, pode-se demonstrar que a função V_∞^* satisfaz o "princípio de otimalidade de Bellman" (BELLMAN, 1957), de forma que, para todo $x \in X$

$$V_\infty^*(x) = \sup_{a \in A(x)} \{ R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V_\infty^*(y) \},$$

em que $V_\infty^*(x)$ é único, e existe uma política estacionária ótima $\pi^* \in \Pi_S$ que satisfaz

$$\pi^*(x) = \arg\sup_{a \in A(x)} \{ R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V_\infty^*(y) \}, \text{ e}$$

$$V_\infty^{\pi^*}(x) = V_\infty^*(x).$$

Definindo-se a máxima utilidade de uma ação $a \in A(x)$, dado um estado observado do sistema $x \in X$, em um horizonte de planejamento infinito e sob o critério de recompensa total descontada esperada, tal que

$$Q_\infty^*(x,a) = R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V_\infty^*(y),$$

segue imediatamente que

$$\sup_{a \in A(x)} Q_\infty^*(x,a) = V_\infty^*(x) \text{ para todo } x \in X,$$

e que Q_{∞}^{*} satisfaz a seguinte equação de ponto fixo

$$Q_{\infty}^{*}(x,a) = R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) \sup_{a' \in A(x)} Q_{\infty}^{*}(x,a') .$$

Para PMDs com dimensões computacionalmente tratáveis, pode-se obter uma política ótima $\pi^{*} \in \prod_{S}$ que maximize a função de recompensa total descontada em um horizonte de planejamento infinito através do AIV ou do AIP, cujos princípios estão descritos, segundo Chang *et al*. (2007), resumidamente a seguir.

Primeiramente, define-se $B(X)$ como um espaço de funções que recebem como argumento um estado $x \in X$ e geram valores reais não negativos limitados. Para $\Phi \in B(X)$, $x \in X$, define-se um operador $T : B(X) \to B(X)$ como

$$T(\Phi)(x) = \sup_{a \in A(x)} \{ R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) \Phi(y) \} .$$

Define-se, também, um operador $T_{\pi} : B(X) \to B(X)$ para $\pi \in \Pi_{S}$ como

$$T_{\pi}(\Phi)(x) = R(x,\pi(x)) + \gamma \sum_{y \in X} P(y \mid (x,\pi(x))) \Phi(y) .$$

Considerando-se o AIP, cada passo do seu processo iterativo é composto por duas partes: a avaliação e o melhoramento da política vigente. As iterações do AIP apresentam uma evolução monotônica em relação à melhora do desempenho das políticas geradas subsequentemente.

Segundo Chang *et al*. (2007), a avaliação da política vigente é baseada no seguinte resultado: para qualquer política $\pi \in \Pi_{S}$ existe uma correspondência com um único $\Phi \in B(X)$ tal que, para todo $x \in X$,

$$T_{\pi}(\Phi)(x) = \Phi(x) \text{ e } \Phi(x) = V_{\infty}^{\pi}(x) .$$

Para a avaliação de uma política vigente $\pi \in \Pi_S$, obtém-se $V_\infty^\pi(x)$ através da solução de um sistema de equações lineares, com $|X|$ equações e $|X|$ incógnitas ($|X|$ representa a cardinalidade do espaço de estados), em que, a cada $x \in X$:

$$V_\infty^\pi(x) = R(x, \pi(x)) + \gamma \sum_{y \in X} P(y \mid (x, \pi(x))) V_\infty^\pi(y).$$

Para o melhoramento de uma política vigente $\pi \in \Pi_S$, parte-se de π para uma nova política $\hat{\pi} \in \Pi_S$ de forma que seja satisfeita a condição

$$T(V_\infty^\pi)(x) = T_{\hat{\pi}}(V_\infty^\pi)(x),$$

ou seja, para todo $x \in X$ deve-se escolher uma ação, tal que

$$\hat{\pi}(x) = \arg\sup_{a \in A(x)} \{ R(x, a) + \gamma \sum_{y \in X} P(y \mid (x, a)) V_\infty^\pi(y) \}.$$

Segundo Chang *et al*. (2007), essa forma de escolha de ações garante que a recompensa descontada esperada da nova política $\hat{\pi}$ não seja menor do que a recompensa descontada esperada da política vigente π,

$$V_\infty^{\hat{\pi}}(x) \geq V_\infty^\pi(x), \text{ para todo } x \in X.$$

Partindo-se de uma política inicial arbitrária, $\pi_0 \in \Pi_S$, repetem-se alternadamente avaliações e melhoramentos das políticas vigentes até que, em uma $k - ésima$ iteração, $V_\infty^{\pi_k}(x) = V_\infty^{\pi_{k-1}}(x)$ para todo $x \in X$; nesse caso, a política $\pi_k \in \Pi_S$ é ótima. Para espaços de estados e de ações finitos, a convergência do AIP a uma solução ótima em um número finito de iterações é garantida.

Considerando-se o AIV, este algoritmo melhora o valor da recompensa descontada esperada através de sucessivas aplicações do operador T, ou seja, para uma função $v \in B(X)$, sendo $B(X)$ um espaço de funções que

recebem como argumento um estado $x \in X$ e geram valores reais não negativos limitados, um novo valor da recompensa descontada esperada, para todo $x \in X$, é obtido pela solução da equação

$$v(x) = \sup_{a \in A(x)} \{ R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) v(y) \}.$$

Conforme o desenvolvimento de Chang *et al*. (2007), define-se $\{ \psi_n \}$ como uma sequência de vetores, $n = 0,1,2,...$, em que cada vetor registra, para cada estado $x \in X$, o valor de uma função de iteração tal que $v_n(x) := T(v_{n-1})(x)$, sendo v_0 uma função arbitrária em $B(X)$. Segue que, para qualquer sequência $n = 0,1,...$, considerando-se que $\| . \|_{\sup}$ representa a norma do supremo de um vetor, a condição

$$\left\| \psi_n - V_\infty^* \right\|_{\sup} \leq \gamma^n \left\| \psi_0 - V_\infty^* \right\|_{\sup}$$

é satisfeita. Portanto, o operador T é um mapa de contração e, pelo teorema do ponto fixo de Banach (GRANAS; DUGUNDJI, 2003), através de sucessivas aplicações de T, $\| \psi_n \|_{\sup}$ converge para V_∞^*.

Particularmente, quando $v_0 = 0$, v_n fornece o valor esperado descontado da recompensa para o horizonte de planejamento finito $H - n$. Ao contrário do AIP, o AIV pode requerer um número infinito de iterações para convergir a uma solução ótima, mesmo para espaços de estados e de ações finitos.

Outro critério de otimização de PMDs é a maximização de uma função de recompensa média (PUTERMAN, 2005). O que se deseja nesse critério é encontrar uma política estacionária ótima $\pi^* \in \prod_S$ que maximize a recompensa média esperada entre instantes de decisão sobre um horizonte de planejamento infinito. A recompensa média esperada por período para uma política $\pi \in \prod_S$, $J_\infty^\pi(x)$, pode ser calculada através da expressão

$$J_{\infty}^{\pi}(x) = \lim_{H \to \infty} \frac{1}{H} E\{ \sum_{t=0}^{H-1} R(x_t, \pi_t(x_t)) \mid x_0 = x \}.$$

A existência de uma política estacionária ótima para o critério de otimização da função de recompensa média fica garantida se o PMD modelado atender à condição de que X seja contável, finito e que exista um número positivo $\alpha < 1$ tal que:

$$\sup_{w, w' \in W} \sum_{y \in X} abs[\, p(y \mid w) - p(y \mid w')] < 2\alpha \text{, em que}$$

o conjunto W é definido como

$$W := \{ (x, a) \mid x \in X,\ a \in A(x) \},$$

$p(y \mid w) = p(y \mid (x, a))$, para todo $(x, a) \in W$ e $y \in X$, e

$abs[.]$ representa o valor absoluto da operação entre colchetes.

O parâmetro α acima é conhecido como o “coeficiente de ergodicidade” (RHODIUS, 1997), sendo a condição descrita uma “condição de ergodicidade”. A condição de ergodicidade garante que, em um número finito de transições, é possível que se alcance qualquer um dos estados do espaço de estados a partir de qualquer estado observado e ação adotada em um instante de decisão. Esta condição é suficiente para garantir a existência de uma política estacionária ótima (PUTERMAN, 2005). Existem outras formas de se elaborar condições que assegurem a ergodicidade das cadeias de Markov embutidas em um PMD, garantindo, consequentemente, a existência de uma política estacionária ótima (COPPERSMITH; WU, 2008).

Segundo Puterman (2005), um PMD pode ser classificado como *unichain* quando a matriz de probabilidades de transições entre estados, correspondente a cada uma das políticas estacionárias $\pi \in \Pi_S$, consistir de uma única classe recorrente mais um conjunto possivelmente vazio de estados transitórios.

Presumindo-se que a condição de ergodicidade seja atendida e que o PMD possa ser classificado como *unichain* (TSITSIKLIS, 2007), conforme demonstrado por Puterman (2005), existe uma função mensurável limitada $h(x)$ e uma constante J_{∞}^{*} tal que:

$$J_{\infty}^{*} + h(x) = \sup_{a \in A(x)} \{ R(x,a) + \sum_{y \in X} P(y \mid (x,a)) h(y) \}, \text{ para todo } x \in X .$$

A constante J_{∞}^{*} é a máxima recompensa média por período considerando-se todas as políticas pertencentes a Π_S, ou seja, $J_{\infty}^{*} = \sup_{\pi \in \Pi_S} J_{\infty}^{\pi}(x)$, para todo $x \in X$. Além disso, fazendo-se $h(x) = 0$ para algum $x \in X$, é possível determinar os valores da função h para todos os outros estados de X.

Ainda sob as condições de ergodicidade e *unichain*, existe uma política estacionária $\pi^{*} \in \prod_S$ que atinge o valor J_{∞}^{*}, estabelecida por

$$\pi^{*}(x) = \arg\sup_{a \in A(x)} \{ R(x,a) + \sum_{y \in X} P(y \mid (x,a)) h(y) \}, \text{ para todo } x \in X .$$

As condições de ergodicidade e *unichain* enunciadas são também suficientes para garantir a convergência do AIV e do AIP aplicados ao critério de maximização da função de recompensa média.

Em relação à convergência dos tradicionais algoritmos AIP e AIV, pode-se demonstrar que o AIP converge para uma política ótima em um número de iterações menor do que o do AIV se ambos os algoritmos iniciarem com políticas de decisão que apresentem o mesmo valor esperado de retorno (PUTERMAN, 2005). Kaelbling *et al.* (1996) e Blondel e Tsitsiklis (2000) mostraram, em aplicações práticas, que o AIP frequentemente apresenta melhor desempenho que o AIV em termos de número de iterações para convergência. Em particular, para PMDs com dimensões pequenas (espaço de estados menor do que 10.000 estados), Rust (1994) mostrou que o AIP tem um

desempenho consideravelmente melhor do que o AIV se o fator de desconto γ for próximo de 1. Littman *et al.* (1995) e Blondel e Tsitsiklis (2000) apresentaram uma discussão detalhada sobre a complexidade computacional dos dois algoritmos. Segundo Chang *et al*. (2007), a complexidade computacional para o pior caso (ARORA; BARAK, 2009) de uma iteração do AIV, $O(|X|^2|A|)$, é menor do que a complexidade para o pior caso de uma iteração do AIP, $O(|X|^2|A|+|X|^3)$.

Sempre que um sistema modelado como um PMD funcionar sob o controle de uma política de decisão estacionária ao longo do horizonte de planejamento, é possível determinar-se uma cadeia de Markov dita "embutida no processo". A cadeia de Markov embutida tem o mesmo espaço de estados que o PMD original, sendo as probabilidades de transições entre estados obtidas por $P(x_{t+1}|(x_t,\pi(x_t)))$, para todo x_t e $x_{t+1} \in X$. Define-se P^{π} como a matriz de probabilidades associada à cadeia de Markov embutida em um PMD sob controle de uma política de decisão estacionária $\pi \in \Pi_S$.

A "probabilidade limite" de um estado componente de uma cadeia de Markov pode ser definida como a probabilidade de se observar o sistema nesse estado depois de um grande número de transições (longo prazo). Define-se π^S como o vetor linha que contém as probabilidades limites dos estados de um PMD sob controle de uma política de decisão estacionária $\pi \in \Pi_S$. Presumindo-se que, sob controle da política π, a cadeia de Markov embutida no processo é *unichain*, π^S pode ser obtido por multiplicações sucessivas de P^{π} até que a soma da diagonal da matriz contendo os resultados das multiplicações convirja para 1; nesse ponto, qualquer uma das linhas da matriz de resultados será igual a π^S. Opcionalmente, π^S pode ser obtido através da solução de um sistema de equações lineares definido por $\pi^S = \pi^S P^{\pi}$ e $\sum \pi^S = 1$. As

probabilidades limites dos estados podem ser empregadas para o cálculo de parâmetros de desempenho do sistema correspondente ao PMD sob controle da política $\pi \in \Pi_S$.

3.2 Política $\varepsilon - ótima$

Uma política $\pi_\varepsilon^* \in \Pi$ é dita $\varepsilon - ótima$ se $V_H^{\pi_\varepsilon^*}(x) \geq \sup_{\pi \in \Pi}(V_H^{\pi}(x)) - \varepsilon$, para todo $x \in X$, sendo $\varepsilon > 0$. Essa definição é válida tanto para o caso do critério de otimização pela função de recompensa total descontada, horizonte de planejamento finito ou infinito, quanto para o caso da função de recompensa média. Para o caso do critério de otimização pela função de recompensa total descontada, seja o horizonte de planejamento finito ou infinito, sempre existe uma política $\varepsilon - ótima$ para algum $\varepsilon > 0$. Isso também é verdade para o caso da função de recompensa média, quando X e A são finitos (CHITASHVILI, 1975; FEINBERG, 1979).

3.3 Horizonte de planejamento reduzido

Pode-se aproximar o resultado da otimização de um PMD formulado originalmente a um horizonte de planejamento infinito por um resultado simplificado a um horizonte de planejamento reduzido finito (PÉRET; GARCIA, 2004). Para tal, fixa-se um horizonte finito, $H < \infty$, e a cada instante de decisão soluciona-se o PMD considerando-se o horizonte reduzido H, obtendo-se uma "ação ótima corrente". Espera-se que, sendo o horizonte H suficientemente longo de forma a prover uma boa estimativa do comportamento estacionário do sistema, a solução ótima para horizonte infinito seja bem aproximada pela solução gerada para horizonte reduzido.

A política de decisão a horizonte de planejamento reduzido, denotada por π_H^{re}, é uma política estacionária para o PMD a horizonte infinito, obtida a partir de uma política ótima possivelmente não estacionária $\{\pi_0^*, ..., \pi_{H-1}^*\}$ para o mesmo PMD a um horizonte finito de tamanho $H < \infty$, sendo que as ações da política π_H^{re} serão aquelas previstas no primeiro instante de decisão π_0^*. Embora a política de decisão a horizonte de planejamento reduzido π_H^{re} seja determinada empregando-se um PMD a horizonte finito, o objetivo de sua obtenção é a aplicação ao PMD a horizonte infinito correspondente.

Define-se formalmente uma política de decisão a horizonte de planejamento reduzido, $H < \infty$, como $\pi_H^{re} = \{\pi_t\}$, $t = 0,1,2...$, tal que para todo t

$$\pi_t(x) = \arg\sup_{a \in A(x)} Q_H^*(x,a), \text{ para todo } x \in X.$$

Para todo $x \in X$, a política π_H^{re} satisfaz a equação de otimalidade

$$V_H^*(x) = R(x, \pi_H^{re}(x)) + \gamma \sum_{y \in X} P(y \mid (x, \pi_H^{re}(x)))V_{H-1}^*(y),$$

em que V_{H-1}^* é a recompensa total descontada esperada ótima alcançável sobre o horizonte de planejamento restante $H-1$.

Dos estudos de Hernández-Lerma e Lasserre (1990) e de Bes e Lasserre (1986), sabe-se que, sob mínimas condições de regularidade, existe um horizonte reduzido finito mínimo $H < \infty$ tal que

$$\pi_H^{re}(x) = \pi^*(x) \text{ para todo } x \in X, \text{ em que}$$

π^* é a política ótima que atinge $\sup_{\pi \in \prod_s} (V_\infty^\pi(x))$ para $0 < \gamma < 1$, e

π^* é a política ótima que atinge $\sup_{\pi \in \prod_s} (J_\infty^\pi(x))$ para $\gamma = 1$.

Hernández-Lerma e Lasserre (1988) demonstraram que a política de decisão obtida a horizonte de planejamento reduzido, quando aplicada ao controle do sistema a um horizonte infinito, converge geometricamente em função de H para o resultado ótimo a horizonte infinito, de forma que para todo $x \in X$:

$$0 \leq V_{\infty}^{*}(x) - V_{\infty}^{\pi_H^{re}}(x) \leq \frac{R_{\max}}{1-\gamma}\gamma^{H} \text{ para } 0<\gamma<1, \text{ e}$$

$$0 \leq J_{\infty}^{*} - J_{\infty}^{\pi_H^{re}}(x) \leq \frac{R_{\max}}{1-\alpha}\alpha^{H-1} \text{ para } \gamma = 1,$$

em que $R_{\max}$ é o valor máximo da função de recompensa e α é o coeficiente de ergodicidade definido na Seção 3.1.

Na prática, para PMDs com espaço de estados grande, determinar o valor esperado ótimo de recompensa para o horizonte reduzido $H < \infty$, sendo H o mínimo horizonte a partir do qual $\pi_H^{re} = \pi^{*}$, é uma tarefa computacionalmente custosa ou mesmo impraticável. Uma solução para viabilizar o tratamento de PMDs com essa característica é reduzir ainda mais o horizonte de planejamento, podendo-se incorrer, consequentemente, em um erro de aproximação da solução ótima. Chang (2001) estabeleceu limites para os erros de aproximação em função da redução do horizonte de planejamento, como se descreve a seguir.

Seja $B(X)$ um espaço de funções em X que geram valores reais não negativos limitados, dado $v \in B(X)$, tal que para algum $n \geq 0$

$$\left| V_n^{*}(x) - v(x) \right| \leq \varepsilon \text{, para todo } x \in X \text{, e}$$

considerando a política π tal que

$$\pi(x) = \underset{a \in A(x)}{\arg\sup}\{ R(x,a) + \gamma \textstyle\sum_{y \in X} p\big(y \mid (x,a)\big) v(y) \} \text{, para todo } x \in X \text{,}$$

tem-se que

$$0 \leq V_{\infty}^{*}(x) - V_{\infty}^{\pi}(x) \leq \frac{R_{\max}}{1-\gamma}\gamma^{n+1} + \frac{2\gamma\varepsilon}{1-\gamma}$$, para todo $x \in X$, se $0 < \gamma < 1$, e

$$\left| J_{\infty}^{*} - J_{\infty}^{\pi} \right| \leq \frac{R_{\max}}{1-\alpha}\alpha^{n} + 2\varepsilon$$, se $\gamma = 1$.

Apresentam-se, a seguir, métodos que exploram o enfoque do controle *on-line* sobre horizonte de planejamento reduzido para solucionar PMDs com espaço de estados grande. Entretanto, observa-se que a ideia de reduzir o horizonte de planejamento e empregar um controle *on-line* tem sido aplicada, além dos PMDs, em diversos problemas em contextos distintos, que incluem, dentre outros, problemas de planejamento e controle de estoques (GRINOLD, 1997; FEDERGRUEN; TZUR, 1998), de roteamento em redes de comunicações (BAGLIETTO *et al.*, 1999), de jogos dinâmicos (VAN DEN BROEK, 2002), de rastreamento de aeronaves (PATTEN; WHITE, 1997), de controle preditivo (MORARI; LEE, 1999), de estabilização de sistemas não lineares variantes no tempo (MAYNE; MICHALSKA, 1990) e de gerenciamento operacional (CHAND *et al.*, 2002).

3.4 Métodos amostrados em horizonte de planejamento reduzido

Os métodos amostrados em horizonte de planejamento reduzido empregam simulação para aproximar o valor da função de utilidade máxima de uma ação a em um estado x, $Q_{H}^{*}(x,a)$, para estabelecer políticas de decisão $\varepsilon - ótimas$. Nessas políticas, a cada instante de decisão, deve-se escolher a ação que apresenta o maior valor aproximado de $Q_{H}^{*}(x,a)$, sendo

$$Q_{H}^{*}(x,a) = R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V_{H-1}^{*}(y)$$, ou

$$Q_{H}^{*}(x,a) = R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) \sup_{a' \in A(y)} Q_{H-1}^{*}(y,a')$$.

Empregando um processo de decisão *on-line* (em que as ações a serem adotadas são escolhidas no instante em que o estado do sistema é observado, sem que exista uma política de decisão estabelecida *off-line* (ver seção 1.5 do Capítulo 1), com ações previamente estabelecidas para cada estado), os métodos amostrados em horizonte de planejamento reduzido evitam a necessidade de se conhecer previamente uma ação para cada estado do espaço de estados, embora seja necessário, dado um estado observado $x \in X$, avaliar a utilidade de todas as possíveis ações do espaço de ações $a \in A(x)$. Assim, esses métodos foram desenvolvidos especificamente para auxiliar o processo decisório em sistemas modelados como PMDs com espaços de estados muito grandes ou mesmo infinitos, mas com espaços de ações pequenos.

Antes de prosseguir, aos leitores interessados em explorar outros métodos que empregaram amostragem do horizonte de planejamento ou simulação para resolver as complicações inerentes ao tratamento de PMDs com grandes dimensões, entre as publicações existentes, em contextos diversos, citam-se Powell (2007), Jain e Varaiya (2006), Marbach e Tsitsiklis (2001), Cooper et al. (2003), e Fang e Cao (2004). Em seu estudo, Powell (2007) desenvolveu uma formulação para se estimar a recompensa esperada de um PMD, dado um estado observado e uma ação adotada, a partir da definição de uma variável a qual denominou *post-decision state variable*. Segundo Powell, a formulação através da *post-decision state variable* facilita a elaboração de algoritmos capazes de gerar políticas de decisão para PMDs com grandes dimensões. Empregando simulação, aproximações parametrizadas e inteligência artificial para a realização de estimativas dos valores da recompensa esperada das políticas de decisão, Powell apresentou uma variedade de métodos através dos quais soluções para problemas de grandes dimensões podem ser obtidas. No estudo de Jain e Varaiya (2006), considerando-se o contexto da solução de PMDs com grandes dimensões, encontra-se a determinação de limitantes para o número de simulações necessárias à convergência uniforme da estimativa

simulada da recompensa esperada de uma política de decisão. Marbach e Tsitsiklis (2001) desenvolveram, para um espaço de políticas parametrizado, um método baseado em simulação para resolver PMDs com grandes dimensões, que, em um processo iterativo, através da estimativa de gradientes de desempenho, atualiza os parâmetros de cada política avaliada, convergindo para uma melhor solução. Cooper e seus colaboradores (2003) apresentaram três algoritmos para utilização em um método de iteração de políticas, demonstrando sua convergência quase certa em função do número de simulações realizadas. Fang e Cao (2004) desenvolveram um método de iteração de políticas através de uma avaliação combinada entre o desempenho potencial simulado de cada política e um gradiente de direcionamento para melhores soluções.

Nas subseções seguintes, apresentam-se os métodos amostrados em horizonte de planejamento reduzido para solução de PMDs que são abordados mais detalhadamente neste estudo.

3.4.1 Método de Kearns, Mansour e Ng

O método KMN, desenvolvido por Kearns, Mansour e Ng (2002), aproxima a esperança $E[V^*_{H-1}] = \sum_{y \in X} P(y \mid (x,a)) V^*_{H-1}(y)$, dado o par (x,a), presente no cálculo de $Q^*_H(x,a)$, por uma média amostrada. Essa média é obtida da seleção de um conjunto $S^n_{x,a}$ de C estados para cada ação $a \in A(x)$, a cada nível n de uma árvore com profundidade H, dado um estado $x \in X$. O conjunto $S^n_{x,a}$ é composto por C estados amostrados do espaço de estados X de forma independente a partir da distribuição de probabilidades de transições entre estados $P(x,a)$ definida para o PMD.

O processo de otimização proposto em KMN, representado esquematicamente na Figura 3.1, pode ser entendido como a criação de uma árvore *look-ahead* com profundidade H, com fator de abertura da árvore igual ao produto $|A|C$, partindo de um estado corrente $x \in X$, em que $|A|$ representa o número de ações no espaço de ações e C representa o número de amostras para cada ação.

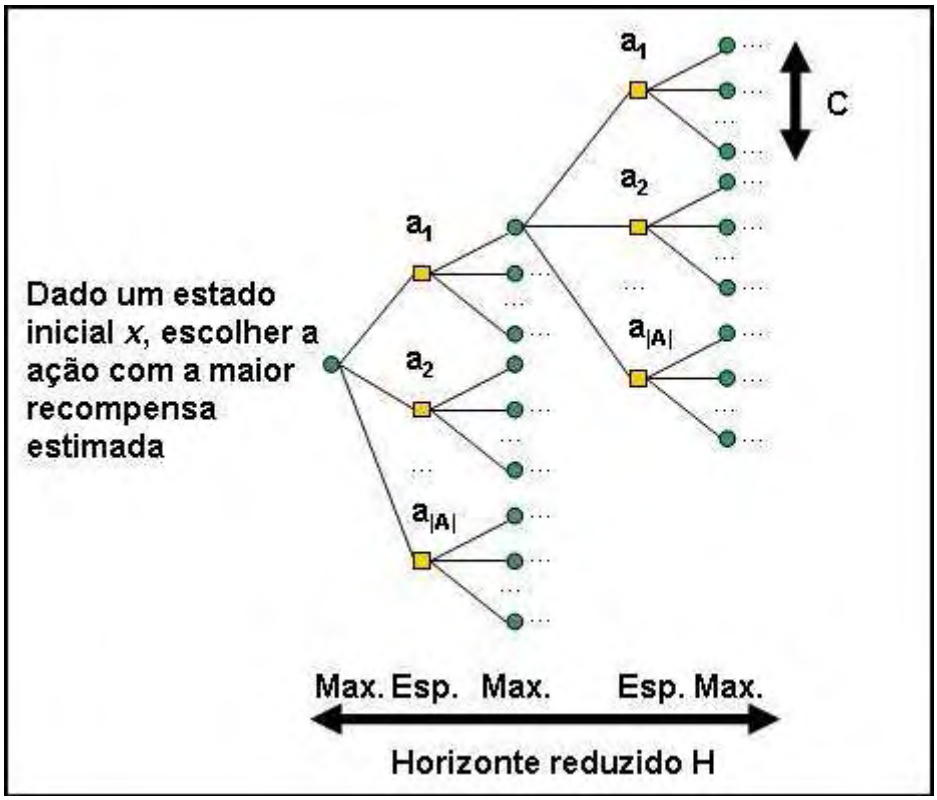


Figura 3.1 – Representação esquemática da árvore de busca do método KMN para solução de processos markovianos de decisão.

Dado um estado $x \in X$ no instante de decisão inicial, para cada $a \in A(x)$, selecionam-se C estados pertencentes a X, segundo $P(x,a)$, para compor os primeiros conjuntos de estados amostrados $S_{x,a}^{H}$. Para cada $y \in S_{x,a}^{H}$ aproxima-se $V_{H-1}^{*}(y)$, o valor ótimo de controle do sistema a partir do estado y em um

horizonte $H-1$, por $\hat{V}^{*}_{H-1}(y)$, de forma que o verdadeiro valor de $Q^{*}_{H}(x,a)$ fica estimado por

$$\hat{Q}^{*}_{H}(x,a) = R(x,a) + \gamma \frac{1}{C} \sum_{y \in S^{H}_{x,a}} \hat{V}^{*}_{H-1}(y).$$

Pode-se estimar recursivamente o valor de controle do sistema em cada nível de profundidade da árvore, dado um estado $x \in X$ e um horizonte restante de planejamento $n \in \{H-1,...,1\}$, utilizando-se a média dos valores aproximados $\hat{V}^{*}_{n-1}(y)$, para todo $y \in S^{n}_{x,a}$, como mostra a equação apresentada a seguir:

$$\hat{V}^{*}_{n}(x) = \sup_{a \in A(x)} \{R(x,a) + \gamma \frac{1}{C} \sum_{y \in S^{n}_{x,a}} \hat{V}^{*}_{n-1}(y)\},$$

fazendo-se $\hat{V}^{*}_{0}(y) = 0$ para todo $y \in X$.

A política KMN é dada por

$$\pi^{kmn}_{H}(x) = \arg\sup_{a \in A(x)} \hat{Q}^{*}_{H}(x,a), \text{ para todo } x \in X.$$

Kearns e seus colaboradores (2002) desenvolveram estimativas para a avaliação dos erros de aproximação realizados pelo método KMN, as quais são apresentadas resumidamente a seguir.

Seja $R_{\max}$ o maior valor que a função de recompensa pode atingir, para todo $x \in X$ e para todo $a \in A(x)$, pode-se limitar em probabilidade o erro devido à amostragem do próximo estado do sistema como segue

$$\Pr\left\{ \left| E_{y}\{V^{*}_{H-1}(y)\} - \frac{1}{C} \sum_{y \in S^{H}_{x,a}} \hat{V}^{*}_{H-1}(y) \right| \leq \lambda \right\} \geq 1 - e^{\left(-\lambda^{2} C \,/\, V^{2}_{\max,H-1} \right)}, \text{ em que:}$$

se $\gamma = 1$

$V_{\max,H-1} = (H-1)\, R_{\max}$;

se $0 < \gamma < 1$

$$V_{\max,H-1} = \sum_{i=1}^{H-1} R_{\max} \Big/ (1-\gamma^i) .$$

Para um horizonte de planejamento H, a aproximação da função de utilidade máxima da ação $a \in A(x)$ em um estado $x \in X$ é definida pela expressão

$$\hat{Q}_H^*(x,a) = R(x,a) + \gamma \frac{1}{C} \sum_{y \in S_{x,a}^H} \hat{V}_{H-1}^*(y) \text{, em que:}$$

$$\hat{V}_{H-1}^*(x) = \sup_{a \in A(x)} \{ \hat{Q}_{H-1}^*(x,a) \} ;$$

$\hat{Q}_0^*(x,a) = 0$, para todo $x \in X$ e para todo $a \in A(x)$.

O erro devido à utilização de aproximações de Q_H^*, através de um número C de estados amostrados, pode ser avaliado em probabilidade como segue

$$\Pr\{ \left| Q_H^*(x,a) - \hat{Q}_H^*(x,a) \right| \leq \alpha_H \} \geq 1 - (|A|C)^H e^{\left(-\lambda^2 C \Big/ V_{\max,H}^2 \right)},$$

para todo $x \in X$ e para todo $a \in A(x)$, em que:

se $0 < \gamma < 1$

$V_{\max,H} = \sum_{i=1}^{H} R_{\max} \Big/ (1-\gamma^i)$, e

$\alpha_H = \left(\sum_{i=1}^{H} \gamma^i \lambda \right) + \gamma^H \left(R_{\max} \Big/ (1-\gamma) \right) \leq \frac{1}{1-\gamma} (\lambda + \gamma^H R_{\max})$;

se $\gamma = 1$

$V_{\max,H} = H \, R_{\max}$, e

$\alpha_H = H \, \lambda$.

Considerando-se as avaliações dos dois erros envolvidos – o erro devido à amostragem do possível próximo estado e o erro devido à utilização de aproximações dos valores ótimos através de horizontes reduzidos –, é possível

derivar o número de estados amostrados C que, em probabilidade, pode garantir uma boa aproximação para o verdadeiro valor de Q_H^*. Para $\lambda > 0$ e $0 < \delta < 1$ é garantido que:

$$\Pr\{ \left| Q_H^*(x,a) - \hat{Q}_H^*(x,a) \right| \leq \lambda \} \geq 1 - \delta ,$$

para todo $x \in X$ e para todo $a \in A(x)$, desde que

$$C \geq \frac{{R_{\max}}^2}{\lambda^2 (1-\gamma)^2} \left(2H \log \frac{|A| {R_{\max}}^2 H}{\lambda^2 (1-\gamma)^2} + \log \frac{1}{\delta} \right), \text{ se } 0 < \gamma < 1, \text{ ou}$$

$$C \geq \frac{H^4 {R_{\max}}^2}{\lambda^2} \left(2H \log \frac{|A| {R_{\max}}^2 H^5}{\lambda^2} + \log \frac{1}{\delta} \right), \text{ se } \gamma = 1,$$

Kearns e seus colegas (2002) avaliaram, também, a diferença esperada de desempenho entre a política ótima estabelecida pelo critério da recompensa média a horizonte infinito e a política a horizonte reduzido e amostrado obtida pelo método KMN, π_H^{kmn}, considerando o fator de desconto $\gamma = 1$. No método KMN, a função $v \in B(X)$, como definida na Seção 3.1, fica aproximada por $\hat{V}_{H-1}^*$. Sendo $\hat{V}_{H-1}^*$ um estimador randômico, então $J_\infty^{\pi_H^{kmn}}$ é uma variável aleatória, de forma que, se

$$\Pr\{ \left| V_{H-1}^*(x) - \hat{V}_{H-1}^*(x) \right| \leq \varepsilon \} \geq 1 - \delta, \text{ para todo } x \in X ,$$

sob a condição de ergodicidade e fazendo $v = \hat{V}_{H-1}^*$,

$$E\left(\left| J_\infty^* - J_\infty^{\pi_H^{kmn}} \right| \right) \leq \frac{R_{\max}}{1-\alpha} \alpha^{H-1} + 2\varepsilon + \delta R_{\max} ,$$

em que α é o coeficiente de ergodicidade. A primeira parcela da soma do lado direito desta equação refere-se à aproximação pelo horizonte reduzido, enquanto as duas últimas parcelas referem-se à aproximação amostrada $\hat{V}_{H-1}^*$. À medida que os valores de δ e ε ficam menores e o horizonte

reduzido aumenta, melhora a aproximação do valor ótimo de controle do sistema.

O método KMN é atrativo para os PMDs com espaço de estados grande e espaço de ações pequeno, pois evita o cálculo da esperança de retorno sobre o espaço de estados completo e também reduz o horizonte de planejamento, o que diminui o esforço computacional requerido para obtenção de políticas de controle. A complexidade do tempo computacional do método KMN para obtenção de uma ação, dado um estado $x \in X$, é $O(C \mid A \mid)^H$, em que C é o número de estados amostrados para cada ação em cada nível da árvore de busca do método, $\mid A \mid$ é a cardinalidade do espaço de ações e H é o horizonte de planejamento reduzido. Observa-se, entretanto, que, para se garantir uma boa aproximação em relação a uma política ótima de controle, geralmente necessita-se de C e H grandes, o que rapidamente torna o método computacionalmente oneroso, visto que a abertura da árvore de busca é exponencial em função de H.

3.4.2 Método *rollout*

O método *rollout*, desenvolvido por Bertsekas e Castañon (1999), é um método que aplica simulação para melhorar uma política de base estacionária preestabelecida. O método *rollout* considera, a cada instante de decisão, o estado observado $x \in X$ e, em um horizonte restante de planejamento reduzido, menor do que o horizonte de planejamento modelado originalmente, avalia a utilidade de todas as ações $a \in A(x)$, simulando o controle do sistema C vezes para cada ação. A cada simulação mantém-se a ação avaliada no primeiro instante de decisão e adotam-se as ações da política de base sobre o horizonte reduzido a partir do segundo instante de decisão. A política de decisão *rollout*, em um instante de decisão, adota a ação com maior utilidade

estimada (Figura 3.2). Ou seja, no método *rollout* “desloca-se” o PMD no tempo, obtendo-se ações a cada instante de decisão através de um processo iterativo simulado, considerando-se um horizonte de planejamento reduzido H.

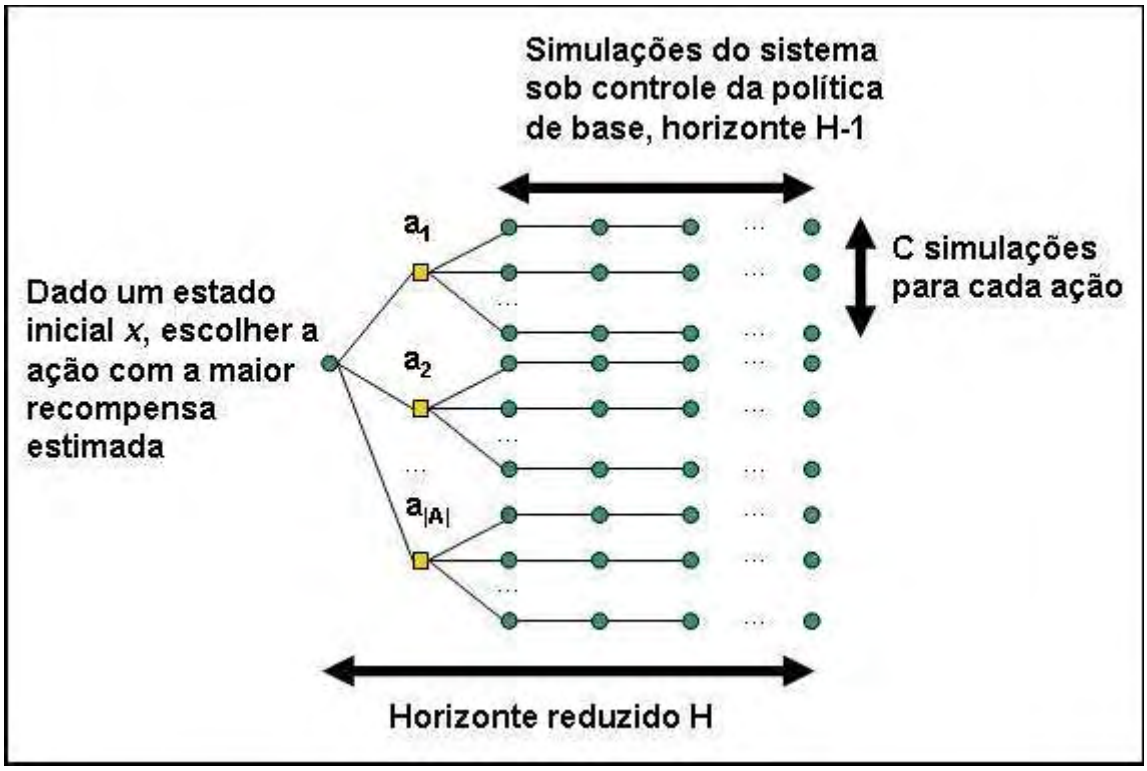


Figura 3.2 – Representação esquemática do método *rollout* para solução de processos markovianos de decisão.

A política *rollout*, π_H^{ro}, é definida pela escolha das ações com máxima utilidade estimada em relação a uma política de base, $\pi \in \prod_s$, em um horizonte de planejamento reduzido, H. A política de base é empregada para controlar o sistema após o primeiro instante de decisão, de forma que

$$\pi_H^{ro}(x) = \arg\sup_{a \in A(x)} \hat{Q}_H^{\pi}(x,a), \text{ para todo } x \in X, \text{ sendo}$$

$$\hat{Q}_H^{\pi}(x,a) = R(x,a) + \frac{1}{C}\sum_{j=1}^{C}(\hat{V}_{H-1}^{\pi}(x,a))_j,$$

em que $(\hat{V}^{\pi}_{H-1}(x,a))_j$ representa o $j-ésimo$ valor simulado de retorno descontado esperado, considerando-se o sistema sob controle da política de base no horizonte de planejamento $H-1$, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada.

Uma forma direta de se obter uma estimativa para o valor de $\hat{Q}^{\pi}_{H}(x,a)$ é utilizando-se simulação de Monte Carlo, à semelhança do estudo de Tesauro e Galperin (1997). Define-se uma função $f(x,a,\omega)$ que, dado um estado $x \in X$, uma $a \in A(x)$ e um vetor ω de números aleatórios, utiliza a distribuição de probabilidades de transições entre estados do modelo, $P(x,a)$, para simular o próximo estado do sistema. Para cada ação $a \in A(x)$, geram-se C sequências de tamanho $H-1$ de vetores de números aleatórios, $\omega_0^j,...,\omega_{H-2}^j$, e estima-se $\hat{V}^{\pi}_{H-1,j}(x,a)$, para cada $j \in \{1,..,C\}$, sendo

$$\hat{V}^{\pi}_{H-1,j}(x,a) = \sum_{i=1}^{H-1} \gamma^i R(x_i, \pi(x_i)) \text{, em que}$$

$x_1 = f(x_0, \pi(x_0), \omega_0^j)$, $x_2 = f(x_1, \pi(x_1), \omega_1^j), \ldots,$ $x_{H-1} = f(x_{H-2}, \pi(x_{H-2}), \omega_{H-2}^j)$ e $x_0 = x$ e $\pi(x_0) = a$.

Segundo o estudo de Chang (2001), pode-se estabelecer o número de simulações C necessário para se garantir um padrão probabilístico de precisão à estimativa da utilidade máxima de uma ação $a \in A(x)$ considerando-se o horizonte de planejamento reduzido H e sob controle de uma política de base $\pi \in \Pi_s$. Dado um $\varepsilon > 0$ e $0 < \delta < 1$, garante-se que

$$\Pr\{ \left| Q^{\pi}_{H}(x,a) - \hat{Q}^{\pi}_{H}(x,a) \right| \leq \varepsilon \} \geq 1 - \delta \text{ se:}$$

$C \geq \dfrac{{R_{\max}}^2 \, \gamma^2 (1-\gamma^H)^2}{\varepsilon^2 \delta (1-\gamma)^2}$, quando $0 < \gamma < 1$, ou

$C \geq \frac{H^2 {R_{\max}}^2}{\varepsilon^2 \delta}$, quando $\gamma = 1$.

Presumindo-se uma aproximação exata da utilidade máxima em relação à política de base $\pi \in \Pi_s$ para qualquer ação $a \in A(x)$, a um horizonte reduzido H, ou seja, $\hat{Q}_H^{\pi}(x,a) = Q_H^{\pi}(x,a)$, a política *rollout* pode ser expressa por

$$\pi_H^{ro}(x) = \arg\sup_{a \in A(x)} Q_H^{\pi}(x,a), \text{ para todo } x \in X, \text{ em que}$$

$$Q_H^{\pi}(x,a) = R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) V_{H-1}^{\pi}(y).$$

Considerando essa assunção, Bertsekas e Castañon (1999) demonstraram que, sob o critério de recompensa total descontada esperada a horizonte de planejamento finito H, a política *rollout* não estacionária

$$\pi_i^{ro}(x_i) = \arg\sup_{a \in A(x)} Q_{H-i}^{\pi}(x_i, a),$$

para cada $i = 0,...,H-1$, alcança desempenho igual ou melhor em relação à política de base π.

No entanto, sob o critério de recompensa descontada a horizonte infinito e sob o critério de recompensa média também a horizonte infinito, a política *rollout* não necessariamente alcança melhor desempenho que a política de base π. Apesar disso, ainda sobre a assunção que $\hat{Q}_H^{\pi}(x,a) = Q_H^{\pi}(x,a)$, é possível descrever padrões de convergência para esses dois casos. Considerando a comparação das políticas sob o critério de recompensa descontada a horizonte infinito, Chang (2001) demonstrou que, para algum $\varepsilon > 0$,

$$\text{se } H \geq 1 + \log_{\gamma} \varepsilon(1-\gamma) \big/ R_{\max},$$

$$\text{então } V_{\infty}^{\pi^{ro}}(x) \geq V_{\infty}^{\pi}(x) - \varepsilon, \text{ para todo } x \in X,$$

sendo $R_{\max}$ o maior valor que a função de recompensa esperada pode atingir. Considerando a comparação das políticas sob o critério de recompensa média a horizonte infinito, sob a condição de que exista um estado $y \in X$ e um número $\eta > 0$ tal que

$p(y \mid k) > \eta$, para todo $k \in K$, em que $K := \{ (x,a) \mid x \in X, a \in A(x) \}$,

Chang (2001) demonstrou que a política *rollout*, $\pi^{ro}(x)$, converge para uma política $\hat{\pi}$, tal que

$J_{\infty}^{\hat{\pi}} \geq J_{\infty}^{\pi}$, se $\gamma = 1$, se o coeficiente de ergodicidade $\alpha = 1 - \eta$ e se $H \to \infty$, e

$J_{\infty}^{\hat{\pi}} \geq J_{\infty}^{\pi} - \varepsilon$, para algum $\varepsilon > 0$, se $\gamma = 1 - \eta$ e se $H \geq 1 + \log_{1-\eta} \frac{\varepsilon}{R_{\max}}$.

A política *rollout* gera um retorno esperado que constitui um limitante inferior para o retorno esperado ótimo, de forma que $V_H^{\pi^{ro}}(x) \leq V_H^*(x)$ para todo $x \in X$. Contudo, independentemente da qualidade da aproximação do retorno esperado ótimo, na prática de um processo decisório, para a seleção de ações, o que realmente interessa é que se mantenha a ordem de grandeza dos valores $Q_H^*(x,a)$ para todo $x \in X$. Ou seja, se a ação ótima sempre tem o maior valor em $\hat{Q}_H^{\pi}(x,a)$, estar-se-á agindo otimamente ao aplicar-se a política *rollout*, não importando a precisão da aproximação de $V_H^{\pi^{ro}}(x)$ em relação ao retorno esperado ótimo. Porém, para se esperar um bom desempenho ao aplicar-se a política *rollout,* é necessário partir-se de uma "boa" política de base. A tarefa de se determinar uma boa política de base pode não ser simples.

Independentemente do tamanho do espaço de estados, a complexidade do tempo de computação do método *rollout* para obtenção de uma ação, dado um estado $x \in X$, é $O(|A| C H)$.

3.4.3 Método *parallel rollout*

O método heurístico *parallel rollout*, desenvolvido por Chang (2001), combina o método *rollout* com os fundamentos de um método de alternância de políticas denominado *policy switching*. O método *policy switching* consiste na escolha, sempre que um estado do sistema for observado, de uma ação dentre as ações estabelecidas pelas políticas pertencentes a um subconjunto de políticas $\Lambda \subseteq \prod_S$, de forma que a política obtida por esse processo, π^{ps}, gera um retorno esperado maior ou no mínimo igual ao retorno esperado de qualquer uma das políticas pertencentes a Λ. Ou seja, em um instante de decisão, dado um estado $x \in X$, a política prevista pelo método *policy switching* determina a ação $\pi^{ps}(x) = \pi(x)$ tal que

$$\pi \in \{\arg\sup_{\pi' \in \Lambda}(V_H^{\pi'}(x))\}.$$

Pelo fato de o método *policy switching* determinar a melhor ação corrente através da avaliação das possíveis ações de um conjunto de políticas de base, espera-se que esse método seja capaz de atingir um desempenho mais uniforme sobre o espaço de estados que o método *rollout*. O método *rollout* é dependente da qualidade da única política de base empregada sobre todos os estados $x \in X$. Por outro lado, o método *rollout* tem a vantagem de dar maior ênfase e liberdade de escolha à ação inicial, a qual é a que realmente importa no processo de controle do sistema. A proposta do método *parallel rollout* é combinar as vantagens dos métodos *rollout* e *policy switching*.

A política *parallel rollout*, π_H^{pr}, é definida pela escolha das ações com máxima utilidade estimada através de simulações de funcionamento do sistema em um horizonte de planejamento reduzido, H, dado um conjunto de políticas de base, $\Lambda \subset \prod_S$, para controlar o sistema após o primeiro instante de decisão, de forma que

$$\pi_H^{pr}(x) = \arg\sup_{a \in A(x)} \hat{Q}_H^{\Lambda}(x,a)$$, para todo $x \in X$, sendo

$$\hat{Q}_H^{\Lambda}(x,a) = R(x,a) + \gamma \frac{1}{C} \sum_{j=1}^{C} (\sup_{\pi \in \Lambda} \hat{V}_{H-1}^{\pi}(x,a))_j ,$$

em que $(\sup_{\pi \in \Lambda} \hat{V}_{H-1}^{\pi}(x,a))_j$ representa o $j-ésimo$ valor simulado de recompensa do sistema segundo o conceito *policy switching*, considerando o conjunto de políticas de base Λ no horizonte de planejamento $H-1$, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada. Uma representação esquemática da obtenção de uma ação através da política π_H^{pr} é apresentada na Figura 3.3.

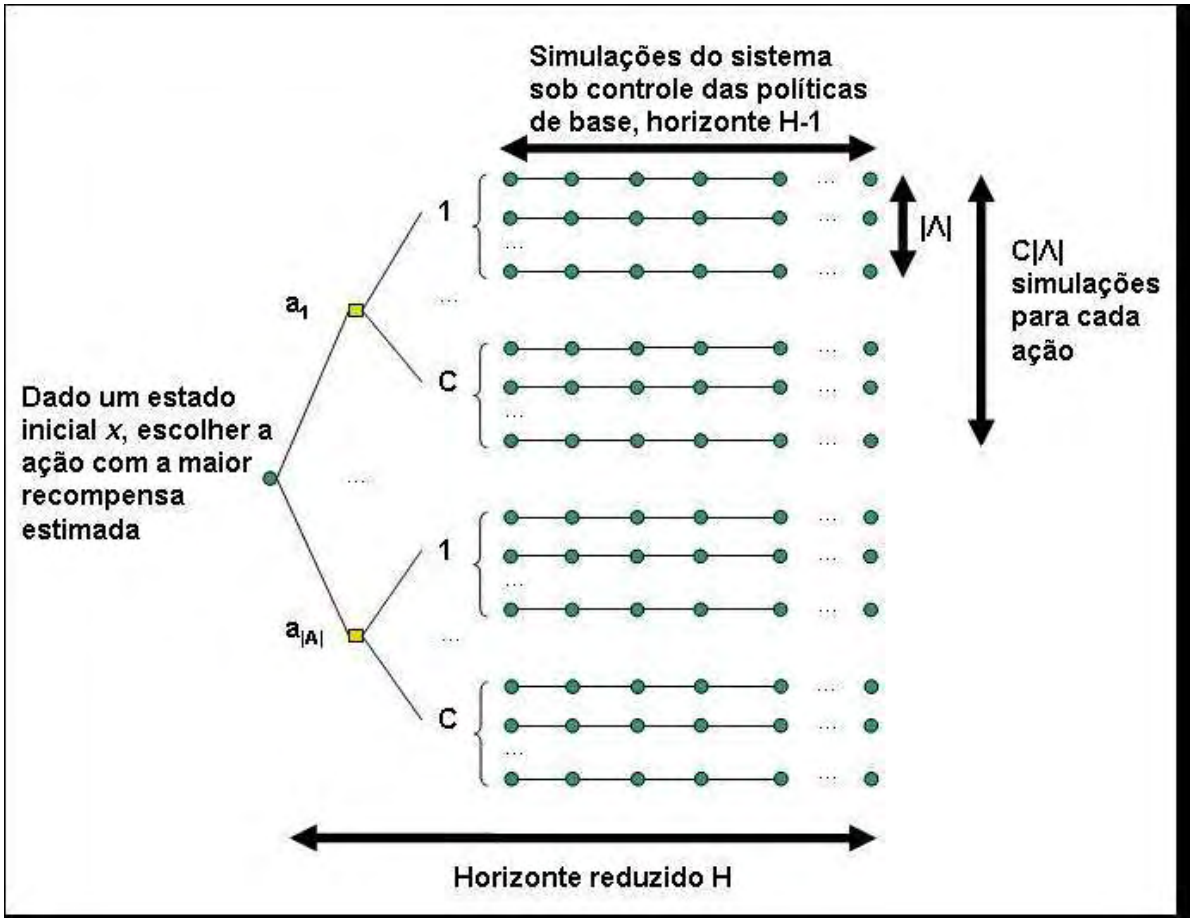


Figura 3.3 – Representação esquemática do método *parallel rollout* para solução de processos markovianos de decisão.

Define-se a máxima utilidade de uma ação $a \in A(x)$, dado um estado $x \in X$, em relação ao conjunto de políticas de base $\Lambda \subset \prod_s$, como

$$Q_H^{\Lambda}(x,a) = R(x,a) + \sum_{y \in X} P(y \mid (x,a) \sup_{\pi \in \Lambda} (V_{H-1}^{\pi}(y)).$$

Chang (2001) demonstrou que a estimativa simulada de $\hat{Q}_H^{\Lambda}(x,a)$ converge para $Q_H^{\Lambda}(x,a)$ à medida que o número de simulações C aumenta, de forma que é possível estimar em probabilidade a precisão dessa aproximação como segue. Dado um $\varepsilon > 0$ e $0 < \delta < 1$, garante-se que

$\Pr\{ \left| Q_H^{\Lambda}(x,a) - \hat{Q}_H^{\Lambda}(x,a) \right| \leq \varepsilon \} \geq 1 - \delta$, para todo $x \in X$ e $a \in A(x)$, se:

$C \geq \dfrac{{R_{\max}}^2 \gamma^2 (1-\gamma^H)^2}{\varepsilon^2 \delta (1-\gamma)^2}$, quando $0 < \gamma < 1$, ou

$C \geq \dfrac{H^2 {R_{\max}}^2}{\varepsilon^2 \delta}$, quando $\gamma = 1$.

Presumindo uma estimativa exata de $Q_H^{\Lambda}(x,a)$ por $\hat{Q}_H^{\Lambda}(x,a)$, Chang (2001) demonstrou que a política resultante da aplicação do método *parallel rollout* atinge um desempenho igual ou melhor do que o de qualquer política $\pi \in \Lambda$. Ou seja, dado um conjunto de políticas de base não vazio $\Lambda \subseteq \prod_s$, para a política π^{pr} definida em Λ tem-se que:

$$V_H^{\pi^{pr}}(x) \geq \sup_{\pi \in \Lambda} V_H^{\pi}(x), \text{ para todo } x \in X,$$

quando se considera a política *parallel rollout* estabelecida de forma não estacionária, comparando-se ao retorno do sistema sob controle *policy switching* das políticas $\pi \in \Lambda$ e sob o critério de recompensa total não descontada em um horizonte de planejamento H;

$$V_{\infty}^{\pi^{pr}}(x) \geq \sup_{\pi \in \Lambda} V_{\infty}^{\pi}(x), \text{ para todo } x \in X,$$

quando se considera a política *parallel rollout* estabelecida a um horizonte de planejamento $H \to \infty$, comparando-se ao retorno do sistema sob controle *policy switching* das políticas $\pi \in \Lambda$ e sob o critério da recompensa total descontada em um horizonte de planejamento infinito;

$$V_{\infty}^{\pi^{pr}}(x) \geq \sup_{\pi \in \Lambda} V_{\infty}^{\pi}(x) - \varepsilon \text{, para todo } x \in X \text{,}$$

quando se considera a política *parallel rollout* estabelecida a um horizonte de planejamento reduzido $H < \infty$, mas aplicada ao controle do sistema em um horizonte infinito, comparando-se ao retorno do sistema sob controle *policy switching* das políticas $\pi \in \Lambda$ e sob o critério de recompensa total descontada em um horizonte de planejamento infinito, dado um $\varepsilon > 0$, com $H \geq 1 + \log_{\gamma} \frac{\varepsilon(1-\gamma)}{R_{\max}}$ e $0 < \gamma < 1$, sendo $R_{\max}$ o maior valor que a função de recompensa pode atingir.

Chang *et al.* (2004) estenderam a formulação do método *parallel rollout* para os PMDs parcialmente observáveis. Chang e Marcus (2003) desenvolveram sua análise sobre a aplicação do *parallel rollout* com foco no critério de recompensa média.

No método *parallel rollout*, a complexidade do tempo computacional para obtenção de uma ação, dado um estado $x \in X$, é $O(|A||\Lambda|CH)$, independentemente do espaço de estados. Observa-se que o número de simulações para a obtenção de aproximações do valor de $Q_H^{\pi}(x,a)$ no método *rollout* é inferior ao número de simulações necessárias para a aproximação de $Q_H^{\Lambda}(x,a)$ no método *parallel rollout*. No entanto, é esperado que a política *parallel rollout* apresente desempenho mais uniforme sobre o espaço de estados, visto que emprega um conjunto de políticas de base. Por outro lado, a necessidade de se obter mais políticas de base, através de heurísticas

preestabelecidas, pode aumentar a dificuldade de preparação do método *parallel rollout* em relação ao método *rollout*.

3.4.4 Método *hindsight*

O método *hindsight*, desenvolvido por Chong *et al.* (2001), fornece um limite superior para o valor de máxima utilidade $Q_H^*(x,a)$ através da aplicação da desigualdade de Jensen (HARDY *et al.*, 1988.) à recursividade da equação que o define, invertendo a ordem entre o cálculo das esperanças e das maximizações. O limite superior estimado pelo método *hindsight*, $\hat{Q}_H^{hd}(x,a)$, para o valor da utilidade máxima de uma ação $a \in A(x)$, dado um estado $x \in X$ e um horizonte de planejamento reduzido H, pode ser obtido por

$$R(x,a) + \frac{1}{C}\sum_{j=1}^{C} \sup_{a_1 \ldots a_{H-1}} (\gamma^1 R(f(x_0, a_0, \omega_1^j), a_1) + \ldots + \gamma^{H-1} R(f(x_{H-2}, a_{H-2}, \omega_{H-1}^j), a_{H-1}))$$

sendo $x_0 = x$, $a_0 = a$, em que a função $f(x_t, a_t, \omega_t^j)$ é capaz de simular o próximo estado do sistema, x_{t+1}, dado um estado $x_t \in X$, uma ação $a_t \in A(x_t)$ e um vetor de números aleatórios ω_t^j, considerando a distribuição de probabilidades de transições entre os estados, $P(x,a)$. Observa-se que o valor superior, $\sup_{a_1 \ldots a_{H-1}}(.)$, presente no interior do somatório, deve ser obtido sobre todas as sequências possíveis $a_1 \ldots a_{H-1}$ de ações pertencentes a A.

A melhor sequência de ações, após adotar-se a ação a, pode ser escolhida deterministicamente se o controlador do sistema, de alguma forma, conhecer de antemão a sequência de números aleatórios gerada para simular um caminho de funcionamento do sistema, ou seja, se o controlador tiver um *hindsight*. *Hindsight* pode ser entendido, a partir da língua inglesa, como o conhecimento ou a compreensão antecipada do desenrolar de um

acontecimento. Portanto, uma vez conhecidas as sequências de vetores de números aleatórios $\omega_1^j,...,\omega_{H-1}^j$, $j=1,...,C$, as C maximizações internas ao somatório do cálculo de $\hat{Q}_H^{hd}(x,a)$ podem ser transformadas em problemas determinísticos. O problema determinístico é escolher a "melhor" sequência de ações sobre os $H-1$ passos restantes, considerando conhecida a sequência de números aleatórios selecionada, com o objetivo de maximizar a recompensa total descontada.

A política *hindsight*, π_H^{hd}, é definida pela escolha das ações com máxima utilidade estimada, de forma que

$$\pi_H^{hd}(x) = \arg\sup_{a \in A(x)} \hat{Q}_H^{hd}(x,a), \text{ para todo } x \in X.$$

Em resumo, como representado esquematicamente na Figura 3.4, dado que o sistema encontra-se em um estado $x \in X$, para cada ação $a \in A(x)$, selecionam-se C sequências de $H-1$ vetores de números aleatórios, solucionam-se os C problemas de otimização determinísticos definidos por cada sequência de vetores de números aleatórios, obtendo-se em seguida $\hat{Q}_H^{hd}(x,a)$, utilizando-se a média dos resultados. A política *hindsight* determina que se escolha sempre a ação com o maior valor $\hat{Q}_H^{hd}(x,a)$.

Presumindo $Q_H^{hd}(x,a)$ como o valor *hindsight* exato de utilidade de uma ação a dado um estado x, Chong *et al.* (2001) demonstraram, através da aplicação imediata da desigualdade de Jensen, que $Q_H^{hd}(x,a)$ é um limitante superior para o valor de utilidade máxima da ação a, de forma que

$$Q_H^{hd}(x,a) \geq Q_H^*(x,a), \text{ para todo } x \in X \text{ e para todo } a \in A(x).$$

Chong *et al.* (2001) demonstraram, também, que a qualidade da aproximação de $Q_H^{hd}(x,a)$ por $\hat{Q}_H^{hd}(x,a)$ pode ser avaliada probabilisticamente em função do número de simulações C. Dado um $\varepsilon > 0$ e $0 < \delta < 1$, garante-se que

$$\Pr\left\{ \left| Q_H^{hd}(x) - \hat{Q}_H^{hd}(x) \right| \leq \varepsilon \right\} \geq 1 - \delta \text{ se:}$$

$C \geq \dfrac{{R_{\max}}^2 \gamma^2 (1-\gamma^H)^2}{\varepsilon^2 \delta (1-\gamma)^2}$, quando $\gamma \in (0,1]$, ou

$C = \dfrac{H^2 {R_{\max}}^2}{\varepsilon^2 \delta}$, quando $\gamma = 1$,

sendo $R_{\max}$ o maior valor que a função de recompensa pode atingir.

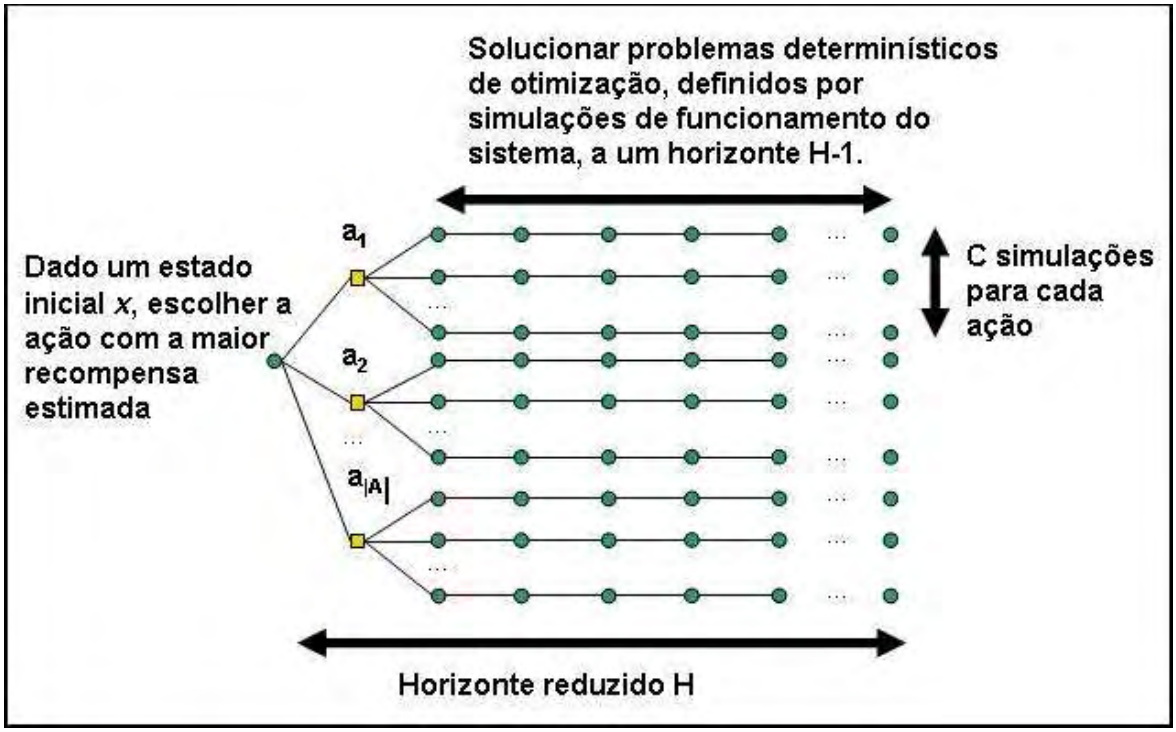


Figura 3.4 – Representação esquemática do método *hindsight* para solução de processos markovianos de decisão.

O método *hindsight* não requer a geração prévia de políticas de base. Porém, como mencionado por Chang (2001), a solução do método *hindsight* pode

valorizar excessivamente a utilidade de ações que, pela suposição de visão antecipada certa e consequente inserção de um problema determinístico no processo de resolução, geram expectativas de resultados bons, mas que na verdade muito provavelmente apresentariam resultados ruins. Isso limita a abrangência de aplicação do método a modelos com características favoráveis a não ocorrência desse fenômeno. Além disso, a resolução dos problemas determinísticos associados ao método *hindsight* pode tornar-se computacionalmente onerosa, principalmente para problemas com maiores espaços de ações e/ou quando o horizonte de planejamento reduzido aumenta. A complexidade do tempo de computação do método *hindsight* para obtenção de uma ação, dado um estado $x \in X$, é $O(C|A|^H)$.

Assim como nos métodos KMN, *rollout* e *parallel rollout*, a chave para um bom desempenho da aplicação de uma política *hindsight* é a conservação da ordem estabelecida entre os verdadeiros valores de utilidade máxima das ações. Mantida essa ordem, um controlador que siga as políticas determinadas por qualquer um dos métodos amostrados em horizonte de planejamento reduzido estará agindo otimamente, independentemente do erro de aproximação em relação a $Q_H^*(x,a)$.

3.5 Métodos evolutivos de iteração de políticas

Nesta seção apresentam-se, a partir do estudo de Chang *et al.* (2007), dois métodos evolutivos de iteração de políticas aplicados à busca de políticas ótimas para PMDs formulados a horizonte de planejamento infinito.

Como mencionado anteriormente, uma política ótima π^* pode ser obtida através dos tradicionais AIP e AIV; porém, esses métodos impõem avaliações sobre todo o espaço de possíveis ações $A(x)$ para cada estado $x \in X$ a cada

iteração do processo de otimização. Isso traz complicações computacionais para a solução de PMDs que apresentam grandes espaços de ações. Os métodos evolutivos de iteração de políticas foram desenvolvidos para solucionar PMDs com grandes espaços de ações. A complexidade computacional em cada iteração dos métodos evolutivos é polinomial sobre o tamanho do espaço de estados, mas, ao contrário do AIP e do AIV, não é afetada significativamente pelo tamanho do espaço de ações. Assim, os dois métodos evolutivos de iteração de políticas que serão apresentados se ajustam particularmente bem aos casos de PMDs com espaço de ações grande mas com espaço de estados pequeno.

Os métodos evolutivos são baseados na realização de buscas diretas no conjunto de políticas de decisões estacionárias Π_S para evitar a realização de maximizações sobre todo o espaço de ações. Nos métodos evolutivos, uma população (conjunto) de políticas de decisão é melhorada a cada geração (iteração), à semelhança do que ocorre nos algoritmos genéticos. Com abordagem distinta da abordagem dos métodos que serão apresentados nesta seção, encontram-se na literatura alguns estudos que empregam algoritmos genéticos para a solução de PMDs. Citam-se, entre eles, Lin *et al.* (2004), que usaram a abordagem dos algoritmos genéticos para construir o conjunto mínimo de funções afins que descrevem a função de recompensa para PMDs parcialmente observáveis, funcionando como uma variante do AIV; Chin e Jafari (1998), que propuseram um método que mapeia algoritmos genéticos heuristicamente "simples" (SRINIVAS; PATNAIK, 1994) para o contexto do AIP; e Barash (1999), que desenvolveu uma busca genética no espaço de políticas, de forma similar a Chin e Jafari (1998), para PMDs com recompensa descontada em horizonte de planejamento infinito.

Seja Λ_k um conjunto de políticas de decisão correspondente à $k-ésima$ geração de populações de políticas, tal que $\Lambda_k \subset \Pi_S$ e Λ_k é composto por um

número constante de políticas ao longo do processo iterativo, $n = |\Lambda_k| > 1$, definem-se, a seguir, dois conceitos-chave que serão empregados nos métodos apresentados.

O primeiro conceito é o conceito de "política de elite". Uma política de elite em uma geração de uma população de políticas é aquela cuja recompensa total descontada esperada é pelo menos tão grande quanto a maior recompensa total descontada esperada entre as políticas de decisão pertencentes à população de políticas da geração imediatamente anterior. Formalmente, $\pi^e \in \Pi_s$ é uma política de elite para uma população de políticas $\Lambda_k \subset \Pi_s$ se

$$V_\infty^{\pi^e}(x) \geq V_\infty^{\pi}(x) \text{ para todo } \pi \in \Lambda_k \text{ e para todo } x \in X .$$

Nos métodos evolutivos de iteração de políticas, as políticas de elite têm a propriedade de serem monotônicas, ou seja, considerando-se gerações sucessivas de populações de políticas $\Lambda_k \subset \Pi_s$ e $\Lambda_{k+1} \subset \Pi_s$,

$$V_\infty^{\pi_{k+1}^e}(x) \geq V_\infty^{\pi_k^e}(x) \text{ para todo } x \in X .$$

O segundo conceito é o conceito de "distribuição de probabilidades para seleção de ações", representada por P_x. Dado um estado $x \in X$, P_x determina as probabilidades de se selecionar cada uma das ações pertencentes ao espaço de ações $A(x)$. Essa distribuição é empregada ao se realizarem "mutações" (alterações) nas políticas de uma população de políticas para a geração de novas políticas, explorando-se de forma randômica o espaço de possíveis políticas estacionárias Π_s. Sendo o espaço de ações discreto, então $P_x(a)$, $a \in A(x)$, denota a probabilidade de se selecionar a ação a, dado que o sistema encontra-se no estado x, sendo que

$$\sum_{a \in A(x)} P_x(a) = 1 \text{ e } P_x(a) > 0 \text{ para todo } a \in A(x) .$$

Uma escolha possível para P_x é a distribuição uniforme em $A(x)$, de forma que a probabilidade de se selecionar uma ação $a \in A(x)$ seja igual a $1/|A(x)|$.

A propriedade de monotonicidade das políticas de elite e a exploração randômica de Π_s, através da distribuição de probabilidades para seleção de uma ação, como definida, asseguram que os métodos evolutivos de iteração de políticas convirjam com probabilidade igual a 1 para uma população na qual a política de elite é uma política ótima.

3.5.1 Método *evolutionary policy iteration*

O método *Evolutionary Policy Iteration* – EPI – foi desenvolvido por Chang *et al.* (2005b). A determinação de uma política de elite é uma das operações centrais do método EPI. Essa operação é realizada através do método *policy switching*, descrito na Subseção 3.4.3. Dada uma população de políticas $\Lambda \subset \Pi_s$, a política de elite π^e é obtida por

$$\pi^e(x) = \pi(x), \text{ tal que } \pi \in \{\arg\sup_{\pi' \in \Lambda}(V_\infty^{\pi'}(x))\}, \text{ para todo } x \in X.$$

Sendo π^e gerada dessa forma, o valor da recompensa descontada esperada de uma política de elite é sempre maior ou igual ao valor da recompensa descontada esperada de qualquer política pertencente à população de políticas Λ, ou seja

$$V_\infty^{\pi^e}(x) \geq \sup_{\pi \in \Lambda} V_\infty^{\pi}(x) \text{ para todo } x \in X.$$

A cada iteração do método EPI, gera-se uma política π_k^e, denominada política de elite relativa à população de políticas corrente $\Lambda_k \subset \Pi_s$, a qual melhora

qualquer política pertencente a $\Lambda_k \subset \Pi_s$ através da aplicação do método *policy switching*. Incluindo-se a política de elite π_k^e na composição da próxima geração de políticas $\Lambda_{k+1} \subset \Pi_s$, garante-se que Λ_{k+1} contém uma política que é maior ou igual a qualquer política pertencente à população de políticas anterior. Assim, a propriedade de monotonicidade, nas sucessivas gerações de políticas de elite fica garantida:

$$V_\infty^{\pi_{k+1}^e}(x) \geq V_\infty^{\pi_k^e}(x) \text{ para todo } x \in X \text{ e para todo } k \geq 0.$$

Fixando-se o tamanho das populações de políticas em um número $n > 1$, depois que uma política de elite foi gerada e o critério de parada do processo iterativo do método EPI não foi atingido, devem-se gerar outras $n-1$ políticas para compor uma nova população de políticas. Para tal, primeiramente geram-se $n-1$ subconjuntos de políticas a partir da população de políticas corrente Λ_k, denominando-se cada subconjunto por S_i, $i = 1,...,n-1$. Esses subconjuntos são gerados como segue:

(a) Seleciona-se, com igual probabilidade, um número $m \in \{1,...,n\}$.

(b) Seleciona-se, com igual probabilidade, m políticas em Λ_k para compor o subconjunto de políticas S_i.

(c) Repete-se o passo (b) na geração de cada subconjunto, $i = 1,...,n-1$.

(d) Pela aplicação do método *policy switching*, geram-se $n-1$ políticas definidas por

$$\pi^{S_i}(x) = \underset{\pi(x)|\pi \in S_i}{\arg\sup} \{V_\infty^{\pi}(x)\}, \text{ para todo } x \in X,\ i = 1,...,n-1.$$

As $n-1$ políticas geradas são, então, mutadas. A mutação de uma política compreende a alteração da ação especificada para cada estado de acordo com uma regra probabilística. A razão principal para se realizar a mutação nas políticas de uma nova população é a necessidade de se assegurar a

possibilidade de exploração completa do espaço de ações, tornando viável a garantia probabilística da convergência do método EPI para uma política ótima.

Para cada subconjunto de políticas π^{S_i} procede-se uma mutação da seguinte forma:

(a) Determina-se se a mutação da política será "global" com probabilidade q_o ou "local" com probabilidade $1-q_o$. Os dois tipos de mutação diferenciam o quanto se espera alterar a política π^{S_i}. Uma mutação local visa a manter a nova política na vizinhança da política π^{S_i}, em busca de uma melhor política nas suas proximidades, enquanto uma mutação global permite ao EPI escapar de políticas ótimas locais.

(b) Se a política π^{S_i} for mutada globalmente, para cada estado $x \in X$ a ação $\pi^{S_i}(x)$ será alterada com probabilidade P_g. Se a mutação for local, então cada ação de π^{S_i} será alterada com probabilidade P_l. Uma alta probabilidade de alteração indica que muitas ações componentes da política serão mudadas, representando uma mutação global, enquanto uma probabilidade de alteração pequena implica que poucas ações da política serão mudadas, permitindo uma busca na vizinhança de π^{S_i}. Sendo assim, devem-se estabelecer as probabilidades de alterações em uma ordem tal que $P_l < P_g$, mantendo-se P_l próxima a zero e P_g próxima a um.

(c) Se for determinada a alteração de uma ação da política π^{S_i}, uma nova ação para compor a política mutada deve ser selecionada do espaço de ações $A(x)$ de acordo com a distribuição de probabilidade para seleção de ações P_x. Caso não seja determinada uma alteração, a ação original de π^{S_i} fica mantida. Por exemplo, uma P_x simples seria uma distribuição uniforme para seleção de ações, de forma que uma ação seria selecionada com igual probabilidade dentre todas as ações pertencentes a $A(x)$.

Assegurando-se que todas as ações pertencentes a $A(x)$ tenham em P_x probabilidades positivas de serem selecionadas, para todo $x \in X$ o mecanismo de mutação descrito, juntamente com o método *policy switching* para determinação de políticas de elite, possibilita que a convergência probabilística do método EPI seja garantida.

Em resumo, uma nova população de políticas Λ_{k+1} é o conjunto formado pela política de elite π_k^e e pelas $n-1$ políticas mutadas a partir da aplicação do método *policy switching* a $n-1$ subconjuntos de políticas de Λ_k.

No método EPI, ao contrário do AIP, se as recompensas descontadas esperadas de duas políticas de elite geradas sucessivamente são idênticas, não se pode afirmar que necessariamente a política de elite correspondente à última iteração é ótima. O método EPI requer a especificação de uma regra de parada. Os autores do método, Chang *et al.* (2005b), sugeriram um critério de parada simples, em que o processo iterativo é interrompido quando nenhum acréscimo ao valor esperado de retorno descontado é obtido em K iterações sucessivas. Ou seja, aceita-se a solução da política de elite π_{k+K}^e quando

$$V_\infty^{\pi_{k+m}^e}(x) = V_\infty^{\pi_k^e}(x) \text{ para todo } x \in X;\ m = 1,2,...,K.$$

Aumentando-se o valor de K, aumenta-se a probabilidade de se estar na vizinhança de uma política ótima; espera-se, também, que a política de elite π_{k+K}^e, se não ótima, tenha maior chance de ser uma boa solução.

Na Figura 3.5 representa-se esquematicamente o método EPI. Partindo-se de uma população inicial de políticas, dois passos são repetidos: (1) a escolha de uma política de elite e (2) a geração de uma nova população. O processo iterativo prossegue até que os valores de K políticas de elite sucessivas sejam idênticos.

Chang *et al.* (2005b) demonstraram a convergência em probabilidade do método EIP, de forma que: sendo o espaço de estados X finito, $q_o \in (0,1)$, $P_g > 0$ e $P_l > 0$, e P_x tal que $\sum_{a \in A(x)} P_x(a) = 1$ e $P_x(a) > 0$ para todo $a \in A(x)$ e para todo $x \in X$, então, com probabilidade igual a 1, uniformemente sobre X, quando $k \to \infty$

$$V_{\infty}^{\pi_k^e}(x) \to V_{\infty}^{*}(x) \text{ para todo } x \in X,$$

independentemente da população inicial Λ_0.

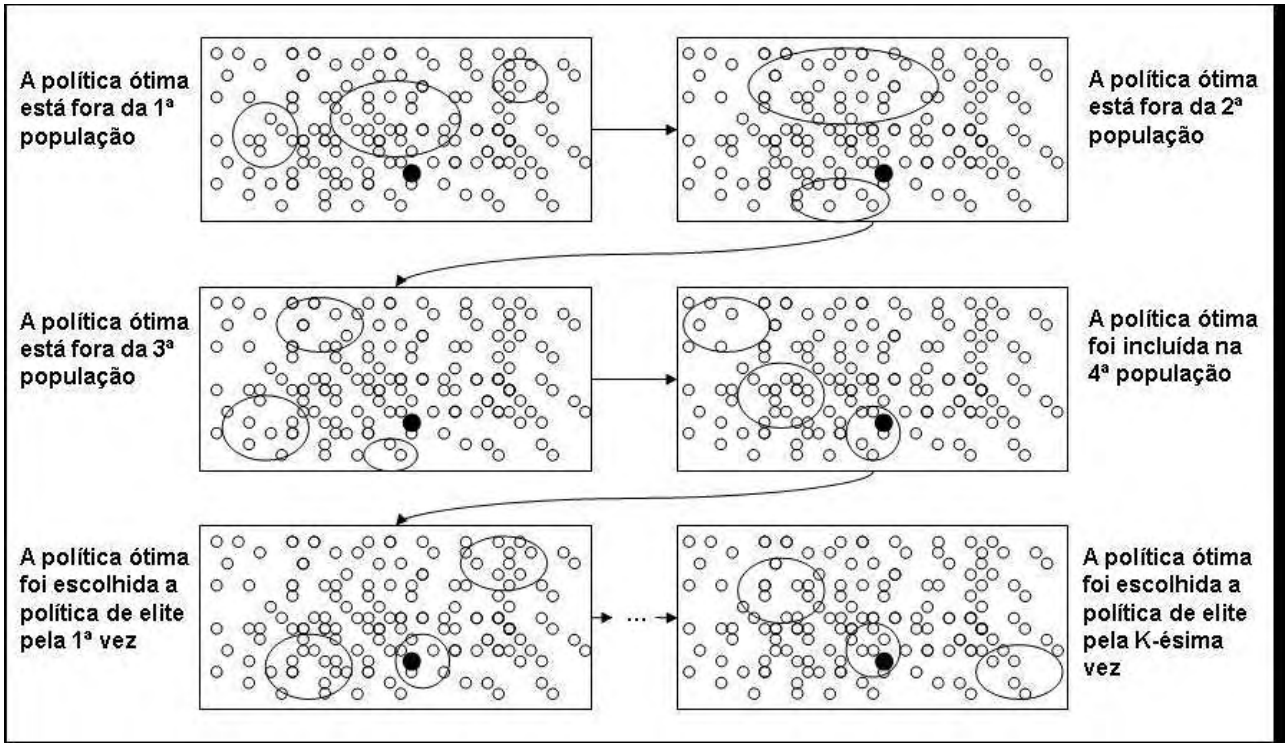


Figura 3.5 – Representação esquemática do método *evolutionary policy iteration* para solução de processos markovianos de decisão. Os pequenos círculos dentro dos retângulos representam todas as possíveis políticas estacionárias; os círculos selecionados nos conjuntos em cada retângulo representam as políticas de uma população de políticas; e o círculo marcado representa a política ótima.

A complexidade computacional no método EPI independe do tamanho do espaço de ações. A técnica *policy switching* manipula diretamente as políticas, eliminando a operação de maximização sobre o espaço de ações completo. A complexidade computacional para obtenção de uma política de elite em uma iteração do método EPI é polinomial de ordem três, definida por $O(nm|X|^3)$, em que: n é o tamanho de Λ_k, a população de políticas avaliadas a cada iteração; m é o número de políticas selecionadas de Λ_k para compor cada subconjunto de políticas S_i empregado no processo de mutações das políticas; e $|X|$ é o tamanho do espaço de estados. Uma vez que o número de operações requeridas por um método direto de solução de um sistema de equações lineares (por exemplo, eliminação gaussiana) é igual a $O(|X|^3)$, a complexidade dominante no método EPI é aquela requerida para a solução de n sistemas de equações lineares relativos ao cálculo da recompensa total descontada esperada de cada uma das políticas pertencentes a Λ_k.

3.5.2 Método *evolutionary random policy search*

O método *Evolutionary Random Policy Search* (ERPS), desenvolvido por Hu *et al.* (2007), é uma evolução do método EIP, que melhora tanto o processo de determinação da política de elite quanto a realização de mutações no processo de geração de uma nova população de políticas. O conceito do método ERPS é subdividir randomicamente um PMD com espaço de ações grande em uma sequência de subPMDs com espaços de ações pequenos, computacionalmente tratáveis, e extrair uma sequência de políticas convergente para a política ótima do PMD original através da resolução dos subPMDs.

A cada iteração o ERPS repetem-se dois passos principais: (1) Uma política de elite é determinada a partir da solução do subPMD construído na iteração prévia. Essa solução é obtida através da aplicação de uma técnica de melhoramento de políticas denominada *Policy Improvement With Reward Swapping* (PIRS). (2) Baseada na política de elite obtida, uma nova população de políticas é gerada através da aplicação randomicamente alternada da heurística *nearest neighbor* e de uma busca amostrada diretamente no espaço de ações do PMD original. Através da redução do espaço de ações do PMD original às ações que compõem a nova população, gera-se um novo subPMD.

A redução do PMD original ocorre como segue. A partir de uma população de políticas Λ, restringem-se as ações possíveis para cada $x \in X$ ao subconjunto de ações $\Gamma(x) = \{\pi(x) \mid \pi \in \Lambda\}$, de forma que um subPMD G_Λ é induzido, sendo representado pela *n-upla* (X, Γ, P, R) em que $\Gamma = \bigcup_{x \in X} \Gamma(x)$ e $\Gamma \subset A$.

No método ERPS, o subPMD G_Λ não é solucionado de forma exata. O ERPS emprega um esquema de aproximação através da técnica PIRS, pelo qual se obtém uma política de elite com uma performance mínima garantida. Dada uma população de políticas $\Lambda \subset \Pi_s$ uma política de elite π^e gerada pela técnica PIRS para o subPMD G_Λ é definida por

$$\pi^e(x) \in \arg\sup_{a \in \Gamma(x)} \{ R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) \sup_{\pi \in \Lambda} V_\infty^\pi(y) \} \text{ para todo } x \in X .$$

Hu *et al.* (2007) demonstraram que a política de elite gerada através da técnica PIRS é melhor ou igual a qualquer política pertencente a Λ, ou seja,

$$V_\infty^{\pi^e}(x) \geq \sup_{\pi \in \Lambda} V_\infty^\pi(x) \text{ para todo } x \in X .$$

Os autores demonstraram, também, que se π^e não é ótima para o subPMD G_Λ, então $V_\infty^{\pi^e}(x) > \sup_{\pi \in \Lambda} V_\infty^\pi(x)$ para pelo menos um $x \in X$.

Considerando-se que a política de elite da $k-ésima$ iteração do método ERPS, π^{e_k}, melhora qualquer política da população de políticas Λ_k, e que a população de políticas Λ_{k+1} inclui em sua formação a política π^{e_k}, a propriedade desejada de monotonicidade na evolução das políticas de elite fica garantida, ou seja, para todo $k \geq 0$

$$V_{\infty}^{\pi_{k+1}^{e}}(x) \geq V_{\infty}^{\pi_{k}^{e}}(x) \text{ para todo } x \in X .$$

Para a geração de uma nova população de políticas, o método ERPS emprega uma técnica de amostragem viesada, que combina exploração local, através do algoritmo *nearest neighbor*, e exploração global, empregando uma busca direta no espaço de ações, através da distribuição de probabilidades de escolha de ações P_x. O balanceamento entre os dois tipos de exploração é determinado pela probabilidade de exploração local q_o. Para uma dada política de elite π^{e_k}, constrói-se uma nova política π para compor a população de políticas Λ_{k+1}. Para cada estado $x \in X$, com probabilidade q_o, $\pi(x)$ é selecionado na vizinhança de $\pi^{e_k}(x)$ ou, com probabilidade $1-q_o$, $\pi(x)$ é selecionado randomicamente entre as possíveis ações do espaço de ações $A(x)$. Esse procedimento é repetido até que sejam obtidas $n-1$ novas políticas. A população de políticas Λ_{k+1} será formada pela política de elite π^{e_k} e pelas $n-1$ novas políticas.

Repetidas realizações do método ERPS geram sequências distintas de políticas de elite; portanto, o método induz a uma distribuição de probabilidade sobre o conjunto de todas as possíveis sequências de políticas de elite. Explorando esse fato, Hu *et al.* (2007) demonstraram, em relação à convergência do método, que: sendo π^* uma política ótima, com recompensa

descontada esperada a horizonte infinito dada por $V_{\infty}^{\pi^*}$, e considerando (1) o espaço de ações finito, (2) a probabilidade de exploração local $q_o < 1$ e a probabilidade de escolha de uma ação $P_x(a) > 0$, para todo $x \in X$ e para todo $a \in A$, existe uma variável aleatória $M > 0$, com $E[M] < \infty$, tal que $V_{\infty}^{\pi_k^e} = V_{\infty}^{\pi^*}$ para todo $k \geq M$.

As demonstrações de convergência em probabilidade dos métodos EPI e ERPS são resultados teóricos. O número de iterações necessário para se atingir uma solução ótima através dos métodos evolutivos de iteração de políticas pode ser extremamente grande. Em relação à eficiência dos métodos, pela utilização de uma busca viesada entre as possíveis políticas e pela aplicação da técnica PIRS para determinar políticas de elite, os autores do método ERPS esperam uma convergência mais rápida desse método comparada à convergência do método EIP.

Independentemente do espaço de ações, a complexidade computacional para obtenção de uma política de elite em uma iteração do método ERPS é $O(n|X|^3)$, semelhante à do método EPI, sendo também dominada pela solução de n sistemas de equações lineares relativos ao cálculo da recompensa total descontada esperada de cada uma das políticas pertencentes a Λ_k, a população de políticas avaliadas a cada iteração.

3.6 Método evolutivo em horizonte de planejamento reduzido e amostrado

Problemas de grandes dimensões têm sido um desafio constante aos pesquisadores que trabalham com PMDs. Na Seção 3.4 foram apresentados os métodos amostrados em horizonte de planejamento reduzido, que proporcionam uma abordagem direcionada à solução de PMDs com espaço de

estados grande e espaço de ações pequeno, enquanto os métodos evolutivos de iteração de políticas, apresentados na Seção 3.5, são adequados para a solução de PMDs com espaço de ações grande e espaço de estados pequeno.

Nesta seção apresenta-se um método combinado com o propósito de solucionar PMDs com espaços de estados e de ações simultaneamente grandes. A combinação em proposição será denominada método “Evolutivo em Horizonte de Planejamento Reduzido e Amostrado” (EHRA).

Mais especificamente, serão combinados os conceitos do método *parallel rollout* (Subseção 3.4.3) e do método *evolutionary random policy search* – ERPS (Subseção 3.5.2). No método EHRA, um método evolutivo fundamentado no ERPS será empregado para gerar, a cada iteração, uma nova população de ações a partir da ação de elite da população de ações imediatamente anterior. Uma ação de elite em uma população de ações é aquela cujo valor de máxima utilidade (Subseção 3.1) estimado para o estado observado é pelo menos tão grande quanto o maior valor de máxima utilidade estimado entre as ações pertencentes à população de ações avaliada. O método *parallel rollout*, amostrado em um horizonte de planejamento reduzido, será empregado para se estimar, dado o estado observado, o valor de máxima utilidade de cada ação pertencente às sucessivas populações de ações avaliadas. Observa-se que a cada iteração do novo método será gerada uma ação de elite, e não uma política de elite, como ocorre no método ERPS, porque, ao se considerar um espaço de estados muito grande, a tarefa de se gerar uma política de decisão com ações estabelecidas para cada estado torna-se computacionalmente onerosa.

Em relação à obtenção da ação de elite, escolheu-se o método *parallel rollout* para compor o novo método, pois, ao considerar em seu processo iterativo um número maior de políticas de base, o método *parallel rollout* apresenta desempenho mais uniforme sobre o espaço de estados que o método *rollout*

(Subseção 3.4.2), sendo capaz de alternar políticas quando o espaço de estados apresenta subconjuntos de estados para os quais uma política de base é melhor do que outra. Além disso, o método *parallel rollout*, quando emprega apenas uma política de base, equivale ao método *rollout*. Quanto aos outros métodos amostrados em horizonte reduzido, descartou-se o método *hindsight* (Subseção 3.4.4) por ser necessário neste método solucionar um problema determinístico que se torna computacional oneroso à medida que o espaço de ações e o horizonte de planejamento reduzido H aumentam (CHANG *et al.*, 2007), e descartou-se o método KMN (Subseção 3.4.1) por ser um método menos eficiente do que o *parallel rollout*, pois sua árvore de busca cresce exponencialmente em função do horizonte de planejamento H.

No tocante ao componente evolutivo do método em proposição, ou seja, em relação à geração de uma sequência de populações de ações e, consequentemente, de uma sequência de ações de elite, o conceito do método ERPS foi escolhido para nortear o novo método, pois traz melhoramentos em relação ao método EPI (Subseção 3.5.1). No ERPS, o método empregado para a determinação de uma política de elite (*policy improvement with reward swapping* – PIRS) e o método de busca local viesada para geração de novas populações de políticas (*nearest neighbor*) proporcionam convergência mais rápida para uma boa solução.

Como apresentado anteriormente, no método ERPS a obtenção de uma política de elite π^{e_k}, em uma iteração k, através da técnica PIRS,

$$\pi^{e_k}(x) \in \underset{a \in \Gamma_k(x)}{\arg\sup} \{ R(x,a) + \gamma \sum_{y \in X} P(y \mid (x,a)) \underset{\pi \in \Lambda_k}{\sup} V_\infty^\pi(y) \} \text{ para todo } x \in X,$$

requer o cálculo da recompensa descontada esperada, $V_\infty^\pi(y)$, de cada uma das políticas π pertencentes à população de políticas Λ_k, para todo estado $y \in X$. Para PMDs com espaço de estados muito grande, essa determinação é

computacionalmente onerosa. No método EHRA o que se pretende é evitar esse cálculo, substituindo-o por um método amostrado.

Considerando-se um PMD com espaço de ações muito grande, o princípio do método EHRA é semelhante ao do método ERPS, ou seja, restringe-se randomicamente o espaço de ações, tornando o PMD reduzido computacionalmente tratável do ponto de vista do espaço de ações. Porém, para viabilizar o tratamento computacional de PMDs com espaços de ações e de estados simultaneamente muito grandes, o processo evolutivo do método EHRA difere do método ERPS. No método EHRA, para se obter uma boa ação em um instante de decisão, dado o estado do sistema $x \in X$, gera-se uma sequência de "ações de elite", $a^{e_k}(x)$, $k = 0,1,...$, ao contrário do método ERPS, no qual, dado um espaço de estados computacionalmente tratável, o processo evolutivo gera uma sequência de políticas de elite, π^{e_k}, $k = 0,1,...$.

Através do método EHRA se estabelece uma política de decisão *on-line* (ver Seção 3.4), π_H^{ehra}, ou seja, a cada instante de decisão, dado um estado $x \in X$, determina-se uma ação através de um processo iterativo. Nesse processo, a cada iteração são realizados três passos: (1) determinação de uma ação de elite $a^{e_k}(x)$, referente à população de ações $\Gamma_k(x)$ vigente na $k-ésima$ iteração; (2) verificação de um critério de parada, através da avaliação do "conjunto de ações de melhor desempenho"; (3) geração de uma nova população de ações $\Gamma_{k+1}(x)$ para a próxima iteração, caso o critério de parada não seja atingido, ou escolha da ação de elite da $k-ésima$ iteração para a tomada de decisão, caso o critério seja atingido. A seguir, descrevem-se em detalhes esses passos.

3.6.1 Determinação de uma ação de elite

Seja $\Lambda \subset \Pi_s$ um conjunto de políticas de base geradas a partir de heurísticas predeterminadas e seja $\Gamma_k(x)$ a população de ações reduzida de $A(x)$ a ser considerada na $k-ésima$ iteração, no método EHRA uma ação de elite $a^{e_k}(x)$ é determinada por

$$a^{e_k}(x) = \arg\sup_{a \in \Gamma_k(x)} \hat{Q}_H^{\Lambda}(x,a), \text{ em que}$$

$$\hat{Q}_H^{\Lambda}(x,a) = R(x,a) + \gamma \frac{1}{C} \sum_{j=1}^{C} (\sup_{\pi \in \Lambda} \hat{V}_{H-1}^{\pi}(x,a))_j ,$$

sendo que $\hat{V}_{H-1}^{\pi}(x,a)$ representa um valor simulado para a recompensa descontada esperada do sistema sob controle de uma das políticas de base em um horizonte de planejamento reduzido $H-1$, dado que no instante de decisão inicial o estado do sistema era x e a ação a foi adotada. Para cada ação $a \in \Gamma_k(x)$, uma estimativa $\hat{Q}_H^{\Lambda}(x,a)$ para $Q_H^{\Lambda}(x,a)$, o verdadeiro valor da utilidade máxima da ação em relação ao conjunto de políticas de base Λ, é obtida através da simulação de $C|\Lambda|$ "caminhos de funcionamento do sistema". Em resumo: dado o estado inicial x e a ação adotada a, simula-se o funcionamento do sistema em um horizonte de planejamento $H-1$, sob o controle de cada uma das políticas de base pertencentes a Λ; seleciona-se o maior valor simulado do retorno descontado esperado; repete-se esse procedimento C vezes; o valor de $\hat{Q}_H^{\Lambda}(x,a)$ será a soma entre o retorno esperado no primeiro intervalo de funcionamento do sistema e a média descontada das C repetições do processo simulado.

Se, em qualquer iteração k, para todo $x \in X$, a população de ações $\Gamma_k(x)$ contiver, além das outras ações em avaliação, as ações determinadas por cada uma das políticas de base, ou seja, $\pi(x) \in \Gamma_k(x)$ para todo $\pi \in \Lambda$, e se o

número de simulações C for grande o suficiente para garantir que $\gamma \frac{1}{C} \sum_{j=1}^{C} (\sup_{\pi \in \Lambda} \hat{V}_{H-1}^{\pi}(x,a))_j$ seja uma aproximação precisa para o valor de $\gamma \sum_{y \in X} P(y \mid (x,a)) \sup_{\pi \in \Lambda} V_{H-1}^{\pi}(y)$, então pode-se garantir, pela definição de $a^{e_k}(x)$, que

$$Q_H^{\Lambda}(x, a^{e_k}(x)) \geq \sup_{\pi \in \Lambda} Q_H^{\Lambda}(x, \pi(x)).$$

Além disso, se a população de ações $\Gamma_k(x)$ contiver também a ação de elite da população de ações da iteração imediatamente anterior, $a^{e_{k-1}}(x)$, pode-se garantir que

$$Q_H^{\Lambda}(x, a^{e_k}(x)) \geq Q_H^{\Lambda}(x, a^{e_{k-1}}(x)).$$

Como mencionado na Subseção 3.4.2, Chang (2001) estabeleceu, em função do horizonte de planejamento reduzido H, do fator de desconto γ e do máximo valor esperado de retorno em um período de funcionamento do sistema $R_{\max}$, o número de simulações C necessário para se garantir uma precisão ε, com probabilidade $1-\delta$, para a estimativa de $Q_H^{\pi}(x,a)$ por $\hat{Q}_H^{\pi}(x,a)$ para cada política de base $\pi \in \Lambda$. Dado um $\varepsilon > 0$ e $0 < \delta < 1$, então

$$\Pr\{ \left| Q_H^{\pi}(x,a) - \hat{Q}_H^{\pi}(x,a) \right| \leq \varepsilon \} \geq 1-\delta \quad \text{se}$$

$C \geq \frac{{R_{\max}}^2 \gamma^2 (1-\gamma^H)^2}{\varepsilon^2 \delta (1-\gamma)^2}$, quando $0 < \gamma < 1$, ou

$C \geq \frac{H^2 {R_{\max}}^2}{\varepsilon^2 \delta}$, quando $\gamma = 1$.

Em relação ao erro de aproximação introduzido pelo horizonte de planejamento reduzido H, fazendo-se ainda referência às demonstrações elaboradas por Chang (2001) para o método *parallel rollout*, e considerando-se a política de

decisão definida para o método EHRA, π_H^{ehra}, pode-se determinar um tamanho para H que garanta que a recompensa descontada esperada para a política π_H^{ehra}, quando aplicada a um horizonte de planejamento infinito, seja maior ou igual à recompensa descontada esperada de qualquer uma das políticas de base $\pi \in \Lambda$, a menos de um erro τ. Assim, presumindo-se que na determinação da política π_H^{ehra} o valor $\hat{Q}_H^{\pi}(x,a)$ seja uma aproximação precisa de $Q_H^{\pi}(x,a)$, se

$$H \geq 1 + \log_{\gamma} \frac{\tau(1-\gamma)}{R_{\max}}, \text{ então}$$

$$V_{\infty}^{\pi^{erha}}(x) \geq \sup_{\pi \in \Lambda} V_{\infty}^{\pi}(x) - \tau, \text{ para todo } x \in X,$$

sendo τ a precisão que se deseja garantir.

O ajuste inter-relacionado dos parâmetros C, δ, ε, H, τ e γ é determinante para a qualidade final da política de decisão π_H^{ehra}. Aumentando-se o número de simulações, C, pode-se garantir com maior probabilidade, $(1-\delta)$, um erro menor, ε, para a aproximação de $Q_H^{\Lambda}(x,a)$ por $\hat{Q}_H^{\Lambda}(x,a)$. Aumentando-se o horizonte de planejamento reduzido H garante-se que a política π_H^{ehra} alcançará uma recompensa descontada esperada maior ou igual à recompensa descontada esperada de qualquer uma das políticas de base, a menos de um erro preestabelecido, τ. Reduzindo-se o fator de desconto γ, dá-se menos importância aos instantes de decisão à medida que estes se afastam do instante de decisão inicial, permitindo uma diminuição tanto no número de simulações C quanto no tamanho do horizonte de planejamento reduzido H, sem perda de precisão. Entretanto, a determinação de γ é dependente do problema modelado; se os instantes de decisão ao final do horizonte de planejamento forem tão relevantes quanto os instantes próximos ao instante inicial de decisão, o fator de desconto γ deve ser calibrado próximo de 1,

tornando necessário que se aumentem os valores de C e H para se garantir a precisão das estimativas de $Q_H^{\Lambda}(x,a)$ e a qualidade da política de decisão proposta.

A calibragem dos parâmetros deve ser realizada empiricamente. Uma sequência lógica possível para essa calibragem seria: (1) determina-se o fator de desconto γ em função das características do problema; (2) determina-se o erro τ de precisão aceitável em relação à redução do horizonte de planejamento; (3) com esses dois parâmetros calcula-se um valor mínimo para H; (4) determina-se a precisão desejada ε em relação às estimativas de $Q_H^{\Lambda}(x,a)$ e a significância probabilística em relação a essa precisão $(1-\delta)$; (5) finalmente, com o valor dos demais parâmetros estabelecidos, calcula-se um valor mínimo recomendável para C.

Definidos os parâmetros γ, τ, H, ε e δ, o cálculo descrito acima para obtenção de um limitante inferior de C, proposto por Chang [7], frequentemente determina um elevado número de simulações para se garantir uma precisão ε para as estimativas de $Q_H^{\Lambda}(x,a)$, com probabilidade $(1-\delta)$. Entretanto, segundo Chang (2001), esse limitante é conservador, sendo que, na prática, valores menores de C resultam em boas aproximações para $Q_H^{\Lambda}(x,a)$. Chang (2001) ressalta, ainda, que a manutenção da ordem entre as ações em relação aos seus valores de utilidade é mais importante que a obtenção de aproximações precisas para $Q_H^{\Lambda}(x,a)$. Mantida essa ordem, a ação de elite $a^{e_k}(x)$ continua sendo a melhor escolha dentre as ações pertencentes à população de ações $\Gamma_k(x)$, independentemente da qualidade da precisão das aproximações.

3.6.2 Verificação do critério de parada

A implementação do método EHRA, assim como nos demais métodos evolutivos, requer a especificação de uma regra de parada, uma vez que a repetição da escolha de uma mesma ação como ação de elite, em duas ou mais iterações sucessivas, não implica necessariamente que essa ação seja a melhor ação disponível.

O método EHRA introduz aleatoriedade nas aproximações simuladas de $Q_H^{\Lambda}(x,a)$. Mesmo que se controle a precisão das aproximações através da calibragem dos parâmetros C, δ, ε, H, τ e γ, pode ocorrer, no decorrer do processo iterativo, principalmente nos casos em que os valores de utilidade das ações sejam muito próximos, que uma ação com valor verdadeiro de utilidade inferior ao de uma ação de elite anterior venha a ser escolhida como a ação de elite vigente. Entretanto, ao longo de um grande número de iterações, espera-se que a ação com o maior valor de utilidade seja escolhida mais vezes como ação de elite do que as demais ações avaliadas.

Define-se, para avaliação do critério de parada do método EHRA, o "conjunto das m ações de melhor desempenho", Ω. Os m elementos de Ω são $n-uplas$ configuradas por $(a,\ d,\ f)$, em que a é uma ação pertencente a $A(x)$, d registra o número de vezes que a ação a foi escolhida como ação de elite e f registra o valor médio de utilidade da ação. A cada iteração, deve-se atualizar Ω da seguinte forma:

- se $a^{e_k}(x)$ compõe uma das $n-uplas$ pertencentes a Ω, então as informações da $n-upla$ $(a^{e_k}(x),\ d,\ f)$ são atualizadas fazendo-se

$$f = {\{(f \times d) + \hat{Q}_H^{\Lambda}(x, a^{e_k}(x))\}} \Big/ {d+1} \quad \text{e} \quad d = d+1;$$

- se $a^{e_k}(x)$ não compõe uma $n-upla$ pertencente a Ω, então se seleciona a $n-upla$ com o menor valor de f para substituí-la pela nova $n-upla$ $(a^{e_k}(x),1,\hat{Q}_H^{\Lambda}(x,a^{e_k}(x)))$.

O critério de parada do EHRA consiste em, ao final da atualização de Ω a cada iteração, verificar na $n-upla$ $(a^{e_k}(x),\ d,\ f)$ se d, o número de vezes que a ação $a^{e_k}(x)$ foi escolhida como ação de elite, atingiu um valor predeterminado K. Quando $d<K$, prossegue-se com a geração de uma nova população de ações $\Gamma_{k+1}(x)$; quando $d=K$, a ação $a^{e_k}(x)$ é a ação a ser adotada, dado o estado observado $x\in X$, e o processo iterativo é encerrado.

Assim como a determinação dos parâmetros C, δ, ε, H, τ e γ, mencionados acima, a determinação dos parâmetros m e K do critério de parada ocorre de forma empírica. O processo de calibragem desses parâmetros é importante para a convergência e a qualidade das ações estabelecidas através do método EHRA. Aumentando-se o valor de K, aumenta-se a chance de se finalizar o processo iterativo com uma ação que tenha valor de utilidade maior ou no mínimo igual ao valor de utilidade de qualquer uma das ações pertencentes ao conjunto de políticas de base. O número m de elementos em Ω deve ser suficiente para se evitar que possíveis imprecisões nas estimativas simuladas de $Q_H^{\Lambda}(x,a)$ ou a existência de ações com valores de utilidade iguais ou muito próximos causem uma alternância cíclica na escolha das políticas de elite, impedindo a convergência do processo iterativo.

3.6.3 Geração de uma população de ações

Considerando-se precisas as aproximações envolvidas na determinação de uma ação de elite pelo método EHRA e considerando-se, também, que uma nova população de ações sempre contém a ação de elite estabelecida na iteração anterior, por definição, o desempenho de uma ação de elite em uma iteração não deve ser pior do que o desempenho das ações de elite das iterações passadas. Portanto, a forma de obtenção de uma ação de elite, por si só, pode ser entendida como uma busca local pela melhor ação. Entretanto, ao se gerar uma nova população de ações, assim como nos métodos evolutivos EPI e ERPS, para se garantir a convergência do método EHRA para uma boa ação é necessário que se aplique um mecanismo para a exploração do espaço de ações tanto local (próximo à ação de elite) quanto global (em todo o espaço de ações).

Para a realização da exploração global do espaço de ações, emprega-se um esquema de amostragem não viesada sobre o espaço de ações (busca global). Utiliza-se, para isso, uma distribuição de probabilidades para escolha de ações, P_X, dado um estado $x \in X$. As populações de ações em sucessivas iterações, geradas dessa forma, são compostas, em parte, por um conjunto de ações independentemente selecionadas diretamente do espaço de ações, ficando intuitivamente evidente a possibilidade de se obter uma melhor ação de elite depois de um número suficiente de iterações.

Esse esquema de amostragem não viesada é bastante efetivo para se escapar de máximos locais e é frequentemente útil para se encontrarem boas ações candidatas a serem escolhidas como solução final. Porém, na prática, a obtenção de melhores ações através apenas de uma busca global torna-se cada vez mais difícil à medida que o processo iterativo avança, visto que a probabilidade de se encontrar uma ação de elite melhor com saltos aleatórios no espaço de ações torna-se cada vez menor. Portanto, assim como nos

métodos evolutivos EPI e ERPS, no método EHRA, além da busca global, emprega-se também um esquema de busca local nas proximidades da ação de elite, aumentando-se, assim, a chance de se obter uma melhor ação a cada iteração. A técnica *nearest neighbor*, empregada no método ERPS, é aplicada no método EHRA para se determinar uma nova ação através de uma busca local viesada na vizinhança de uma ação de elite.

Define-se a probabilidade de busca local q_o para controlar a alternância entre a seleção de ações através de busca local ou busca global. Assim, no método EHRA, quando se realiza a composição de uma nova população de ações, sempre que for necessário selecionar uma nova ação, com probabilidade q_o a ação será selecionada na vizinhança da ação de elite vigente, através da técnica *nearest neighbor*, e com probabilidade $1-q_o$ a ação será selecionada diretamente do espaço de ações $A(x)$, segundo a distribuição de probabilidades de seleção de ações P_x.

Entretanto, em função da aleatoriedade introduzida na estimativa simulada de $Q_H^{\Lambda}(x,a)$ e de uma possível proximidade entre os valores de utilidade das melhores ações, sabe-se que a ação de elite vigente pode ter valor real de utilidade menor do que o valor de utilidade de ações de elite de iterações anteriores. Sendo assim, ao invés de se incluir em cada nova população apenas a ação de elite vigente, no método EHRA incluem-se todas as ações componentes das $n-uplas$ do conjunto de ações de melhor desempenho, Ω (definido na seção anterior, componente do critério de parada do processo iterativo). Espera-se, com isso, que a ação de elite da próxima iteração tenha desempenho igual ou superior não só em relação à ação de elite vigente, mas também em relação às demais ações pertencentes as $n-uplas$ de Ω. Também com o propósito de garantir que as ações de elite, em cada iteração, tenham desempenho igual ou superior ao desempenho das ações prescritas

pelas políticas de base, incluem-se, ainda, nas novas populações as ações das políticas pertencentes ao conjunto de políticas de base Λ.

Dessa forma, uma nova população de ações é constituída: (1) pelas ações componentes das $n-uplas$ do conjunto de ações com melhor desempenho, Ω (conjunto que inclui a ação de elite da iteração vigente); (2) pelas ações correspondentes às políticas do conjunto de políticas de base Λ; e (3) por um conjunto de novas ações geradas ou através de uma busca local na vizinhança da ação de elite vigente (*nearest neighbor*), com probabilidade q_o, ou, com probabilidade $1-q_o$, através de uma busca global entre as possíveis ações $A(x)$ regida pela distribuição de probabilidades de seleção de ações P_x.

O número de ações em cada população de ações ao longo do processo iterativo pode ser mantido fixo. Seja n o número fixo de ações em cada população de ações da sequência $\Gamma_k(x)$, $k=0,1,2,...$, define-se que $(|\Lambda|+|\Omega|) < n < |A(x)|$, ou seja, n deve ser no mínimo maior do que a soma entre o número de políticas de base e o número de $n-uplas$ no conjunto Ω, e, obviamente, deve ser menor do que o número de ações possíveis para o estado x, $|A(x)|$. Fazendo-se $n>(|\Lambda|+|\Omega|)$ garante-se a possibilidade de que, a cada iteração, pelo menos uma ação possa ser escolhida, independentemente, entre as ações pertencentes ao conjunto de possíveis ações $A(x)$.

A determinação do parâmetro n, assim como os demais parâmetros do método EHRA, ocorre de forma empírica. Além das ações relacionadas aos conjuntos Λ e Ω, as populações de ações devem ser suficientemente grandes para incluir um conjunto de ações selecionadas randomicamente através de uma busca global ou local em $A(x)$, de forma que a exploração do espaço de ações fique garantida. Por outro lado, a cardinalidade das populações de ações deve

ser tal que o processamento computacional a cada iteração possa ser realizado eficientemente.

3.6.4 Técnica *nearest neighbor*

A técnica *nearest neighbor* (HU *et al.*, 2007) é empregada pelo EHRA para gerar uma ação na vizinhança de uma ação de elite (busca local). Presumindo-se que A é um espaço de ações discreto, sendo um espaço métrico não vazio com uma métrica $d(.,.)$ bem definida, a técnica *nearest neighbor* pode ser implementada como descrito a seguir: dada uma ação de elite $a^{e_k}(x)$, estabelece-se um número inteiro positivo, $r < |A|$, para limitar a amplitude desejada da busca local, sendo $|A|$ a cardinalidade do espaço de ações; gera-se um valor para a variável aleatória $l \sim DU[1, r]$, em que $DU[1, r]$ representa a distribuição uniforme discreta no intervalo entre 1 e r, incluindo os limites do intervalo; escolhe-se para ser a "ação vizinha" a $l - ésima$ ação mais próxima de $a^{e_k}(x)$, medida por $d(.,.)$.

A escolha de uma métrica adequada é importante para a eficiência da busca local através da técnica *nearest neighbor*. Em geral, boas ações tendem a permanecer "próximas" dentro do conjunto de possíveis ações; assim sendo, uma boa métrica deve explorar essa característica. Como destacam Hu *et al.* (2007), a distância euclidiana tem sido empregada com bons resultados.

3.6.5 Algoritmo EHRA

O método EHRA gera uma política de decisão *on-line*. Ou seja, em vez de obter uma política de decisão completa, com ações previamente determinadas (*off-line*) para todos os estados do sistema, sob o critério de maximização do retorno descontado esperado a um horizonte de planejamento infinito, o método EHRA visa, dado o estado observado do sistema a cada instante de decisão e considerando um horizonte de planejamento reduzido H, a determinar uma "boa ação", com retorno descontado esperado melhor ou no mínimo igual ao das ações previstas pelas políticas de base.

Apresenta-se, a seguir, o pseudocódigo para implementação do método EHRA. Considera-se a solução de PMDs descritos através da *n-upla* (X, A, P, R).

Especificações de entrada:

- estado observado, $x \in X$;
- fator de desconto, γ;
- horizonte de planejamento reduzido, H;
- número de simulações de funcionamento do sistema, C;
- θ heurísticas geradoras do conjunto de políticas de base, $\Lambda = \{\pi_1, ..., \pi_\theta\}$;
- número m de $n-uplas$ para compor o "conjunto de ações de melhor desempenho" $\Omega = \{(a_1, d_1, f_1), ..., (a_m, d_m, f_m)\}$ sendo que, para $i = 1, ..., m$, a_i representa uma ação pertencente a $A(x)$, d_i registra o número de vezes que a ação a_i foi escolhida como ação de elite e f_i registra o valor médio de utilidade da ação a_i;
- Ω inicial, sendo a_i uma ação qualquer pertencente a $A(x)$, $d_i = 0$ e $f_i = 0$, para $i = 1, ..., m$;
- número máximo K de repetições da escolha de uma ação como ação de elite, para verificação do critério de parada;

- número n de ações para compor as populações de ações $\Gamma_k(x) = \{a_1^k, ..., a_n^k\}$, a serem avaliadas em cada iteração k, $k = 1,2,...$; $(\theta + m) < n < |A(x)|$;
- função de recompensa esperada $R(x,a)$ em um período entre dois instantes de decisão consecutivos, dado o estado e a ação adotada no instante inicial;
- função $f(x,a)$ para simular o próximo estado, dado um estado e uma ação adotada, baseada no conjunto de distribuições de probabilidades de transições entre estados P, definido para o PMD;
- população de ações inicial $\Gamma_0(x) = \{a_1^0, ..., a_n^0\}$, incluindo nas posições de $n-(\theta-1)$ até n as ações estabelecidas pelas θ políticas de base, dado o estado observado x, nas posições de $n-(\theta+m-1)$ até $n-\theta$ as ações correspondentes às $n-uplas$ de Ω, e nas posições de 1 até $n-(\theta+m)$ ações quaisquer pertencentes a $A(x)$;
- distribuição de probabilidades P_x para a escolha de ações em $A(x)$, dado o estado observado x, a ser utilizada na busca global;
- probabilidade q_o para realização de uma busca local na vizinhança da ação de elite, ou $1-q_o$ para realização de uma busca global entre as ações de $A(x)$;
- amplitude r para o intervalo de busca local na vizinhança da ação de elite, através da heurística *nearest neighbor*;

Vetores auxiliares:

- vetor $x[H]$ de estados, para armazenar temporariamente um caminho de funcionamento do sistema de tamanho H;
- vetor $EstQ[n]$, para armazenar a estimativa do valor de utilidade de cada uma das n ações pertencentes à população de ações;

- vetor $EstV[\theta]$, para armazenar a estimativa do valor de retorno descontado de um caminho de funcionamento amostrado para cada uma das θ políticas de base pertencente a Λ;

Algoritmo:

"Seja x o estado observado do sistema:"

fazer o contador $k = 0$

fazer o contador $l = 1$

(0) repetir enquanto $d_l < K$ (até que a regra de parada seja atingida, em que d_l é o número de vezes que a ação a_l foi escolhida como ação de elite, sendo a_l e d_l componentes da $l-ésima \;\; n-upla$ de Ω)

"Determinação de uma ação de elite para a população de ações $\Gamma_k(x)$:"

(1) para o contador $j = 1$ até n fazer

$EstQ[j] = 0$

(2) repetir C vezes

(3) para o contador $i = 1$ até θ fazer

$EstV[i] = 0$

$x[0] = x$

(4) para o contador $t = 1$ até H fazer

apenas quando $t = 1$, fazer $\pi_i(x[0]) = a_j^k$

$EstV[i] = EstV[i] + \gamma^{t-1} R(x[t-1], \pi_i(x[t-1]))$

$x[t] = f(x[t-1], \pi_i(t-1))$

(4) finalizar

(3) finalizar

$EstQ[j] = EstQ[j] + \max_{i=1,\ldots,\theta}(EstV[i])$

(2) finalizar

$$EstQ[j] = EstQ[j] \,/\, C$$

(1) finalizar

fazer $e = \arg\max_{j=1,...,n} EstQ[j]$

"O índice da ação de elite da população k é dado por e."

"Atualização de $\Omega = \{(a_1, d_1, f_1),...,(a_m, d_m, f_m)\}$:"

fazer o contador $l = 1$

(1) repetir enquanto $l < m+1$ e $a_l \neq a_e^k$

$$l = l + 1$$

(1) finalizar

(1) se $l < m+1$, então

$$f_l = \{(f_l \times d_l) + EstQ[e]\} \,/\, d_l + 1$$

$$d_l = d_l + 1$$

(1) finalizar

(1) se $l = m+1$, então

localizar no conjunto Ω o índice s correspondente ao menor f_s, valor médio de utilidade da ação a_s pertencente a $s-ésima$ $n-upla$ de Ω, e então fazer

$$a_s = a_e^k$$

$$f_s = EstQ[e]$$

$$d_s = 1$$

$$l = s$$

(1) finalizar

(1) se $d_l < K$, o critério de parada não foi atingido, então

"Determinação de uma nova população de ações, $k+1$:"

(2) para o contador $i = 1$ até $n - (\theta + m)$ fazer

gerar um número aleatório u, seguindo a distribuição uniforme $U \sim [0,1]$

(3) se $u \leq q_o$, então

escolher a ação a_i^{k+1} na vizinhança de a_e^k, através da heurística *nearest neighbor*

(3) finalizar

(3) se $u > q$, então

escolher a ação a_i^{k+1} empregando a distribuição de probabilidades P_x

(3) finalizar

(2) finalizar

(2) para o contador $i = (n+1) - (\theta + m)$ até $n - \theta$ fazer

$a_i^{k+1} = a_{(n+1)-(i+\theta)}$ (carregar as ações do conjunto Ω)

(2) finalizar

"A nova população de ações é $\Gamma_{k+1}(x) = \{a_1^{k+1}, ..., a_n^{k+1}\}$, composta por $n - (\theta + m)$ ações escolhidas randomicamente de $A(x)$, por m ações que compõem as $n-uplas$ de Ω e por θ ações correspondentes às decisões do conjunto de políticas de base Λ."

fazer o contador $k = k + 1$

(1) finalizar

(0) finalizar

"Após k iterações, a ação escolhida pelo método EHRA para o estado observado x é a_e^k."

A complexidade computacional do pior caso para o método EHRA em cada iteração realizada para obtenção de uma ação de elite é dada por $O(n\,\theta\,C\,H\,g)$, ou seja, uma multiplicação linear entre o tamanho da população

de ações, a quantidade de políticas de base, o número de amostras de caminhos, o tamanho do horizonte de planejamento reduzido e o coeficiente g. O coeficiente g representa a complexidade relativa ao cálculo da função de recompensa esperada $R(x,a)$ em um período. Os parâmetros n, θ, C e H são independentes das dimensões dos espaços de estados e de ações, sendo definidos para as operações das iterações do método. O coeficiente g depende da complexidade da função $R(x,a)$ em relação aos espaços de estados e de ações, podendo ser relevante ou não a sua contribuição para a complexidade do método de acordo com a estrutura da função de custo considerada.

3.7 Considerações sobre os métodos

Neste capítulo foram revistos os conceitos fundamentais dos PMDs e dos dois tradicionais métodos para obtenção de políticas de decisão ótimas, o AIV e o AIP. Também foram apresentados os conceitos dos métodos meta-heurísticos amostrados em horizonte de planejamento reduzido – KMN, *rollout*, *parallel rollout* e *hindsight* – e evolutivos de iteração de políticas – EPI e ERPS –. Em seguida, foi proposta uma combinação entre os métodos *parallel rollout* e ERPS para formação de um novo método evolutivo em horizonte de planejamento reduzido e amostrado, o EHRA.

Os métodos tradicionais de otimização AIV e AIP sofrem perda de eficiência computacional à medida que os espaços de estados e de ações aumentam. Os métodos amostrados em horizonte de planejamento reduzido não garantem a otimalidade, tratando PMDs com espaço de estados maiores que os suportados pelos métodos tradicionais, porém sofrem perda de eficiência computacional conforme o espaço de ações aumenta. Os métodos evolutivos de iteração de políticas são capazes de convergir em probabilidade para uma

política ótima, tratando PMDs com espaço de ações maiores do que os suportados pelos AIV e AIP, porém sofrem perda de eficiência computacional à medida que o espaço de estados aumenta.

Espera-se que o método EHRA seja capaz de melhorar as políticas de base iniciais, tratando PMDs com espaços de estados e de ações conjuntamente maiores do que os suportados pelos outros métodos considerados. No Capítulo 4, os resultados das subseções 4.2.4 e 4.2.5 corroboram para a sustentação dessa expectativa. Contudo, mesmo com um desempenho esperado melhor do que o das outras meta-heurísticas, a complexidade computacional das iterações do método EHRA ainda pode, dependendo da estrutura da função de recompensa do modelo considerado, ser agravada pelas dimensões do PMD.

A relação entre as dimensões do PMD e a complexidade das iterações do EHRA ocorre através da estrutura da função de recompensa esperada $R(x,a)$. Considerando-se o modelo para o controle de admissão de pacientes apresentado no Capítulo 2, nota-se que o cálculo da recompensa esperada no próximo período a partir do par (estado observado; ação) nesse modelo exige que se determinem todos os estados que podem ser atingidos a partir desse par; também exige que, para cada estado atingível, se somem as probabilidades de todas as combinações possíveis de movimentações de pacientes que liguem o par (estado; ação) ao estado que pode ser atingido e que, finalmente, se some para todos os estados atingíveis a multiplicação da probabilidade de ser atingido pela recompensa associada ao estado. Como se pode observar nos resultados apresentados na Subseção 4.3.5 do Capítulo 4, a função de recompensa esperada do PMD para o controle da admissão de pacientes não permite que o EHRA solucione instâncias muito grandes do modelo. Mesmo diante dessa obstância, o EHRA é capaz de obter ações para instâncias do modelo maiores do que as que poderiam ser tratadas pelos métodos tradicionais. Por outro lado, assim como se fez para o valor descontado esperado de retorno após o primeiro período do horizonte de

planejamento, é possível substituir o valor exato da função de recompensa em um período por uma estimativa obtida através de simulação. Essa possibilidade, que potencializa a capacidade do EHRA de solucionar PMDs com maiores dimensões, é apresentada e testada na Subseção 4.3.6 do Capitulo 4.

CAPÍTULO 4

RESULTADOS

Por meio de implementação computacional, os objetivos principais deste capítulo são: (1) mostrar que, através do método evolutivo em horizonte de planejamento reduzido e amostrado, apresentado no capítulo anterior, é possível obter uma política de decisão *on-line* que melhora as políticas de base empregadas no EHRA, sendo o retorno esperado do sistema sob controle da política EHRA mais próximo do retorno esperado ótimo; (2) mostrar que o método EHRA é computacionalmente viável em situações em que os métodos tradicionais de obtenção de políticas ótimas (algoritmos de iteração de valores, AIV, e de iteração de políticas, AIP) são ineficientes ou inviáveis; e (3) empregando dados obtidos de um hospital de reabilitação com atendimento exclusivamente eletivo, testar as possibilidades do uso da modelagem apresentada no Capítulo 2 para auxiliar os gestores do hospital na tomada de decisão sobre o número de admissões de pacientes a serem realizadas em intervalos discretos de tempo, sendo o objetivo do controle das admissões manter a utilização dos recursos considerados em níveis preestabelecidos, levando em conta as importâncias relativas dos recursos.

O restante do capítulo está organizado como segue. Na Seção 4.1, descrevem-se os recursos computacionais empregados. Na Seção 4.2, apresentam-se os resultados referentes ao método EHRA para um pequeno modelo hipotético, comparando-os aos resultados alcançados pelo controle das políticas gulosas e da política ótima obtida via AIV. Na Seção 4.3, apresenta-se a implementação do controle de admissões de pacientes, empregando-se dados reais obtidos do sistema de informações de um hospital de reabilitação com atendimento eletivo. Para essa implementação, consideraram-se: pacientes de duas especialidades, complicações na coluna e complicações

musculoesqueléticas no aparelho locomotor em geral; dois recursos, consultas médicas e sessões de fisioterapia; e quatro padrões de atendimento, sendo que o quarto padrão representa a alta hospitalar. Ainda na Seção 4.3, avalia-se a possibilidade de se obter o valor da função de custos via simulação. Finalmente, na Seção 4.4, apresenta-se uma breve discussão sobre os resultados obtidos.

4.1 Recursos computacionais

As análises computacionais representaram parte importante dos esforços desta pesquisa. Considerando-se as etapas de planejamento, desenvolvimento, implementação, teste dos programas e obtenção dos resultados, muitos meses de trabalho foram empenhados. Alguns dos principais recursos de computação utilizados estão descritos, um pouco mais detalhadamente, nas seguintes subseções: (4.1.1) o *hardware* e o *software*; (4.1.2) a técnica para simular os próximos estados do sistema, empregada pelo método amostrado de solução dos PMDs; e (4.1.3) a técnica de compactação de matrizes esparsas, empregada pelo método utilizado para se obter as probabilidades limites dos estados sob controle das políticas de decisão consideradas.

4.1.1 *Hardware* e *software*

Toda a experimentação computacional foi realizada em um microcomputador comum, com processador *Intel Core 2 Duo* e 2 GB de memória RAM.

A linguagem de programação utilizada foi a linguagem C (HARBISON; STEELE, 2002). Utilizou-se o compilador Dev-C++, também conhecido como Dev-Cpp. O Dev-C++ é um ambiente de desenvolvimento integrado livre que utiliza os compiladores do projeto GNU para compilar programas para o

sistema operacional *Microsoft Windows*. Esse compilador suporta as linguagens de programação C e C++ e possui toda a biblioteca ANSI C, além de algumas bibliotecas similares às da Borland Turbo C.

O *software* SPSS 13.0 *for Windows* foi utilizado para se obterem os resultados do método das k-médias, empregado na Subseção 4.3.3 para determinação do número de padrões de atendimento. Os *softwares Excel* 2003 e *Word* 2003, do *Microsoft Office*, foram utilizados para elaboração das tabelas, dos gráficos e do texto em geral.

4.1.2 Simulação

Em uma definição generalista, simulação consiste em empregar técnicas matemáticas associadas a técnicas de processamento computacional com o propósito de imitar um processo ou uma operação do mundo real. A simulação também é aplicada à aproximação numérica de equações complexas. Neste estudo, o termo simulação é empregado com o sentido particular da simulação estocástica, ou simulação de Monte Carlo, em que a simulação é empregada para gerar realizações da dinâmica de um sistema formulado como um PMD. Especificamente, o que se pretende aqui é simplesmente gerar o próximo estado do sistema, dado um estado observado e uma ação adotada, de acordo com a distribuição de probabilidades de transição entre os estados determinada pelo par (estado; ação).

Considerando que não é o foco deste estudo explorar os diversos aspectos da simulação em profundidade, não serão abordados importantes tópicos da área, tais como geração de números aleatórios, geração de variáveis aleatórias, análise de *input* e *output*, verificação, validação e técnicas de redução de variância. Para os leitores interessados em pesquisar esses temas, indicam-se os abrangentes estudos de Fishman (2005) e de Law e Kelton (2000).

A simulação de Monte Carlo toma sequências de números aleatórios como base elementar para reproduzir a dinâmica de funcionamento de um sistema. Uma sequência de números aleatórios é, por definição, uma sequência de variáveis aleatórias independentes e identicamente distribuídas no intervalo entre zero e um de acordo com a distribuição de probabilidades uniforme, representada por $U[0,1]$. Partindo-se de números aleatórios gerados de acordo com a distribuição $U[0,1]$, através de métodos específicos, podem-se obter valores para variáveis aleatórias com distribuições de probabilidade diversas. Neste estudo, emprega-se o "método direto", descrito a seguir, para gerar valores de uma variável aleatória com distribuição de probabilidade multinomial (DAVIS, 1993).

Seja $D = \{D_1,...,D_k\}$ um conjunto de k variáveis aleatórias com distribuição de probabilidades multinomial, $M_k(N, p)$, em que N é um número inteiro positivo e $p = \{p_1,..., p_k\}$ é um conjunto de probabilidades, sendo $p_i > 0$ para todo $i \in \{1,...,k\}$ e $\sum_{i=1}^{k} p_i = 1$, segue que a distribuição de probabilidades de D pode ser calculada pela expressão

$$\Pr(D_1 = n_1,..., D_k = n_k) = \frac{N!}{n_1!...n_k!} p_1^{n_1}...p_k^{n_k},$$

em que $n_1,...,n_k$ são inteiros não negativos e satisfazem a condição $\sum_{i=1}^{k} n_i = N$.

Os passos do método direto para se gerar valores de $D \sim M_k(N, p)$ são:

1) Monta-se um vetor p^* com k posições para acumular as probabilidades, de forma que $p_i^* = \sum_{j=1}^{i} p_j$ para $i = 1,...,k$.

2) Iniciam-se com valor zero os correspondentes $D_1,...,D_k$ pertencentes a D.

3) Gera-se uma sequência de N números aleatórios, u_l, $l = 1,...,N$, independentes com distribuição $U[0,1]$.

4) Para $l = 1,...,N$, o $i-ésimo$ componente de D deve ser acrescentado de 1 se $p^*_{i-1} < u_l \leq p^*_i$, sendo $p^*_0 = 0$.

Neste estudo, emprega-se a função rand(), geradora de números pseudoaleatórios, pertencente à biblioteca padrão “stdlib.h” da linguagem C, para gerar números aleatórios com distribuição $U[0, RAND\_MAX]$, sendo RAND_MAX uma constante definida que pode variar entre os diferentes compiladores. Para o compilador DEV-C++ a constante RAND_MAX é igual a 32.767. Emprega-se, também, a função srand((unsigned)time(0)), que gera, quando necessário, uma nova semente para a função rand() cujo valor é equivalente ao instante de tempo registrado pelo sistema do computador. O valor para um número aleatório u com distribuição de probabilidades $U[0,1]$ pode ser obtido através da expressão

$$u = \frac{rand()}{RAND\_MAX}.$$

4.1.3 Multiplicação e armazenamento de matrizes esparsas

As probabilidades limites para cada um dos estados de um sistema formulado como um PMD, sob controle de uma política de decisão predeterminada, podem ser obtidas através de um método logicamente simples (ver Capítulo 2, Subseção 2.1.1). Esse método consiste em multiplicar a matriz de probabilidades de transição entre estados por ela mesma até que a soma dos elementos de sua diagonal convirja para 1, a menos de um erro tão pequeno quanto o desejado. Nesse processo, frequentemente é necessário armazenar e realizar multiplicações entre matrizes esparsas.

Uma matriz é dita esparsa quando a maioria de seus elementos é igual a zero. Espaço de memória e tempo de processamento são economizados quando apenas os elementos não nulos da matriz esparsa são armazenados e entram nas operações. Neste estudo, aplica-se a técnica de armazenamento de matrizes esparsas conhecida como *row-indexed sparse storage mode* (PRESS *et al.*, 2007).

Nos apêndices A.1 e A.2 apresentam-se, respectivamente, um pseudocódigo para se armazenar matrizes esparsas quadradas e um pseudocódigo para multiplicar duas matrizes quadradas esparsas.

4.2 Avaliação do método EHRA

Nesta seção apresenta-se a implementação de um modelo hipotético pequeno para o controle de admissões de pacientes em um hospital eletivo (ver Capítulo 2). O objetivo é testar os desempenhos das políticas obtidas através do método EHRA comparado-os ao desempenho da política ótima e de duas políticas gulosas que são empregadas como políticas de base no método EHRA. Avalia-se, também, a capacidade do método EHRA para abordar instâncias do modelo com espaço de estados e de ações maiores.

4.2.1 Configuração do modelo

Nas tabelas 4.1 e 4.2, apresenta-se a configuração do modelo que será empregada na realização dos testes do método EHRA. Para que o desempenho das políticas comparadas pudesse ser verificado, os parâmetros do modelo foram escolhidos com diferenciação nos custos de desvio, no consumo esperado médio dos recursos e nas probabilidades de transição e de entrada nos padrões de atendimento.

O modelo para teste (Tabela 4.1) consiste de duas especialidades, dois recursos hospitalares e três padrões de atendimento. A capacidade máxima de admissão de pacientes por período é de dois pacientes para cada especialidade. A capacidade disponibilizada por período para cada um dos dois recursos é de cinco unidades. O nível desejado de utilização de cada um dos recursos é de quatro unidades por período.

Tabela 4.1 - Parâmetros do modelo para teste.

Nº de especialidades	**2**											
Capacidade máxima de admissões												
Especialidade 1	2											
Especialidade 2	2											
Nº de recursos	**2**											
Capacidade disponibilizada dos recursos												
Recurso 1	5											
Recurso 2	5											
Nível de utilização												
Nível de utilização desejado para cada recurso												
Nível do rec. 1	4											
Nível do rec. 2	4											
Custos												
Custo do desvio relativo ao nível de utilização desejado												
	Rec. 1	Rec. 2										
Custo de ociosidade	4,5	1,0										
Custo de exces.: nível	1,5	1,0										
Custo de exces.: capacidade	1,0	1,0										
Nº de padrões de atendimento	**3**											
Consumo médio de recursos em cada padrão												
	Configuração 1		Configuração 2		Configuração 3		Configuração 4		Configuração 5		Configuração 6	
	Rec. 1	Rec. 2	Rec. 1	Rec. 2	Rec. 1	Rec. 2	Rec. 1	Rec. 2	Rec. 1	Rec. 2	Rec. 1	Rec. 2
Padrão 1	1,85	2,25	1,90	2,30	1,95	2,35	2,00	2,40	2,05	2,45	2,10	2,50
Padrão 2	2,25	1,85	2,30	1,90	2,35	1,95	2,40	2,00	2,45	2,05	2,50	2,10
Padrão 3 (alta)	-	-	-	-	-	-	-	-	-	-	-	-
Nº de Estados	**9.774**		**8.166**		**8.166**		**7.626**		**7.206**		**6.756**	

Para os custos associados aos desvios em relação ao nível desejado de utilização por período, consideram-se quantitativos adimensionais para controlar a importância relativa dos recursos. Cada unidade de desvio menor

do que o nível de utilização desejado para o recurso L_1 possui um custo de ociosidade de 4,5, e cada unidade de desvio maior do que o nível de utilização desejado tem um custo de excesso de 1,5. Se for necessário utilizar mais recurso do que o disponibilizado para o período, considera-se que o hospital providenciará o recurso a um custo excedente. Assim, para o recurso L_1, cada unidade de desvio acima da capacidade disponibilizada do recurso possui um custo adicional de 1,0. Considerando-se o recurso L_2, temos os seguintes custos: custo de ociosidade 1,0, custo de excesso sobre o nível de utilização desejado 1,0 e custo de excesso sobre a capacidade disponível 1,0.

Com o objetivo de comparar o desempenho das políticas em diferentes instâncias do modelo, foram consideradas seis configurações de padrões de atendimento distintas. Partindo da Configuração 1 até a Configuração 6, aumentou-se o consumo de recursos em 0,05 unidades a cada configuração. Dessa forma, à medida que o consumo esperado de recursos em cada padrão aumenta, reduz-se o número de estados possíveis do espaço de estados. Por exemplo, na Configuração 1 o consumo médio esperado de um paciente que passa um período no padrão de atendimento E_1 é de 1,85 unidades do recurso L_1 e 2,25 unidades do recurso L_2, enquanto que um paciente que passa um período no padrão de atendimento E_2 apresenta um consumo médio esperado de 2,25 unidades do recurso L_1 e 1,85 unidades do recurso L_2. O espaço de estados da Configuração 1 é composto por 9.774 possíveis estados, ao passo que a Configuração 6 apresenta um espaço de estados composto por 6.756 possíveis estados. Observa-se que não ocorre consumo de recursos no padrão de atendimento E_3, o qual representa a alta hospitalar.

Tabela 4.2 - Probabilidades do modelo para teste.

Probabilidades de transições entre os padrões de tratamento				
		Pad. 1	Pad. 2	Pad. 3
Especialidade 1				
	Padrão 1	0,4	0,1	0,5
	Padrão 2	0,1	0,3	0,6
	Padrão 3	0,0	0,0	1,0
Especialidade 2				
	Padrão 1	0,2	0,1	0,7
	Padrão 2	0,1	0,2	0,7
	Padrão 3	0,0	0,0	1,0

Probabilidades de entrada nos padrões para cada especialidade		
	Espec. 1	Espec. 2
Padrão 1	0,50	0,40
Padrão 2	0,50	0,60
Padrão 3	0,00	0,00

Na Tabela 4.2, apresentam-se as probabilidades de transição entre os padrões de atendimento para os pacientes de cada especialidade e, para os pacientes admitidos no próximo período, as probabilidades de entrada em cada padrão de atendimento. O Padrão de Atendimento 3 representa a alta hospitalar, sendo um estado absorvente nas matrizes de probabilidades de transição das duas especialidades. Os pacientes da Especialidade 2 afluem mais rapidamente para a alta hospitalar e, quando admitidos, têm maior probabilidade de iniciarem o atendimento no Padrão de Atendimento 2. Os pacientes admitidos não podem ingressar diretamente no padrão correspondente à alta, por isso a probabilidade de entrada no Padrão de Atendimento 3 é igual a zero para as duas especialidades.

4.2.2 Políticas de controle

O AIV foi aplicado para obtenção de uma política ótima (PO) para o controle do modelo de teste, considerando-se a minimização do custo descontado a um horizonte de planejamento infinito (ver Capítulo 3). Além da PO, foram elaboradas duas políticas gulosas, a gulosa 1 (G1) e a gulosa 2 (G2).

O controle da política G1 prescreve que, dado um estado observado $x_t \in X$ em um instante t, deve-se escolher sempre a ação que atinja o menor $R(x_t, a)$ – custo esperado até o próximo instante de decisão (ver Capítulo 2) –. Ou seja, escolhe-se a ação a_{G1} tal que

$$a_{G1} = \arg\min_{a \in A(x_t)} R(x_t, a).$$

Para se determinar a ação prescrita pela política gulosa G2, dado um estado observado $x_t \in X$ em um instante t, definem-se as três funções descritas a seguir:

(1) A função $N_{E_{i,(x_t,a)}}$, $i = 1...n$, calcula o número esperado de pacientes em cada um dos n padrões de atendimento no próximo período $t+1$, dado que a ação $a \in A(x_t)$ foi adotada no período t, sendo

$$N_{E_{i,(x_t,a)}} = \sum_{d=1}^{m} \sum_{j=1}^{n} p_{ji}^{d} E_{j,x_t}^{d} + \sum_{d=1}^{m} p_i^d S_a^d ,$$

em que E_{j,x_t}^{d} representa o número de pacientes da especialidade d presentes no padrão de atendimento j no instante t de decisão, p_{ji}^{d} representa a probabilidade de um paciente da especialidade d passar do padrão de atendimento E_j para o padrão de atendimento E_i no próximo período, p_i^d representa a probabilidade de um paciente da especialidade d ser admitido em tratamento no padrão E_i, e S_a^d representa o número de pacientes da

especialidade d a serem admitidos no próximo período segundo a prescrição da ação a.

(2) A função $D_{L_{j,(x_t,a)}}$ calcula o consumo esperado no próximo período $t+1$ de cada um dos k recursos L_j, $j=1,...,k$, dado que a ação $a \in A(x_t)$ foi adotada no período t, sendo

$$D_{L_{j,(x_t,a)}} = \sum_{i=1}^{n} N_{E_{i,t+1}} L_{ij} \ ,$$

em que L_{ij} representa o consumo médio esperado do recurso L_j realizado por um paciente que passa um período no padrão de atendimento E_i.

(3) A função $R^{G2}_{(x_t,a)}$ calcula o valor da ação $a \in A(x_t)$ considerando $D_{L_{j,(x_t,a)}}$, sendo

$$R^{G2}_{(x_t,a)} = \sum_{j=1}^{k} \left\{ \begin{array}{l} O_j \times \max(N_j - D_{L_{j,(x_t,a)}};0) + \\ B_j \times \max(D_{L_{j,(x_t,a)}} - N_j;0) + \\ C_j \times \max(D_{L_{j,(x_t,a)}} - \max L_j;0) \end{array} \right\},$$

em que $\max L_j$ representa a quantidade do recurso L_j disponibilizada em um período, N_j representa o nível de utilização desejado do recurso L_j, O_j representa o custo de ociosidade de uma unidade do recurso L_j em relação à N_j, B_j representa o custo de excesso de uma unidade do recurso L_j em relação à N_j e C_j representa o custo de excesso sobre $\max L_j$.

Apresenta-se, a seguir, um pseudocódigo para obtenção da ação a_{G2}, dado um estado observado $x_t \in X$ em um instante de decisão t:

montar a ação inicial $a_0 = \{S_0^1,...,S_0^m\}$, tal que $S_0^d = 0$, para $d = 1,...,m$

$i = 1$

(0) repetir enquanto $i > 0$

$i = 0$

$aux = R^{G2}_{(x_t, a_0)}$

(1) para o contador $d = 1$ até m, fazer

(2) se $S_0^d + 1 \leq$ "*capacidade máxima de admissão da especialidade d*", fazer

montar uma nova ação $a_1 = \{S_1^1, ..., S_1^m\}$ tal que, para $j = 1, ..., m$,

se $j \neq d$ então $S_1^j = S_0^j$ e se $j = d$ então $S_1^j = S_0^j + 1$

$aux1 = R^{G2}_{(x_t, a_1)}$

(3) se $aux1 < aux$, fazer

$aux = aux1$

$i = d$

(3) finalizar

(2) finalizar

(1) finalizar

(1) se $i > 0$, fazer

atualizar a ação a_0, fazendo $S_0^i = S_0^i + 1$

(1) finalizar

(0) finalizar

$a_{G2} = a_0$

Em resumo, a política G2 parte de uma ação em que não se admitem pacientes, testa o acréscimo de um paciente a cada especialidade e soma um paciente à especialidade que apresenta o menor valor associado; esse procedimento se repete até que não seja vantajoso acrescentarem-se pacientes. O valor de comparação do acréscimo de um paciente é determinado em função do número esperado de pacientes em cada padrão de atendimento e do correspondente consumo esperado de cada recurso.

O método EHRA foi aplicado para gerar três políticas de controle: a política EHRA1, que emprega como política de base apenas a política gulosa G1; a

política EHRA2, que emprega como política de base apenas a G2; e a política EHRA12, que emprega como políticas de base a G1 e a G2. Os parâmetros de entrada do algoritmo (ver Capítulo 3) para a obtenção das políticas EHRA foram: o tamanho da população de ações igual a 5 ações, o tamanho do horizonte de planejamento reduzido igual a 20 períodos, o número de amostras de realizações de funcionamento do sistema para cada política de base igual a 30 horizontes reduzidos, o fator de desconto igual a 0,8, a probabilidade para escolha de uma ação aleatoriamente no espaço de ações igual a 0,1, a probabilidade para escolha de uma ação na vizinhança da ação de elite igual a 0,9, e o número de repetições da ação de elite para o critério de parada igual a 10.

Tabela 4.3 - Decisões da política ótima, políticas gulosa 1, gulosa 2 e políticas EHRA1, EHRA2 e EHRA12, para cinco estados selecionados.

Estados [a]: $\{E_1^1, E_2^1, E_3^1, E_1^2, E_2^2, E_3^2\}$	Pacientes atendidos no último período: no Hospital	por Espec. 1	por Espec. 2	por Padrão E_1	por Padrão E_2	PO [b] (S^1, S^2)	G1 [b] (S^1, S^2)	G2 [b] (S^1, S^2)	EHRA1 [b] (S^1, S^2)	EHRA2 [b] (S^1, S^2)	EHRA12 [b] (S^1, S^2)
{0,1,1,0,0,3}	1	1	0	0	1	(1,1)	(0,2)	(2,0)	(1,1)	(2,0)	(1,1)
{0,0,0,0,2,3}	2	0	2	0	2	(0,1)	(1,1)	(0,1)	(2,0)	(0,1)	(0,1)
{0,1,1,1,1,2}	3	1	2	1	2	(0,1)	(0,1)	(1,0)	(0,1)	(1,0)	(1,0)
{1,0,3,1,1,2}	3	1	2	2	1	(0,1)	(0,1)	(1,0)	(0,1)	(1,0)	(0,1)
{1,2,0,1,0,1}	4	3	1	1	3	(0,0)	(0,1)	(0,0)	(0,1)	(0,0)	(0,0)

a – Estado: número de pacientes de cada especialidade atendidos em cada padrão de atendimento no último período. O padrão de atendimento E_3 corresponde à alta hospitalar.
b – Decisão da política: número de pacientes de cada especialidade a ser admitido no próximo período.

Exemplificando a estrutura dos estados e as decisões determinadas pelas políticas PO, G1, G2, EHRA1, EHRA2 e EHRA12, apresentam-se, na Tabela 4.3, as ações de cada política para cinco estados selecionados do espaço de estados, considerando-se o modelo constituído pelos parâmetros apresentados nas tabelas 4.1 e 4.2, com a Configuração 1 para o consumo médio de recursos dos padrões de atendimento. Também são apresentados, para cada

estado selecionado, os correspondentes quantitativos de pacientes atendidos no último período, distribuídos por especialidade e por padrão de atendimento. Por exemplo, na primeira linha da Tabela 4.3 (linha sombreada), o estado indica que, no último período, um paciente da Especialidade 1 e três pacientes da Especialidade 2 receberam alta hospitalar, e um paciente da Especialidade 1 esteve em atendimento no Padrão de Atendimento 2. Para esse estado, a PO determina a admissão de um paciente de cada especialidade, a G1 determina a admissão de dois pacientes da Especialidade 2, a G2 determina a admissão de dois pacientes da Especialidade 1, a EHRA1 e a EHRA12 concordam com a PO, e a EHRA2 concorda com a G2.

4.2.3 Parâmetros de desempenho

Os desempenhos das políticas de controle geradas pelo método EHRA foram comparados ao desempenho da PO e das políticas gulosas G1 e G2 através da avaliação de parâmetros calculados com base nas probabilidades limites dos estados do modelo sob controle de cada política (ver Seção 3.1 do Capítulo 3), da avaliação das estimativas dos custos médios esperados de cada política e da avaliação da proporção de concordâncias entre as ações prescritas pela PO e as ações prescritas pelas políticas EHRA1, EHRA2, EHRA12, G1 e G2.

Observa-se que, embora as ações das políticas EHRA tenham sido obtidas através do processo iterativo deste método, que emprega os custos descontados ao longo do horizonte de planejamento reduzido (ver Seção 3.6 do Capítulo 3), optou-se por se comparar os custos médios de cada política ao longo do horizonte de planejamento infinito, obtidos através do AIV. Para se estimar o custo médio de cada política (EHRA1, EHRA2, EHRA12 e também as gulosas G1 e G2), após a determinação de uma ação para cada estado, manteve-se fixa a política de decisão assim elaborada e fez-se o AIV iterar até

a convergência para o custo médio esperado ao longo do horizonte de planejamento infinito da política (considerando-se um erro máximo de +/- 10^{-6}, estabelecido para o critério de parada do AIV).

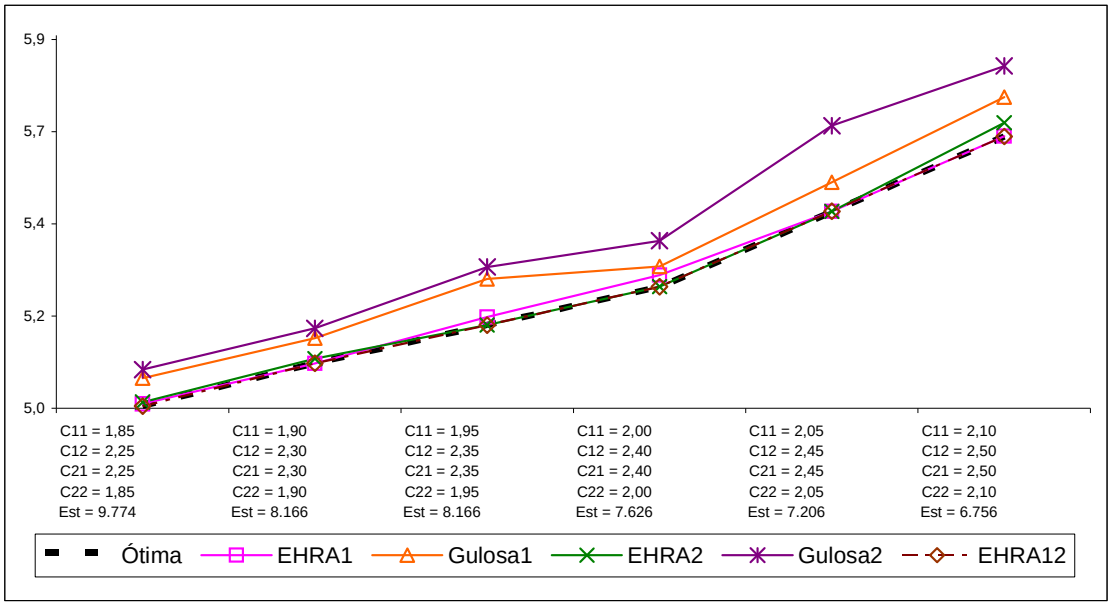
Os parâmetros de desempenho avaliados, baseados nas probabilidades limites (sistema em equilíbrio após longo tempo de funcionamento (CHI, 1996)), foram os seguintes: o consumo esperado dos recursos, o número esperado de pacientes nos padrões de atendimento, o número esperado de pacientes em tratamento de cada especialidade e o número esperado de admissões de pacientes de cada especialidade. Para obtê-los, os quantitativos correspondentes aos parâmetros em cada estado foram multiplicados pela respectiva probabilidade limite, e os resultados foram acumulados para todos os estados. As probabilidades limites foram obtidas da seguinte forma: primeiramente foram preparadas as cadeias de Markov de probabilidades de transição entre estados associadas a cada uma das políticas; em seguida, a matriz de probabilidades de transição de cada política foi multiplicada por si própria até que a soma da diagonal convergisse para 1, a menos de um erro (erro empregado, 10^{-6}).

As concordâncias das ações em relação à PO foram verificadas diretamente, estado a estado, após a preparação de cada política. A preparação (determinação de uma ação para cada estado) das políticas EHRA1, EHRA2 e EHRA12 foi realizada através da execução do algoritmo EHRA para cada estado para cada uma das políticas. As políticas gulosas G1 e G2 foram preparadas prontamente como parte do processo de obtenção das políticas EHRA.

4.2.4 Comparação dos parâmetros de desempenho

Apresentam-se, a seguir, através de representações gráficas, os resultados do desempenho das políticas obtidas pelo método EHRA comparados aos resultados da PO, obtida pelo AIV, e das políticas gulosas G1 e G2. As instâncias distintas do modelo foram obtidas com o aumento gradual do consumo dos recursos em cada padrão de atendimento, partindo da Configuração 1 (ver Tabela 4.1), com 9.744 estados, até a Configuração 6, com 6.756 estados.

Gráfico 4.1 - Custo médio esperado por período sob controle de cada política.



Observa-se, no Gráfico 4.1, que o custo médio esperado por período, considerando o sistema sob controle de cada política em um horizonte de planejamento infinito, é menor para as políticas EHRA do que para as políticas gulosas em todas as instâncias avaliadas. As políticas EHRA tiveram desempenho bem próximo ao da PO, enquanto que a política G2 apresentou o pior desempenho, seguida pela G1. Entre as políticas EHRA, a política EHRA12 registrou o desempenho mais consistente nas seis configurações: sua curva no Gráfico 4.1 praticamente se confunde com a do resultado ótimo. A

política EHRA12 apresentou a maior proximidade em relação ao custo médio esperado ótimo em cinco configurações. Apenas na Configuração 5 a política EHRA2 apresentou o custo médio esperado mais próximo ao da PO.

Considerando o consumo médio esperado dos recursos por período (gráficos 4.2 e 4.3), observa-se que, com o modelo em equilíbrio, após longo período em funcionamento, todas as políticas mantêm o consumo dos recursos em um patamar acima de 4 unidades, nível desejado para ambos os recursos, variando entre 4,02 unidades e 4,59 unidades. Isso ocorre como consequência de os custos de ociosidade estabelecidos serem maiores do que os custos de excesso acima dos níveis de utilização desejados e excesso acima da quantidade de recursos disponibilizada. As políticas gulosas, que consideram em suas ações apenas o período até o próximo instante de decisão, apresentaram um padrão de atendimento mais afastado em relação ao padrão da PO. Enquanto na G1 o consumo dos recursos é sempre superior ao da PO, a G2 apresenta um comportamento mais variável, mantendo o consumo dos recursos ora acima, ora abaixo em relação ao da PO. As políticas EHRA acompanharam o padrão de atendimento da PO, sendo que a EHRA12 apresentou consumo praticamente idêntico ao da PO em todas as configurações, a EHRA2 apresentou consumo muito próximo ao da PO nas configurações 2, 3, 4 e 5 e a EHRA1 teve consumo similar ao da PO nas configurações 5 e 6.

Observa-se que manter o consumo esperado dos recursos mais próximo do nível de utilização desejado nem sempre implica incorrer em um menor custo de operação em um longo prazo. Por exemplo, a política G2 apresentou uma maior proximidade de consumo esperado dos recursos em relação aos níveis de utilização desejados nas configurações 1 e 6, porém o seu custo médio esperado por período foi sempre o mais oneroso (Gráfico 4.1). O custo de uma política é determinado pela combinação entre a proximidade do nível de utilização desejado, a especialidade dos pacientes mantidos em atendimento, a

quantidade de pacientes mantida em cada padrão de atendimento e o custo dos desvios em relação ao nível de utilização desejado.

Gráfico 4.2 - Consumo esperado do Recurso 1 sob controle de cada política.

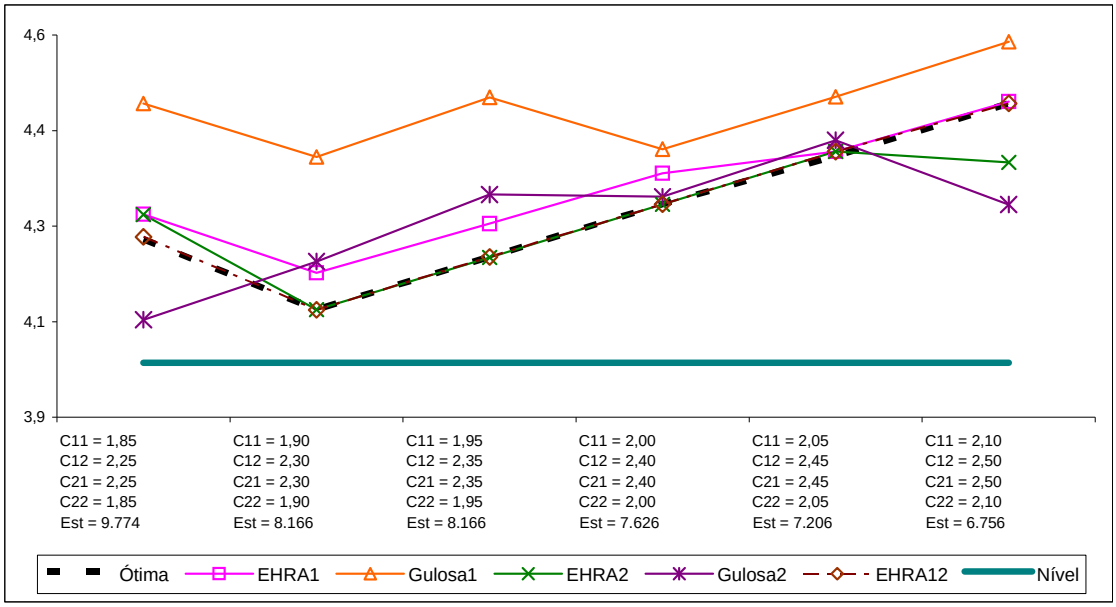
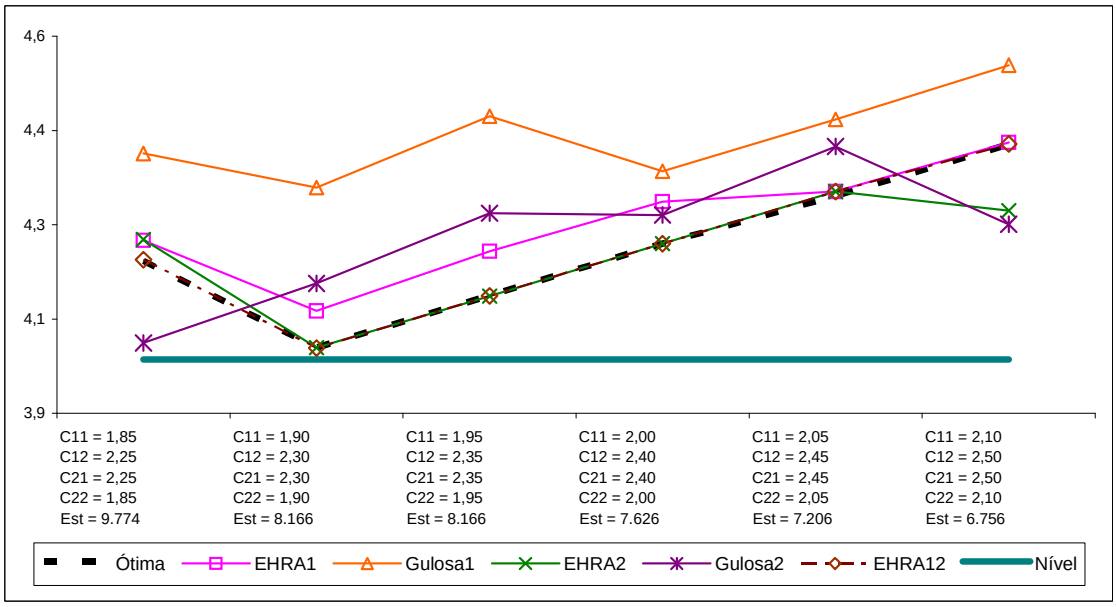


Gráfico 4.3 - Consumo esperado do Recurso 2 sob controle de cada política.



Considerando o número médio esperado por período de pacientes em cada padrão de atendimento após um longo período de funcionamento do sistema (gráficos 4.1 e 4.5), todas as políticas mantêm um número maior de pacientes no Padrão de Atendimento 2. Após a Configuração 1, em que a PO mantém em média 0,97 pacientes por período no Padrão 1, a PO apresenta uma constância nas demais configurações, mantendo, em média, 0,87 pacientes no Padrão 1; em relação ao Padrão 2, a PO mantém uma média em torno de 1,06 pacientes em todas as configurações. As políticas gulosas apresentaram comportamento sempre mais distante da PO quando comparadas às políticas EHRA. As políticas EHRA acompanharam a PO, sendo que a EHRA12 apresentou comportamento praticamente idêntico ao da PO em todas as configurações, a EHRA2 foi similar à PO nas configurações 2, 3, 4 e 5 e a EHRA1 se aproximou mais da PO nas configurações 5 e 6.

Gráfico 4.4 - Número esperado de pacientes no Padrão 1.

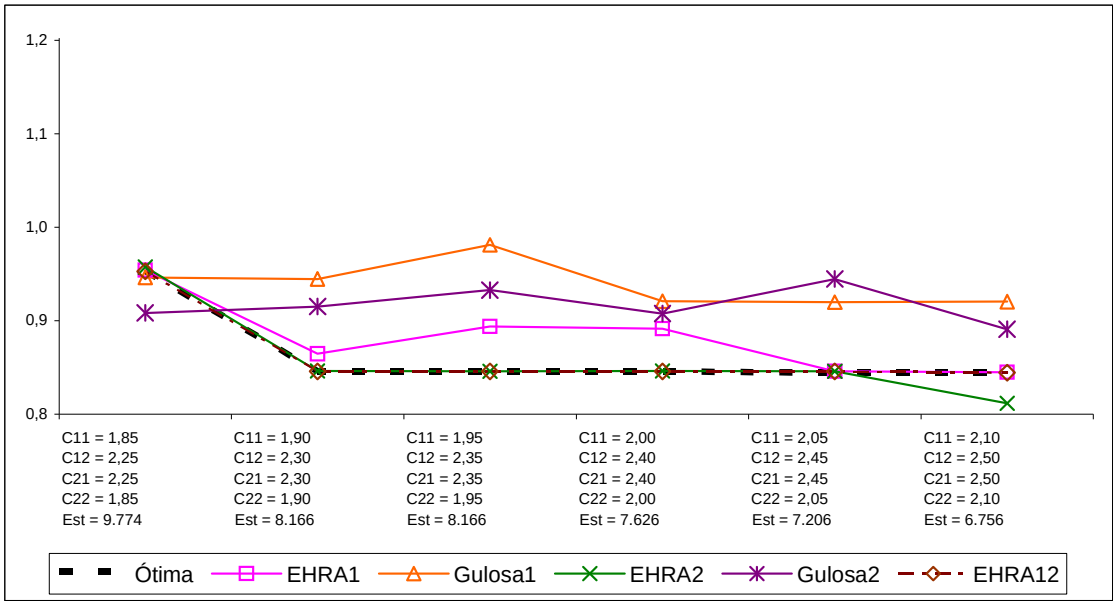
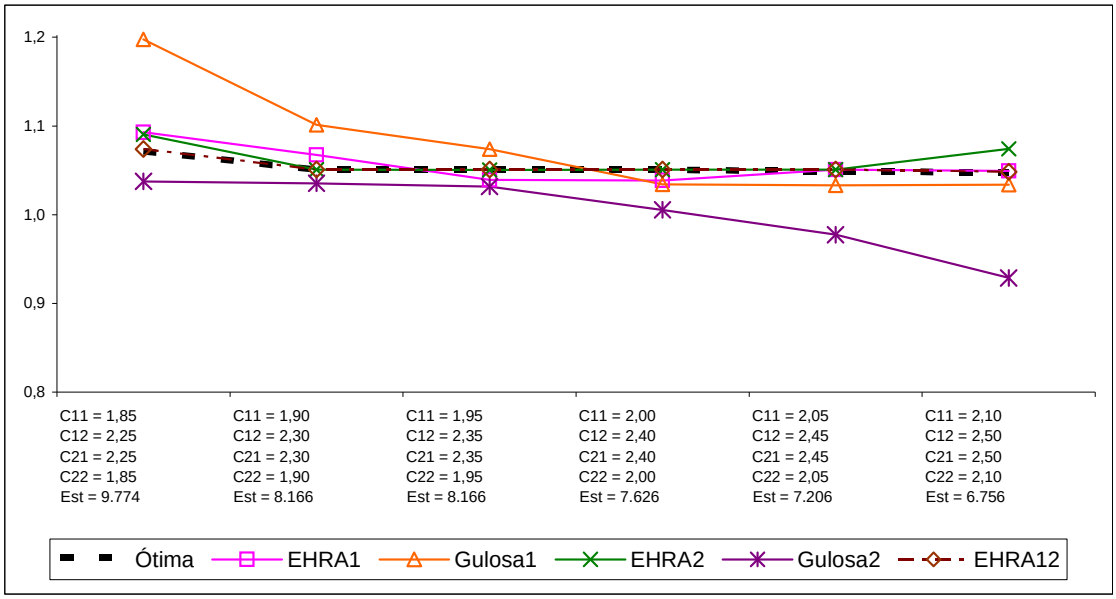


Gráfico 4.5 - Número esperado de pacientes no Padrão 2.



Considerando o número médio esperado de admissões de pacientes por período com o modelo em equilíbrio, após um longo período de controle das políticas (gráficos 4.6, 4.7 e 4.8), observa-se que a PO e as políticas EHRA mantêm uma média de admissões entre 1,3 e 1,4 pacientes por período em todas as configurações, sendo as admissões de pacientes da Especialidade 2, com média de 0,89 admissões por período, sempre em maior número do que as admissões de pacientes da Especialidade 1, com média de 0,38 admissões por período. A política gulosa G1 é a que mais admite pacientes nas configurações 1, 2 e 3, sendo os pacientes da Especialidade 2 admitidos em maior quantidade. A política gulosa G2, que admite mais pacientes da Especialidade 1 nas configurações 2, 4, 5 e 6, é a mais restritiva, admitindo menos pacientes em todas as configurações. Observa-se que as políticas EHRA mantiveram um padrão de admissão mais próximo ao da PO, sendo que a EHRA12 apresentou praticamente a mesma curva de admissões que a PO em todas as configurações, a EHRA1 foi similar à PO nas configurações 5 e 6 e a EHRA2 foi próxima à PO nas configurações 2, 3, 4 e 5.

Gráfico 4.6 - Admissões esperadas de pacientes da Especialidade 1.

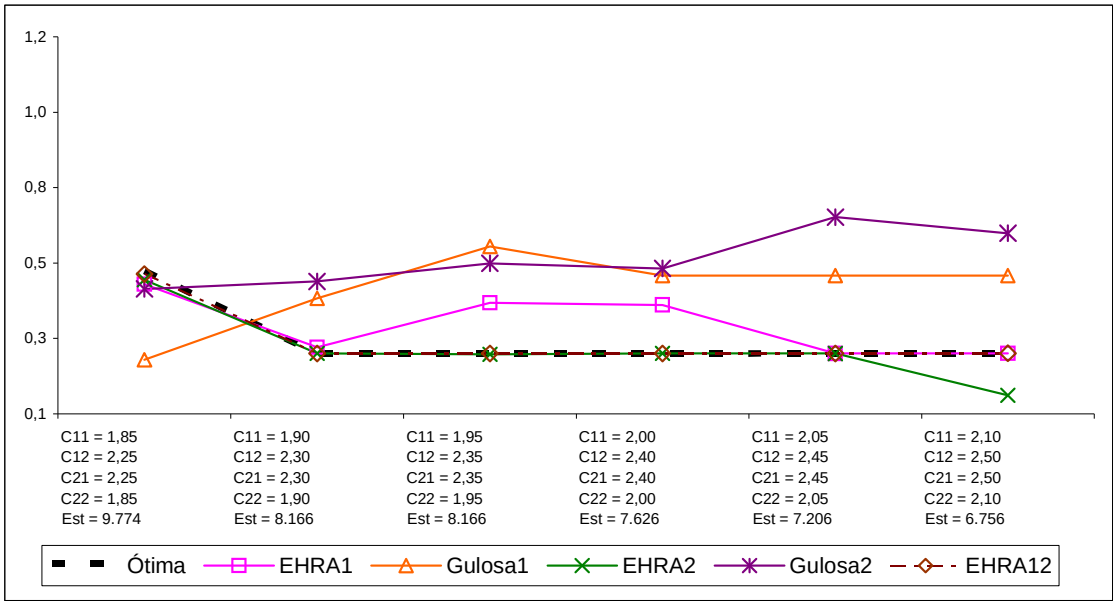


Gráfico 4.7 - Admissões esperadas da Especialidade 2.

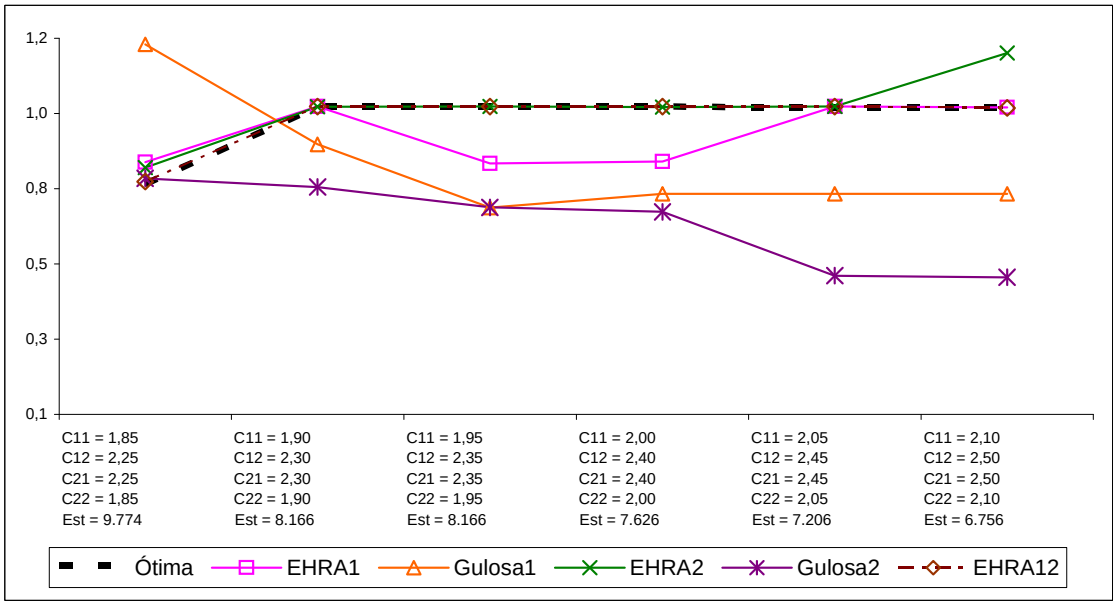
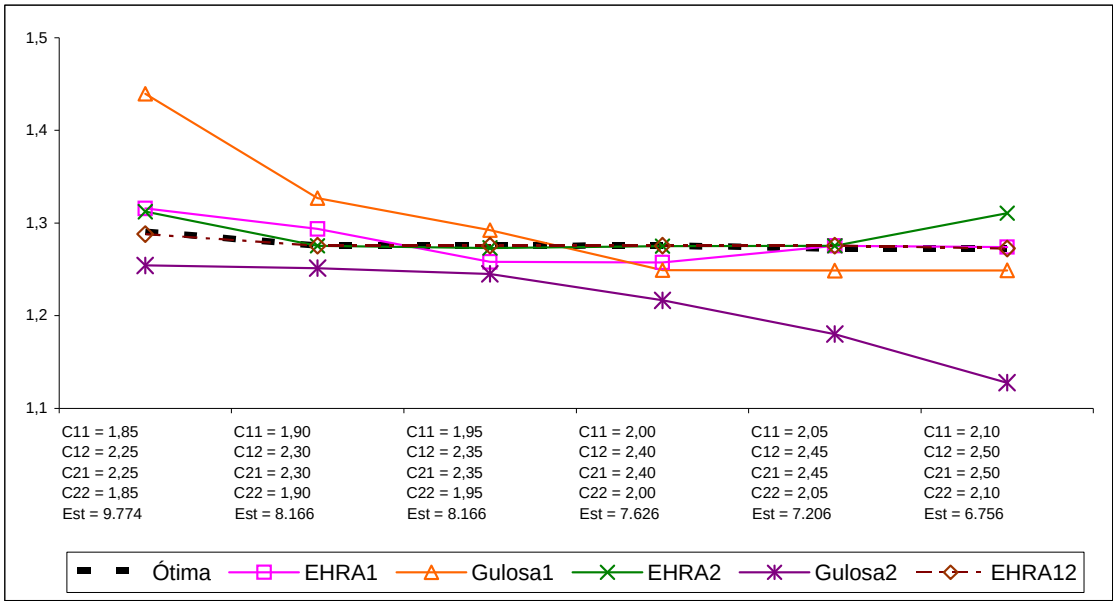


Gráfico 4.8 - Número esperado de admissões de pacientes.



Considerando a média esperada de pacientes em atendimento, após longo tempo de funcionamento do modelo sob controle de cada política (gráficos 4.9, 4.10 e 4.11), observa-se que a PO na Configuração 1 mantém, em média, 2,05 pacientes em atendimento, sendo a média reduzida e mantida no patamar de 1,93 pacientes em atendimento nas demais configurações. A PO mantém em atendimento um quantitativo maior de pacientes da Especialidade 2. A política gulosa G1 mantém maior número médio de pacientes em atendimento do que o da PO em todas as configurações, sendo também os pacientes da Especialidade 2 os que permanecem em atendimento em maior número. A política gulosa G2 mantém em atendimento um número menor de pacientes da Especialidade 2, sendo que nas configurações 1 e 6 o número esperado de pacientes em atendimento das duas especialidades é menor do que o apresentado pela PO. Esta situação se inverte nas configurações de 2 a 5, nas quais a G2 mantém mais pacientes em atendimento do que a PO. As políticas EHRA em geral seguiram o comportamento da PO, sendo que a política EHRA12 apresentou um comportamento praticamente idêntico ao da PO, a política EHRA1 foi mais próxima da PO nas configurações 5 e 6 e a política EHRA2 foi similar à PO nas configurações 2, 3, 4 e 5.

Gráfico 4.9 - Número esperado de pacientes da Espec. 1 em atendimento.

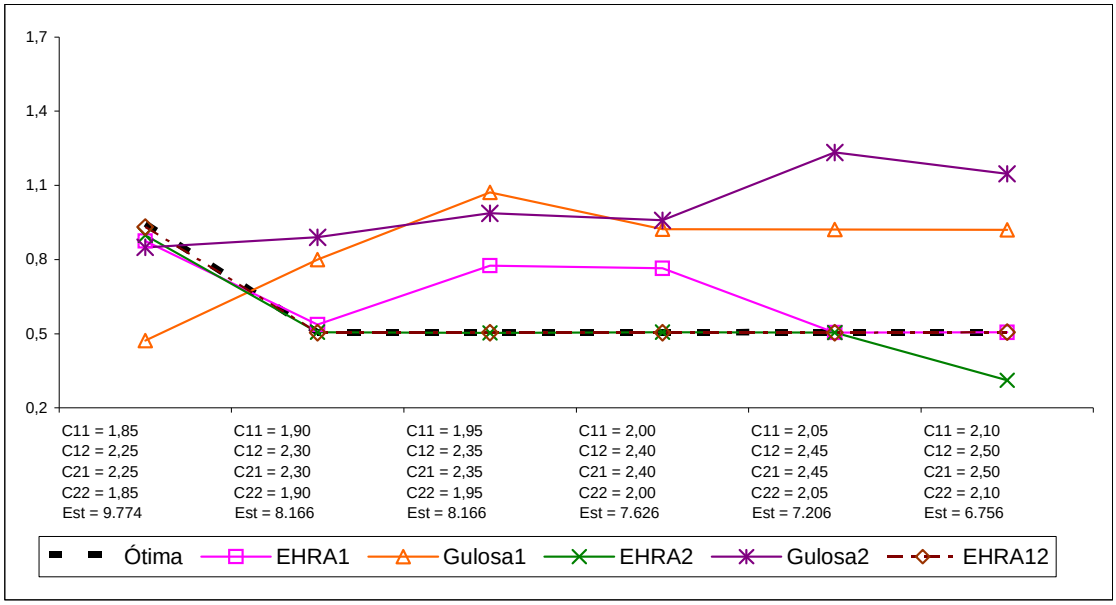


Gráfico 4.10 - Número esperado de pacientes da Espec. 2 em atendimento.

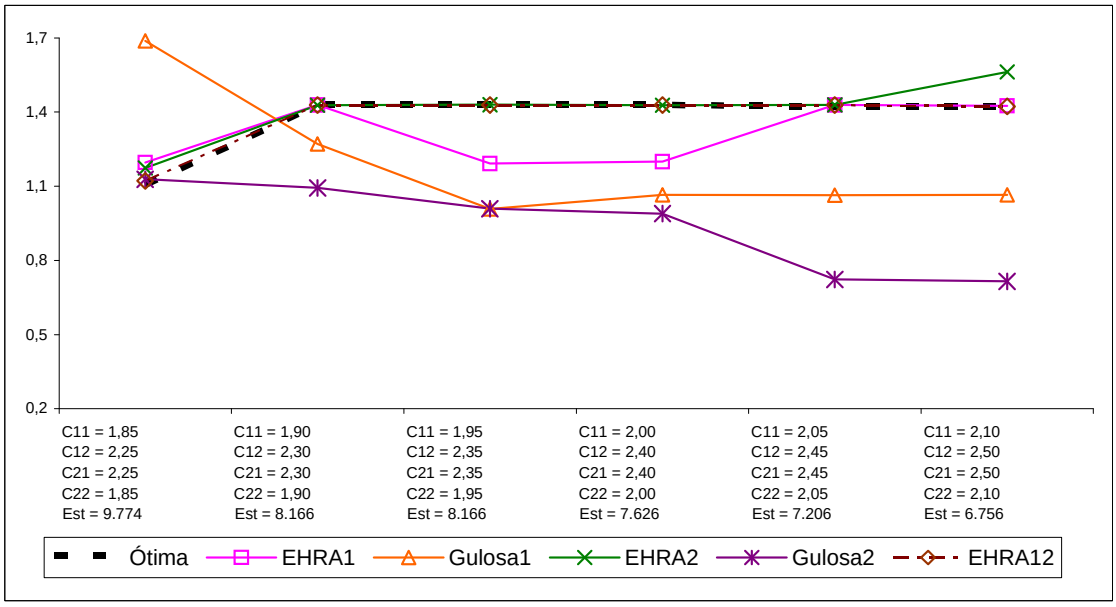
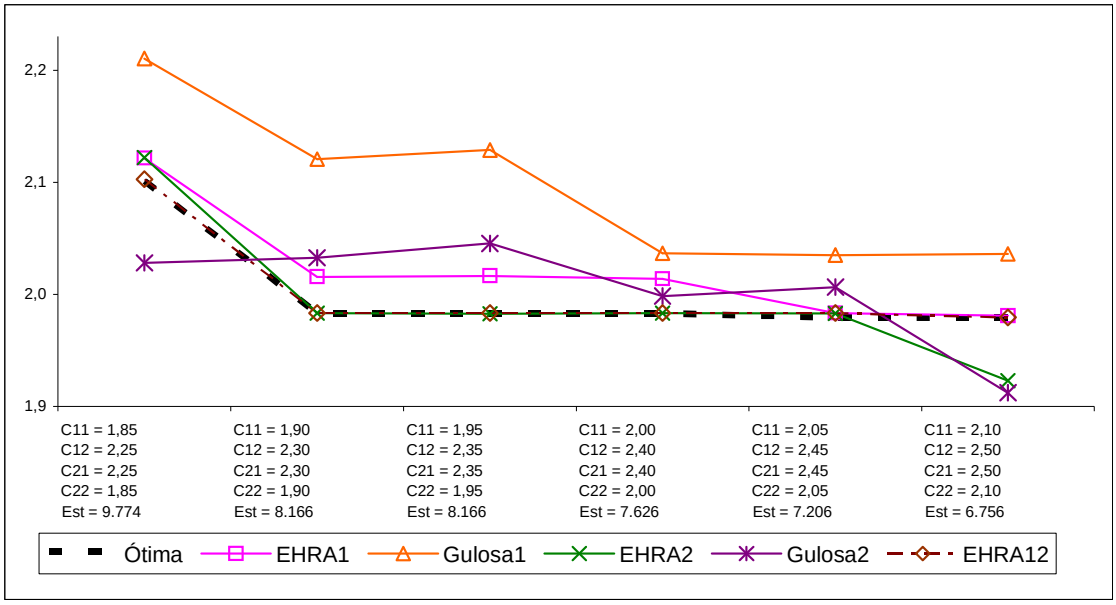


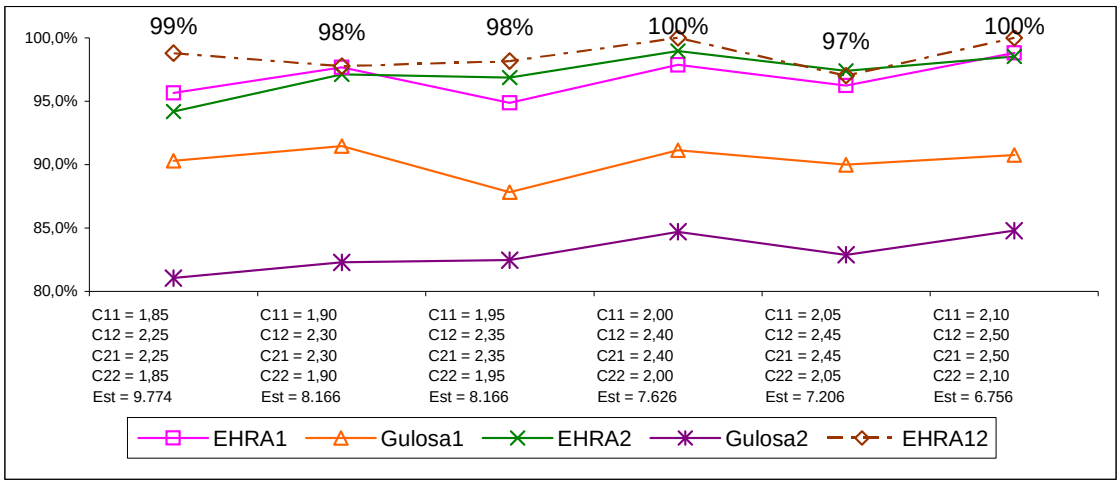
Gráfico 4.11 - Número esperado de pacientes em atendimento.



No Gráfico 4.12, apresentam-se as proporções de concordâncias entre as ações prescritas pela PO e as ações prescritas por cada uma das políticas EHRA e gulosas. Este é um importante indicador de qualidade das políticas; espera-se que uma boa proporção de concordância esteja associada a uma política que apresente um custo médio esperado por período próximo do ótimo. Observa-se, entretanto, que não há garantia de que a política que apresenta a maior proporção de concordâncias com a PO seja sempre a política que possua o custo médio esperado por período mais próximo do ótimo. Nas configurações testadas neste estudo, sempre que a proporção de concordância entre as ações de uma das políticas e as ações da PO foi maior em comparação a outra política, o custo médio esperado da política em questão foi menor. A política gulosa G2 apresentou a menor proporção de concordância em relação às ações da PO, variando entre 81% e 85%. A política G1 apresentou o segundo pior desempenho em relação à concordância com a PO, variando entre 87% e 92%. As políticas EHRA apresentaram percentual de concordância com a PO variando entre 94% e 100%. A política EHRA12 apresentou a maior proporção de concordância nas configurações 1, 2, 3, 4 e

6, e a política EHRA2 apresentou a maior proporção de concordância na Configuração 5.

Gráfico 4.12 - Concordância com as ações da política ótima.



4.2.5 Aumentando o espaço de estados e o espaço de ações

A complexidade de uma iteração do método EHRA, quando aplicado ao controle das admissões de pacientes como formulado no Capítulo 2, é estimada em seu esforço máximo de processamento (pior caso) pela expressão $O(|a||\Lambda|CHg^{m(n-1)})$, em que $|a|$ representa o tamanho da população de ações a ser testada a cada iteração, $|\Lambda|$ representa o número de políticas de base empregadas pelo método, C representa o número de simulações do horizonte de planejamento reduzido a ser realizado para cada política de base em cada iteração, H representa o tamanho do horizonte de planejamento reduzido, m representa o número de especialidades e n o número de padrões de atendimento, g representa o número máximo de pacientes em um padrão de atendimento, considerando-se os n padrões. Os últimos três parâmetros são dependentes da estrutura do modelo relativa à dinâmica estocástica da movimentação dos pacientes. O termo $g^{m(n-1)}$ é

derivado da necessidade de se avaliar as possíveis combinações geradoras dos estados atingíveis a partir do par de informações (estado; ação), para então se calcular as probabilidades de transição entre os estados, baseadas na distribuição multinomial, e para, enfim, se calcular o custo esperado até o próximo instante de decisão. Os parâmetros $|a|$, $|\Lambda|$, C e H são independentes das dimensões do modelo, sendo definidos para as operações das iterações do método.

A complexidade computacional do AIV é polinomial de grau dois sobre o espaço de estados (LITTMAN *et al.*, 1995), sendo calculada pela expressão $O(|X|^2|A|)$, em que $|X|$ é o tamanho do espaço de estados e $|A|$ é o tamanho do espaço de ações. Ressalta-se que o AIV constrói uma política ótima completa, uma ação para cada estado, a qual é empregada geralmente em um processo de decisão *off-line*, em que, de antemão, as ações são conhecidas quando o estado do sistema é observado. Por outro lado, o método EHRA trata um estado por vez, sendo empregado mais comumente em processos de decisão do tipo *on-line*, em que se observa o estado do sistema e então se utiliza o método para obtenção de uma ação.

Com o objetivo de exemplificar a diferença do esforço computacional demandado pelos dois métodos, apresentam-se, nas tabelas 4.4 e 4.5, os tempos de processamento requeridos para se obter soluções em diferentes instâncias do modelo de controle de admissões de pacientes. Os parâmetros e as probabilidades do modelo são apresentados nas tabelas 4.1 e 4.2 da Seção 4.2.1, sendo adotada a seguinte configuração para os consumos médios de cada padrão de atendimento: o consumo médio esperado de um paciente que passa um período no padrão de atendimento E_1 é de 2,3 unidades do recurso L_1 e 2,7 unidades do recurso L_2, enquanto que um paciente que passa um período no padrão de atendimento E_2 apresenta um consumo médio esperado de 2,7 unidades do recurso L_1 e 2,3 unidades do recurso L_2. Para obtenção de

uma ação pelo método EHRA2 em todas as instâncias testadas, fixou-se o estado (1,0,0,0,1,0): um paciente da Especialidade 1 no Padrão de Atendimento 1 e um paciente da Especialidade 2 no Padrão de Atendimento 2.

Na Tabela 4.4, apresenta-se o efeito do crescimento do espaço de estados, mantido o espaço de ações com a mesma dimensão. Aumentou-se a disponibilidade dos recursos em uma unidade a cada instância, passando de 10 unidades de cada recurso até atingir 14 unidades. Com isso, ampliou-se a capacidade de atendimento de pacientes, e, consequentemente, o espaço de possíveis estados aumentou de 54.947 estados até 207.706 estados. O número máximo de pacientes nos padrões de atendimento, g, passou de 15 para 20 pacientes. O espaço de possíveis ações foi mantido com 9 ações, partindo da ação que não admite novos pacientes, ação (0,0), até a ação correspondente à capacidade máxima de admissões, fixada em 2 pacientes de cada especialidade, (2,2). Observa-se que o AIV rapidamente aumenta sua complexidade computacional, medida pela estimativa do pior caso, passando de cerca de 27 bilhões de operações a cada iteração na primeira instância a mais de 388 bilhões de operações na última instância, com um aumento de mais de 14 vezes. O tempo de processamento medido para obtenção de uma política ótima através do AIV passou de 2 horas e 37 minutos na primeira instância para 1 dia, 15 horas e 52 minutos na última instância testada, registrando um aumento de 15 vezes. O método EHRA2, que utiliza apenas a política gulosa 2 como política de base, aumentou sua complexidade computacional em 3 vezes, em função do aumento do parâmetro g, passando de cerca de 151 milhões de operações a cada iteração a 480 milhões. O tempo de processamento de vinte iterações para obtenção de uma ação, considerando o estado observado (1,0,0,0,1,0), passou de 1 minuto e 24 segundos na primeira instância a 3 minutos e 51 segundos na última instância, registrando um aumento de cerca de 3 vezes. As ações obtidas pelo EHRA2 coincidiram com as ações prescritas pelo AIV.

Tabela 4.4 - Complexidade computacional e tempo de processamento do AIV e do EHRA2, incrementando a disponibilidade de cada recurso.

Unidades de cada recurso	Max. de adm. de pacient. de cada espec.	nº de ações	nº de estados	AIV			EHRA2		
				$O(\|X\|^2\|A\|)$	Tempo de process.: para uma iteração	Tempo de process. para a política ótima	$O(\|a\|\|\Lambda\|CHg^{m(n-1)})$	Tempo de process.: para uma iteração	Tempo de process. para vinte iterações
10	2	9	54.947	27.172.555.281	00:10h	02:37h	151.875.000	00:00'04h	00:01'24h
11	2	9	77.078	53.469.162.756	00:22h	05:35h	196.608.000	00:00'07h	00:02'23h
12	2	9	110.852	110.593.493.136	00:44h	11:03h	314.928.000	00:00'09h	00:02'51h
13	2	9	151.684	207.072.322.704	01:14h	21:07h	390.963.000	00:00'10h	00:03'12h
14	2	9	207.706	388.276.041.924	02:40h	1 dia e 15:52h	480.000.000	00:00'12h	00:03'51h

Um aumento do espaço de ações, considerando o modelo de controle de admissões de pacientes, gera, consequentemente, um aumento do espaço de estados, visto que a possibilidade de admitir maior número de pacientes aumenta a quantidade de possíveis combinações de pacientes no período seguinte. Por exemplo, partindo-se do estado (1,0,0,0,1,0) com o espaço de ações composto pelas ações relativas à capacidade máxima de 1 admissão em cada especialidade, considerando-se todas as ações, seriam possíveis de serem atingidos 64 estados, enquanto que com capacidade máxima de 2 admissões por especialidade seriam 225 os estados possíveis no próximo período.

Tabela 4.5 - Complexidade computacional e tempo de processamento do AIV e do EHRA2, incrementando a capacidade de admissão de pacientes.

Unidades de cada recurso	Max. de adm. de pacient. de cada espec.	nº de ações	nº de estados	AIV			EHRA2		
				$O(\|X\|^2\|A\|)$	Tempo de process.: para uma iteração	Tempo de process. para a política ótima	$O(\|a\|\|\Lambda\|CHg^{m(n-1)})$	Tempo de process.: para uma iteração	Tempo de process. para vinte iterações
10	2	9	54.947	27.172.555.281	00:10h	02:37h	151.875.000	00:00'04h	00:01'24h
10	3	16	99.931	159.779.276.176	00:46h	7:40h	196.608.000	00:00'05h	00:01'36h
10	4	25	170.853	729.768.690.225	02:29h	1 dia e 00:46h	250.563.000	00:00'05h	00:01'46h
10	5	36	277.843	2.779.082.375.364	06:46h	2 dias e 22:52h	314.928.000	00:00'06h	00:02'18h

Na Tabela 4.5, apresenta-se o efeito do crescimento do espaço de ações, quando mantida a quantidade disponível de cada recurso em 10 unidades, e aumentando-se de 2 até 5 pacientes a capacidade de admissão de pacientes

de cada especialidade. O espaço de ações passou de 9 a 36 ações possíveis, enquanto que o espaço de estados cresceu rapidamente de 54.947 a 277.843 estados. A complexidade computacional de uma iteração do AIV, estimada para o pior caso, passou de cerca de 27 bilhões de operações para mais de 2,7 trilhões de operações, apresentando crescimento maior do que 100 vezes o tamanho inicial, enquanto que o espaço de estados cresceu 5 vezes. O tempo de processamento medido para obtenção de uma política ótima pelo AIV passou de 2 horas e 37 minutos para 2 dias, 22 horas e 52 minutos, aumentando mais do que 27 vezes. Em relação ao método EHRA2, a complexidade de uma iteração para se obter uma ação dado o estado (1,0,0,0,1,0), medida pela estimativa do pior caso, sofreu um acréscimo relativamente pequeno, passando de cerca de 152 milhões de operações a cerca de 315 milhões, aumentando em 2 vezes seu tamanho, enquanto que o espaço de estados aumentou em 5 vezes. Observa-se que o parâmetro g, componente da expressão empregada para o cálculo da complexidade do método EHRA, cresce mais lentamente quando se aumenta o espaço de ações do que quando se disponibiliza maior quantidade de recursos. A cada unidade acrescida à capacidade de admissão de pacientes, o número máximo de pacientes em um dos padrões de atendimento, g, aumenta também em uma unidade. Nas instâncias testadas, g partiu de 15 pacientes até atingir 18 pacientes. O tempo medido de processamento de vinte iterações do método EHRA2 para obtenção de uma ação, dado o estado (1,0,0,0,1,0), apresentou um acréscimo relativo pequeno à medida que se aumentou a dimensão do espaço de ações. O tempo de processamento passou de 1 minuto e 24 segundos, na primeira instância, a 2 minutos e 18 segundos na quarta instância, com um crescimento de apenas 1,6 vezes. As ações obtidas pelo EHRA2 coincidiram com as ações prescritas pelo AIV.

4.3 Avaliação do modelo de controle de admissões

Apresenta-se, nesta seção, a implementação do modelo de controle de admissões de pacientes aplicado aos dados obtidos de um hospital de reabilitação com atendimento exclusivamente eletivo, o Hospital SARAH de Brasília, integrante da Rede SARAH de Hospitais de Reabilitação. Os recursos hospitalares considerados no modelo foram as consultas médicas e os atendimentos fisioterápicos. Os pacientes foram divididos em dois grupos. Cada grupo continha pacientes pertencentes a uma de duas especialidades. Inicialmente, foram identificados cinco padrões de atendimento, que com alguns ajustes no modelo foram posteriormente reduzidos a quatro padrões. As subseções seguintes detalham os elementos do modelo e os ajustes realizados para viabilizar a obtenção de soluções.

4.3.1 Características das especialidades

Duas especialidades de pacientes foram consideradas. A Especialidade 1 refere-se aos pacientes adultos com problemas de coluna incluindo dores irradiadas para os membros superiores e inferiores. Na Especialidade 2 foram agrupados os demais casos de pacientes adultos com dores e/ou complicações musculoesqueléticas envolvendo o aparelho locomotor.

Ao longo do ano de 2005, no Hospital SARAH de Brasília, foram admitidos 271 pacientes classificados como pertencentes à Especialidade 1 e 2.661 pacientes classificados como pertencentes à Especialidade 2. Essa coorte de pacientes foi acompanhada até 31 de julho de 2009. Registraram-se cronologicamente as atividades realizadas pelo hospital para cada paciente.

4.3.2 Características dos recursos

Para essa implementação foram considerados dois recursos do hospital: as consultas médicas e as sessões de fisioterapia. Dessa forma, para cada paciente registrou-se a data da realização das consultas e dos atendimentos fisioterápicos ao longo do período de observação.

Os custos relativos de cada recurso foram estipulados considerando-se as diferenças entre os salários bases dos médicos e dos fisioterapeutas. Os custos relativos às consultas médicas foram estabelecidos com valores 25% superiores aos custos relativos às sessões fisioterápicas. Os valores atribuídos aos recursos são adimensionais, guardando apenas as diferenças proporcionais. Assim, em relação ao nível desejado de utilização dos recursos, o custo de ociosidade de uma consulta foi estabelecido em 1,6, enquanto que o custo de ociosidade de uma sessão fisioterápica ficou em 1,28. O custo de excesso em relação ao nível de utilização desejado e sobre a capacidade disponibilizada do recurso para o período foram, ambos, estabelecidos em 1,25 para uma consulta médica e em 1 para uma sessão de fisioterapia.

4.3.3 Características dos padrões de atendimento

O período total de observação da evolução dos tratamentos dos pacientes, de 2005 a 2009, foi suficiente para que todos recebessem alta hospitalar relativa à causa inicial do tratamento. O período total de atendimento de cada paciente, da admissão à alta, foi subdividido em períodos de sete dias. Dentro de cada período de sete dias contabilizou-se o número de consultas médicas realizadas e de sessões de fisioterapia. Considerou-se o período de alta como o próximo período de sete dias após o último período com registro de pelo menos uma atividade.

Utilizou-se o método das k-médias (HARTIGAN; WONG, 1979) para a identificação de padrões de atendimento com consumo semelhante dos recursos considerados. Analisando-se os resultados, chegou-se à identificação de quatro padrões de atendimento: Padrão de Atendimento 1, em que dentro do período de sete dias ocorre a realização de consultas médicas e de sessões de fisioterapia; Padrão de Atendimento 2, em que dentro do período de sete dias ocorre a realização apenas de consultas médicas; Padrão de Atendimento 3, em que dentro do período de sete dias ocorre a realização apenas de sessões de fisioterapia; e Padrão de Atendimento 4, em que dentro do período de sete dias não ocorre atendimento. O Padrão 4 representa um período intermediário em que o paciente aguarda a evolução do seu quadro ou realiza outras intervenções terapêuticas para então retornar a um dos padrões anteriores. Acrescentou-se, ainda, o Padrão de Atendimento 5 para representar a alta hospitalar.

Tabela 4.6 - Utilização dos recursos por padrão de atendimento e especialidade.

Padrões de Atendimento	Consumo médio Especialidade 1		Consumo médio Especialidade 2	
	Consultas	Fisioterapias	Consultas	Fisioterapias
1	1,08	1,70	1,12	1,29
2	1,03	-	1,06	-
3	-	3,64	-	2,45
4	-	-	-	-
5	-	-	-	-

As duas especialidades consideradas apresentaram utilizações médias dos recursos consideravelmente distintas em cada padrão de atendimento (Tabela 4.6). Para contabilizar a utilização dos recursos levando-se em conta essas diferenças, optou-se por realizar um pequeno ajuste no modelo original, apresentado no Capítulo 2. Segundo a formulação original, L_{ij}, a quantidade média esperada de recursos do tipo L_j utilizada por um paciente em um período é dependente apenas de E_i, o padrão de atendimento em que o

paciente se encontra. Segundo a formulação ajustada para essa implementação, L_{dij}, a quantidade média esperada de recursos do tipo L_j utilizada por um paciente em um período passa a ser dependente tanto do padrão de atendimento E_i como da especialidade do paciente, sendo o índice d relativo à especialidade. Seguindo essa modificação, as fórmulas originais para os cálculos do consumo dos recursos podem ser facilmente ajustadas.

4.3.4 Probabilidades de entrada e das transições

Partindo das observações dos atendimentos dos pacientes admitidos no ano de 2005, as estimativas de máxima verossimilhança para as probabilidades de entrada de um novo paciente em cada padrão de atendimento e para as probabilidades de passagem em um período de um padrão de atendimento a outro foram obtidas pela contagem das ocorrências desses eventos em cada especialidade.

Tabela 4.7 - Registros de entradas nos padrões de atendimento e estimativa das probabilidades de entrada nos padrões de atendimento, por especialidade.

Padrões de Atendimento	Especialidade 1		Especialidade 2	
	Reg. de entradas	Probabilidades	Reg. de entradas	Probabilidades
1	88	0,325	1.076	0,404
2	183	0,675	1.585	0,596
3	-	-	-	-
4	-	-	-	-
5	-	-	-	-
Registros	271		2.661	

As probabilidades de entrada em cada padrão de atendimento para cada especialidade estão apresentadas na Tabela 4.7. Nota-se que os pacientes só podem ingressar nos padrões de atendimento que incluem pelo menos uma

consulta médica. Em ambas as especialidades, os pacientes têm mais chance de entrarem no Padrão de Atendimento 2, que inclui apenas consultas médicas, sendo que a Especialidade 1 apresenta a maior probabilidade de entrada nesse padrão.

Tabela 4.8 - Registros de transições entre padrões de atendimento e estimativa das probabilidades de transição entre os padrões de atendimento, por especialidade.

Padrões de chegada	Padrões de partida 1		2		3		4		Registros	
	Reg.	Probab.	Reg.	Probab.	Reg.	Probab.	Reg.	Probab.		
Especialidade 1										
1	6	0,0160	11	0,0178	10	0,0305	259	0,0277	286	2,7%
2	3	0,0080	13	0,0210	5	0,0152	415	0,0444	436	4,1%
3	68	0,1818	14	0,0226	180	0,5488	66	0,0071	328	3,1%
4	214	0,5722	420	0,6785	106	0,3232	8.605	0,9208	9.345	87,6%
5	83	0,2219	161	0,2601	27	0,0823	-	-	271	2,5%
Registros	374		619		328		9.345		10.666	
	3,5%		5,8%		3,1%		87,6%			
Especialidade 2										
1	31	0,0106	172	0,0358	39	0,0234	1.603	0,0223	1.845	2,3%
2	56	0,0192	204	0,0424	42	0,0252	2.920	0,0406	3.222	4,0%
3	219	0,0750	104	0,0216	524	0,3143	820	0,0114	1.667	2,0%
4	1.510	0,5169	3.069	0,6384	764	0,4583	66.634	0,9258	71.977	88,5%
5	1.105	0,3783	1.258	0,2617	298	0,1788	-	-	2.661	3,3%
Registros	2.921		4.807		1.667		71.977		81.372	
	3,6%		5,9%		2,0%		88,5%			

Na Tabela 4.8 apresentam-se, para cada especialidade, os registros de passagens de um padrão de atendimento a outro e as estimativas de probabilidades de transição entre os padrões de atendimento. O Padrão de Atendimento 5 é um padrão absorvente, representando a alta hospitalar. Observa-se, em ambas as especialidades, que a maior chance de mudança de padrão é para o Padrão de Atendimento 4 e que, uma vez nesse padrão, a maior chance é que o paciente nele permaneça. O Padrão de Atendimento 4 é um padrão de espera, sem consumo de recursos e sem possibilidade de passar ao padrão de alta. Uma vez que todos os pacientes acompanhados

neste estudo receberam alta hospitalar, ou seja, atingiram o Padrão 5, e que as novas entradas ocorreram apenas nos padrões 1 e 2, observa-se que o número de transições que chegam é igual ao número de transições que partem nos padrões 3 e 4 e que a soma das diferenças entre partidas e chegadas aos padrões 1 e 2 é igual ao número de entradas de pacientes.

O Padrão de Atendimento 4 tem um efeito adverso sobre o modelo proposto no Capítulo 2, pois à medida que os períodos de funcionamento do hospital se sucedem, naturalmente ocorre a formação de uma grande concentração de pacientes no Padrão 4. Essa concentração rapidamente torna o processamento computacional para o cálculo das probabilidades de transição entre os estados, que é baseado na multiplicação de distribuições de probabilidade multinomiais, muito custoso ou mesmo inviável. Tentando tratar instâncias maiores do problema, é possível reduzir o número inicial de padrões de atendimento, o que requer alguns ajustes na estrutura do modelo para acomodar as modificações consequentes. A simplificação aqui proposta passa a entender o Padrão 4 como um *buffer* de pacientes, retendo-os por algum tempo para retorná-los ao processo de atendimento períodos adiante.

A Tabela 4.9 mostra as probabilidades de transição entre padrões de atendimento, agora para o modelo simplificado com quatro padrões. No modelo simplificado, o Padrão de Atendimento 4 é o padrão absorvente, que agrupa as transições de chegadas registradas originalmente para os padrões de atendimento 4 e 5 do modelo anterior.

As probabilidades de reentrada em cada padrão de atendimento para os pacientes que aguardam o prosseguimento do tratamento estão apresentadas na Tabela 4.10. As transições de saída do Padrão de Atendimento 4 do modelo original foram empregadas para o cálculo das probabilidades de reentrada dos pacientes nos padrões de atendimento 1, 2 e 3 do modelo simplificado.

Tabela 4.9 - Para o modelo com quatro padrões de atendimento, registros de transições entre padrões de atendimento e estimativa das probabilidades de transição entre os padrões de atendimento, por especialidade.

Padrões de chegada	Padrões de partida 1		2		3		Registros	
	Registros	Probab.	Registros	Probab.	Registros	Probab.		
Especialidade 1								
1	6	0,0160	11	0,0178	10	0,0305	27	2,0%
2	3	0,0080	13	0,0210	5	0,0152	21	1,6%
3	68	0,1818	14	0,0226	180	0,5488	262	19,8%
4	297	0,7941	581	0,9386	133	0,4055	1.011	76,5%
Registros	374		619		328		1.321	
	28,3%		46,9%		24,8%			
Especialidade 2								
1	31	0,0106	172	0,0358	39	0,0234	242	2,6%
2	56	0,0192	204	0,0424	42	0,0252	302	3,2%
3	219	0,0750	104	0,0216	524	0,3143	847	9,0%
4	2.615	0,8952	4.327	0,9001	1.062	0,6371	8.004	85,2%
Registros	2.921		4.807		1.667		9.395	
	31,1%		51,2%		17,7%			

Tabela 4.10 - Para o modelo com quatro padrões de atendimento, registros das reentradas nos padrões de atendimento e estimativa das probabilidades de reentrada nos padrões de atendimento, por especialidade.

Padrões de Atendimento	Especialidade 1		Especialidade 2	
	Reg. de reentradas	Probabilidades	Reg. de reentradas	Probabilidades
1	259	0,350	1.603	0,300
2	415	0,561	2.920	0,547
3	66	0,089	820	0,153
4	-	-	-	-
Registros	740		5.343	

O número esperado de pacientes que deve reentrar no sistema a cada período é dado pela multiplicação do número de pacientes que permanecem aguardando para prosseguir o tratamento (*buffer*) *versus* a probabilidade de um paciente retornar ao sistema, dado que ele se encontra em espera. A

estimativa desta probabilidade é igual ao valor de uma unidade menos a probabilidade de permanência no Padrão 4 do modelo original. Partindo-se dos dados observados, para a Especialidade 1 a probabilidade estimada de um paciente que se encontra em espera retornar ao atendimento é de 0,0792, e para a Especialidade 2 esta probabilidade é de 0,0742. Optou-se por considerar que a quantidade de pacientes que permanecem aguardando no *buffer* é constante; isso implica considerar que a quantidade de pacientes retornando ao atendimento em cada período também é constante e que deve ser determinada previamente.

Para ajustar o modelo original proposto no Capítulo 2 à modificação introduzida pelo *buffer* descrito acima, basta considerar, em cada período, em conjunto com as entradas de pacientes previstas pelas ações de controle de admissões, o fluxo constante de reentradas dos pacientes que aguardam o prosseguimento do tratamento. Os espaços de estados e de ações permanecem os mesmos, sendo que as ações, simbolizadas por $a = \{S^1,...,S^m\}$, em que S^d representa a quantidade de pacientes da especialidade d, $d \in \{1,...,m\}$, a ser admitida no próximo período, ficam acrescidas das constantes de reentradas c^d, podendo ser representadas por $a' = \{S^1,...,S^m, c^1,...,c^m\}$. As probabilidades de reentradas em cada padrão de atendimento, mostradas na Tabela 4.10, são distintas das probabilidades de entrada de novos pacientes em cada padrão de atendimento, mostradas na Tabela 4.7. Essas diferenças podem ser facilmente incorporadas à formulação original do modelo para o cálculo das probabilidades de transição entre estados e dos custos esperados associados ao par (estado; ação).

4.3.5 Aplicação do modelo

Com o objetivo de avaliar o potencial de aplicação do modelo de controle de admissões proposto utilizando-se os dados observados e os ajustes descritos nas subseções acima (tabelas 4.9 e 4.10), apresentam-se, a seguir, para algumas instâncias do modelo, as ações obtidas pelos métodos gulosos G1 e G2 e pelo método EHRA (ver Seção 3.6 do Capítulo 3), empregando a gulosa G2 como política de base (EHRA2). Para cada instância testada, foram registrados os tempos de processamento para a determinação das ações. As instâncias avaliadas foram obtidas com o aumento de alguns parâmetros dimensionais do modelo, quais sejam: a quantidade disponibilizada dos recursos, o número de pacientes aguardando para prosseguir o tratamento e o número máximo de admissões por período.

Optou-se por empregar a política gulosa G2 como política de base para o método EHRA, por ser esta menos custosa em relação ao tempo de processamento se comparada à política gulosa G1. Os parâmetros de entrada (ver Capítulo 3) para obtenção das ações EHRA2 foram: o tamanho da população de ações igual a 4 ações, o tamanho do horizonte de planejamento reduzido igual a 4 períodos, o número de amostras de realizações de funcionamento do sistema para cada política de base igual a 4 horizontes reduzidos, o fator de desconto igual a 0,8, a probabilidade para escolha de uma ação aleatoriamente no espaço de ações igual a 0,1, a probabilidade para escolha de uma ação na vizinhança da ação de elite igual a 0,9, e o número de repetições da ação de elite para o critério de parada igual a 16. Em cada instância, considerou-se como nível de utilização desejado para cada recurso hospitalar uma unidade a menos do que a capacidade disponibilizada por período. Os custos de desvio em relação ao nível de utilização desejado foram estabelecidos da seguinte forma: cada unidade de desvio menor do que o nível de utilização desejado para o recurso L_1 (consulta médica) possui um custo de ociosidade de 1,6, cada unidade de desvio maior do que o nível de utilização

desejado tem um custo de excesso de 1,25 e cada unidade de desvio acima da capacidade disponibilizada do recurso L_1 possui um custo adicional de 1,25. Considerando-se o recurso L_2 (sessão de fisioterapia), temos os seguintes custos: custo de ociosidade 1,28, custo de excesso sobre o nível de utilização desejado 1,0 e custo de excesso sobre a capacidade disponível 1,0.

Tabela 4.11 - Ações prescritas pelos métodos G1, G2 e EHRA2, e tempo de processamento para obtenção das ações para algumas instâncias do modelo de adm. de pacientes.

Estado:[a]	Max de adm[b]	Reentr[c]	Rec[d]	G1		G2		EHRA2			
				ação[e]	tp proc[f]	ação	tp proc	ação	itera[g]	tp proc	tp itera[h]
{1,1,1,0,1,1,1,0}	{2,2}	{2,2}	{8,14}	{2,1}	00:00'10h	{1,2}	00:00'01h	{2,1}	47	05:00'17h	00:06'23h
{1,1,1,0,1,1,1,0}	{2,2}	{2,2}	{9,15}	{2,2}	00:00'10h	{2,2}	00:00'01h	{2,2}	20	04:28'13h	00:13'25h
{1,1,1,0,1,1,1,0}	{2,2}	{3,3}	{8,14}	{1,0}	00:00'26h	{0,1}	00:00'01h	{1,0}	36	06:28'16h	00:10'47h
{1,1,1,0,1,1,1,0}	{3,3}	{2,2}	{8,14}	{3,0}	00:00'41h	{1,2}	00:00'01h	{1,2}	34	04:43'18h	00:08'20h
{2,2,2,0,2,2,2,0}	{2,2}	{2,2}	{8,14}	{0,2}	00:03'37h	{1,2}	00:00'01h	{2,1}	40	06:05'57h	00:09'09h
{2,2,2,0,2,2,2,0}	{3,3}	{3,3}	{9,15}	{0,2}	00:19'31h	{1,1}	00:00'01h	{1,1}	40	15:35'26h	00:23'24h
{9,9,9,0,9,9,9,0}	{20,20}	{16,15}	{60,80}	{5,20}	696:19'31h	{6,20}	00:00'01h	-	-	-	-

a - Estado: $\{E_1^1, E_2^1, E_3^1, E_4^1, E_1^2, E_2^2, E_3^2, E_4^2\}$
b - Máximo de admissões em cada especialidade
c - Número de reentradas em cada especialidade
d - Quantidade disponibilizada de cada recurso: consultas e sessões de fisioterapia
e - Ação: $\{S^1, S^2\}$
f - Tempo de processamento
g - Número de iterações
h - Tempo médio para uma iteração

Para exemplificar os resultados apresentados na Tabela 4.11, descreve-se a seguir as informações referentes à primeira instância testada, apresentada na primeira linha de dados: o estado observado informa que estiveram em atendimento no último período 3 pacientes da Especialidade 1 e 3 pacientes da Especialidade 2, sendo 1 paciente de cada especialidade em cada padrão de atendimento, sem altas hospitalares; podem ser admitidos no máximo 4 pacientes por período, 2 de cada especialidade; são admitidos em reentradas 4 pacientes por período, 2 de cada especialidade (isto implica que ficam em espera por atendimento aproximadamente 25 pacientes da Especialidade 1 e 27 pacientes da Especialidade 2); estão disponibilizadas 8 consultas médicas e 14 sessões de fisioterapia por período; para esse estado, a ação gulosa G1 prescreveu a admissão no próximo período de 2 pacientes da Especialidade 1

e 1 paciente da Especialidade 2; o tempo de processamento da G1 foi de 10 segundos; a ação gulosa G2 prescreveu a admissão de 1 paciente da Especialidade 1 e 2 pacientes da Especialidade 2; o tempo de processamento da G2 foi menor que 1 segundo; o método EHRA2 realizou 47 iterações para prescrever sua ação; a ação EHRA2 foi a mesma prescrita pela G1, admitir 2 pacientes da Especialidade 1 e 1 paciente da Especialidade 2; o tempo de processamento para as 47 iterações foi de 5 horas e 17 segundos; o tempo médio para uma iteração do EHRA2 foi de 6 minutos e 23 segundos.

Na segunda instância, disponibilizou-se mais uma unidade de cada recurso. A ação prescrita pelos três métodos foi a de admitir 2 pacientes de cada especialidade no próximo período. O tempo de processamento para obtenção das ações gulosas não se alterou em relação à instância anterior, pois o acréscimo da disponibilidade de recursos não aumenta o número de ações a serem avaliadas e não aumenta, a partir do estado observado e das ações disponíveis, o número de possíveis estados que podem ser alcançados no próximo período. Já o tempo médio de processamento das iterações do EHRA2 aumentou, visto que o acréscimo dos recursos aumenta o espaço de possíveis estados, e as ações são mais permissivas em relação ao número de admissões, o que leva a estados com maior número de pacientes em atendimento e, consequentemente, a um maior número de combinações de próximos estados possíveis a serem avaliados.

As reentradas foram acrescidas em 1 paciente para cada especialidade na terceira instância. A ação G1 prescreveu, em 26 segundos, a admissão de 1 paciente da Especialidade 1 no próximo período. A ação G2 prescreveu, em menos de 1 segundo, a admissão de 1 paciente da Especialidade 2. A ação EHRA2 prescreveu, em 36 iterações, após 6 horas, 28 minutos e 16 segundos, as mesmas quantidades determinadas pela gulosa G1. O tempo de processamento para obtenção da ação G1 aumentou, pois o acréscimo das reentradas aumenta o número de próximos estados que podem ser atingidos

no próximo período e devem ser avaliados para a determinação da ação. O tempo de processamento para obtenção da ação G2, que não realiza a avaliação de todos os próximos estados possíveis, permaneceu menor do que 1 segundo. O tempo médio de processamento de uma iteração para obtenção da ação EHRA2 aumentou em relação à primeira instância, já que o acréscimo das reentradas, assim como para a obtenção da ação G1, aumenta o número de próximos estados que podem ser atingidos no próximo período. Porém, em relação à segunda instância, o tempo médio de processamento de uma iteração do EHRA2 diminuiu, pois o efeito de se acrescentarem mais recursos permite que se atinjam estados com maior número de pacientes em atendimento, aumentando, consequentemente, o número de combinações possíveis a serem avaliadas para o próximo estado, enquanto que o efeito do aumento de reentradas sem aumento de recursos causa uma restrição de admissões, e o número de pacientes em atendimento não aumenta tanto quanto com o acréscimo de recursos.

Na quarta instância, a capacidade de admissão foi acrescida em 1 paciente para cada especialidade. A ação G1 prescreveu, em 41 segundos, a admissão de 3 pacientes da Especialidade 1 no próximo período. A ação G2 prescreveu, em menos de 1 segundo, a admissão de 1 paciente da Especialidade 1 e 2 pacientes da Especialidade 2. A ação EHRA2 prescreveu, em 34 iterações, após 4 horas, 43 minutos e 18 segundos, as mesmas quantidades determinadas pela gulosa G2. O tempo de processamento para a ação G1 aumentou, pois o acréscimo da capacidade de admissões faz aumentar o número de possíveis ações que deve ser avaliado para a escolha da ação. O tempo de processamento para a ação G2 permaneceu menor do que 1 segundo, visto que no método G2 não se avaliam todas as ações nem todos os possíveis próximos estados (ver a Subseção 4.2.2). O tempo médio de uma iteração para obtenção da ação EHRA2 aumentou levemente em relação ao tempo médio da primeira instância, pois o acréscimo de capacidade de admissão aumenta as opções de ações disponíveis, o que não causa um

aumento no tempo de processamento tão evidente como o causado pelo aumento da capacidade de atendimento, com o acréscimo de mais recursos, ou o causado pela reentrada de mais pacientes em cada período.

Modificou-se o estado observado na quinta instância, acrescentando-se um paciente de cada especialidade em cada padrão de atendimento. A ação G1 prescreveu, em 3 minutos e 37 segundos, a admissão de 2 pacientes da Especialidade 2 no próximo período. A ação G2 prescreveu, em menos de 1 segundo, a admissão de 1 paciente da Especialidade 1 e 2 pacientes da Especialidade 2. A ação EHRA2 prescreveu, em 40 iterações, após 6 horas, 5 minutos e 57 segundos, uma ação distinta das ações prescritas pelos métodos gulosos. A ação EHRA2 determina a admissão no próximo período de 2 pacientes da Especialidade 1 e 1 paciente da Especialidade 2. O tempo de processamento para a ação G1 aumentou, pois o estado observado acrescido de pacientes faz aumentar a quantidade de possíveis estados que podem ser atingidos no próximo período, o que, consequentemente, aumenta o esforço computacional para determinar todas as combinações possíveis e calcular as probabilidades associadas. O tempo de processamento para a ação G2 permaneceu menor do que 1 segundo, já que o acréscimo de pacientes ao estado observado não provoca incremento significativo nas operações realizadas pelo método guloso G2. O tempo médio de processamento de uma iteração para obtenção da ação EHRA2 aumentou, pois o acréscimo de pacientes ao estado observado implica que o estado inicial, em todos os horizontes de planejamento simulados, contém mais pacientes, aumentando, assim, a quantidade de estados que podem ser atingidos a partir do estado inicial, o que consequentemente leva a um maior esforço computacional para avaliar as combinações de estados possíveis e calcular as probabilidades associadas.

Na sexta instância, foram acrescentadas à primeira instância todas as modificações realizadas da segunda instância à quinta. A ação G1 prescreveu

a admissão de 2 pacientes da Especialidade 2 no próximo período. A ação G2 prescreveu a admissão de 1 paciente da Especialidade 1 e 1 paciente da Especialidade 2. A ação EHRA2 prescreveu as mesmas quantidades determinadas pela gulosa G2. Pela conjunção dos motivos destacados acima na descrição de cada instância, os tempos de processamento na sexta instância foram bem maiores para obtenção das ações G1 e EHRA2, enquanto que para a ação G2 permaneceu abaixo de 1 segundo. O tempo de processamento para a ação G1 foi de 19 minutos e 31 segundos. A ação EHRA2 foi obtida em 6 horas, 35 minutos e 26 segundos, após 40 iterações, com um tempo médio de 23 minutos e 24 segundos para cada iteração.

A sétima instância apresenta uma configuração que retrata de forma mais realista a estrutura de uma pequena equipe composta por um médico e dois fisioterapeutas. O número disponibilizado de consultas médicas por semana é de 60 consultas, e o número de sessões de fisioterapia é de 80 sessões. Considera-se que, em média, 200 pacientes de cada especialidade ficam na condição de aguardo para o prosseguimento do tratamento, o que gera uma reentrada de 16 pacientes da Especialidade 1 e 15 pacientes da Especialidade 2 por semana. O número máximo de admissões disponibilizado por semana é de 20 admissões para cada especialidade. O estado observado considerado registrou o atendimento de 54 pacientes na semana anterior, sendo 27 de cada especialidade, 9 em cada padrão de atendimento. Para a semana subsequente, a ação G2 prescreveu, em menos de 1 segundo, a admissão de 6 pacientes da Especialidade 1 e 20 pacientes da Especialidade 2. A ação G1 prescreveu, após 29 dias de processamento, a admissão de 5 pacientes da Especialidade 1 e 20 pacientes da Especialidade 2. O tempo de processamento para determinação da ação EHRA2 é evidentemente proibitivo.

4.3.6 Simulação da função de recompensa

A utilidade de uma ação $a \in A(x)$ dado um estado observado $x \in X$ é estimada no método EHRA (ver Subseção 3.6.1 do Capítulo 3) através da operação

$$\hat{Q}_H^{\Lambda}(x,a) = R(x,a) + \gamma \frac{1}{C} \sum_{j=1}^{C} (\sup_{\pi \in \Lambda} \hat{V}_{H-1}^{\pi}(x,a))_j .$$

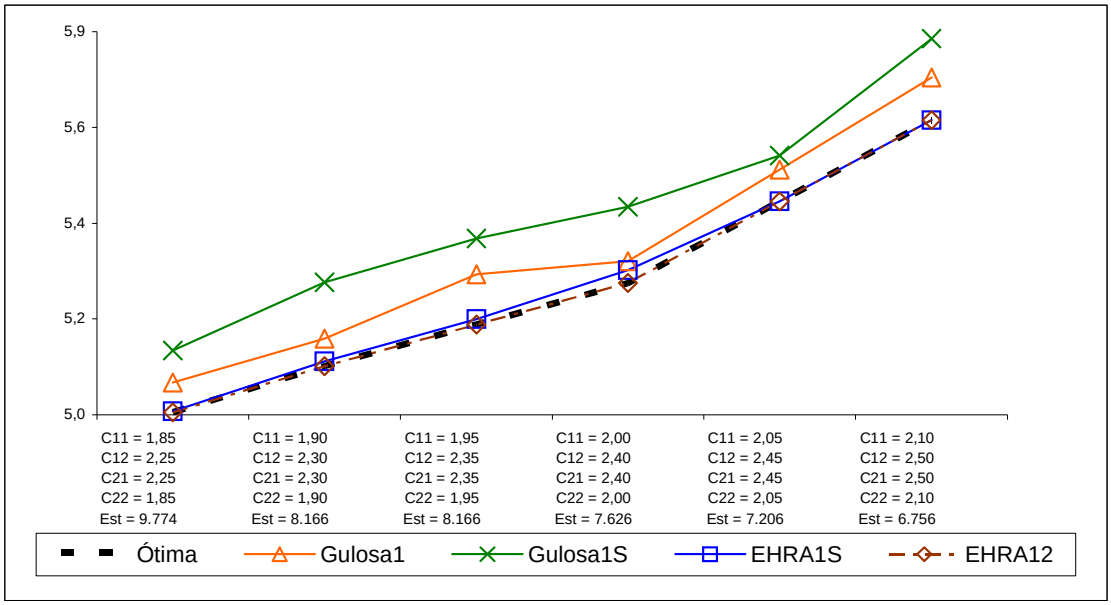
A última parcela da soma acima, $\gamma \frac{1}{C} \sum_{j=1}^{C} (\sup_{\pi \in \Lambda} \hat{V}_{H-1}^{\pi}(x,a))_j$, corresponde à recompensa descontada esperada após o primeiro período do horizonte de planejamento, e é estimada através de C simulações do funcionamento do sistema ao longo do horizonte de planejamento reduzido H para cada política de base $\pi \in \Lambda$. A primeira parcela da soma, correspondente à recompensa esperada no primeiro período do horizonte de planejamento, é calculada diretamente pela função de recompensa $R(x,a)$. Observa-se que, ao longo do processo para a estimativa do valor da utilidade de uma ação, o cálculo da função de recompensa é repetido $C \times H \times |\Lambda| + 1$ vezes. Dessa forma, a complexidade computacional do método EHRA depende da complexidade da função de recompensa (Subseção 3.6.5 do Capítulo 3).

Para PMDs como o proposto nesse estudo, em que $R(x,a)$ demanda grande esforço computacional proporcional às dimensões do modelo, pode-se pensar em obter a função de recompensa através de simulação. De maneira que, dado um estado observado, $x \in X$, e uma ação adotada, $a \in A(x)$, o valor esperado da recompensa no próximo período seja estimado por

$$\hat{R}(x,a) = \frac{1}{D} \sum_{j=1}^{D} R'_j(x,a),$$

em que $R'_j(x,a)$ representa o valor da $j-ésima$ recompensa simulada em um período a partir do par (x,a).

Gráfico 4.13 - Custo médio esperado por período sob controle de cada política.



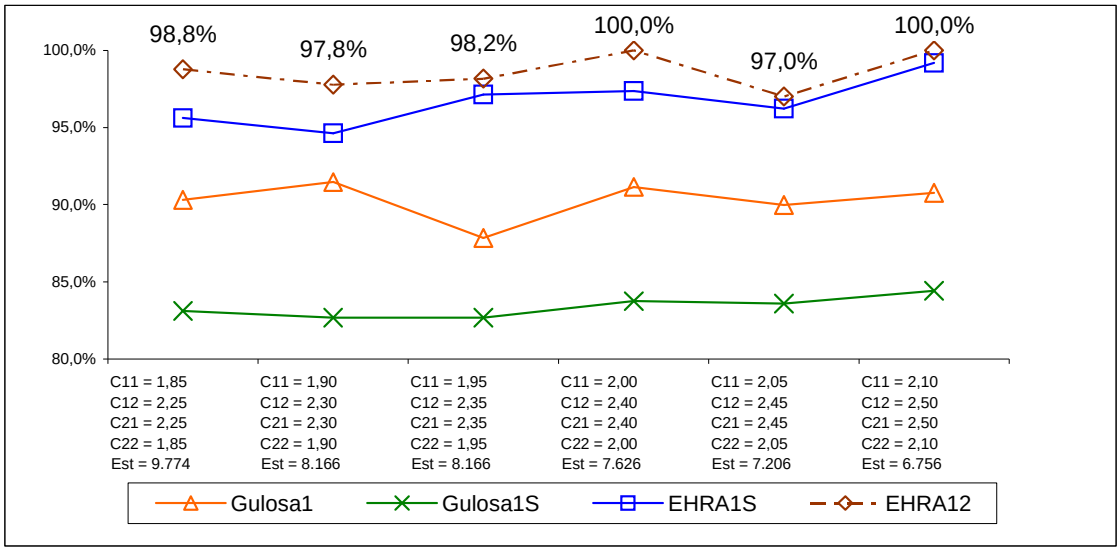
Com o objetivo de testar a qualidade da política de decisão gerada empregando-se o valor simulado da função de recompensa, implementou-se, para as seis configurações do modelo experimental apresentado na Subseção 4.2.1, tabelas 4.1 e 4.2, uma política EHRA1S, cuja política de decisão de base é a política gulosa G1S, que nada mais é que a política gulosa G1 (Subseção 4.2.2) obtida a partir da função de recompensa simulada $\hat{R}(x,a)$. No caso do modelo de controle de admissões o que se deseja é minimizar custos, portanto, sem perda da generalidade, a função de recompensa pode ser chamada de função de custo. As comparações entre as políticas G1, G1S e EHRA1S, em relação ao custo médio esperado por período e à proporção de concordâncias com a PO são apresentadas nos gráficos 4.13 e 4.14.

Observa-se que, como descrito anteriormente na Subseção 4.2.3, embora as ações da política EHRA1S tenham sido obtidas através do processo iterativo do método EHRA, o qual emprega custos descontados ao longo do horizonte de planejamento reduzido (ver Seção 3.6 do Capítulo 3), optou-se por se comparar os custos médios de cada política elaborada em relação ao horizonte

de planejamento infinito, obtidos através do AIV. No processo de geração das ações das políticas G1S e EHRA1S, sempre que foi necessário obter-se uma estimativa da função de custo esperado em um período do horizonte de planejamento, dado um estado observado e uma ação adotada, realizaram-se 5.000 simulações do valor desta função.

Observa-se no Gráfico 4.13 que a política gulosa que usa a função de custo simulada, G1S, apresenta maior custo médio esperado por período que a gulosa exata G1. A política EHRA1S, que emprega como política de base a G1S e usa a função de custo simulada em seu processo iterativo, apresentou bom desempenho, comparável ao da política EHRA12 que emprega como políticas de base a G1 e a G2 (ver Subseção 4.2.2).

Gráfico 4.14 - Concordância com as ações da política ótima.

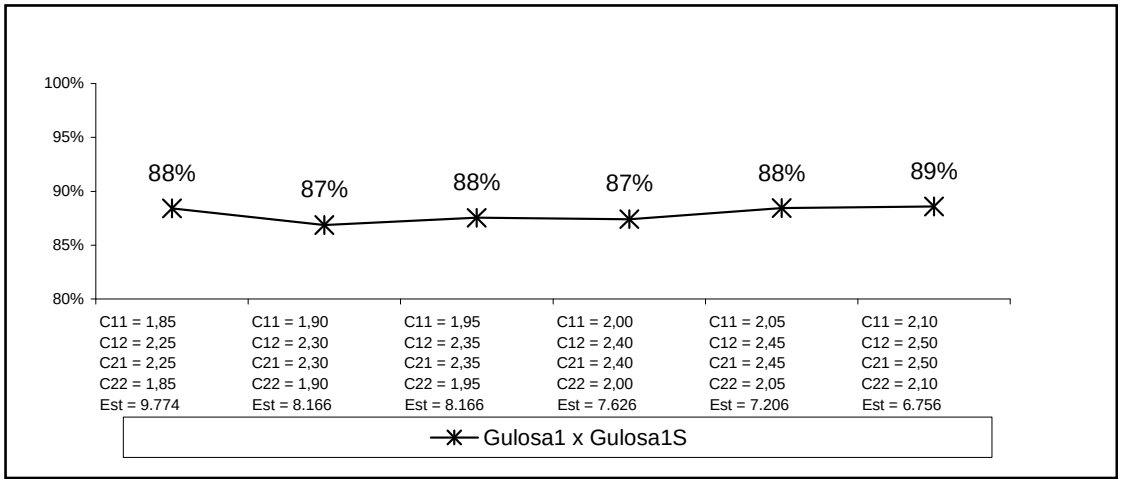


O Gráfico 4.14 apresenta a porcentagem de concordância das ações de cada política com as da PO. Observa-se que, embora a G1S tenha apresentado o pior desempenho, a EHRA1S teve uma porcentagem de concordância acima da G1 e próxima da política EHRA12, a qual apresentou sempre o melhor desempenho.

As percentagens de concordância entre as ações das políticas G1 e G1S nas instâncias distintas do modelo são apresentadas no Gráfico 4.15. Observa-se uma concordância variando entre 87% e 89%.

Com o objetivo de avaliar o ganho obtido em relação ao tempo de processamento e à possibilidade de tratamento de PMDs com maiores dimensões, pelo emprego da simulação do valor da função de custo, foram obtidos alguns parâmetros comparativos para obtenção de ações via G1S e EHRA1S, considerando-se as mesmas instâncias apresentadas na Tabela 4.11 da Subseção 4.3.5 referentes ao modelo estudado na Seção 4.3. O programa utilizado para obtenção dos resultados se encontra descrito no Apêndice B.3. Para obtenção das ações G1S, repetiu-se dez vezes o processo simulado de seleção de uma ação e apresentou-se na tabela a ação que foi selecionada o maior número de vezes. As ações do método EHRA1S foram selecionadas em uma única realização do método iterativo. Sempre que foi necessário obter uma estimativa da função de custo, realizaram-se 5.000 simulações do seu valor.

Gráfico 4.15 - Concordância entre as ações das políticas Gulosa 1 e Gulosa 1S.



Observa-se na Tabela 4.12 que as ações mais freqüentes prescritas pela G1S foram as mesmas prescritas pela G1. Porém, para as instâncias com maiores

dimensões dos espaços de estados e de ações a concordância entre as ações foi mais instável. Na maior instância, a ação G1S igualou a ação G1 em apenas três das dez repetições do processo. As ações prescritas pelo EHRA1S, obtidas uma única vez para cada instância, também foram concordantes com as ações da G1, com exceção da última instância. Embora não se tenha avaliado mais que uma ação da EHRA1S para cada instância, espera-se que, assim como ocorreu com a G1S, o aumento das dimensões do modelo cause instabilidade em relação à repetição das ações obtidas.

Tabela 4.12 - Ações prescritas pelas políticas G1, G1S e EHRA1S, e tempo de processamento para obtenção das ações para algumas instâncias do modelo de admissão de pacientes.

Estado:[a]	Max de adm[b]	Reentr[c]	Rec[d]	G1		G1S			EHRA1S		
				ação[e]	tp proc[f]	ação[g]	repet[h]	tp proc	ação	itera[i]	tp proc
{1,1,1,0,1,1,1,0}	{2,2}	{2,2}	{8,14}	{2,1}	00:00'10h	{2,1}	7	00:00'01h	{2,1}	36	00:03'23h
{1,1,1,0,1,1,1,0}	{2,2}	{2,2}	{9,15}	{2,2}	00:00'10h	{2,2}	10	00:00'01h	{2,2}	27	00:04'17h
{1,1,1,0,1,1,1,0}	{2,2}	{3,3}	{8,14}	{1,0}	00:00'26h	{1,0}	7	00:00'01h	{1,0}	25	00:04'12h
{1,1,1,0,1,1,1,0}	{3,3}	{2,2}	{8,14}	{3,0}	00:00'41h	{3,0}	7	00:00'02h	{3,0}	25	00:06'46h
{2,2,2,0,2,2,2,0}	{2,2}	{2,2}	{8,14}	{0,2}	00:03'37h	{0,2}	8	00:00'01h	{0,2}	41	00:06'24h
{2,2,2,0,2,2,2,0}	{3,3}	{3,3}	{9,15}	{0,2}	00:19'31h	{0,2}	4	00:00'02h	{0,2}	29	00:09'17h
{9,9,9,0,9,9,9,0}	{20,20}	{16,15}	{60,80}	{5,20}	696:19'31h	{5,20}	3	00:00'34h	{8,17}	36	25:28'12h

a - Estado: $\{E_1^1, E_2^1, E_3^1, E_4^1, E_1^2, E_2^2, E_3^2, E_4^2\}$
b - Máximo de admissões em cada especialidade
c - Número de reentradas em cada especialidade
d - Quantidade disponibilizada de cada recurso: consultas e sessões de fisioterapia
e - Ação: $\{S^1, S^2\}$
f - Tempo de processamento
g - Ação que mais foi selecionada em 10 simulações.
h - Número de vezes que a ação foi selecionada em 10 simulações.
i - Número de iterações

O tempo de processamento da G1S é bem inferior ao da G1. Enquanto a G1 gastou mais que 29 dias para apresentar uma ação para a sétima instância, a G1S gastou apenas 34 segundos. Embora se saiba que a precisão da G1S é afetada pelo aumento das dimensões do problema modelado, torna-se possível repetir muitas vezes o processo de geração da G1S para se escolher a ação que mais vezes resultar destas repetições, e espera-se que essa ação seja a mais próxima da ação G1. O tempo de processamento para as ações geradas

pela EHRA1S, embora seja menor que o da G1 na última instância testada, ainda é proibitivo para modelos com maiores dimensões. Para a sétima instância do modelo, o EHRA1S gastou mais que 25 horas para prescrever uma ação.

4.4 Considerações sobre os resultados

As duas principais vantagens do método EHRA – evolutivo em horizonte de planejamento reduzido e amostrado – são as seguintes: (1) através da amostragem de horizontes de funcionamento do sistema, o método pode estimar os custos de controle das ações sem a necessidade de que sejam avaliados todos os estados e sem a necessidade de resolução de sistemas de equações lineares; e (2) através da busca evolutiva no espaço de ações, o método é capaz de encontrar melhores ações sem que seja necessário avaliar em cada iteração todas as ações para cada estado. A principal desvantagem do método EHRA é a dependência de boas políticas de base para se obterem ações de qualidade. Uma política EHRA será tão melhor quanto melhor forem as políticas de base empregadas em seu processo iterativo.

Os gráficos 4.1 e 4.12 da Subseção 4.2.4, particularmente, mostram que as políticas EHRA geraram decisões para o PMD formulado que melhoraram o desempenho das políticas de base empregadas, tanto em termos de custos quanto em termos de concordância de ações com a política ótima. A política EHRA12, que empregou como base as duas políticas gulosas, G1 e G2, foi mais constante com relação à manutenção de um desempenho próximo ao da política ótima, evidenciando que a utilização de maior quantidade de “boas” políticas de base, apesar de onerar o tempo de processamento do método, pode gerar melhores soluções.

Os resultados da Seção 4.2 confirmam que o método EHRA é capaz de fornecer, para PMDs com grandes espaços de estados e de ações, políticas de decisão *on-line,* subótimas, que melhoram ou no mínimo equivalem às soluções das políticas de base empregadas e em um tempo de processamento significativamente menor do que o que seria necessário para obter uma política ótima através do tradicional AIV, como mostram as tabelas 4.4 e 4.5.

Em relação à avaliação do modelo de controle de admissões de pacientes, apresentada na Seção 4.3, constatou-se que os parâmetros básicos do modelo (padrões de atendimento, utilização dos recursos, probabilidades, estado observado), em um hospital com um bom sistema de registros, preferencialmente informatizado, podem ser obtidos com simplicidade. Constatou-se, também, que o modelo geral proposto no Capítulo 2 pode ser facilmente modificado para acomodar as particularidades dos serviços a serem considerados, como foi o caso da transformação do modelo de 5 padrões de atendimento em um modelo de 4 padrões com uma taxa constante de reentradas.

Percebe-se, analisando-se a estrutura do modelo testado, que o espaço de estados cresce rapidamente quando se considera um maior número de padrões de atendimento e que um maior número de especialidades faz crescer fortemente tanto o espaço de estados como o espaço de ações. Os parâmetros quantitativos do modelo também contribuem para um rápido aumento de suas dimensões: o crescimento da disponibilidade de recursos faz aumentar rapidamente o espaço de estados e o acréscimo da capacidade de admissões aumenta o espaço de ações.

As instâncias do modelo apresentadas na Tabela 4.11 geraram espaços de estados com mais de dez milhões de estados. Anteriormente, na Tabela 4.4 da Seção 4.2, pode-se observar que o AIV gastou mais do que 1 dia para obter uma política de decisão para um modelo com pouco mais de 200.000 estados.

Portanto, nota-se claramente que a aplicação do AIV é inviável para o modelo de controle de admissões, como foi proposto na Seção 4.3. O método guloso G1 e o método EHRA2, embora tenham sido capazes de gerar resultados para as pequenas instâncias apresentadas na Tabela 4.11, também se mostraram inviáveis para utilização em situações práticas mais próximas da realidade de um hospital, como a situação apresentada na sétima instância, a qual retrata a estrutura de uma equipe de atendimento formada por 1 médico e 2 fisioterapeutas.

Os resultados da Tabela 4.11 mostraram que apenas o método guloso G2, apesar de ser o que gera as políticas com pior desempenho, como mostraram os resultados da Seção 4.2, é capaz de apresentar ações para o modelo proposto em tempo hábil para aplicação prática em situações realistas. Isso ocorre porque o método de obtenção da política G2, apresentado na Subseção 4.2.2, não envolve o cálculo das probabilidades de transição entre estados para a estimativa do custo esperado no próximo período, como ocorre com a política EHRA2 e com a gulosa G1. A obtenção de ações para a política EHRA2 e para a política gulosa G1 torna-se computacionalmente onerosa com o crescimento dos espaços de estados, pois, considerando-se a função de recompensa definida para o modelo proposto, a estimativa do custo esperado no próximo período requer que se determinem todos os estados que podem ser atingidos a partir do par (estado observado; ação), que se utilize uma convolução de distribuições multinomiais para o cálculo da probabilidade de alcançar cada estado atingível, para, finalmente, calcular-se o custo esperado no próximo período.

Na Subseção 4.3.6 apresentou-se a possibilidade de se simular a função de recompensa em um período, na tentativa de se contornar a dificuldade com a complexidade do cálculo exato dessa função. Testou-se a qualidade da política gulosa simulada G1S e da política gerada pelo método simulado EHRA1S. Para o modelo pequeno estudado na Seção 4.2 a política EHRA1S apresentou

boa qualidade, com resultados próximos ao da política ótima (ver gráficos 4.13 e 4.14). Entretanto, para o modelo estudado na Seção 4.3, o tempo de processamento para obtenção de uma ação através do método EHRA1S, apesar de ser bastante inferior ao do EHRA2, mostrou-se ainda proibitivo para instâncias mais realistas, como a sétima instância da Tabela 4.12. O tempo de processamento para obtenção de ações para a G1S, embora esta política tenha mostrado alguma instabilidade em relação à concordância com as ações prescritas pela G1 exata, permite o tratamento de instâncias mais realistas.

Concluindo, os resultados deste capítulo mostram que o modelo de controle de admissões proposto tem uma estrutura flexível, com parâmetros que podem ser obtidos do próprio sistema de informações do hospital. Sendo observado um estado, a ação determinada pode servir prontamente para orientar o gestor na tomada de decisão operacional. Porém, os melhores métodos estudados para obtenção de políticas de decisão, em ordem de qualidade das políticas, o AIV, os métodos EHRA12, EHRA2, EHRA1, EHRA1S e o G1, em virtude da complexidade da função de recompensa como foi definida para o modelo, não foram capazes de oferecer soluções para instâncias mais próximas de situações realistas, restando ao gestor hospitalar, antes que sejam desenvolvidos melhores métodos, a solução da política gulosa G2 e da política simulada G1S.

CAPÍTULO 5

CONCLUSÃO

Controlar a admissão de pacientes eletivos é uma atividade-chave que permite ao gestor de um hospital balancear a demanda dos pacientes por atendimento e a disponibilidade dos recursos hospitalares (Capítulo 1). Com a finalidade de prover uma ferramenta de auxílio à tomada de decisão nesse contexto, formulou-se o controle de admissões de pacientes eletivos como um PMD (Capítulo 2). O modelo formulado tem como característica gerar grandes espaços de estados e de ações, o que torna o emprego dos métodos tradicionais de obtenção de políticas de decisões ótimas impraticável para instâncias realistas do modelo. Com o intuito de viabilizar a aplicação prática do modelo proposto, foram pesquisados métodos alternativos existentes capazes de tratar PMDs com grandes dimensões, mesmo que sem a garantia de uma solução ótima. Maior enfoque foi dado aos métodos amostrados em horizonte de planejamento reduzido e aos métodos de busca evolutiva, baseados nos algoritmos genéticos. Um novo método combinado, evolutivo em horizonte de planejamento reduzido e amostrado, foi desenvolvido (Capítulo 3) e testado para o modelo proposto (Capítulo 4).

Nas seções subsequentes, apresentam-se as considerações finais e sugestões para pesquisas futuras sobre o modelo proposto (Seção 5.1), sobre o método proposto (Seção 5.2) e sobre o estudo de uma forma geral (Seção 5.3).

5.1 O PMD para o controle de admissão de pacientes eletivos

Através do PMD proposto no Capítulo 2, obtêm-se políticas de decisão cujas ações determinam a quantidade de pacientes de cada especialidade a ser admitida no próximo período, dado o estado observado de funcionamento do

hospital no último período (número de pacientes tratados em cada padrão de atendimento). Nos instantes de decisão, o estado observado e a ação adotada determinam as probabilidades de transição para os próximos estados e o custo esperado no próximo período. Os custos do modelo foram definidos em função dos desvios da utilização esperada dos recursos hospitalares em relação aos níveis desejados de consumo, sendo consideradas as importâncias relativas de cada recurso.

A política de decisão ótima, gerada pelo AIV (Subseção 4.2.4 do Capítulo 4), minimiza o custo conjunto dos desvios em relação aos níveis desejados de utilização dos recursos. A política ótima, nos protótipos testados, foi capaz de prevenir ociosidade ou utilização excessiva dos recursos, observando suas importâncias relativas. Certamente, a política ótima é restritiva; em muitos estados a ação indicada é a de não se admitirem pacientes no próximo período; mas, por outro lado, sob controle da política ótima, espera-se que seja maior a chance de um paciente ser atendido sem que ocorra interrupção do seu curso de tratamento por escassez de um dos recursos necessários ao atendimento.

A formulação do modelo proposto (espaços de estados e de ações, padrões de atendimento, estruturas estocástica e de custos) é bastante flexível, podendo ser facilmente ajustada para se adaptar a possíveis situações específicas de cada hospital. Assim, sugere-se que sejam exploradas derivações deste modelo original, representando-se outras situações, como, por exemplo, a inclusão do serviço de emergência, a consideração no modelo do tamanho da fila de espera e a priorização de pacientes da lista de espera em função da gravidade dos seus casos.

Em relação à estrutura da política de decisão ótima gerada a partir do modelo, nos exemplos considerados, não foi identificado um padrão regular de admissões do tipo encontrado, por exemplo, nas políticas de decisão da área

de controle de estoques (AXSÄTER, 2006). Em virtude do aumento rápido do espaço de estados, à medida que se abordam instâncias mais realistas do modelo, seria importante continuar a investigar se a política ótima de decisão delineia regiões no espaço de estados onde se aplica a mesma ação. O reconhecimento de uma estrutura regular determinada pela política ótima poderia viabilizar a aplicação de métodos de decomposição ou redução de PMDs de grandes dimensões (BOUTILIER *et al.*, 1995; CAO *et al.*, 2002; DEAN; GIVAN, 1997; GIVAN *et al.*, 2003; HAUSKRECHT *et al*., 1998; KUSHNER; CHEN, 1974; LIN, 1997; MEULEAU *et al.*, 1998; PARR, 1998b), facilitando, assim, o tratamento de instâncias maiores do modelo.

A distribuição de probabilidades multinomial, que rege a dinâmica estocástica das transições entre estados do modelo, gera uma formulação matemática complexa e de difícil tratamento computacional. Empregando-se a distribuição multinomial, não é tarefa fácil elaborar um programa computacional que seja capaz de calcular as probabilidades de transição entre estados para modelos com configurações distintas em relação ao número de recursos, de padrões de atendimento e de especialidades. Qualquer alteração em um dos parâmetros de configuração do modelo requer a reformulação minuciosa do código do programa. Consequentemente, seguem duas sugestões para estudos futuros. Uma é a de tornar mais eficientes os códigos dos programas desenvolvidos no presente estudo (apêndices B.1, B.2 e B.3). Outra é a de estudar a viabilidade da aproximação da distribuição multinomial por outra distribuição de probabilidades, talvez empregando a distribuição de Poisson (McDONALD, 1980; BLANC, 1991). Essa aproximação poderia simplificar os cálculos necessários para obtenção das probabilidades de transição entre estados e do custo esperado em um período de planejamento. Possivelmente, uma distribuição de probabilidades mais simples permitiria o tratamento computacional de instâncias maiores do modelo.

A função de custos do modelo proposto (Subseção 2.3.5 do Capítulo 2), que

utiliza as probabilidades de transição entre estados para estimar o custo esperado no período até o próximo instante de decisão, inviabiliza, mesmo empregando-se o método EHRA ou a política gulosa G1, a obtenção de políticas de decisão para instâncias realistas do modelo, as quais geram grandes espaços de estados. Assim, sugere-se que sejam investigadas funções de custos alternativas, que permitam a aplicação do método EHRA a instâncias maiores do modelo. Poder-se-ia avaliar, por exemplo, a formulação da função de custos proposta para a política gulosa G2 (Subseção 4.2.2 do Capítulo 4) como substituta para a função de custos original do modelo. Um aprofundamento no estudo de métodos simulados para a estimativa da função de custos, como o que foi proposto na Subseção 4.3.6, seria também um bom projeto de pesquisa. Como ressalta Chang (2001), em um processo encadeado de decisões, se a ordem do valor real da utilidade das ações possíveis para cada estado observado for mantida, independentemente da precisão da estimativa desse valor, o decisor que escolher a melhor ação de acordo com essa ordem estará, em última análise, agindo de forma ótima.

5.2 O método EHRA para solução de PMDs

Partindo-se das demonstrações de Chang (2001) sobre a convergência do método *parallel rollout* para uma política com desempenho melhor ou igual ao desempenho das políticas de base e de Hu *et al.* (2007) sobre a convergência do método ERPS para uma política ótima, é possível que o método EHRA, assim como o *parallel rollout*, convirja para uma política que melhora as políticas de base. Os resultados obtidos no Capítulo 4, relativos ao PMD proposto para o controle de admissões de pacientes eletivos, confirmam que a meta-heurística EHRA proporciona uma política de decisão capaz de melhorar a qualidade das políticas de base empregadas no método, aproximando seu desempenho ao da política ótima. Seria interessante que se desenvolvesse,

em estudo futuro, uma demonstração formal da convergência do método EHRA.

Em virtude da estrutura da dinâmica estocástica do modelo de controle de admissões, o método EHRA, apesar de ter suportado instâncias maiores do que os métodos tradicionais de otimização, não foi capaz de gerar ações para instâncias mais realistas do modelo. Com o intuito de avaliar o real potencial do método EHRA para solucionar PMDs com espaços de estados e de ações simultaneamente com grandes dimensões, sugere-se que em estudos futuros o método seja explorado em outros modelos com características estocásticas mais favoráveis ou no próprio modelo proposto, com alguma simplificação na estrutura da função de custos esperados ou através de métodos simulados de estimação dessa função.

Nos experimentos do Capítulo 4, não foi realizada uma análise de sensibilidade para se avaliar o impacto, sobre a qualidade das políticas de decisão e sobre a capacidade de se abordar PMDs com grandes dimensões, causado pela variação dos valores dos parâmetros empregados para o controle empírico do processo iterativo do método EHRA (Subseção 3.6.5 do Capítulo 3). Sugere-se que esta análise de sensibilidade seja realizada em estudos futuros.

5.3 Considerações finais

A modelagem apresentada neste estudo para o controle de admissões de pacientes eletivos como um PMD é uma abordagem inovadora, a qual pode proporcionar um processo decisório mais eficiente em relação ao balanço entre a entrada de novos pacientes e a utilização dos recursos hospitalares disponíveis. Combinado a um método eficiente para solução de PMDs com grandes dimensões, esse modelo tem amplo potencial para aplicação.

Considerando o longo período de tempo percebido em relação aos últimos trabalhos científicos publicados sobre a modelagem markoviana aplicada ao controle do fluxo de pacientes em hospitais, espera-se que este estudo possa ter lançado novas luzes sobre o assunto e que venha a revigorar essa importante linha de pesquisa.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAN, I.; VISSERS, J. Patient mix optimization in hospital admission planning: a case study. **International Journal of Operations and Production Management**, v. 22, n. 4, p. 445-461, 2002.

ARORA, S.; BARAK, B. **Computational complexity**: a modern approach. New York, NY: Cambridge University Press, 2009. 594 p. ISBN 9780521424264.

AXSÄTER, S. **Inventory control**. 2. ed. New York, NY: Springer, 2006. 336 p. ISBN 0387332502.

BAGLIETTO, M.; PARISINI, T.; ZOPPOLI, R. Neural approximators and team theory for dynamic routing: a receding horizon approach. In: IEEE CONFERENCE ON DECISION AND CONTROL, n. 38, 7 Dec. 1999, Phoenix. **Proceedings…**, New York: IEEE Publisher, 1999. v. 4, p. 3283-3288, ISBN 0780352505.

BARASH, D. A genetic search in policy space for solving Markov decision processes. In: AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE SYMPOSIUM ON SEARCH TECHNIQUES FOR PROBLEM SOLVING UNDER UNCERTAINTY AND INCOMPLETE INFORMATION, 22 Mar. 1999, Stanford. **Proceedings…**, Menlo Park: AAAI Press, 1999. p. 8-12, ISBN 1577351061. Disponível em: < http://www.aaai.org/Papers/Symposia/Spring/1999/SS-99-07/SS99-07-002.pdf >. Acesso em: 10 jan. 2010.

BELDERRAIN, M. **Abordagem estocástica no processo de admissões hospitalares**. 90 p. Tese (Doutorado em Organização Industrial) – Instituto Tecnológico de Aeronáutica, São José dos Campos, 1998. Disponível em: < http://www.bd.bibl.ita.br/tesesdigitais/000407905.pdf >. Acesso em: 13 abr. 2010.

BELLMAN, D. **Dynamic programming**. Princeton, NJ: Princeton University Press, 1957. 342 p. ISBN 069107951X.

BERNSTEIN, D.; GIVAN, R.; IMMERAN, N.; ZILBERSTEIN, S. The complexity of decentralized control of Markov decision processes. **Mathematics of Operations Research**, v. 27, n. 4, p. 819-840, 2002.

BERTSEKAS, D.; CASTAÑON, D. Adaptive aggregation methods for infinite horizon dynamic programming. **IEEE Transactions on Automatic Control**, v. 34, n. 6, p. 589-598, 1989.

BERTSEKAS, D.; CASTAÑON, D. Rollout algorithms for stochastic scheduling problems. **Journal of Heuristics**, v. 5, p. 89-108, 1999.

BERTSEKAS, D.; SHREVE, S. **Stochastic control:** the discrete time case. New York, NY: Academic Press, 1978. 330 p. ISBN 0120932601.

BERTSEKAS, D.; TSITSIKLIS, J. **Neuro-dynamic programming**. Belmont, MA: Athena Scientific, 1996. 512 p. ISBN 1886529108.

BES, C.; LASSERRE, J. An on-line procedure in discounted infinite-horizon stochastic optimal control. **Journal of Optimization Theory and Applications**, v. 50, n.1, p. 61-67, 1986.

BLANC, S. On the Poisson approximation for some multinomial distributions. **Statistics & Probability Letters**, v. 11, n. 1, p. 1-6, 1991.

BLONDEL, V.; TSITSIKLIS, J. A survey of computational complexity results in systems and control. **Automatica**, v. 36, n. 9, p. 1249-1274, 2000.

BOUTILIER, C.; DEARDEN, R.; GOLDSZMIDT, M. Exploiting Structure in Policy Construction. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, n. 14, 20 Aug. 1995, Montreal. **Proceedings…**, San Francisco: Morgan Kaufmann Publishers, 1995. v. 2, p. 1104-1111. Disponível em: < http://ijcai.org/Past%20Proceedings/IJCAI-95-VOL2/PDF/012.pdf >. Acesso em: 10 jan. 2010.

CAO, X.; REN, Z.; BHATNAGAR, S.; FU, M.; MARCUS, S. A time aggregation approach to Markov decision processes. **Automatica**, v. 38, n. 6, p. 929-943, 2002.

CHAND, S.; HSU, V.; SETHI, S. Forecast, solutions, and rolling horizons in operations management problems: a classified bibliography. **Manufacturing & Service Operations Management**, v. 4, n. 1, p. 25-43, 2002.

CHANG, H. **On-line sampling-based control for network queueing problems**. 169 p. Tese (Doutorado em Engenharia Elétrica e da Computação) – Purdue University, West Lafayette, 2001. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.9369&rep=rep1&type=pdf >. Acesso em: 08 jan. 2010.

CHANG, H.; FU, M.; HU, J.; MARCUS, S. An adaptive sampling algorithm for solving Markov decision processes. **Operations Research**, v. 53, n. 1, p. 126-139, 2005a.

CHANG, H.; FU, M.; MARCUS, S. **Simulation-based algorithms for Markov decision processes**. London: Springer-Verlag, 2007. 189 p. ISBN 9781846286896.

CHANG, H.; GIVAN, R.; CHONG, E. Parallel rollout for on-line solution of partially observable Markov decision processes. **Discrete Event Dynamic Systems: Theory and Application**, v. 15, n. 3, p. 309-341, 2004.

CHANG, H.; LEE, H.; FU, M.; MARCUS, S. Evolutionary policy iteration for solving Markov decision processes. **IEEE Transactions on Automatic Control**, v. 50, n. 11, p. 1804-1808, 2005b.

CHANG, H.; MARCUS, S. Approximate receding horizon approach for Markov decision processes: average reward case. **Journal of Mathematical Analysis and Applications**, v. 286, n. 2, p. 636-651, 2003.

CHI, M. The long-run behavior of Markov chains. **Linear Algebra and Its Applications**, v. 244, n. 1-3, p. 111-121, 1996.

CHIN, H.; JAFARI, A. Genetic algorithm methods for solving the best stationary policy of finite Markov decision processes. In: SOUTHEASTERN SYMPOSIUM ON SYSTEM THEORY, n. 30, 8 Mar. 1998, Morgantown, WV. **Proceedings…**, New York: IEEE Publisher, 1998. v. 2, p. 538-543, ISBN 0780345479.

CHITASHVILI, R. A controlled finite Markov chain with an arbitrary set. of decisions. **Theory of Probability and Its Applications**, v. 20, n. 11, p. 839-846, 1975.

CHONG, E.; GIVAN, R.; CHANG, H. A framework for simulation-based network control via hindsight optimization. In: IEEE CONFERENCE ON DECISION AND CONTROL, n. 39, 12 Dec. 2000, Sydney. **Proceedings…**, New York: IEEE Publisher, 2001. v. 2, p. 1433-1438, ISBN 0780366387. Disponível em: < http://cobweb.ecn.purdue.edu/~givan/brown/papers/cdc00.pdf > Acesso em: 10 jan. 2010.

COOPER, W.; HENDERSON, S.; LEWIS, M. Convergence of simulation-based policy iteration. **Probability in the Engineering and Informational Sciences**, v. 17, n. 2, p. 213-234, 2003.

COPPERSMITH, D.; WU, C. Conditions for weak ergodicity of inhomogeneous Markov chains. **Statistics & Probability Letters**, v. 78, n. 17, p. 3082-3085, 2008.

CÔTÉ, M.; STEIN, W. An Erlang-based stochastic model for patient flow. **Omega**, v. 28, n. 3, p. 347-359, 2000.

DAVIS, C. The computer generation of multinomial random variates. **Computational Statistics and Data Analysis**, v. 16, n. 2, p. 205-217, 1993.

DEAN, T.; GIVAN, R. Model minimization in Markov decision processes. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, n. 14, 27 July 1997, Providence. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1997. p. 106-111, ISBN 9780262510950. Disponível em: < http://eprints.kfupm.edu.sa/51224/1/51224.pdf > Acesso em: 10 jan. 2010.

DEAN, T.; GIVAN, R.; LEACH, S. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 13, 1 Aug. 1997, Providence. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1997. p. 124-131, ISBN 1558604855. Disponível em: < http://eprints.kfupm.edu.sa/51231/1/51231.pdf > Acesso em: 10 jan. 2010.

DEAN, T.; KIM, K.; GIVAN, R. Solving stochastic planning problems with large state and action spaces. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE PLANNING SYSTEMS, n. 4, 7 June 1998, Pittsburgh. **Proceedings…** Menlo Park: AAAI Press, 1998. p. 102-110, ISBN 1577350529. Disponível em: < http://cobweb.ecn.purdue.edu/~givan/papers/aips98.ps > Acesso em: 10 jan. 2010.

DEAN, T.; LIN, S. Decomposition techniques for planning in stochastic domains. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, n. 14, 20 Aug. 1995, Montreal. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1995. v. 2, p. 1121-1127. Disponível em: < http://ijcai.org/Past%20Proceedings/IJCAI-95-VOL2/PDF/014.pdf >. Acesso em: 10 jan. 2010.

FANG, H.; CAO, X. Potential-based on-line policy iteration algorithms for Markov decision processes. **IEEE Transactions on Automatic Control**, v. 49, n. 11, p. 493-505, 2004.

FEINBERG, E. The existence of a stationary $\varepsilon - optimal$ policy for a finite Markov chain. **Theory of Probability and Its Applications**, v. 23, n. 2, p. 297-313, 1979.

FETTER, R.; THOMPSON, J. A decision model for design and operation of a progressive patient care hospital. **Medical Care**, v. 7, n. 6, p. 450-462, 1969.

FEDERGRUEN, A.; TZUR, M. Detection of minimal forecast horizons in dynamic programs with multiple indicators of the future. **Naval Research Logistics**, v. 43, n. 2, p. 169-189, 1998.

FISHMAN, G. **A first course in Monte Carlo**. Belmont: Duxbury Press, 2005. 350 p. ISBN 053442046X.

GEMMEL, P.; VAN DIERDONCK, R. Admission scheduling in acute care hospitals: does the practice fit with the theory. **International Journal of Operations and Production Management**, v. 19, n. 9, p. 863-878, 1999.

GIVAN, R.; DEAN, T.; GREIG, M. Equivalence notions and model minimization in Markov decision processes. **Artificial Intelligence**, v. 147, n. 1-2, p. 163-223, 2003.

GIVAN, R.; LEACH, S.; DEAN, T. Bounded parameter Markov decision processes. **Artificial Intelligence**, v. 122, n. 1-2, p. 71-109, 2000.

GRANAS, A.; DUGUNDJI, J. **Fixed point theory**. New York, NY: Springer-Verlag, 2003. 690 p. ISBN 0387001735.

GRINOLD, R. Finite horizon approximations of infinite horizon linear programs. **Mathematical Programming**, V22, n. 1, p. 1-17, 1997.

GROOT, P. **Decision support for admission planning under multiple resource constraints**. 181 p. Tese (Doutorado em Engenharia Industrial) – Eindhoven University of Technology, Eindhoven, 1993. Disponível em: < http://alexandria.tue.nl/extra3/proefschrift/PRF9B/9303423.pdf >. Acesso em: 08 jan. 2010.

HARBISON, S.; STEELE, G. **C**: a reference manual. 5. ed. Englewood Cliffs, NJ: Prentice Hall, 2002. 560 p. ISBN 013089592X

HARDY, G.; LITTLEWOOD, J.; PÓLYA, G. **Inequalities**. 2. ed. Cambridge: Cambridge University Press, 1998. 340 p. ISBN 0521358809.

HARTIGAN, J.; WONG, M. Algorithm AS 136: a k-means clustering algorithm. **Applied Statistics**, v. 28, n. 1, p. 100-108, 1979.

HAUSKRECHT, M.; MEULEAU, N.; DEAN, T. Hierarchical solution of Markov decision processes using macro-actions. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 14, 24 July 1998, Madison. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1998. p. 220-229, ISBN 155860555X. Disponível em: < http://eprints.kfupm.edu.sa/43895/1/43895.pdf >. Acesso em: 08 jan. 2010.

HERNÁNDEZ-LERMA, O.; LASSERRE, J. A forecast horizon and a stopping rule for general Markov decision processes. **Journal of Mathematical Analysis and Applications**, v. 132, p. 388-400, 1988.

HERNÁNDEZ-LERMA, O.; LASSERRE, J. Error bounds for rolling horizon policies in discrete-time Markov control processes. **IEEE Transactions on Automatic Control**, v. 35, n. 10, p. 118-1124, 1990.

HERNÁNDEZ-LERMA, O.; LASSERRE, J. **Discrete-time Markov control processes**: basic optimality criteria. New York, NY: Springer-Verlag, 1996. 216 p. ISBN 0387945792.

HERSHEY, J.; WEISS, E.; COHEN, M. A stochastic service network model with application to hospital facilities. **Operations Research**, v. 29, n. 1, p. 1-22, 1981.

HINCAPIÉ, D.; OSPINA, J.; GRISALES, H.; ARROYAVE, M.; VALENCIA, M.; GONZÁLEZ, G. Análisis markoviano de un proceso de estancia hospitalaria en un hospital de tercer nivel de complejidad. **Revista Facultad Nacional de Salud Pública**, v. 22, n. 1, p. 61-71, 2004.

HOBSBAWM, E. **A era das revoluções**. 25. ed. São Paulo, SP: Paz e Terra, 2009. 535 p. ISBN 9788577530991.

HU, J.; FU, M.; RAMEZANI, V.; MARCUS, S. An evolutionary random policy search algorithm for solving Markov decision processes. **Informs Journal on Computing**, v. 19, n. 2, p. 161-174, 2007.

JAIN, R.; VARAIYA, P. Simulation-based uniform value function estimates of Markov decision processes. **SIAM Journal on Control and Optimization**, v. 45, n. 5, p. 1633-1656, 2006.

KAELBLING, L.; LITTMAN, M.; MOORE, A. Reinforcement learning: a survey. **Journal of Artificial Intelligence Research**, v. 4, p. 237-285, 1996.

KAO, E. A semi-Markov model to predict recovery of coronary patients. **Health Services Research**, v. 7, n. 3, p. 191-208, 1972.

KAO, E. A semi-Markovian population model with application to hospital planning. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 3, n. 4, p. 327-336, 1973.

KAO E. Modeling the movement of coronary patients within a hospital semi-Markov process. **Operations Research**, v. 22, n. 4, p. 683-699, 1974.

KAPADIA, A.; VINEBERG, S.; ROSSI, D. Predicting course of treatment in a rehabilitation hospital: a Markovian model. **Computers & Operations Research**, v. 12, n. 5, p. 459-469, 1985.

KAPADIA, A.; CHAN, W.; SACHDEVA, R. Predicting duration of stay in a pediatric intensive care unit: a Markovian approach. **European Journal of Operational Research**, v. 124, n. 2, p. 353-359, 2000.

KEARNS, M.; MANSOUR, Y.; NG, A. Approximate planning in large POMDPs via reusable trajectories. In: CONFERENCE ON ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, n. 10, 29 Nov. 1999, Denver. **Proceedings…** Cambridge: MIT Press, 2000. p. 1001-1007, ISBN 0262194503. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/ download?doi=10.1.1.50.2708&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

KEARNS, M.; MANSOUR, Y.; NG, A. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. **Machine Learning**, v. 49, n. 2-3, p. 193-208, 2002.

KIM, K.; DEAN, T. Solving factored Markov decision processes using non-homogeneous partitions. **Artificial Intelligence**, v. 147, n. 1-2, p. 225-251, 2003.

KOGAN, J.; NICHOLAS, C.; TEBOULLE, M. (Ed.). **Grouping multidimensional data**: recent advances in clustering. New York, NY: Springer, 2006. 268 p. ISBN 354028348X.

KOLLER, D.; PARR, R. Computing factored value functions for policies in structured MDPs. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, n. 16, 31 July 1999, Stockholm. **Proceedings…**, San Francisco: Morgan Kaufmann Publishers, 1999. v. 2, p. 1332-1339, ISBN 1558606130. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/ download?doi=10.1.1.89.5203&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

KOLLER, D.; PARR, R. Policy iteration for factored MDPs. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 16, 30 June 2000, Stanford. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 2000. p. 326-334, ISBN 1558607099. Disponível em: < https://eprints.kfupm.edu.sa/ 58258/1/58258.pdf >. Acesso em: 10 jan. 2010.

KUSHNER, H.; CHEN, C. Decomposition of systems governed by Markov chains. **IEEE Transactions on Automatic Control**, v. 19, n. 5, p. 501-507, 1974.

KUSTERS, R.; GROOT P. Modelling resource availability in general hospitals design and implementation of a decision support model. **European Journal of Operational Research**, v. 88, n. 3, p. 428-445, 1996.

LANE, T.; KAELBLING, L. Approaches to macro decompositions of large Markov decision process planning problems. In: PHOTOICS EAST: INTELLIGENT SYSTEMS AND ADVANCED MANUFACTURING – MOBILE ROBOTS, n. 16, 28 Oct. 2001, Boston. **Proceedings…** Bellingham: SPIE, 2002. v. 4573, p. 104-113, ISBN 9780819443014. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.3769&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

LAW, A.; KELTON, W. **Simulation modeling and analysis**. 3. ed. New York, NY: McGraw-Hill, 2000. 800 p. ISBN 0070592926.

LIN, S. **Exploiting structure for planning and control**. 177 p. Tese (Doutorado em Ciência da Computação) – Brown University, Providence, 1997. Disponível em: < https://eprints.kfupm.edu.sa/39888/1/39888.pdf >. Acesso em: 10 jan. 2010.

LIN, A.; BEAN, J.; WHITE, C. A hybrid genetic/optimization algorithm for finite horizon partially observed Markov decision processes. **Informs Journal on Computing**, v. 16, n. 1, p. 27-38, 2004.

LITTMAN, M.; DEAN, T.; KAELBLING, L. On the complexity of solving Markov decision problems. In: INTERNATIONAL CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 11, 18 Aug. 1995, Montreal. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1995. p. 394-402, ISBN 1558603859. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.6423&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

McDONALD, D. On the Poisson approximation to the multinomial distribution. **The Cadaian Journal of Statistics**, v. 8, n. 1, p. 115-118, 1980.

MARBACH, P.; TSITSIKLIS, J. Simulation-based optimization of Markov reward processes. **IEEE Transactions on Automatic Control**, v. 46, n. 2, p. 191-209, 2001.

MAYNE, D.; MICHALSKA, H. Receding horizon control of nonlinear system. **IEEE Transactions on Automatic Control**, v. 35, n. 7, p. 814-824, 1990.

MEULEAU, N.; HAUSKRECHT, M.; DEAN, T. Solving very large weakly coupled Markov decision processes. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 14, 24 July 1998, Madison. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1998. p. 165-172, ISBN

155860555X. Disponível em: < https://eprints.kfupm.edu.sa/65781/1/65781.pdf >. Acesso em: 10 jan. 2010.

MORARI, M.; LEE, J. Model predictive control: past, present, and future. **Computers and Chemical Engineering**, v. 23, n. 4, p. 667-682, 1999.

NAVARRO, V. A systems approach to health planning. **Health Services Research**, v. 4, n. 2, p. 96-111, 1969.

NG, A.; JORDAN, M. Pegasus: a policy search method for large MDPs and POMDPs. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 16, 30 June 2000, *Stanford*. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 2000. p. 406-415, ISBN 1558607099. Disponível em: < http://www-cs.stanford.edu/~ang/papers/uai00-pegasus.pdf >. Acesso em: 10 jan. 2010.

NUNES, L.; CARVALHO, S.; RODRIGUES, R. Markov decision process applied to the control of hospital elective admissions. **Artificial Intelligence in Medicine**, v. 47, n. 2, p. 159-171, 2009.

PAPADIMITRIOU, C.; TSITSIKLIS, J. The complexity of Markov decision processes. **Mathematics of Operations Research**, v. 12, n. 3, p. 441-450, 1987.

PARR, R. Flexible decomposition algorithms for weakly coupled Markov decision problems. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, n. 14, 24 July 1998, Madison. **Proceedings…** San Francisco: Morgan Kaufmann Publishers, 1998a. p. 422-430, ISBN 155860555X. Disponível em: < http://eprints.kfupm.edu.sa/41465/1/41465.pdf >. Acesso em: 10 jan. 2010.

PARR, R. **Hierarchical control and learning for Markov decision processes**. 153 p. Tese (Doutorado em Ciência da Computação) – University of California, Berkeley, 1998b. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.4106&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

PARR, R.; RUSSELL, S. Reinforcement learning with hierarchies of machines. In: CONFERENCE ON ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, n. 10, 1 Dec. 1997, Denver. **Proceedings…** Cambridge: MIT Press, 1998. p. 1043-1049, ISBN 0262100762. Disponível em: < http://www.cs.berkeley.edu/~russell/papers/nips97-ham.ps.gz >. Acesso em: 10 jan. 2010.

PATTEN, W.; WHITE, L. A sliding horizon feedback control problem with feedforward and disturbance. **Journal of Mathematical Systems, Estimation, and Control**, v. 7, n. 2, p. 1-33, 1997.

PÉRET, L.; GARCIA, F. On-line search for solving Markov decision processes via heuristic sampling. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, n. 16, 22 Aug. 2004, Valencia. **Proceedings…** Amsterdam: IOS Press, 2004. p. 1043-1049, ISBN 1586034529. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.4531&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

PORTEUS, E. Conditions for characterizing the structure of optimal strategies in infinite-horizon dynamic programs. **Journal of Optimization Theory and Applications**, v. 36, n. 3, p. 419-432, 1982.

POWELL, W. **Approximate dynamic programming**. Hoboken, NJ: Wiley-Interscience, 2007. 488 p. ISBN 0470171553.

PRESS, W.; TEUKOLSKY, S.; VETTERLING, W.; FLANNERY, B. **Numerical recipes in C++**: the art of scientific computing. 3. ed. Cambridge: Cambridge University Press, 2007. 1256 p. ISBN 0521880688.

PUTERMAN, M. **Markov decision processes**: discrete stochastic dynamic programming. Hoboken, NJ: John Wiley & Sons, 2005. 649 p. ISBN 0471727822.

RHODIUS, A. On the maximum of ergodicity coefficients, the Dobrushin ergodicity coefficient, and products of stochastic matrices. **Linear Algebra and its Applications**, v. 253, n. 1-3, p. 141-154, 1997.

ROTH, A.; VAN DIERDONCK, R. Hospital resource planning: concepts, feasibility, and framework. **Production and Operations Management**, v. 4, n. 1, p. 2-29, 1995.

RUST, J. Structural estimation of Markov decision processes In: ENGLE, R.; MCFADDEN, D. (Ed.). **Handbook of Econometrics Volume IV**. Amsterdam, NH: NORTH-HOLLAND, 1994. cap. 51, p. 3081-3143. 1078 p. ISBN 0444887660.

SALLANS, B. **Reinforcement learning for factored Markov decision processes**. 159 p. Tese (Doutorado em Ciência da Computação) – University of Toronto, Toronto, 2002. Disponível em: < http://members.chello.at/hoebertz-sallans/sallans/pthesis/pthesis.pdf >. Acesso em: 10 jan. 2010.

SAUL, L.; SINGH, S. Learning curve bounds for Markov decision processes with undiscounted rewards. In: CONFERENCE ON COMPUTATION

LEARNING THEORY, n. 9, 28 July 1996, Desenzano del Garda. **Proceedings…** New York: ACM, 1996. p. 147-156, ISBN 0897918118. Disponível em: < https://eprints.kfupm.edu.sa/48085/1/48085.pdf >. Acesso em: 10 jan. 2010.

SINGH, S.; COHN, D. How to dynamically merge Markov decision processes. In: CONFERENCE ON ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, n. 10, 1 Dec. 1997, Denver. **Proceedings…** Cambridge: MIT Press, 1998. p. 1057-1063, ISBN 0262100762. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.7806&rep=rep1&type=pdf >. Acesso em: 10 jan. 2010.

SMALLWOOD, R.; MURRAY, G.; SILVA, D.; SONDIK, E.; KLAINER, L. A medical service requirements model for health system design. **Proceedings of the IEEE**, v. 57, n. 11, p. 1880-1887, 1969.

SMITH, J.; MCCARDLE, K. Structural properties of stochastic dynamic programs. **Operations Research**, v.50, n. 5, p. 796-809, 2002.

SRINIVAS, M.; PATNAIK, L. Genetic algorithms: a survey. **IEEE Computer**, v.27, n. 6, p. 17-26, 1994.

TESAURO, G.; GALPERIN, G. On-line policy improvement using Monte Carlo search. In: CONFERENCE ON ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, n. 9, 2 Dec. 1996, Denver. **Proceedings…** Cambridge: MIT Press, 1997. p. 1068-1074, ISBN 0262100657. Disponível em: < http://books.nips.cc/papers/files/nips09/1068.pdf >. Acesso em: 13 jan. 2010.

THOMAS, W. A model for predicting recovery progress of coronary patients. **Health Services Research**, v. 3, n. 3, p. 185-213, 1968.

TSITSIKLIS, J. NP-Hardness of checking the unichain condition in average cost MDPs. **Operations Research Letters**, v. 35, p. 319-323, 2007.

VAN DEN BROEK, W. Moving horizon control in dynamic games. **Journal of Economic Dynamics and Control**, v. 26, n. 6, p. 937-961, 2002.

WATKINS, C. Q-learning. **Machine Learning**, v. 8, n. 3, p. 279-292, 1992.

WEISS E.; COHEN, M.; HERSHEY, J. An iterative estimation and validation procedure for specification of semi-Markov models with application to hospital patient flow. **Operations Research**, v. 30, n. 6, p. 1082-1104, 1982.

WHITE, D. **Markov decision processes**. Chichester, WS: John Wiley & Sons, 1993. 238 p. ISBN 0471936278.

APÊNDICE A

A.1 Armazenamento de uma matriz esparsa

A seguir, apresenta-se a estrutura lógica para o armazenamento, pela técnica *row-indexed sparse storage mode* (PRESS *et al.*, 2007), de uma matriz quadrada esparsa A de dimensões $N \times N$:

1) Preparam-se dois vetores: o primeiro, sa, para armazenar os valores dos elementos da matriz, e o segundo, ija, para armazenar índices de direcionamento sobre o vetor sa.

2) As primeiras N posições do vetor sa são ocupadas pelos valores dos elementos da diagonal da matriz A, em ordem. Os valores dos elementos da diagonal devem ser armazenados mesmo que sejam iguais a zero.

3) As primeiras N posições do vetor ija são ocupadas pelos índices de armazenamento em sa do primeiro valor diferente de zero fora da diagonal, para cada linha da matriz A. Se não existir valor diferente de zero fora da diagonal em uma linha da matriz A, ija deve assumir o valor do índice do último elemento armazenado em sa acrescido de uma unidade.

4) A primeira posição de ija deve armazenar o valor $N+2$.

5) A posição $N+1$ de ija deve armazenar o índice de sa correspondente ao valor do último elemento diferente de zero da última linha da matriz A. O valor de $ija[N+1]$ pode ser lido para determinar o número de elementos diferente de zero da matriz A ou o número de elementos nos vetores ija e sa. A posição $N+1$ de sa não tem utilidade, podendo assumir qualquer valor.

6) As posições acima de $N+1$ em sa devem armazenar os valores diferentes de zero dos elementos da matriz A fora da diagonal, ordenados pelas linhas e, em cada linha, ordenados pelas colunas.

7) As posições acima de $N+1$ em ija devem armazenar as posições das colunas correspondentes aos elementos armazenados em sa.

Segue abaixo, um pseudocódigo para o armazenamento de uma matriz quadrada A de dimensões $N \times N$, representada por $a[1...n][1...n]$, empregando-se dois vetores $sa[n\max]$ e $ija[n\max]$, sendo $n\max$ o tamanho preestabelecido dos vetores:

(0) para o contador $j = 1$ até n fazer

$sa[j] = a[j][j]$

(0) finalizar

fazer $ija[1] = n + 2$

fazer o contador $k = n + 1$

(0) para o contador $i = 1$ até n, fazer

(1) para o contador $j = 1$ até n, fazer

(2) se $a[i][j] > 0$ e $i \neq j$, fazer

$k = k + 1$

se $k > n\max$, então: "$n\max$ incompatível", parar o algoritmo

$sa[k] = a[i][j]$

$ija[k] = j$

(2) finalizar

(1) finalizar

$ija[i + 1] = k + 1$

(0) finalizar

A.2 Multiplicação de duas matrizes esparsas

Sejam A e B duas matrizes de dimensões $N \times N$ armazenadas nos vetores, respectivamente, sa, ija, sb e ijb, pela técnica *row-indexed sparse storage mode*, o produto das matrizes $A \times B$ pode ser realizado segundo o pseudocódigo descrito abaixo. A matriz resultante da operação fica armazenada nos vetores sr e ijr, sendo $n = N$ e $n\max$ o tamanho preestabelecido dos vetores empregados.

(0) para o contador $j = 1$ até n, fazer

se $sa[j] > 0$ e $sb[j] > 0$, fazer $sr[j] = sa[j] \times sb[j]$

em caso contrário, fazer $sr[j] = 0$

(1) para o contador $k = ija[j]$ até $ija[j+1] - 1$, fazer

(2) para o contador $l = ijb[ija[k]]$ até $ijb[ija[k+1]] - 1$, fazer

se $ijb[l] = ija[k]$, fazer $sr[j] = sr[j] + sa[k] \times sb[l]$

(2) finalizar

(1) finalizar

(0) finalizar

fazer $ijr[1] = n + 2$

fazer o contador $k = n + 1$

(0) para o contador $j = 1$ até n, fazer

(1) para o contador $i = 1$ até n, fazer

(2) se $j \neq i$, fazer

$aux = 0$

(3) se $sa[j] > 0$, fazer

(4) para o contador $l = ijb[j]$ até $ijb[j+1] - 1$ fazer

se $ijb[l] = i$, fazer $aux = sa[j] \times sb[l]$

(4) finalizar

(3) finalizar

(3) se $sb[i] > 0$, fazer

(4) para o contador $l = ija[\,j\,]$ até $ija[\,j+1] - 1$, fazer

se $ija[l] = i$, fazer $aux = aux + sb[i] \times sa[l]$

(4) finalizar

(3) finalizar

(3) para o contador $h = ija[\,j\,]$ até $ija[\,j+1] - 1$, fazer

(4) para o contador $l = ijb[ija[h]]$ até $ijb[ija[h+1]] - 1$, fazer

se $ijb[l] = ija[h]$, fazer $aux = aux + sa[h] \times sb[l]$

(4) finalizar

(3) finalizar

(3) se $aux > 0$, fazer

$k = k + 1$

se $k > n\max$, "$n\max$ incompatível", parar o algoritmo

$sr[k] = aux$

$ijr[k] = i$

(3) finalizar

(2) finalizar

(1) finalizar

$ijr[\,j+1] = k + 1$

(0) finalizar

APÊNDICE B

B.1 Determinação de políticas e obtenção de probabilidades limites

Apresenta-se, a seguir, o código em linguagem C para determinação das políticas gulosas G1 e G2, da política ótima via AIV e da política EHRA12 (Subseção 4.2.2 do Capítulo 4), considerando o PMD modelado para o controle de admissão de pacientes eletivos (Capítulo 2) com os parâmetros definidos na Tabela 4.1 (sexta configuração para o consumo de recursos) e probabilidades apresentadas na Tabela 4.2 (Subseção 4.2.1 do Capítulo 4). Através do código abaixo, também são obtidas as probabilidades limites para as políticas ótima e EHRA12 (Seção 3.1 do Capítulo 3).

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <time.h>
#include <math.h>
#include <conio.h>
#include "mehra4.h"

FILE *fp;   /* file pointer */

/*-------------------------------------------------------*/
/* Criacao e alocacao de valores banais pras variaveis */
/*-------------------------------------------------------*/

int Nesp   = 0;// Numeros de especialidades
int Npad   = 0;// Numero de padroes
int Nrec   = 0;// Numero de recursos
int N      = 0;// Tamanho da populacao de acoes
int Nalemx = 0;// Quantidade msxima de numeros aleatorios gerados
int Na     = 0;// Tamanho do vetor de acoes para criterio de parada
int Nr     = 0;// Acoes com menores custos a serem mantidas na pop
int C      = 0;// Numero de amostras de caminhos de funcionamento
int K      = 0;// Politicas para o metodo parallel rollout
double GAMA   = 0.0;// Fator de desconto
double Q      = 0.0;// Probabilidade de busca local ou global
int Stp    = 0;// Numero de repeticoes de uma acao para parada
int Nale   = 0;// Contador para o numero de numeros aleatorios
int H      = 0;// Tamanho do horizonte de planejamento
int Ttot   = 0;// Tamanho do espaco de estados
int Tact   = 0;// Numero de estados que aceitam novas admissoes

/*-------------------------------------------------------*/
/*        Processo de inicializacao efetivas...        */
/*-------------------------------------------------------*/
/* segundo Processo: Variaveis dependentes - Vetores*/

double LE[2]; // Custo de excesso sobre o nivel desejado para cada recurso
double LR[2]; // Custo de excesso sobre a capacidade maxima p/cada recurso
double LO[2]; // Custo de ociosidade para cada recurso
double NL[2]; // Nivel de consumo desejado para cada recurso
double LK[2]; // Capacidade maxima de consumo de cada recurso
double L[3][2]; // Consumo: para cada padrao, consumo de cada recurso
int x[3][2]; // Estado: para cada padrao, paciente por especialidade
int a[2]; // Acao: quantos pacientes de cada especialidade
int b[2]; // Acao: quantos pacientes de cada especialidade
int ma[2]; // Vetor aux: quantos pacientes de cada especialidade
int Smx[2]; // Capacidade maxima de admissao para cada especialidade
double P[2][3][3]; // Por especialidade: prob de transicoes entre padroes
double PE[3][2]; // Por padrao: prob de entrada de cada especialidade
double V[3]; // Para acumular o numero de pacientes em cada padrao
int A[5][2]; // Matriz: armazena as acoes de uma populacao de acoes
int R[2]; // Amplitude para a alteracao de acoes na busca local
struct Cstp VCstp[4]; // Criadas em mehra4.h: para o genetico e para a parada
int xx[3][2]; // Auxiliar: mesma definicao que x[4][3]
int S[150000][3][2]; // Espaco de estados
int G[150000][2];// matriz para a politica gulosa
int G1[150000][2]; // matriz para a politica gulosa
int P1[150000][2]; // matriz para a politica corrente, e no final otima
int AL[150000][2]; // matriz para a politica EHRA
double V1[150000]; // vetor para os custos no periodo anterior
double V2[150000]; // vetor para os custos no periodo corrente
int ija[30000000]; // Para compactar matriz esparsa: enderecos
double sa[30000000]; // Para compactar matriz esparsa: elementos
```

```
int ijt[30000000]; // Para compactar matriz esparsa: enderecos
double st[30000000]; // Para compactar matriz esparsa: elementos

/*---------------------------------------------------------*/
/*          Variaveis Auxiliares Globais                   */
/*---------------------------------------------------------*/

double CLT;// Custo total estimado
double u;// Numero aleatorio
int    e;// Acao de elite
int    i5;// Indice do processo iterativo principal

/*---------------------------------------------------------*/
/*             Prototipos                                  */
/*---------------------------------------------------------*/

void gera_acao_aleatoria();  // Gera acao aleatoria para iniciar as interacoes
void gera_ale();             // Gera um numero aleatorio: distribuicao uniforme
void gera_proximo_estado_custo();  // Simulacao: proximo estado e custo
int compara_vetor(int v1, int v2);  // Compara dois vetores
void gera_acao_nearest_neighbor();  // Gera acao na vizinhanca de uma dada acao
void gera_espaco_estados();  // Gera espaco de estados
double gera_probabilidade(int i1, int i2, int ac[3]);  // Probab de transicao
double fat(int arg); // Calcula o fatorial do argumento
double gera_probabilidade_1(int i2, int ac[3]); // Probab de transicao 1 estado



/*---------------------------------------------------------*/
/*          main                                           */
/*---------------------------------------------------------*/

int main(int argc, char** argv)
{

    double aux, caux, caux1, caux3, caux2, caux4, dxbase,
           aux0, naux, naux1, auxstp, raux,
           cust, custtot, mcust, ep, mn, mx, time1,
           EspPad0, EspPad1, time2, Cons[2], Vaux[3], EspEsp0, EspEsp1,
           tocio, texce, tover, cocio, cexce, cover;
      int  a1, a2, a3, c, cont, d, d1, f, f1, f2, f3, f4, f5, f6, FIM1, FIM,
           g, g1, g2, ggg, h, i, i1, i2, j, j1, k, l, n, n1, n2, na, nr, nmax,
           Nok, t, t1, xbase;


//Iniciar o Programa:

 time1 = time(0);

/*---------------------------------------------------------*/
/*       Parametros do modelo                              */
/*---------------------------------------------------------*/

Nesp = 2;

Npad = 3;

Nrec = 2;

LE[0] = 1.5; LE[1] = 1.0;

LR[0] = 1.0; LR[1] = 1.0;
```

```
LO[0] = 4.1; LO[1] = 1.0;

NL[0] = 4; NL[1] = 4;

LK[0] = 5; LK[1] = 5;

Smx[0] = 2; Smx[1] = 2;

P[0][0][0] = 0.4; P[0][0][1] = 0.1;  P[0][0][2] = 0.5;
P[0][1][0] = 0.1; P[0][1][1] = 0.3;  P[0][1][2] = 0.6;
P[0][2][0] = 0.0; P[0][2][1] = 0.0;  P[0][2][2] = 1.0;

P[1][0][0] = 0.2; P[1][0][1] = 0.1;  P[1][0][2] = 0.7;
P[1][1][0] = 0.1; P[1][1][1] = 0.2;  P[1][1][2] = 0.7;
P[1][2][0] = 0.0; P[1][2][1] = 0.0;  P[1][2][2] = 1.0;

PE[0][0] = 0.50; PE[0][1] = 0.40;
PE[1][0] = 0.50; PE[1][1] = 0.60;
PE[2][0] = 0.00; PE[2][1] = 0.00;

L[0][0] = 2.1; L[0][1] = 2.5;
L[1][0] = 2.5; L[1][1] = 2.1;
L[2][0] = 0.0; L[2][1] = 0.0;

/*-----------------------------------------------------*/
/*          Parametros do EHRA                         */
/*-----------------------------------------------------*/

H = 20;

C = 30;

K = 2;

GAMA = 0.8;

N = 5;

Na = 4;

Nr = 2;

Q = 0.9;

R[0] = 1; R[1] = 1;

Stp = 10;

Nalemx = 30000;

srand((unsigned int)time(null));

Nale = 0;

/*-------------------------------------------------------------------------*/
/*        Gera o espaco de estados                                         */
/*-------------------------------------------------------------------------*/

 gera_espaco_estados();
```

```
/*-------------------------------------------------------------------------*/
/*       Gera politica Gulosa 1                                            */
/*-------------------------------------------------------------------------*/

 for (i = 0; i < Ttot; i++)
 {
    mcust = 1000000;
    if (((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][0] +
         (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
          S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][0] <= LK[0]) &&
        ((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][1] +
         (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
          S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][1] <= LK[1]))
    {
       for (a1 = 0; a1 < Smx[0] + 1; a1++)
       {
          for (a2 = 0; a2 < Smx[1] + 1; a2++)
          {
                a[0] = a1; a[1] = a2;
                g  = S[i][0][0] + S[i][1][0] + a[0];
                g1 = S[i][0][1] + S[i][1][1] + a[1];
                caux = 0;
                custtot = 0;
                for (j = 0; j < Ttot; j++)
                {
                   if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
                       (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
                       (S[j][0][0] + S[j][1][0] > a[0] - 1) && (S[j][2][0] <
                                      S[i][0][0] + S[i][1][0] + 1) &&
                       (S[j][0][1] + S[j][1][1] > a[1] - 1) && (S[j][2][1] <
                                      S[i][0][1] + S[i][1][1] + 1))
                   {
                      caux1 = gera_probabilidade(i, j, a);
                      caux = caux + caux1;
                      cust = 0;
                      for (l = 0; l < Nrec; l++)
                      {
                         caux2 = 0;
                         for (d = 0; d < Npad; d++)
                         {
                            caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
                         }
                         if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
                         if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
                         if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
                      }
                      cust = cust * caux1;
                      custtot = custtot + cust;
                   }
                }
                if (custtot < mcust)
                {
                   ma[0] = a[0]; ma[1] = a[1];
                   mcust = custtot;
                }
          }
       }
       G[i][0] = ma[0]; G[i][1] = ma[1];
    }
    else
    {
```

```
      G[i][0] = 0; G[i][1] = 0;
      a[0] = 0; a[1] = 0;
      g  = S[i][0][0] + S[i][1][0] + a[0];
      g1 = S[i][0][1] + S[i][1][1] + a[1];
      caux = 0;
      custtot = 0;
      for (j = 0; j < Ttot; j++)
      {
         if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
             (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
             (S[j][0][0] + S[j][1][0] > a[0] - 1) && (S[j][2][0] < S[i][0][0] +
                                      S[i][1][0] + 1) &&
             (S[j][0][1] + S[j][1][1] > a[1] - 1) && (S[j][2][1] < S[i][0][1] +
                                      S[i][1][1] + 1))
         {
            caux1 = gera_probabilidade(i, j, a);
            caux = caux + caux1;
            cust = 0;
            for (l = 0; l < Nrec; l++)
            {
               caux2 = 0;
               for (d = 0; d < Npad; d++)
               {
                  caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
               }
               if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
               if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
               if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
            }
            cust = cust * caux1;
            custtot = custtot + cust;
         }
      }
      mcust = custtot;
   }
   V1[i] = mcust;
 }

/*-------------------------------------------------------------------------*/
/*        Gera politica Gulosa 2                                           */
/*-------------------------------------------------------------------------*/

 for (i = 0; i < Ttot; i++)
 {
    if (((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][0] +
         (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
          S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][0] <= LK[0]) &&
        ((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][1] +
         (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
          S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][1] <= LK[1]))
    {
       mcust = 1000000;
       for (i1 = 0; i1 < Npad; i1++)
       {
          V[i1] = 0;
          for (d1 = 0; d1 < Nesp; d1++)
          {
             for (j1 = 0; j1 < Npad; j1++)
             {
                V[i1] = V[i1] + S[i][j1][d1] * P[d1][j1][i1];
             }
```

```
         }
      }
      for (a1 = 0; a1 < Smx[0] + 1; a1++)
      {
         for (a2 = 0; a2 < Smx[1] + 1; a2++)
         {
            a[0] = a1; a[1] = a2;
            caux = 0;
            for (i1 = 0; i1 < Npad; i1++)
            {
               Vaux[i1] = V[i1];
               for (d1 = 0; d1 < Nesp; d1++)
               {
                  Vaux[i1] = Vaux[i1] + a[d1] * PE[i1][d1];
               }
            }
            for (l = 0; l < Nrec; l++)
            {
               aux = 0;
               for (i1 = 0; i1 < Npad; i1++)
               {
                  aux = aux + Vaux[i1] * L[i1][l];
               }
               if (aux < NL[l]) {caux = caux + (NL[l] - aux) * LO[l];}
               if (aux > NL[l]) {caux = caux + (aux - NL[l]) * LE[l];}
               if (aux > LK[l]) {caux = caux + (aux - LK[l]) * LR[l];}
            }
            if (caux < mcust)
            {
               ma[0] = a[0]; ma[1] = a[1];
               mcust = caux;
            }
         }
      }
      G1[i][0] = ma[0]; G1[i][1] = ma[1];
   }
   else
   {
      G1[i][0] = 0; G1[i][1] = 0;
   }
 }

/*-------------------------------------------------------------------------*/
/*       Gera politica EHRA12                                              */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*  Teste primario: se o consumo medio estimado no ultimo periodo de um dos */
/*  recursos for maior que a quantidade disponivel, a unica acao permitida  */
/*  e "nao admitir pacientes no proximo periodo". E o programa se encerra.  */
/*-------------------------------------------------------------------------*/

t = 0;
for (i5 = 0; i5 < Ttot; i5++)
{
 t1 = 0;
 for (i = 0; i < Npad; i++)
 {
    for (d = 0; d < Nesp; d++)
    {
        x[i][d] = S[i5][i][d];
    }
 }
```

```
if (i5 < Tact)
{

   /*-----------------------------------------------*/
   /*   Geracao de uma populacao inicial de acoes   */
   /*   Mantendo as acoes das politicas de base     */
   /*-----------------------------------------------*/

   for (n = 0; n < N; n++ )
   {
     if (n < N - K)
     {
       gera_acao_aleatoria();
       for (d = 0; d < Nesp; d++)
       {
         A[n][d] = a[d];
       }
     }
     if ((n >= N - K) && (n < N - 1))
     {
       for (d = 0; d < Nesp; d++)
       {
         A[n][d] = G1[i5][d];
       }
     }
     if (n == N - 1)
     {
       for (d = 0; d < Nesp; d++)
       {
         A[n][d] = G[i5][d];
       }
     }
   }

  /*-----------------------------------------------*/
  /* Limpeza dos registros para o criterio de parada  */
  /*-----------------------------------------------*/

  for (na = 0; na < Na; na++)
  {
      for (d = 0; d < Nesp; d++)
      {
          VCstp[na].b[d]  = Smx[d] + 1;
      }
      VCstp[na].som   = 0;
      VCstp[na].med   = bilhao;
      VCstp[na].N1    = 0;
  }

 /*-----------------------------------------------*/
 /*       Inicio do processo iterativo            */
 /*-----------------------------------------------*/

  FIM = 0;
  while (FIM == 0)
  {
    t1 = t1 + 1;
    printf("estado: %d   iteracao: %d\n", t + 1, t1);
//Determinacao de uma acao de elite via "parallel rollout":
    e = 0;
    naux1 = bilhao;
    for (n = 0; n < N; n++)
    {
```

```
naux = 0;
for (c = 0; c < C; c++)
{
  caux1 = bilhao;
  for (k = 0 ; k < K; k++)
  {
    for (i = 0; i < Npad; i++)
    {
      for (d = 0; d < Nesp; d++)
      {
        xx[i][d] = x[i][d];
      }
    }
    for (d = 0; d < Nesp; d++)
    {
      a[d] = A[n][d];
    }
    gera_proximo_estado_custo();
    caux = CLT;
    for (h = 0; h < H; h++)
    {
      if (k < K - 1)
      {
        i2 = -1;
        f6 = 0;
        while  (f6 == 0)
        {
          f6 = 1;
          i2 = i2 + 1;
          for (i = 0; i < Npad - 1; i++)
          {
            for (d = 0; d < Nesp; d++)
            {
              if (xx[i][d] != S[i2][i][d]) {f6 = 0;}
            }
          }
        }
        for (d = 0; d < Nesp; d++)
        {
          a[d] = G1[i2][d];
        }
      }
      if (k == K - 1)
      {
        i2 = -1;
        f6 = 0;
        while (f6 == 0)
        {
          f6 = 1;
          i2 = i2 + 1;
          for (i = 0; i < Npad - 1; i++)
          {
            for (d = 0; d < Nesp; d++)
            {
              if (xx[i][d] != S[i2][i][d]) {f6 = 0;}
            }
          }
        }
        for (d = 0; d < Nesp; d++)
        {
          a[d] = G[i2][d];
        }
      }
```

```
        gera_proximo_estado_custo();
        caux = caux + pow(GAMA,(h+1)) * CLT;
      }
      if (caux < caux1)
      {
        caux1 = caux;
      }
    }
    naux = naux + caux1;
  }
  naux = naux / C;
  if (naux < naux1)
  {
    naux1 = naux;
    e = n;
  }
}

/*-----------------------------------------------*/
/*     O indice da acao de elite e dado por e    */
/*-----------------------------------------------*/

/*-----------------------------------------------*/
/*     Verificacao do criterio de parada         */
/*-----------------------------------------------*/

g = 0;
f = 0;
while ((g < Na) && (f == 0))
{
  if ( compara_vetor(e, g) == 1)
  {
    ggg = g;
    f = 1;
    VCstp[g].som = VCstp[g].som + naux1;
    VCstp[g].N1++;
    VCstp[g].med = VCstp[g].som / VCstp[g].N1;
    if (VCstp[g].N1 == Stp)
    {
      FIM = 1;
    }
  }
  g = g + 1;
}
if (f == 0)
{
  g1 = 0;
  for (na = 0; na < Na - 1; na++)
  {
    if (VCstp[g1].med < VCstp[na + 1].med)
    {
      g1 = na + 1;
    }
  }
  for (d = 0; d < Nesp; d++)
  {
    VCstp[g1].b[d] = A[e][d];
  }
  VCstp[g1].som = naux1;
  VCstp[g1].N1  = 1;
  VCstp[g1].med = naux1;
}
```

```
/*-------------------------------------------------*/
/* Determinacao de uma nova populacao de acoes     */
/* via ERPS, partindo da acao de elite A[e],       */
/* mantendo as acoes das politicas de base,        */
/* e evitando repeticao de acoes.                  */
/*-------------------------------------------------*/

if (FIM == 0)
{

/*-------------------------------------------------*/
/* Incluir as NR acoes com menores custos,         */
/* componentes do conjunto de acoes em             */
/* avaliacao no criterio de parada.                */
/*-------------------------------------------------*/

  raux = 0.0;
  n2 = 0;
  nr = 1;
  while (nr <= Nr && n2 ==0)
  {
    g1 = 0;
    for (na = 0; na < Na - 1; na++)
    {
      if ((VCstp[g1].med > VCstp[na + 1].med) &&
          (VCstp[na + 1].med > raux)) {g1 = na + 1;}
    }
    if ((VCstp[g1].med == bilhao) || (VCstp[g1].med <= raux))
    {
      n2 = 1;
      nr = nr - 1;
    }
    if ((VCstp[g1].med < bilhao) && (VCstp[g1].med > raux))
    {

    /*-------------------------------------------------*/
    /* Evita repeticao de acoes identicas as acoes     */
    /* das politicas de base                           */
    /*-------------------------------------------------*/

      f4 = 1;
      for (n1 = (N - K); n1 < N; n1++)
      {
        f5 = 0;
        for (d = 0; d < Nesp; d++)
        {
          if (VCstp[g1].b[d] == A[n1][d]) {f5 = f5 + 1;}
        }
        if (f5 == Nesp) {f4 = 0;}
      }
      if (f4 == 1)
      {
        for (d = 0; d < Nesp; d++)
        {
          A[N - K - nr][d] = VCstp[g1].b[d];
        }
      }
      if (f4 == 0)
      {
        f1 = 0;
        while (f1 == 0)
        {
          f1 = 1;
```

```
        gera_ale();
        if (u > Q)
        {
          gera_acao_nearest_neighbor();
          for (d = 0; d < Nesp; d++)
          {
            A[N - K - nr][d] = a[d];
          }
        }
        if (u <= Q)
        {
          gera_acao_aleatoria();
          for (d = 0; d < Nesp; d++)
          {
            A[N - K - nr][d] = a[d];
          }
        }
        for (n1 = (N - K); n1 < N; n1++)
        {
          f3 = 0;
          for (d = 0; d < Nesp; d++)
          {
            if (A[N - K - nr][d] == A[n1][d]) {f3 = f3 + 1;}
          }
          if (f3 == Nesp) {f1 = 0;}
        }
      }
    }
    raux = VCstp[g1].med;
    nr = nr + 1;
  }
}

/*-------------------------------------------------*/
/* Completar com N - k - nr acoes                  */
/* a nova populacao de acoes.                      */
/*                                                 */
/* A primeira acao e a acao de elite da ultima     */
/* iteracao.                                       */
/*                                                 */
/* Evita-se repeticoes de acoes na nova populacao.*/
/*-------------------------------------------------*/

f2 = 1;
for (d = 0; d < Nesp; d++)
{
  A[0][d] = A[e][d];
}
for (n1 = (N - K - nr); n1 < N; n1++)
{
  f3 = 0;
  for (d = 0; d < Nesp; d++)
  {
    if (A[0][d] == A[n1][d]) {f3 = f3 + 1;}
  }
  if (f3 == Nesp) {f2 = 0;}
}
for (n = f2; n < N - K - nr; n++)
{
  f1 = 0;
  while (f1 == 0)
  {
    f1 = 1;
```

```
          gera_ale();
          if (u > Q)
          {
            gera_acao_nearest_neighbor();
            for (d = 0; d < Nesp; d++)
            {
              A[n][d] = a[d];
            }
          }
          if (u <= Q)
          {
            gera_acao_aleatoria();
            for (d = 0; d < Nesp; d++)
            {
              A[n][d] = a[d];
            }
          }
          if (n > 0)
          {
            for (n1 = 0; n1 < n; n1++)
            {
              f3 = 0;
              for (d = 0; d < Nesp; d++)
              {
                if (A[n][d] == A[n1][d]) {f3 = f3 + 1;}
              }
              if (f3 == Nesp) {f1 = 0;}
            }
          }
          if (f1 == 1)
          {
            for (n1 = (N - K - nr); n1 < N; n1++)
            {
              f3 = 0;
              for (d = 0; d < Nesp; d++)
              {
                if (A[n][d] == A[n1][d]) {f3 = f3 + 1;}
              }
              if (f3 == Nesp) {f1 = 0;}
            }
          }
        }
      }
    }
  }

  /*-------------------------------------------------*/
  /*     “Resultados do processo iterativo.”         */
  /*-------------------------------------------------*/

  AL[i5][0] = VCstp[ggg].b[0]; AL[i5][1] = VCstp[ggg].b[1];
}
if (i5 > Tact - 1)
{
  for (i = 0; i < Npad; i++)
  {
    V[i] = 0;
    for (d = 0; d < Nesp; d++)
    {
      for (j = 0; j < Npad; j++)
      {
        V[i] = V[i] + x[j][d] * P[d][j][i];
      }
```

```
    }
   }
   FIM1 = 0;
   for (l = 0; l < Nrec; l++)
   {
     aux0 = 0;
     for (i = 0; i < Npad; i++)
     {
       aux0 = aux0 + V[i] * L[i][l];
     }
     if (aux0 > LK[l]) {FIM1 = 1;}
   }
   if (FIM1 == 1) {AL[i5][0] = 0; AL[i5][1] = 0;}
   else
   {
     cont = 0;
     i1 = 0;
     while ((i1 < Tact) && (cont == 0))
     {
       if (S[i1][0][0] == x[0][0] && S[i1][0][1] == x[0][1] && S[i1][1][0] == x[1][0] &&
           S[i1][1][1] == x[1][1])
       {
         cont = 1;
       }
       i1 = i1 + 1;
     }
     if (cont == 1)
     {
       AL[i5][0] = AL[i1-1][0]; AL[i5][1] = AL[i1-1][1];
     }
     if ((i1 == Tact) && (cont == 0))
     {
       printf("erro de logica\n");
       getchar();
     }
   }
 }
 t = t + 1;
 printf("estado: %d \n", t);
}

/*--------------------------------------------------------------------------*/
/*  Politica Otima pelo Algoritmo de Iteracao de Valores                    */
/*--------------------------------------------------------------------------*/

for (i = 0; i < Ttot; i++)
{
  V1[i] = 0;
}
ep = 1;
cont = 0;
while (ep > 0.000001)
{
  time2 = time(0);
  mn = 1000000;
  cont = cont + 1;
  printf("  Via - numero de iteracoes: %d\n", cont);
  mx = 0;
  for (i = 0; i < Ttot; i++)
  {
    mcust = 1000000;
    if (((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][0] +
```

```
     (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
      S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][0] <= LK[0]) &&
    ((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
      S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][1] +
     (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
      S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][1] <= LK[1]))
{
  for (a1 = 0; a1 < Smx[0] + 1; a1++)
  {
    for (a2 = 0; a2 < Smx[1] + 1; a2++)
    {
      a[0] = a1; a[1] = a2;
      g  = S[i][0][0] + S[i][1][0] + a[0];
      g1 = S[i][0][1] + S[i][1][1] + a[1];
      caux = 0;
      custtot = 0;
      for (j = 0; j < Ttot; j++)
      {
        if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
            (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
            (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
            (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
            (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
            (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
        {
          caux1 = gera_probabilidade(i, j, a);
          caux = caux + caux1;
          cust = 0;
          for (l = 0; l < Nrec; l++)
          {
            caux2 = 0;
            for (d = 0; d < Npad; d++)
            {
              caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
            }
            if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
            if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
            if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
          }
          cust = cust * caux1;
          custtot = custtot + cust + caux1 * V1[j];
        }
      }
      if (custtot < mcust)
      {
        ma[0] = a[0]; ma[1] = a[1];
        mcust = custtot;
      }
    }
  }
  P1[i][0] = ma[0]; P1[i][1] = ma[1];
}
else
{
  P1[i][0] = 0; P1[i][1] = 0;
  a[0] = 0; a[1] = 0;
  g  = S[i][0][0] + S[i][1][0];
  g1 = S[i][0][1] + S[i][1][1];
  caux = 0;
  custtot = 0;
  for (j = 0; j < Ttot; j++)
  {
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
```

```
            (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
            (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
            (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
            (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
            (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
        {
          caux1 = gera_probabilidade(i, j, a);
          caux = caux + caux1;
          cust = 0;
          for (l = 0; l < Nrec; l++)
          {
            caux2 = 0;
            for (d = 0; d < Npad; d++)
            {
              caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
            }
            if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
            if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
            if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
          }
          cust = cust * caux1;
          custtot = custtot + cust + caux1 * V1[j];
        }
      }
      mcust = custtot;
    }
    V2[i] = mcust;
    if (V2[i] - V1[i] < mn) {mn = V2[i] - V1[i];}
    if (V2[i] - V1[i] > mx) {mx = V2[i] - V1[i];}
  }
  ep = (mx - mn) / mn;
  for (i = 0; i < Ttot; i++)
  {
    V1[i] = V2[i];
  }
  printf("  tempo da ultima iteracao em segundos: %f\n", time(0) - time2);
  printf(" \n");
  printf("  mn: %f   mx: %f  ep: %f\n", mn, mx, ep);
  printf(" \n");
}
time1 = time(0) - time1;
printf("  tempo total de processamento em minutos: %f\n", time1 / 60);
printf("  mn: %f   mx: %f\n", mn, mx);

/* open file for output */
if ((fp = fopen("saida1.log", "w+t"))==NULL)
{
  printf("Cannot open file \n");
  exit(1);
}
fprintf(fp,"   VIA - custo medio: minimo %f  e maximo %f\n", mn, mx);

/*-------------------------------------------------------------------------*/
/*  Iteracao de Valores para determinar o custo medio                      */
/*  para uma politica fixa. (Pode ser a G1 a G2 ou a EHRA12.)              */
/*  No caso abaixo trata-se a politica EHRA12.                             */
/*-------------------------------------------------------------------------*/

for (i = 0; i < Ttot; i++)
{
  V1[i] = 0;
}
ep = 1;
```

```
cont = 0;
time1 = time(0);
while (ep > 0.00001)
{
  time2 = time(0);
  mn = 1000000;
  cont = cont + 1;
  printf("  EHRA12 - numero de iteracoes: %d\n", cont);
  mx = 0;
  for (i = 0; i < Ttot; i++)
  {
    mcust = 1000000;
    if (((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][0] +
         (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
          S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][0] <= LK[0]) &&
        ((S[i][0][0] * P[0][0][0]+ S[i][1][0] * P[0][1][0] +
          S[i][0][1] * P[1][0][0]+ S[i][1][1] * P[1][1][0]) * L[0][1] +
         (S[i][0][0] * P[0][0][1]+ S[i][1][0] * P[0][1][1] +
          S[i][0][1] * P[1][0][1]+ S[i][1][1] * P[1][1][1]) * L[1][1] <= LK[1]))
    {
      a[0] = AL[i][0]; a[1] = AL[i][1];
      g  = S[i][0][0] + S[i][1][0] + a[0];
      g1 = S[i][0][1] + S[i][1][1] + a[1];
      caux = 0;
      custtot = 0;
      for (j = 0; j < Ttot; j++)
      {
        if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
            (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
            (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
            (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
            (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
            (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
        {
          caux1 = gera_probabilidade(i, j, a);
          caux = caux + caux1;
          cust = 0;
          for (l = 0; l < Nrec; l++)
          {
            caux2 = 0;
            for (d = 0; d < Npad; d++)
            {
              caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
            }
            if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
            if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
            if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
          }
          cust = cust * caux1;
          custtot = custtot + cust + caux1 * V1[j];
        }
      }
    }
    else
    {
      a[0] = AL[i][0]; a[1] = AL[i][1];
      g  = S[i][0][0] + S[i][1][0];
      g1 = S[i][0][1] + S[i][1][1];
      caux = 0;
      custtot = 0;
      for (j = 0; j < Ttot; j++)
      {
```

```
        if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
            (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
            (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
            (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
            (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
            (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
        {
          caux1 = gera_probabilidade(i, j, a);
          caux = caux + caux1;
          cust = 0;
          for (l = 0; l < Nrec; l++)
          {
            caux2 = 0;
            for (d = 0; d < Npad; d++)
            {
              caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
            }
            if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
            if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
            if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
          }
          cust = cust * caux1;
          custtot = custtot + cust + caux1 * V1[j];
        }
      }
    }
    V2[i] = custtot;
    if (V2[i] - V1[i] < mn) {mn = V2[i] - V1[i];}
    if (V2[i] - V1[i] > mx) {mx = V2[i] - V1[i];}
  }
  ep = (mx - mn) / mn;
  for (i = 0; i < Ttot; i++)
  {
    V1[i] = V2[i];
  }
  printf("  tempo da ultima iteracao em segundos: %f\n", time(0) - time2);
  printf(" \n");
}
time1 = time(0) - time1;
printf("  tempo total de processamento em minutos: %f\n", time1 / 60);
printf("  mn: %f   mx: %f\n", mn, mx);

fprintf(fp,"   EHRA12 - custo medio: minimo %f  e maximo %f\n", mn, mx);

/*---------------------------------------------------------------------------*/
/*  Gera custo e outros parametros esperados da Politica Otima               */
/*---------------------------------------------------------------------------*/

fprintf(fp,"   Politica Otima e valores esperados\n");
printf("  Gera custo esperado da Politica Otima\n");
for (i = 0; i < Ttot; i++)
{
  a[0] = P1[i][0]; a[1] = P1[i][1];
  g  = S[i][0][0] + S[i][1][0] + a[0];
  g1 = S[i][0][1] + S[i][1][1] + a[1];
  caux = 0;
  custtot = 0; tocio = 0; texce = 0; tover = 0;
  Cons[0] = 0; Cons[1] = 0;
  EspPad0 = 0; EspPad1 = 0;
  EspEsp0 = 0; EspEsp1 = 0;
  for (j = 0; j < Ttot; j++)
  {
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
```

```
        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
    {
      caux1 = gera_probabilidade(i, j, a);
      caux = caux + caux1;
      cust = 0;
      cocio = 0; cexce = 0; cover = 0;
      for (l = 0; l < Nrec; l++)
      {
        caux2 = 0;
        for (d = 0; d < Npad; d++)
        {
          caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
        }
        if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];
                            cocio = cocio + (NL[l] - caux2) * LO[l];}
        if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];
                            cexce = cexce + (caux2 - NL[l]) * LE[l];}
        if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];
                            cover = cover + (caux2 - LK[l]) * LR[l];}
        Cons[l] = Cons[l] + caux2 * caux1;
      }
      EspPad0 = EspPad0 + (S[j][0][0] + S[j][0][1]) * caux1;
      EspPad1 = EspPad1 + (S[j][1][0] + S[j][1][1]) * caux1;
      EspEsp0 = EspEsp0 + (S[j][0][0] + S[j][1][0]) * caux1;
      EspEsp1 = EspEsp1 + (S[j][0][1] + S[j][1][1]) * caux1;
      custtot = custtot + cust * caux1;
      tocio = tocio + cocio * caux1;
      texce = texce + cexce * caux1;
      tover = tover + cover * caux1;
    }
  }
  fprintf(fp,"%d+%d+%d+%d+%d+%d+%d+%d+%f+%f+%f+%f+%f+%f+%f+%f+%f+%f\n", S[i][0][0],
              S[i][1][0], S[i][2][0], S[i][0][1], S[i][1][1], S[i][2][1], P1[i][0],
              P1[i][1], custtot, Cons[0], Cons[1], EspPad0, EspPad1, EspEsp0, EspEsp1,
              tocio, texce, tover);
}

/*-------------------------------------------------------------------------*/
/*  Gera custo e outros parametros esperados da EHRA12                     */
/*-------------------------------------------------------------------------*/

fprintf(fp,"   Politica EHRA12 e valores esperados\n");
printf("  Gera custo esperado da Politica EHRA12\n");
for (i = 0; i < Ttot; i++)
{
  a[0] = AL[i][0]; a[1] = AL[i][1];
  g  = S[i][0][0] + S[i][1][0] + a[0];
  g1 = S[i][0][1] + S[i][1][1] + a[1];
  caux = 0;
  custtot = 0; tocio = 0; texce = 0; tover = 0;
  Cons[0] = 0; Cons[1] = 0;
  EspPad0 = 0; EspPad1 = 0;
  EspEsp0 = 0; EspEsp1 = 0;
  for (j = 0; j < Ttot; j++)
  {
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
```

```
        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
    {
      caux1 = gera_probabilidade(i, j, a);
      caux = caux + caux1;
      cust = 0;
      cocio = 0; cexce = 0; cover = 0;
      for (l = 0; l < Nrec; l++)
      {
        caux2 = 0;
        for (d = 0; d < Npad; d++)
        {
          caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
        }
        if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];
                            cocio = cocio + (NL[l] - caux2) * LO[l];}
        if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];
                            cexce = cexce + (caux2 - NL[l]) * LE[l];}
        if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];
                            cover = cover + (caux2 - LK[l]) * LR[l];}
        Cons[l] = Cons[l] + caux2 * caux1;
      }
      EspPad0 = EspPad0 + (S[j][0][0] + S[j][0][1]) * caux1;
      EspPad1 = EspPad1 + (S[j][1][0] + S[j][1][1]) * caux1;
      EspEsp0 = EspEsp0 + (S[j][0][0] + S[j][1][0]) * caux1;
      EspEsp1 = EspEsp1 + (S[j][0][1] + S[j][1][1]) * caux1;
      cust = cust * caux1;
      custtot = custtot + cust;
      tocio = tocio + cocio * caux1;
      texce = texce + cexce * caux1;
      tover = tover + cover * caux1;
    }
  }
  fprintf(fp,"%d+%d+%d+%d+%d+%d+%d+%d+%f+%f+%f+%f+%f+%f+%f+%f+%f+%f\n", S[i][0][0],
              S[i][1][0], S[i][2][0], S[i][0][1], S[i][1][1], S[i][2][1], AL[i][0],
              AL[i][1], custtot, Cons[0], Cons[1], EspPad0, EspPad1, EspEsp0, EspEsp1,
              tocio, texce, tover);
}
fclose(fp);

/*-------------------------------------------------------------------------*/
/*  Gera matriz de probabilidades condensada - Politica Otima              */
/*-------------------------------------------------------------------------*/

nmax = 30000000;
for (j = 0; j < Ttot; j++)
{
  a[0] = P1[j][0]; a[1] = P1[j][1];
  g  = S[j][0][0] + S[j][1][0] + a[0];
  g1 = S[j][0][1] + S[j][1][1] + a[1];
  if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
      (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
      (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
      (S[j][2][0] < S[j][0][0] + S[j][1][0] + 1) &&
      (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
      (S[j][2][1] < S[j][0][1] + S[j][1][1] + 1))
  {
    sa[j] = gera_probabilidade(j, j, a);
  }
  else {sa[j] = 0;}
}
ija[0] = Ttot + 1;
k = Ttot;
```

```
for (i = 0; i < Ttot; i++)
{
  a[0] = P1[i][0]; a[1] = P1[i][1];
  g  = S[i][0][0] + S[i][1][0] + a[0];
  g1 = S[i][0][1] + S[i][1][1] + a[1];
  for (j = 0; j < Ttot; j++)
  {
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
    {
      caux1 = gera_probabilidade(i, j, a);
    }
    else {caux1 = 0;}
    if (caux1 > 0 && i != j)
    {
      k = k + 1;
      if (k > nmax) {printf("  erro a lista tem tamanho pequeno\n"); getchar();}
      else
      {
        sa[k] = caux1;
        ija[k] = j;
      }
    }
  }
  ija[i + 1] = k + 1;
}
cont = 0;
for (k = 0; k < Ttot; k++)
{
  if (sa[k] == 0) {cont = cont + 1;}
}
printf("    VIA - Estados da matriz de probabilidades condensada: ija[0]-1 %d\n",
         ija[0]-1);
printf("    Numero de entradas diferentes de zero: ija[ija[0]-1] %d\n",
         ija[ija[0]-1]-1);
printf("    Diagonal zero: %d\n", cont);

/*-------------------------------------------------------------------------------*/
/*  Produto da matriz acumulada pela matriz de probabilidades - Politica Otima  */
/*-------------------------------------------------------------------------------*/

caux1 = 2;
cont = 0;
while (caux1 > 1.000001 || caux1 - 1 < (0 - 0.000001))
{
  time2 = time(0);
  printf("   \n");
  printf("    Via - produto da matriz: %d\n", cont + 1);
  for (j = 0; j < Ttot; j++)
  {
    a[0] = P1[j][0]; a[1] = P1[j][1];
    g  = S[j][0][0] + S[j][1][0] + a[0];
    g1 = S[j][0][1] + S[j][1][1] + a[1];
    if ( sa[j] > 0 && (S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
                      (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
                      (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
                      (S[j][2][0] < S[j][0][0] + S[j][1][0] + 1) &&
                      (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
                      (S[j][2][1] < S[j][0][1] + S[j][1][1] + 1))
```

```
  {
    st[j] = sa[j] * gera_probabilidade(j, j, a);
  }
  else {st[j] = 0;}
  for (k = ija[j]; k < ija[j + 1]; k++)
  {
    i = ija[k];
    a[0] = P1[i][0]; a[1] = P1[i][1];
    g  = S[i][0][0] + S[i][1][0] + a[0];
    g1 = S[i][0][1] + S[i][1][1] + a[1];
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
    {
      st[j] = st[j] + sa[k] * gera_probabilidade(i, j, a);
    }
  }
}
ijt[0] = Ttot + 1;
k = Ttot;
for (i = 0; i < Ttot; i++)
{
  for (j = 0; j < Ttot; j++)
  {
    if (j != i)
    {
      a[0] = P1[i][0]; a[1] = P1[i][1];
      g  = S[i][0][0] + S[i][1][0] + a[0];
      g1 = S[i][0][1] + S[i][1][1] + a[1];
      if ( sa[i] > 0 && (S[j][0][0] + S[j][1][0] + S[j][2][0] == g) &&
                        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
                        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
                        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
                        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
                        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
      {
        caux1 = sa[i] * gera_probabilidade(i, j, a);
      }
      else {caux1 = 0;}
      for (l = ija[i]; l < ija[i + 1]; l++)
      {
        a[0] = P1[ija[l]][0]; a[1] = P1[ija[l]][1];
        g  = S[ija[l]][0][0] + S[ija[l]][1][0] + a[0];
        g1 = S[ija[l]][0][1] + S[ija[l]][1][1] + a[1];
        if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
            (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
            (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
            (S[j][2][0] < S[ija[l]][0][0] + S[ija[l]][1][0] + 1) &&
            (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
            (S[j][2][1] < S[ija[l]][0][1] + S[ija[l]][1][1] + 1))
        {
          caux1 = caux1 +  sa[l] * gera_probabilidade(ija[l], j, a);
        }
      }
      if (caux1 > 0)
      {
        k = k + 1;
        if (k > nmax) {printf("  erro a lista tem tamanho pequeno\n"); getchar();}
        else
        {
```

```
            st[k] = caux1;
            ijt[k] = j;
          }
        }
      }
    }
    ijt[i + 1] = k + 1;
  }
  printf(" \n");
  printf("  VIA - Fim do produto da matriz: %d      numero de estados: ijt[0]-1 %d\n",
          cont + 1, ijt[0]-1);
  printf("    Numero de entradas diferentes de zero: ijt[ija[0]-1]-1 %d\n",
          ijt[ijt[0]-1] - 1);
  printf("    Tempo de processamento em minutos  %f\n", (time(0) - time2)/60);
  caux1 = 0;
  for (k = 0; k < ijt[ijt[0]-1] - 1; k++)
  {
    if (k < Ttot) {caux1 = caux1 + st[k];}
    sa[k] = st[k];
    ija[k] = ijt[k];
  }
  printf("    VIA - Soma da diagonal: %f\n", caux1);
  cont = cont + 1;
}

fclose(fp);
/* open file for output */
if ((fp = fopen("Probab.log", "w+t"))==NULL)
{
  printf("Cannot open file \n");
  exit(1);
}
fprintf(fp,"\n");
fprintf(fp,"   VIA - Soma da diagonal: %f\n", caux1);
for (i = 0; i < Ttot; i++)
{
  fprintf(fp," %f\n", sa[i]);
}

/*--------------------------------------------------------------------------*/
/*  Gera matriz de probabilidades condensada - Politica EHRA12              */
/*--------------------------------------------------------------------------*/

nmax = 30000000;
for (j = 0; j < Ttot; j++)
{
  a[0] = AL[j][0]; a[1] = AL[j][1];
  g  = S[j][0][0] + S[j][1][0] + a[0];
  g1 = S[j][0][1] + S[j][1][1] + a[1];
  if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
      (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
      (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
      (S[j][2][0] < S[j][0][0] + S[j][1][0] + 1) &&
      (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
      (S[j][2][1] < S[j][0][1] + S[j][1][1] + 1))
  {
    sa[j] = gera_probabilidade(j, j, a);
  }
  else {sa[j] = 0;}
}
ija[0] = Ttot + 1;
k = Ttot;
for (i = 0; i < Ttot; i++)
```

```
{
  a[0] = AL[i][0]; a[1] = AL[i][1];
  g  = S[i][0][0] + S[i][1][0] + a[0];
  g1 = S[i][0][1] + S[i][1][1] + a[1];
  for (j = 0; j < Ttot; j++)
  {
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
    {
      caux1 = gera_probabilidade(i, j, a);
    }
    else {caux1 = 0;}
    if (caux1 > 0 && i != j)
    {
      k = k + 1;
      if (k > nmax) {printf("  erro a lista tem tamanho pequeno\n"); getchar();}
      else
      {
        sa[k] = caux1;
        ija[k] = j;
      }
    }
  }
  ija[i + 1] = k + 1;
}
printf("    EHRA12 - Estados matriz de probabilidades condensada: ija[0]-1 %d\n",
          ija[0]-1);
printf("    Numero de entradas diferentes de zero: ija[ija[0]-1] %d\n",
          ija[ija[0]-1]-1);

/*-----------------------------------------------------------------------------*/
/*  Produto da matriz acumulada pela matriz de probabilidades - EHRA12         */
/*-----------------------------------------------------------------------------*/

caux1 = 2;
cont = 0;
while (caux1 > 1.000001 || caux1 - 1 < (0 - 0.000001))
{
  time2 = time(0);
  printf("   \n");
  printf("    EHRA12 - produto da matriz: %d\n", cont + 1);
  for (j = 0; j < Ttot; j++)
  {
    a[0] = AL[j][0]; a[1] = AL[j][1];
    g  = S[j][0][0] + S[j][1][0] + a[0];
    g1 = S[j][0][1] + S[j][1][1] + a[1];
    if ( sa[j] > 0 && (S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
                      (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
                      (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
                      (S[j][2][0] < S[j][0][0] + S[j][1][0] + 1) &&
                      (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
                      (S[j][2][1] < S[j][0][1] + S[j][1][1] + 1))
    {
      st[j] = sa[j] * gera_probabilidade(j, j, a);
    }
    else {st[j] = 0;}
    for (k = ija[j]; k < ija[j + 1]; k++)
    {
      i = ija[k];
```

```
    a[0] = AL[i][0]; a[1] = AL[i][1];
    g  = S[i][0][0] + S[i][1][0] + a[0];
    g1 = S[i][0][1] + S[i][1][1] + a[1];
    if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
    {
      st[j] = st[j] + sa[k] * gera_probabilidade(i, j, a);
    }
  }
}
ijt[0] = Ttot + 1;
k = Ttot;
for (i = 0; i < Ttot; i++)
{
  for (j = 0; j < Ttot; j++)
  {
    if (j != i)
    {
      a[0] = AL[i][0]; a[1] = AL[i][1];
      g  = S[i][0][0] + S[i][1][0] + a[0];
      g1 = S[i][0][1] + S[i][1][1] + a[1];
      if ( sa[i] > 0 && (S[j][0][0] + S[j][1][0] + S[j][2][0] == g) &&
                        (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
                        (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
                        (S[j][2][0] < S[i][0][0] + S[i][1][0] + 1) &&
                        (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
                        (S[j][2][1] < S[i][0][1] + S[i][1][1] + 1))
      {
        caux1 = sa[i] * gera_probabilidade(i, j, a);
      }
      else {caux1 = 0;}
      for (l = ija[i]; l < ija[i + 1]; l++)
      {
        a[0] = AL[ija[l]][0]; a[1] = AL[ija[l]][1];
        g  = S[ija[l]][0][0] + S[ija[l]][1][0] + a[0];
        g1 = S[ija[l]][0][1] + S[ija[l]][1][1] + a[1];
        if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
            (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
            (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
            (S[j][2][0] < S[ija[l]][0][0] + S[ija[l]][1][0] + 1) &&
            (S[j][0][1] + S[j][1][1] > a[1] - 1) &&
            (S[j][2][1] < S[ija[l]][0][1] + S[ija[l]][1][1] + 1))
        {
          caux1 = caux1 +  sa[l] * gera_probabilidade(ija[l], j, a);
        }
      }
      if (caux1 > 0)
      {
        k = k + 1;
        if (k > nmax) {printf("  erro a lista tem tamanho pequeno\n"); getchar();}
        else
        {
          st[k] = caux1;
          ijt[k] = j;
        }
      }
    }
  }
  ijt[i + 1] = k + 1;
```

```
  }
  printf(" \n");
  printf("  EHRA12 - Fim do produto da matriz: %d    numero de estados: ijt[0]-1 %d\n",
           cont + 1, ijt[0]-1);
  printf("    Numero de entradas diferentes de zero: ijt[ija[0]-1]-1 %d\n",
            ijt[ijt[0]-1] - 1);
  printf("    Tempo de processamento em minutos  %f\n", (time(0) - time2)/60);
  printf(" \n");
  caux1 = 0;
  for (k = 0; k < ijt[ijt[0]-1] - 1; k++)
  {
    if (k < Ttot) {caux1 = caux1 + st[k];}
    sa[k] = st[k];
    ija[k] = ijt[k];
  }
  printf("    EHRA12 - Soma da diagonal: %f\n", caux1);
  cont = cont + 1;
}

fprintf(fp,"\n");
fprintf(fp,"   EHRA12 - Soma da diagonal: %f\n", caux1);
for (i = 0; i < Ttot; i++)
{
  fprintf(fp," %f\n", sa[i]);
}
fclose(fp);
printf(" \n");
printf("  Os resulatdos foram gravados no arquivo Probab.log\n");
printf(" \n");
printf("***  FIM  ***");
printf(" \n");

getchar();
return (EXIT_SUCCESS);
}

/*-------------------------------------------------------*/
/*          Gera espaco de estados                       */
/*-------------------------------------------------------*/

void gera_espaco_estados()
{
  int maxesp[3][2];
  int d, i, j, aux, cont, cont1, i1, i2,
      a1, a2, a3, b1, b2, b3, c1, c2, c3, d1, d2, d3,
      a4, a5, a6, n, nt, nt1, maxA, maxB, maxC;

/*-------------------------------------------------------------------------*/
/*      Estabelece o maximo de pacientes de cada especialidade             */
/*      em cada padrao, para os estados que podem admitir novos pacientes  */
/*      segundo as restricoes estabelecidas pelo modelo                    */
/*-------------------------------------------------------------------------*/

for (i = 0; i < Npad-1; i++)
{
  for (d = 0; d < Nesp; d++)
  {
    aux = 0;
    while (((P[d][i][0] * aux * L[0][0] + P[d][i][1] * aux * L[1][0]) <= LK[0]) &&
           ((P[d][i][0] * aux * L[0][1] + P[d][i][1] * aux * L[1][1]) <= LK[1]))
    {
      aux = aux + 1;
    }
```

```
    maxesp[i][d] = aux -1;
  }
}
printf("Para os estados que podem admitir pacientes");
printf("\n");
for (i = 0; i < Npad - 1; i++)
{
  printf("pad: %d", i);
  printf("\n");
  for (d = 0; d < Nesp; d++)
  {
    printf(" esp: %d maximo de pac: %d\n", d, maxesp[i][d]);
  }
}

/*-----------------------------------------------------------------*/
/*      Gera os estados que podem admitir novos pacientes          */
/*-----------------------------------------------------------------*/

n = 0;
a1 = -1;
while (a1 < maxesp[0][0])
{
  a1 = a1 + 1;
  a2 = -1;
  while (a2 < maxesp[0][1])
  {
    a2 = a2 + 1;
    b1 = -1;
    while (b1 < maxesp[1][0])
    {
      b1 = b1 + 1;
      b2 = -1;
      while (b2 < maxesp[1][1])
      {
        b2 = b2 + 1;
        if (((a1 * P[0][0][0]+ b1 * P[0][1][0] +
              a2 * P[1][0][0]+ b2 * P[1][1][0]) * L[0][0] +
             (a1 * P[0][0][1]+ b1 * P[0][1][1] +
              a2 * P[1][0][1]+ b2 * P[1][1][1]) * L[1][0] <= LK[0]) &&
            ((a1 * P[0][0][0]+ b1 * P[0][1][0] +
              a2 * P[1][0][0]+ b2 * P[1][1][0]) * L[0][1] +
             (a1 * P[0][0][1]+ b1 * P[0][1][1] +
              a2 * P[1][0][1]+ b2 * P[1][1][1]) * L[1][1] < LK[1]))
        {
          S[n][0][0] = a1;
          S[n][0][1] = a2;
          S[n][1][0] = b1;
          S[n][1][1] = b2;
          S[n][2][0] = 0;
          S[n][2][1] = 0;
          n = n + 1;
        }
      }
    }
  }
}
printf("\n");
printf("quantidade de estados onde ainda se pode admitir pacientes n: %d", n);
printf("\n");
Tact = n;
```

```
/*-------------------------------------------------------------------------*/
/*      Gera os estados correspondentes a admissao de novos pacientes      */
/*-------------------------------------------------------------------------*/

nt = n;
aux = 0;
for (i = 0; i < nt; i++)
{
  for (a4 = 0; a4 < 1 + Smx[0]; a4++)
  {
    for (a5 = 0; a5 < 1 + Smx[1]; a5++)
    {
      a1 = -1;
      while (a1 < a4)
      {
        a1 = a1 + 1;
        a2 = -1;
        while (a2 < a5)
        {
          a2 = a2 + 1;
          b1 = -1;
          while (b1 < a4)
          {
            b1 = b1 + 1;
            b2 = -1;
            while (b2 < a5)
            {
              b2 = b2 + 1;
              aux = aux + 1;
              if ((a1 + b1 == a4) && (a2 + b2 == a5))
              {
                cont = 0;
                i1 = 0;
                while ((i1 < n) && (cont == 0))
                {
                  if (S[i1][0][0] == a1 + S[i][0][0] &&
                      S[i1][0][1] == a2 + S[i][0][1] &&
                      S[i1][1][0] == b1 + S[i][1][0] &&
                      S[i1][1][1] == b2 + S[i][1][1])
                  {
                    cont = 1;
                  }
                  i1 = i1 + 1;
                }
                if (cont == 0)
                {
                  S[n][0][0] = a1 + S[i][0][0];
                  S[n][0][1] = a2 + S[i][0][1];
                  S[n][1][0] = b1 + S[i][1][0];
                  S[n][1][1] = b2 + S[i][1][1];
                  S[n][2][0] = 0;
                  S[n][2][1] = 0;
                  n = n + 1;
                }
              }
            }
          }
        }
      }
    }
  }
}
printf("\n");
```

```
printf("\n");
printf("acrescidos os estados apos a admissao de pacientes: %d", n);
printf("\n");

/*-------------------------------------------------------------------------*/
/*      Gera os estados resultantes das possiveis movimentacoes            */
/*-------------------------------------------------------------------------*/

aux = 0;
nt = n;
for (i = 0; i < nt; i++)
{
  maxA = S[i][0][0] + S[i][1][0];
  maxB = S[i][0][1] + S[i][1][1];
  a1 = -1;
  while (a1 < maxA)
  {
    a1 = a1 + 1;
    a2 = -1;
    while (a2 < maxB)
    {
      a2 = a2 + 1;
      b1 = -1;
      while (b1 < maxA)
      {
        b1 = b1 + 1;
        b2 = -1;
        while (b2 < maxB)
        {
          b2 = b2 + 1;
          aux = aux + 1;
          if ((a1 + b1 == maxA) && (a2 + b2 == maxB))
          {
            cont = 0;
            i1 = 0;
            while ((i1 < n) && (cont == 0))
            {
              if (S[i1][0][0] == a1 &&
                  S[i1][0][1] == a2 &&
                  S[i1][1][0] == b1 &&
                  S[i1][1][1] == b2)
              {
                cont = 1;
              }
              i1 = i1 + 1;
            }
            if (cont == 0)
            {
              S[n][0][0] = a1;
              S[n][0][1] = a2;
              S[n][1][0] = b1;
              S[n][1][1] = b2;
              S[n][2][0] = 0;
              S[n][2][1] = 0;
              n = n + 1;
            }
          }
        }
      }
    }
  }
}
```

```
printf("\n");
printf("\n");
printf("acrescidos os estados gerados pelas movimentacoes: %d", n);
printf("\n");

/*-------------------------------------------------------------------------*/
/*      Inclui os estados relacionados as possiveis movimentacoes          */
/*      para o padrao correspondente a alta hospitalar                     */
/*-------------------------------------------------------------------------*/

aux = 0;
nt = n;
for (i = 0; i < nt; i++)
{
  maxA = S[i][0][0] + S[i][1][0];
  maxB = S[i][0][1] + S[i][1][1];
  cont1 = 0;
  i2 = 0;
  while ((i2 < i) && (cont1 == 0))
  {
    if (maxA == S[i2][0][0] + S[i2][1][0] &&
        maxB == S[i2][0][1] + S[i2][1][1])
    {
      cont1 = 1;
    }
    i2 = i2 + 1;
  }
  if (cont1 == 0)
  {
    a1 = -1;
    while (a1 < maxA)
    {
      a1 = a1 + 1;
      a2 = -1;
      while (a2 < maxB)
      {
        a2 = a2 + 1;
        b1 = -1;
        while (b1 < maxA)
        {
          b1 = b1 + 1;
          b2 = -1;
          while (b2 < maxB)
          {
            b2 = b2 + 1;
            d1 = -1;
            while (d1 < maxA)
            {
              d1 = d1 + 1;
              d2 = -1;
              while (d2 < maxB)
              {
                d2 = d2 + 1;
                aux = aux + 1;
                if ((a1 + b1 + d1 == maxA) && (a2 + b2 + d2 == maxB))
                {
                  cont = 0;
                  i1 = 0;
                  while ((i1 < n) && (cont == 0))
                  {
                    if (S[i1][0][0] == a1 &&
                        S[i1][0][1] == a2 &&
                        S[i1][1][0] == b1 &&
```

```
                        S[i1][1][1] == b2 &&
                        S[i1][2][0] == d1 &&
                        S[i1][2][1] == d2)
                    {
                      cont = 1;
                    }
                    i1 = i1 + 1;
                  }
                  if (cont == 0)
                  {
                    S[n][0][0] = a1;
                    S[n][0][1] = a2;
                    S[n][1][0] = b1;
                    S[n][1][1] = b2;
                    S[n][2][0] = d1;
                    S[n][2][1] = d2;
                    n = n + 1;
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
printf("\n");
printf("\n");
printf("acrescidos os estados correspondentes as variacoes das altas: %d", n);
printf("\n");
printf("\n");
Ttot = n;
}

/*---------------------------------------------------------*/
/*      Fatorial                                           */
/*---------------------------------------------------------*/

double fat(int arg)
{
  double n, Fatx, x;
  if (arg < 0)
  {
    printf("\n");
    printf("Valor negativo no fatorial");
    printf("\n");
    getchar();
  }
  Fatx = 1;
  x = (double) arg;
  for (n=x; n>1; n--)
  {
    Fatx*=n;
  }
  return Fatx;
}
```

```
/*-------------------------------------------------------*/
/*  gera probabilidade de transicao entre dois estados, */
/*  com metodo de entrada correspondente a declaracao   */
/*  do estado observado, do estado a ser atingido e da  */
/*  acao adotada                                        */
/*-------------------------------------------------------*/

double gera_probabilidade(int i1, int i2, int ac[2])
{
  int    d, i, y0, y1, y2, x00, x01, x02, x03,
         x10, x11, x12, x13, x20, x21, x22, x23,
         infy0, supy0, infy1, supy1, infx00, supx00,
         infx01, supx01, infx02, supx02, infx03, supx03,
         infx10, supx10, infx11, supx11, infx12, supx12,
         infx13, supx13, infx20, supx20, infx21, supx21,
         infx22, supx22, infx23, supx23,cont;
  double Pesp, Ptot;

/*-------------------------------------------------------*/
/*  Determinando os limites inferiores e superiores das */
/*  somatorias                                          */
/*-------------------------------------------------------*/

Ptot = 1;
for (i = 0; i < Nesp; i++)
{
  cont = 0;
  Pesp = 0;
  if (S[i2][0][i] - (S[i1][0][i] + S[i1][1][i] - S[i2][2][i]) > 0)
  {
    infy0 = S[i2][0][i] - (S[i1][0][i] + S[i1][1][i] - S[i2][2][i]);
  }
  else
  {
    infy0 = 0;
  }
  if (S[i2][0][i] < ac[i])
  {
    supy0 = S[i2][0][i];
  }
  else
  {
    supy0 = ac[i];
  }
  for (y0 = infy0; y0 < supy0 + 1; y0++)
  {
    if (S[i2][0][i] - (S[i1][1][i] + y0) > 0)
    {
      infx00 = S[i2][0][i] - (S[i1][1][i] + y0);
    }
    else
    {
      infx00 = 0;
    }
    if (S[i2][0][i] - y0 < S[i1][0][i])
    {
      supx00 = S[i2][0][i] - y0;
    }
    else
    {
      supx00 = S[i1][0][i];
    }
    for (x00 = infx00; x00 < supx00 + 1; x00++)
```

```
{
  if (S[i2][0][i] - (x00 + y0) > 0)
  {
    infx10 = S[i2][0][i] - (x00 + y0);
  }
  else
  {
    infx10 = 0;
  }
  if (S[i2][0][i] - (x00 + y0) < S[i1][1][i])
  {
    supx10 = S[i2][0][i] - (x00 + y0);
  }
  else
  {
    supx10 = S[i1][1][i];
  }
  for (x10 = infx10; x10 < supx10 + 1; x10++)
  {
    y1 = ac[i] - y0;
    if (S[i2][1][i] - (S[i1][1][i] - x10 + y1) > 0)
    {
      infx01 = S[i2][1][i] - (S[i1][1][i] - x10 + y1);
    }
    else
    {
      infx01 = 0;
    }
    if (S[i2][1][i] - y1 < S[i1][0][i] - x00)
    {
      supx01 = S[i2][1][i] - y1;
    }
    else
    {
      supx01 = S[i1][0][i] - x00;
    }
    for (x01 = infx01; x01 < supx01 + 1; x01++)
    {
      if (S[i2][1][i] - (x01 + y1) > 0)
      {
        infx11 = S[i2][1][i] - (x01 + y1);
      }
      else
      {
        infx11 = 0;
      }
      if (S[i2][1][i] - (x01 + y1) < S[i1][1][i] - x10)
      {
        supx11 = S[i2][1][i] - (x01 + y1);
      }
      else
      {
        supx11 = S[i1][1][i] - x10;
      }
      for (x11 = infx11; x11 < supx11 + 1; x11++)
      {
        if (S[i2][2][i] - (S[i1][1][i] - x10 - x11) > 0)
        {
          infx02 = S[i2][2][i] - (S[i1][1][i] - x10 - x11);
        }
        else
        {
          infx02 = 0;
```

```
            }
            if (S[i2][2][i] < S[i1][0][i] - x00 - x01)
            {
              supx02 = S[i2][2][i];
            }
            else
            {
              supx02 = S[i1][0][i] - x00 - x01;
            }
            for (x02 = infx02; x02 < supx02 + 1; x02++)
            {
              if (S[i2][2][i] - (x02) > 0)
              {
                infx12 = S[i2][2][i] - (x02);
              }
              else
              {
                infx12 = 0;
              }
              if (S[i2][2][i] - x02 < S[i1][1][i] - x10 - x11)
              {
                supx12 = S[i2][2][i] - x02;
              }
              else
              {
                supx12 = S[i1][1][i] - x10 - x11;
              }
              for (x12 = infx12; x12 < supx12 + 1; x12++)
              {
                Pesp = Pesp +
                     (fat(ac[i])/(fat(y0)*fat(y1)))*(pow(PE[0][i],y0)*pow(PE[1][i],y1))*
                     (fat(S[i1][0][i])/(fat(x00)*fat(x01)*fat(x02)))*
                       (pow(P[i][0][0],x00)*pow(P[i][0][1],x01)*pow(P[i][0][2],x02))*
                     (fat(S[i1][1][i])/(fat(x10)*fat(x11)*fat(x12)))*
                       (pow(P[i][1][0],x10)* pow(P[i][1][1],x11)* pow(P[i][1][2],x12));
              }
            }
          }
        }
      }
    }
  }
  Ptot = Ptot * Pesp;
}
return Ptot;
}

/*-------------------------------------------------------*/
/*            gera_acao_aleatoria                        */
/*-------------------------------------------------------*/

void gera_acao_aleatoria()
{
  int d;
  for (d = 0; d < Nesp; d++)
  {
    gera_ale();
    if (u <1)
    {
      a[d] = (int)floor(( Smx[d] +1) * u);
    }
    if (u == 0.999)
    {
```

```
      a[d] = Smx[d];
    }
  }
}

/*---------------------------------------------------------*/
/*                    gera_ale                             */
/*---------------------------------------------------------*/

void gera_ale()
{
  if (Nale >= Nalemx)
  {
    srand((unsigned int)time(NULL));
    Nale = 0;
  }
  u = rand();
  u = u / RAND_MAX;
  if (u == 0.0)
  {
    u = 0.001;
  }
  if (u == 1)
  {
    u = 0.999;
  }
  Nale = Nale + 1;
}

/*---------------------------------------------------------*/
/*           gera_proximo_estado_custo                     */
/*---------------------------------------------------------*/

void gera_proximo_estado_custo()
{
  int    d, i, l, j, g, g1;
  int    aux, cont;
  int    xaux[3][2];
  double aux2, caux1, cust, caux2;

/*-----------------------------------------------------*/
/*  Gera uma matriz auxiliar contendo o estado "sem */
/*  pacientes"                                        */
/*-----------------------------------------------------*/

  for (d = 0; d < Nesp; d++)
  {
    for (i = 0; i < Npad; i++)
    {
      xaux[i][d] = 0;
    }
  }

  /*-----------------------------------------------------*/
  /* Gera, de acordo com uma distribuicao Multinomial,*/
  /* o número esperado de pacientes, de cada          */
  /* especialidade em cada padrao de consumo, no      */
  /* proximo periodo de funcionamento do sistema      */
  /*-----------------------------------------------------*/

  for (d = 0; d < Nesp; d++)
  {
    for (i = 0; i < Npad; i++)
```

```
  {
    aux = xx[i][d];
    cont = 0;
    while (cont < aux)
    {
      j = -1;
      gera_ale();
      if (u == 0) {j = 0;}
      aux2 = 0;
      while (u > aux2)
      {
        j = j + 1;
        aux2 = aux2 + P[d][i][j];
      }
      xaux[j][d] = xaux[j][d] + 1;
      cont = cont + 1;
    }
  }
  aux = a[d];
  cont = 0;
  while (cont < aux)
  {
    j = -1;
    gera_ale();
    if (u == 0) {j = 0;}
    aux2 = 0;
    while (u > aux2)
    {
      j = j + 1;
      aux2 = aux2 + PE[j][d];
    }
    xaux[j][d] = xaux[j][d] + 1;
    cont = cont + 1;
  }
}

/*-------------------------------------------------*/
/* Gera custo esperado, dado o estado xx[i,d]    */
/* e a acao a[d]                                   */
/*-------------------------------------------------*/

g  = xx[0][0] + xx[1][0] + a[0];
g1 = xx[0][1] + xx[1][1] + a[1];
CLT = 0;
for (j = 0; j < Ttot; j++)
{
  if ((S[j][0][0] + S[j][1][0] + S[j][2][0] == g)  &&
      (S[j][0][1] + S[j][1][1] + S[j][2][1] == g1) &&
      (S[j][0][0] + S[j][1][0] > a[0] - 1) &&
      (S[j][2][0] < xx[0][0] + xx[1][0] + 1) &&
      (S[j][0][1] + S[j][1][1] > a[ 1] - 1) &&
      (S[j][2][1] < xx[0][1] + xx[1][1] + 1))
  {
    caux1 = gera_probabilidade_1(j, a);
    cust = 0;
    for (l = 0; l < Nrec; l++)
    {
      caux2 = 0;
      for (d = 0; d < Npad; d++)
      {
        caux2 = caux2 + (S[j][d][0] + S[j][d][1]) * L[d][l];
      }
      if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
```

```
        if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
        if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
      }
      CLT = CLT + (cust * caux1);
    }
  }

  /*-----------------------------------------------------*/
  /* Gera uma matriz contendo o novo estado              */
  /*                                                     */
  /*-----------------------------------------------------*/

  for (d = 0; d < Nesp; d++)
  {
    for (i = 0; i < Npad; i++)
    {
      xx[i][d] = xaux[i][d];
    }
  }
}

/*-----------------------------------------------------------*/
/*          gera_acao_nearest_neighbor                       */
/*-----------------------------------------------------------*/

void gera_acao_nearest_neighbor()
{
  int lmi, lmx;
  int d;
  for (d = 0; d < Nesp; d++)
  {
    lmi = A[e][d] - R[d];
    lmx = A[e][d] + R[d];
    if (lmi < 0)
    {
      lmi = 0;
    }
    if (lmx > Smx[d])
    {
      lmx = Smx[d];
    }
    gera_ale();
    if (u < 1)
    {
      a[d] = (int)floor(lmi+ (lmx-lmi +1) * u);
    }
    if (u == 0.999)
    {
      a[d] = lmx;
    }
  }
}

/*-----------------------------------------------------------*/
/*                compara_vetor                              */
/*-----------------------------------------------------------*/

int compara_vetor(int v1, int v2)
{
  int d;
  for (d = 0; d < Nesp; d++)
  {
    if (A[v1][d] != VCstp[v2].b[d])
```

```
    {
      return 0;
    }
  }
  return 1;
}

/*---------------------------------------------------------*/
/*  Gera a probabilidade de transicao entre dois           */
/*  estados, sendo o estado observado dado por um vetor    */
/*  corrente xx[i][d]. A entrada corresponde ao estado     */
/*  a ser atingido (i2) e a acao adotada (ac[2]).          */
/*---------------------------------------------------------*/

double gera_probabilidade_1(int i2, int ac[2])

{
  int    d, i, y0, y1, y2, x00, x01, x02, x03,
         x10, x11, x12, x13, x20, x21, x22, x23,
         infy0, supy0, infy1, supy1, infx00, supx00,
         infx01, supx01, infx02, supx02, infx03, supx03,
         infx10, supx10, infx11, supx11, infx12, supx12,
         infx13, supx13, infx20, supx20, infx21, supx21,
         infx22, supx22, infx23, supx23,cont;
  double Pesp, Ptot;

/*---------------------------------------------------------*/
/*  Determinando os limites inferiores e superiores das    */
/*  somatorias                                             */
/*---------------------------------------------------------*/

 Ptot = 1;
 for (i = 0; i < Nesp; i++)
 {
   cont = 0;
   Pesp = 0;
   if (S[i2][0][i] - (xx[0][i] + xx[1][i] - S[i2][2][i]) > 0)
   {
     infy0 = S[i2][0][i] - (xx[0][i] + xx[1][i] - S[i2][2][i]);
   }
   else
   {
     infy0 = 0;
   }
   if (S[i2][0][i] < ac[i])
   {
     supy0 = S[i2][0][i];
   }
   else
   {
     supy0 = ac[i];
   }
   for (y0 = infy0; y0 < supy0 + 1; y0++)
   {
     if (S[i2][0][i] - (xx[1][i] + y0) > 0)
     {
       infx00 = S[i2][0][i] - (xx[1][i] + y0);
     }
     else
     {
       infx00 = 0;
     }
     if (S[i2][0][i] - y0 < xx[0][i])
```

```
{
  supx00 = S[i2][0][i] - y0;
}
else
{
  supx00 = xx[0][i];
}
for (x00 = infx00; x00 < supx00 + 1; x00++)
{
  if (S[i2][0][i] - (x00 + y0) > 0)
  {
    infx10 = S[i2][0][i] - (x00 + y0);
  }
  else
  {
    infx10 = 0;
  }
  if (S[i2][0][i] - (x00 + y0) < xx[1][i])
  {
    supx10 = S[i2][0][i] - (x00 + y0);
  }
  else
  {
    supx10 = xx[1][i];
  }
    for (x10 = infx10; x10 < supx10 + 1; x10++)
    {
      y1 = ac[i] - y0;
      if (S[i2][1][i] - (xx[1][i] - x10 + y1) > 0)
      {
        infx01 = S[i2][1][i] - (xx[1][i] - x10 + y1);
      }
      else
      {
        infx01 = 0;
      }
      if (S[i2][1][i] - y1 < xx[0][i] - x00)
      {
        supx01 = S[i2][1][i] - y1;
      }
      else
      {
        supx01 = xx[0][i] - x00;
      }
      for (x01 = infx01; x01 < supx01 + 1; x01++)
      {
        if (S[i2][1][i] - (x01 + y1) > 0)
        {
          infx11 = S[i2][1][i] - (x01 + y1);
        }
        else
        {
          infx11 = 0;
        }
        if (S[i2][1][i] - (x01 + y1) < xx[1][i] - x10)
        {
          supx11 = S[i2][1][i] - (x01 + y1);
        }
        else
        {
          supx11 = xx[1][i] - x10;
        }
        for (x11 = infx11; x11 < supx11 + 1; x11++)
```

```
            {
              if (S[i2][2][i] - (xx[1][i] - x10 - x11) > 0)
              {
                infx02 = S[i2][2][i] - (xx[1][i] - x10 - x11);
              }
              else
              {
                infx02 = 0;
              }
              if (S[i2][2][i] < xx[0][i] - x00 - x01)
              {
                supx02 = S[i2][2][i];
              }
              else
              {
                supx02 = xx[0][i] - x00 - x01;
              }
              for (x02 = infx02; x02 < supx02 + 1; x02++)
              {
                if (S[i2][2][i] - (x02) > 0)
                {
                  infx12 = S[i2][2][i] - (x02);
                }
                else
                {
                  infx12 = 0;
                }
                if (S[i2][2][i] - x02 < xx[1][i] - x10 - x11)
                {
                  supx12 = S[i2][2][i] - x02;
                }
                else
                {
                  supx12 = xx[1][i] - x10 - x11;
                }
                for (x12 = infx12; x12 < supx12 + 1; x12++)
                {
                  Pesp = Pesp +
                    (fat(ac[i])/(fat(y0)*fat(y1)))*
                      (pow(PE[0][i],y0)*pow(PE[1][i],y1))*
                    (fat(xx[0][i])/(fat(x00)*fat(x01)*fat(x02)))*
                      (pow(P[i][0][0],x00)* pow(P[i][0][1],x01)* pow(P[i][0][2],x02))*
                    (fat(xx[1][i])/(fat(x10)*fat(x11)*fat(x12)))*
                      (pow(P[i][1][0],x10)* pow(P[i][1][1],x11)* pow(P[i][1][2],x12));
                }
              }
            }
          }
        }
      }
      Ptot = Ptot * Pesp;
    }
    return Ptot;
}
```

```
/*-------------------------------------------------------*/
/*  DeDefinicao do mehra4.h                              */
/*-------------------------------------------------------*/

#ifndef _MEHRA_H
#define _MEHRA_H
/* _Nesp: Numero de especialidades médicas*/
/* ========>>>>>>> PROBLEMAS NO STRUCT    <<<<<<<========== */
#define _Nesp 3
/* _Npad: Numero de padroes de consumo */
#define _Npad 4
/* _Nrec: Numero de recursos */
#define _Nrec 2
/* _N: Tamanho da populacao de acoes a ser avaliada a cada iteracao, N >1 */
#define _N  20
/* Definicao de null... nao presente em C */
#define bilhao 1000000000
#define null 0
#define _Nalemx 100
#define _Na 20
#define _C 40
#define _K 1
#define _H 30
struct Cstp
{
  int    b[3];
  double som;
  double med;
  int    N1;
};
struct Custos
{
  int    y[4][2];
  int    aa[2];
  double custo;
};
#endif
```

B.2 Determinação de ações

Dado um estado observado, apresenta-se, a seguir, o código em linguagem C para determinação de uma ação pelos métodos gulosos G1 e G2 e EHRA2 (Subseção 4.2.2 do Capítulo 4), considerando o PMD modelado para o controle de admissão de pacientes eletivos (Capítulo 2), com as modificações, as probabilidades e os parâmetros definidos na Seção 4.3 do Capítulo 4.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <time.h>
#include <math.h>
#include "mehra4.h"

/*-------------------------------------------------------*/
/* Criacao e alocacao de valores banais pras variaveis */
/*-------------------------------------------------------*/

int Nesp   = 0; // Numeros de especialidades
int Npad   = 0; // Numero de padroes
int Nrec   = 0; // Numero de recursos
int N      = 0; // Tamanho da populacao de acoes
int Nalemx = 0; // Quantidade msxima de numeros aleatorios gerados
int Na     = 0; // Tamanho do vetor de acoes para criterio de parada
int Nr     = 0; // Acoes com menores custos a serem mantidas na pop.
int C      = 0; // Numero de amostras de caminhos de funcionamento
int K      = 0; // Politicas para o metodo parallel rollout
double GAMA   = 0.0; // Fator de desconto
double Q      = 0.0; // Probabilidade de busca local ou global
int Stp    = 0; // Numero de repeticoes de uma acao para parada
int Nale   = 0; // Contador para o numero de numeros aleatorios
int H      = 0; // Tamanho do horizonte de planejamento

/*-------------------------------------------------------*/
/*        Processo de inicializacao efetivas           */
/*-------------------------------------------------------*/

double    LE[2]; // Custo de excesso sobre o nivel desejado para cada recurso
double    LR[2]; // Custo de excesso sobre a capacidade maxima p/cada recurso
double    LO[2]; // Custo de ociosidade para cada recurso
double    NL[2]; // Nivel de consumo desejado para cada recurso
double    LK[2]; // Capacidade maxima de consumo de cada recurso
double L[2][4][2]; // Consumo: para cada padrao, consumo de cada recurso
int    x[4][2]; // Estado: para cada padrao, paciente por especialidade
int    a[2]; // Acao: quantos pacientes de cada especialidade
int    y[2]; // Acao: quantos pacientes de cada especialidade
int    Smx[2]; // Capacidade maxima de admissao para cada especialidade
double P[2][4][4]; // Por especialidade: prob de transicoes entre padroes
double PE[4][2]; // Por padrao: prob de entrada de cada especialidade
double PRE[4][2]; // Por padrao: prob de reentrada de cada especialidade
double V[2][4]; // Para acumular o numero de pacientes em cada padrao
int    A[40][2]; // Matriz: armazena as acoes de uma populacao de acoes
int    R[2]; // Amplitude para a alteracao de acoes na busca local
struct Cstp VCstp[10]; // Criada em mehra4.h: para o genetico e para a parada
int    xx[4][2]; // Auxiliar: mesma definicao que x[4][2]
int    xxx[4][2]; // Auxiliar: mesma definicao que x[4][2]
int    npermuta[10]; // Auxiliar: para permutacao, tamanho Nesp * Npad
int    mpermuta[25]; // Auxiliar: para permutacao, tamanho Npad * Npad
struct Custos CC[100000]; // Criada em mehra4.h: para armazenar custos gerados
int    RE[2]; // Pacientes que retomam seus tratamentos

/*-------------------------------------------------------*/
/*       Variaveis Auxiliares                            */
/*-------------------------------------------------------*/

double CLT; // Custo total estimado
double custtot, prob1; // Custo auxiliar para a procedure gulosa
double u, probesp; // Numero aleatorio e probabilidade de uma especialidade
int e; // Acao de elite
int auxint, auxmax, auxint1, auxmax1, nn, m, mm; // Variaveis auxiliares
```

```
/*---------------------------------------------------------*/
/*             Prototipos                                  */
/*---------------------------------------------------------*/

void gera_acao_aleatoria(); // Gera acao aleatoria para iniciar as interacoes
void gera_ale(); // Gera um numero aleatorio: distribuicao uniforme
void gera_proximo_estado_custo(); // Simulacao: proximo estado e custo
void gera_acao_gulosa2(); // Dado um estado, gera uma acao gulosa1
int compara_vetor(int v1, int v2); // Compara dois vetores
void gera_acao_nearest_neighbor(); // Gera acao na vizinhanca de uma dada acao
void permuta(int nivel); // Gera os possiveis proximos estados, sem admissao
double fat(int arg); // Calcula o fatorial do argumento
double gera_probabilidade(int ac[2]); // Probab de transicao
void gera_acao_gulosa1(); // Dado um estado, gera uma acao gulosa

/*---------------------------------------------------------*/
/*         main                                            */
/*---------------------------------------------------------*/

int main(int argc, char** argv)
{
  int    FIM1, FIM, e, cont;
  double aux0, naux, naux1, time1, time2, time3, timeG1, timeG2,
         caux, caux1, auxstp, raux;
  int    n, n1, n2, na, nr, i, j, k, d, h, t, l,
         ggg, g, g1, f, f1, f2, f3, f4, f5;

//Iniciar o Programa:
time1 = time(null);

/*---------------------------------------------------------*/
/*         Parametros do modelo                            */
/*---------------------------------------------------------*/

  Nesp = 2;

  Npad = 4;

  Nrec = 2;

  LE[0] = 1.25; LE[1] = 1.0;

  LR[0] = 1.25; LR[1] = 1.0;

  LO[0] = 1.6; LO[1] = 1.28;

  NL[0] = 5; NL[1] = 5;

  LK[0] = 6; LK[1] = 6;

  Smx[0] = 2; Smx[1] = 2;

  RE[0] = 1; RE[1] = 1;

  P[0][0][0] = 0.0160; P[0][0][1] = 0.0080; P[0][0][2] = 0.1820; P[0][0][3] = 0.7940;
  P[0][1][0] = 0.0178; P[0][1][1] = 0.0210; P[0][1][2] = 0.0226; P[0][1][3] = 0.9386;
  P[0][2][0] = 0.0305; P[0][2][1] = 0.0152; P[0][2][2] = 0.5488; P[0][2][3] = 0.4055;
  P[0][3][0] = 0.0000; P[0][3][1] = 0.0000; P[0][3][2] = 0.0000; P[0][3][3] = 1.0000;

  P[1][0][0] = 0.0106; P[1][0][1] = 0.0192; P[1][0][2] = 0.0750; P[1][0][3] = 0.8952;
  P[1][1][0] = 0.0360; P[1][1][1] = 0.0420; P[1][1][2] = 0.0220; P[1][1][3] = 0.9000;
  P[1][2][0] = 0.0234; P[1][2][1] = 0.0252; P[1][2][2] = 0.3143; P[1][2][3] = 0.6371;
  P[1][3][0] = 0.0000; P[1][3][1] = 0.0000; P[1][3][2] = 0.0000; P[1][3][3] = 1.0000;
```

```
  PE[0][0] = 0.3247; PE[0][1] = 0.4044;
  PE[1][0] = 0.6753; PE[1][1] = 0.5956;
  PE[2][0] = 0.0000; PE[2][1] = 0.0000;
  PE[3][0] = 0.0000; PE[3][1] = 0.0000;

  PRE[0][0] = 0.3500; PRE[0][1] = 0.3000;
  PRE[1][0] = 0.5610; PRE[1][1] = 0.5470;
  PRE[2][0] = 0.0890; PRE[2][1] = 0.1530;
  PRE[3][0] = 0.0000; PRE[3][1] = 0.0000;

  L[0][0][0] = 1.078; L[0][0][1] = 1.698;
  L[0][1][0] = 1.032; L[0][1][1] = 0.000;
  L[0][2][0] = 0.000; L[0][2][1] = 3.643;
  L[0][3][0] = 0.000; L[0][3][1] = 0.000;

  L[1][0][0] = 1.118; L[1][0][1] = 1.286;
  L[1][1][0] = 1.057; L[1][1][1] = 0.000;
  L[1][2][0] = 0.000; L[1][2][1] = 2.454;
  L[1][3][0] = 0.000; L[1][3][1] = 0.000;

/*-----------------------------------------------------------*/
/*     Parametros do EHRA                                    */
/*-----------------------------------------------------------*/

  H = 4;

  C = 4;

  K = 1;

  GAMA = 0.8;

  N = 4;

  Na = 5;

  Nr = 1;

  Q = 0.9;

  R[0] = 1; R[1] = 1;

  Stp = 16;

  Nalemx = 30000;

  srand((unsigned int)time(null));

  Nale = 0;

/*-------------------------------------------------------------------------*/
/*     Estado observado                                                    */
/*-------------------------------------------------------------------------*/

  x[0][0] = 1; x[0][1] = 1;
  x[1][0] = 1; x[1][1] = 1;
  x[2][0] = 1; x[2][1] = 1;
  x[3][0] = 0; x[3][1] = 0;
```

```
/*--------------------------------------------------------------------------*/
/*     Gera acao EHRA2                                                      */
/*--------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------*/
/*  Teste primario: se o consumo medio estimado do proximo periodo de um dos*/
/*  recursos for maior que a quantidade disponivel, a unica acao permitida  */
/*  e "nao admitir pacientes no proximo periodo". E o programa se encerra.  */
/*--------------------------------------------------------------------------*/

 auxmax = 0;
 for (d = 0; d < Nesp; d++)
 {
   auxint = 0;
   for (i = 0; i < Npad; i++)
   {
     auxint = auxint + x[i][d];
     xx[i][d] = x[i][d];
   }
   auxint = auxint + RE[d];
   if (auxint > auxmax) {auxmax = auxint;}
 }

/*--------------------------------------------------------------------------*/
/*  Gera acao para a politica gulosa G1, depois contiunua                   */
/*  com o processo do EHRA2                                                 */
/*--------------------------------------------------------------------------*/

 time3 = time(null);
 gera_acao_gulosa1();
 timeG1 = time(null) - time3;
 y[0] = a[0]; y[1] = a[1];

 auxint = Npad * Nesp;
 for (i = 0; i < auxint; i++)
 {
   npermuta[i] = 0;
 }
 for (d = 0; d < Nesp; d++)
 {
   for (i = 0; i < Npad; i++)
   {
     V[d][i] = 0;
   }
 }
 mm = 0; nn = 0;
 a[0]= 0; a[1]= 0;
 custtot = 0; prob1 = 0;
 permuta(0);
 FIM1 = 0;
 for (l = 0; l < Nrec; l++)
 {
   aux0 = 0;
   for (d = 0; d < Nesp; d++)
   {
     for (i = 0; i < Npad; i++)
     {
       aux0 = aux0 + V[d][i] * L[d][i][l];
     }
   }
   if (aux0 > LK[l]) {FIM1 = 1;}
 }
if (FIM1 == 0)
```

```
{

/*---------------------------------------------------------------------*/
/*  Alimenta banco de estados acoes e custos que sera utilizado para   */
/*  acelerar o processamento do algoritmo                              */
/*---------------------------------------------------------------------*/

 for (d = 0; d < Nesp; d++)
 {
   for (i = 0; i < Npad; i++)
   {
     CC[mm].y[i][d] = xx[i][d];
   }
   CC[mm].aa[d] = a[d];
 }
 CC[mm].custo = custtot;
 mm = mm + 1;

/*-----------------------------------------------------*/
/*      Geracao de uma populacao inicial               */
/*      Mantendo as acoes das politicas de base        */
/*-----------------------------------------------------*/

 for (n = 0; n < N; n++ )
 {
   if (n == 0)
   {
     for (d = 0; d < Nesp; d++)
     {
       A[n][d] = y[d];
     }
   }
   if (n < N - K && n > 0)
   {
     gera_acao_aleatoria();
     for (d = 0; d < Nesp; d++)
     {
       A[n][d] = a[d];
     }
   }
   if ((n >= N - K) && (n < N - 1))
   {
     for (d = 0; d < Nesp; d++)
     {
       A[n][d] = (int) ceil(Smx[d] * (N - (n + 2)) * 0.25);
     }
   }
   if (n == N - 1)
   {
     time3 = time(null);
     gera_acao_gulosa2();
     timeG2 = time(null) - time3;
     for (d = 0; d < Nesp; d++)
     {
       A[n][d] = a[d];
     }
   }
 }
```

```
 /*-----------------------------------------------------*/
 /*  Limpeza dos registros para o criterio de parada */
 /*-----------------------------------------------------*/

 for (na = 0; na < Na; na++)
 {
   for (d = 0; d < Nesp; d++)
   {
     VCstp[na].b[d]  = Smx[d] + 1;
   }
   VCstp[na].som   = 0;
   VCstp[na].med   = bilhao;
   VCstp[na].N1    = 0;
 }

 /*-----------------------------------------------------*/
 /*       Inicio do processo iterativo                  */
 /*-----------------------------------------------------*/

 t = 0;
 FIM = 0;
 while (FIM == 0)
 {
//Determinacao de uma acao de elite via "parallel rollout":
   t = t + 1;
   printf("iteracao: %d \n", t);
   e = 0;
   naux1 = bilhao;
   for (n = 0; n < N; n++)
   {
     time3 = time(null);
     naux = 0;
     for (c = 0; c < C; c++)
     {
       caux1 = bilhao;
       for (k = 0 ; k < K; k++)
       {
         for (i = 0; i < Npad; i++)
         {
           for (d = 0; d < Nesp; d++)
           {
             xx[i][d] = x[i][d];
           }
         }
         for (d = 0; d < Nesp; d++)
         {
           a[d] = A[n][d];
         }
         gera_proximo_estado_custo();
         caux = CLT;
         if (k < K - 1)
         {
           cont = 0;
           for (l = 0; l < Nrec; l++)
           {
             aux0 = 0;
             for (d = 0; d < Nesp; d++)
             {
               for (i = 0; i < Npad; i++)
               {
                 aux0 = aux0 + (xx[i][d] + RE[d] * PRE[i][d]) * L[d][i][l];
               }
             }
```

```
          if (aux0 > LK[l]) {cont = 1;}
        }
        if (cont == 0)
        {
          for (d = 0; d < Nesp; d++)
          {
            a[d] = (int) ceil(Smx[d] * k * 0.25);
          }
        }
        if (cont == 1)
        {
          for (d = 0; d < Nesp; d++)
          {
            a[d] = 0;
          }
        }
      }
      for (h = 0; h < H; h++)
      {
        if (k == K - 1)
        {
          gera_acao_gulosa2();
        }
        gera_proximo_estado_custo();
        caux = caux + pow(GAMA,(h+1)) * CLT;
      }
      if (caux < caux1)
      {
        caux1 = caux;
      }
    }
    naux = naux + caux1;
  }
  naux = naux / C;
  if (naux < naux1)
  {
    naux1 = naux;
    e = n;
  }
  printf("  tempo: %f\n", time(null) - time3);
}

/*-----------------------------------------------*/
/*     O indice da acao de elite e dado por e    */
/*-----------------------------------------------*/

/*-----------------------------------------------*/
/*     Verificacao do criterio de parada         */
/*-----------------------------------------------*/

g = 0;
f = 0;
while ((g < Na) && (f == 0))
{
  if ( compara_vetor(e, g) == 1)
  {
    ggg = g;
    f = 1;
    VCstp[g].som = VCstp[g].som + naux1;
    VCstp[g].N1++;
    VCstp[g].med = VCstp[g].som / VCstp[g].N1;
    if (VCstp[g].N1 == Stp)
    {
```

```
      FIM = 1;
    }
  }
  g = g + 1;
}
if (f == 0)
{
  g1 = 0;
  for (na = 0; na < Na - 1; na++)
  {
    if (VCstp[g1].med < VCstp[na + 1].med)
    {
      g1 = na + 1;
    }
  }
  for (d = 0; d < Nesp; d++)
  {
    VCstp[g1].b[d] = A[e][d];
  }
  VCstp[g1].som = naux1;
  VCstp[g1].N1  = 1;
  VCstp[g1].med = naux1;

  /*-------------------------------------------------*/
  /* Determinacao de uma nova populacao de acoes     */
  /* via ERPS, partindo da acao de elite A[e],       */
  /* mantendo as acoes das politicas de base,        */
  /* e evitando repeticao de acoes.                  */
  /*-------------------------------------------------*/

  if (FIM == 0)
  {

  /*-------------------------------------------------*/
  /* Incluir as NR acoes com menores custos,         */
  /* componentes do conjunto de acoes em             */
  /* avaliacao no criterio de parada.                */
  /*-------------------------------------------------*/

  raux = 0.0;
  n2 = 0;
  nr = 1;
  while (nr <= Nr && n2 ==0)
  {
    g1 = 0;
    for (na = 0; na < Na - 1; na++)
    {
      if ((VCstp[g1].med > VCstp[na + 1].med) && (VCstp[na + 1].med > raux))
      {g1 = na + 1;}
    }
    if ((VCstp[g1].med == bilhao) || (VCstp[g1].med <= raux))
    {
      n2 = 1;
      nr = nr - 1;
    }
    if ((VCstp[g1].med < bilhao) && (VCstp[g1].med > raux))
    {

      /*-------------------------------------------------*/
      /* Evita repeticao de acoes identicas as acoes     */
      /* das politicas de base                           */
      /*-------------------------------------------------*/
```

```
    f4 = 1;
    for (n1 = (N - K); n1 < N; n1++)
    {
      f5 = 0;
      for (d = 0; d < Nesp; d++)
      {
        if (VCstp[g1].b[d] == A[n1][d]) {f5 = f5 + 1;}
      }
      if (f5 == Nesp) {f4 = 0;}
    }
    if (f4 == 1)
    {
      for (d = 0; d < Nesp; d++)
      {
        A[N - K - nr][d] = VCstp[g1].b[d];
      }
    }
    if (f4 == 0)
    {
      f1 = 0;
      while (f1 == 0)
      {
        f1 = 1;
        gera_ale();
        if (u > Q)
        {
          gera_acao_nearest_neighbor();
          for (d = 0; d < Nesp; d++)
          {
            A[N - K - nr][d] = a[d];
          }
        }
        if (u <= Q)
        {
          gera_acao_aleatoria();
          for (d = 0; d < Nesp; d++)
          {
            A[N - K - nr][d] = a[d];
          }
        }
        for (n1 = (N - K); n1 < N; n1++)
        {
          f3 = 0;
          for (d = 0; d < Nesp; d++)
          {
            if (A[N - K - nr][d] == A[n1][d]) {f3 = f3 + 1;}
          }
          if (f3 == Nesp) {f1 = 0;}
        }
      }
    }
    raux = VCstp[g1].med;
    nr = nr + 1;
  }
}
```

```
/*-------------------------------------------------*/
/* Completar com N - k - nr acoes                  */
/* a nova populacao de acoes.                      */
/*                                                 */
/* A primeira acao e a acao de elite da ultima     */
/* iteracao.                                       */
/*                                                 */
/* Evita-se repeticoes de acoes na nova populacao.*/
/*-------------------------------------------------*/

f2 = 1;
for (d = 0; d < Nesp; d++)
{
  A[0][d] = A[e][d];
}
for (n1 = (N - K - nr); n1 < N; n1++)
{
  f3 = 0;
  for (d = 0; d < Nesp; d++)
  {
    if (A[0][d] == A[n1][d]) {f3 = f3 + 1;}
  }
  if (f3 == Nesp) {f2 = 0;}
}
for (n = f2; n < N - K - nr; n++)
{
  f1 = 0;
  while (f1 == 0)
  {
    f1 = 1;
    gera_ale();
    if (u > Q)
    {
      gera_acao_nearest_neighbor();
      for (d = 0; d < Nesp; d++)
      {
        A[n][d] = a[d];
      }
    }
    if (u <= Q)
    {
      gera_acao_aleatoria();
      for (d = 0; d < Nesp; d++)
      {
        A[n][d] = a[d];
      }
    }
    if (n > 0)
    {
      for (n1 = 0; n1 < n; n1++)
      {
        f3 = 0;
        for (d = 0; d < Nesp; d++)
        {
          if (A[n][d] == A[n1][d]) {f3 = f3 + 1;}
        }
        if (f3 == Nesp) {f1 = 0;}
      }
    }
    if (f1 == 1)
    {
      for (n1 = (N - K - nr); n1 < N; n1++)
      {
```

```
            f3 = 0;
            for (d = 0; d < Nesp; d++)
            {
              if (A[n][d] == A[n1][d]) {f3 = f3 + 1;}
            }
            if (f3 == Nesp) {f1 = 0;}
          }
        }
      }
    }
  }
}

/*-------------------------------------------------*/
/*     “Resultados do processo iterativo.”         */
/*-------------------------------------------------*/

time2 = time(null);
printf("\n");
printf("\n");
printf("struct:\n");
for (n = 0; n < Na; n++)
{
  printf("\n");
  printf("\n");
  printf("posicao n:, %d, \n", n);
  for (d = 0; d < Nesp; d++)
  {
    printf("admitir, %d, pacientes da especialidade, %d \n", VCstp[n].b[d], d);
  }
  printf("VCstp[n].som, %f \n", VCstp[n].som);
  printf("VCstp[n].N1, %d \n", VCstp[n].N1);
  printf("VCstp[n].med, %f \n", VCstp[n].med);
}
printf("\n");
printf("\n");
printf("\n");
printf("Estado inicial do sistema:\n");
for (i = 0; i < Npad - 1; i++)
{
  printf("pad: %d", i);
  printf("\n");
  for (d = 0; d < Nesp; d++)
  {
    printf(" esp: %d pac: %d\n", d, x[i][d]);
  }
}
printf("\n");
printf("Considerando-se um horizonte de planejamento de tamanho: %d\n", H);
printf("\n");
printf("Acao com maior numero de repeticoes:\n");
for (d = 0; d < Nesp; d++)
{
  printf("admitir, %d, pacientes da especialidade %d\n",
  VCstp[ggg].b[d], d);
}
printf("\n");
printf("Custo total descontado estimado de operacao do sistema: %f\n", VCstp[ggg].med);
printf("Repeticoes: %d\n", VCstp[ggg].N1);
printf("Posicao na populacao de acoes   : %d\n", ggg);
printf("\n");
printf("Acao gulosa 1:\n");
for (d = 0; d < Nesp; d++)
```

```
 {
   printf("admitir, %d, pacientes da especialidade %d\n", y[d], d);
 }
 printf("\n");
 printf("Tempo de processamento G1: %f", timeG1);
 printf("\n");
 printf("\n");
 printf("Acao gulosa 2:\n");
 for (d = 0; d < Nesp; d++)
 {
   printf("admitir, %d, pacientes da especialidade %d\n",
   A[N -1][d], d);
 }
 printf("\n");
 printf("Tempo de processamento G2: %f", timeG2);
 printf("\n");
 g1 = 0;
 for (na = 0; na < Na - 1; na++)
 {
   if (VCstp[g1].med > VCstp[na + 1].med)
   {
     g1 = na + 1;
   }
 }
 printf("\n");
 printf("Acao com menor custo:\n");
 for (d = 0; d < Nesp; d++)
 {
   printf("admitir, %d, pacientes da especialidade %d\n", VCstp[g1].b[d], d);
 }
 printf("\n");
 printf("Custo total descontado estimado de operacao do sistema: %f\n", VCstp[g1].med);
 printf("Repeticoes: %d\n", VCstp[g1].N1);
 printf("Posicao na populacao de acoes   : %d", g1);
 printf("\n");
 printf("\n");
 time2 = time2 - time1;
 printf("Tempo de processamento: %f", time2);
 printf("\n");
}
 if (FIM1 == 1)
{
 printf("\n");
 printf("\n");
 printf("\n");
 printf("Estado inicial do sistema:\n");
 for (i = 0; i < Npad - 1; i++)
 {
   printf("pad: %d", i);
   printf("\n");
   for (d = 0; d < Nesp; d++)
   {
     printf(" esp: %d pac: %d\n", d, x[i][d]);
   }
 }
 printf("\n");
 printf("Neste estado, o consumo medio esperado no ultimo periodo");
 printf("\n");
 printf("ultrapassa a capacidade disponivel de pelo menos um recurso.");
 printf("\n");
 printf("A unica acao permitida e:\n");
 for (d = 0; d < Nesp; d++)
 {
```

```
   printf("admitir, %d, pacientes da especialidade %d\n", 0, d);
 }
    printf("\n");
}

getchar();
return (EXIT_SUCCESS);
}

/*---------------------------------------------------------*/
/*            gera_acao_aleatoria                          */
/*---------------------------------------------------------*/

void gera_acao_aleatoria()
{
  int d;
  for (d = 0; d < Nesp; d++)
  {
    gera_ale();
    if (u <1)
    {
      a[d] = (int)floor(( Smx[d] +1) * u);
    }
    if (u == 0.999)
    {
      a[d] = Smx[d];
    }
  }
}

/*---------------------------------------------------------*/
/*                gera numero aleatorio  U (0,1)           */
/*---------------------------------------------------------*/

void gera_ale()
{
  if (Nale >= Nalemx)
  {
    srand((unsigned int)time(NULL));
    Nale = 0;
  }
  u = rand();
  u = u / RAND_MAX;
  if (u == 0.0)
  {
    u = 0.001;
  }
  if (u == 1)
  {
    u = 0.999;
  }
  Nale = Nale + 1;
}

/*---------------------------------------------------------*/
/*           gera_proximo_estado_custo                     */
/*---------------------------------------------------------*/

void gera_proximo_estado_custo()
{
  int    d, i, l, j;
  int    aux, cont, cont1;
  int    xaux[4][2];
```

```
double aux2;

/*-----------------------------------------------------*/
/* Gera uma matriz auxiliar contendo o estado "sem  */
/* pacientes"                                          */
/*-----------------------------------------------------*/

for (d = 0; d < Nesp; d++)
{
  for (i = 0; i < Npad; i++)
  {
    xaux[i][d] = 0;
  }
}

/*-----------------------------------------------------*/
/* Gera, de acordo com uma distribuicao Multinomial,*/
/* o numero esperado de pacientes, de cada          */
/* especialidade em cada padrao de consumo, no      */
/* proximo periodo de funcionamento do sistema      */
/*-----------------------------------------------------*/

for (d = 0; d < Nesp; d++)
{
  for (i = 0; i < Npad - 1; i++)
  {
    aux = xx[i][d];
    cont = 0;
    while (cont < aux)
    {
      j = -1;
      gera_ale();
      if (u == 0) {j = 0;}
      aux2 = 0;
      while (u > aux2)
      {
        j = j + 1;
        aux2 = aux2 + P[d][i][j];
      }
      xaux[j][d] = xaux[j][d] + 1;
      cont = cont + 1;
    }
  }
  aux = a[d];
  cont = 0;
  while (cont < aux)
  {
    j = -1;
    gera_ale();
    if (u == 0) {j = 0;}
    aux2 = 0;
    while (u > aux2)
    {
      j = j + 1;
      aux2 = aux2 + PE[j][d];
    }
    xaux[j][d] = xaux[j][d] + 1;
    cont = cont + 1;
  }
  aux = RE[d];
  cont = 0;
  while (cont < aux)
  {
```

```
    j = -1;
    gera_ale();
    if (u == 0) {j = 0;}
    aux2 = 0;
    while (u > aux2)
    {
      j = j + 1;
      aux2 = aux2 + PRE[j][d];
    }
    xaux[j][d] = xaux[j][d] + 1;
    cont = cont + 1;
  }
}

/*-----------------------------------------------*/
/* Gera custo esperado, dado o estado xx[i,d]    */
/* e a acao a[d]                                 */
/*-----------------------------------------------*/

CLT = 0;
cont1 = 0; m = 0;

/*-----------------------------------------------*/
/* Compara com struct que armazena os estados    */
/* acoes e custos ja gerados. O objetivo desta   */
/* comparacao e acelerar o processo, evintando o */
/* recalculo.                                    */
/*-----------------------------------------------*/

while (cont1 == 0 && m < mm)
{
  if (CC[m].y[0][0] == xx[0][0] && CC[m].y[1][0] == xx[1][0] &&
      CC[m].y[2][0] == xx[2][0] && CC[m].y[3][0] == xx[3][0] &&
      CC[m].y[0][1] == xx[0][1] && CC[m].y[1][1] == xx[1][1] &&
      CC[m].y[2][1] == xx[2][1] && CC[m].y[3][1] == xx[3][1] &&
      CC[m].aa[0] == a[0] && CC[m].aa[1] == a[1])
  {
    CLT = CC[m].custo;
    cont1 = 1;
  }
  m = m + 1;
}
if (cont1 == 0)
{
  auxmax = 0;
  for (d = 0; d < Nesp; d++)
  {
    auxint = 0;
    for (i = 0; i < Npad - 1; i++)
    {
      auxint = auxint + xx[i][d];
    }
    auxint = auxint + a[d] + RE[d];
    if (auxint > auxmax) {auxmax = auxint;}
  }
  auxint = Npad * Nesp;
  for (i = 0; i < auxint; i++)
  {
    npermuta[i] = 0;
  }
  for (d = 0; d < Nesp; d++)
  {
    for (i = 0; i < Npad; i++)
```

```
      {
        V[d][i] = 0;
      }
    }
    custtot = 0; prob1 = 0;
    nn = 0;
    permuta(0);
    CLT = custtot;

   /*-------------------------------------------------------------------------*/
   /*  Alimenta banco de estados, acoes e custos que sera utilizado para  */
   /*  acelerar o processamento do algoritmo                                  */
   /*-------------------------------------------------------------------------*/

    for (d = 0; d < Nesp; d++)
    {
      for (i = 0; i < Npad; i++)
      {
        CC[mm].y[i][d] = xx[i][d];
      }
      CC[mm].aa[d] = a[d];
    }
    CC[mm].custo = custtot;
    mm = mm + 1;
  }

  /*-----------------------------------------------------*/
  /*     Gera uma matriz contendo o novo estado       */
  /*-----------------------------------------------------*/

  for (d = 0; d < Nesp; d++)
  {
    for (i = 0; i < Npad; i++)
    {
      xx[i][d] = xaux[i][d];
    }
  }
}

/*-----------------------------------------------------*/
/*          gera_acao_gulosa2                          */
/*-----------------------------------------------------*/

void gera_acao_gulosa2()
{
  double V[2][4], Vaux[2][4];
  int    d, d1, i, i1, j, l, cont, ma[2];
  double aux, caux, comp, comp1;

  /*-----------------------------------------------------*/
  /* Gera uma acao inicial que nao admite pacientes   */
  /* no proximo período e gera matrizes de estado     */
  /* auxiliar.                                           */
  /*-----------------------------------------------------*/

  for (d = 0; d < Nesp; d++)
  {
    a[d] = 0;
  }
```

```
/*--------------------------------------------------------------------------*/
/* Teste inicial: se o consumo medio estimado do proximo periodo de um dos  */
/* recursos for maior que a quantidade disponivel, a unica acao permitida e */
/* "nao admitir pacientes no proximo periodo".                              */
/*--------------------------------------------------------------------------*/

for (i = 0; i < Npad; i++)
{
  for (d = 0; d < Nesp; d++)
  {
    V[d][i] = 0;
    for (j = 0; j < Npad; j++)
    {
      V[d][i] = V[d][i] + xx[j][d] * P[d][j][i];
    }
    V[d][i] = V[d][i] + RE[d] * PRE[i][d];
  }
}
cont = 0;
for (l = 0; l < Nrec; l++)
{
  aux = 0;
  for (d = 0; d< Nesp; d++)
  {
    for (i = 0; i < Npad; i++)
    {
      aux = aux + V[d][i] * L[d][i][l];
    }
  }
  if (aux > LK[l]) {cont = 1;}
}
if (cont == 0)
{
  i1 = 0;
  while (i1 >= 0)
  {
    i1 = -1;
    for (i = 0; i < Npad; i++)
    {
      for (d1 = 0; d1 < Nesp; d1++)
      {
        Vaux[d1][i] = V[d1][i] + a[d1] * PE[i][d1];
      }
    }
    caux = 0;
    for (l = 0; l < Nrec; l++)
    {
      aux = 0;
      for (d1 = 0; d1 < Nesp; d1++)
      {
        for (i = 0; i < Npad; i++)
        {
          aux = aux + Vaux[d1][i] * L[d1][i][l];
        }
      }
      if (aux < NL[l]) {caux = caux + (NL[l] - aux) * LO[l];}
      if (aux > NL[l]) {caux = caux + (aux - NL[l]) * LE[l];}
      if (aux > LK[l]) {caux = caux + (aux - LK[l]) * LR[l];}
    }
    comp = caux;
    for (d = 0; d < Nesp; d++)
    {
      if (a[d] < Smx[d])
```

```
        {
          for (d1 = 0; d1 < Nesp; d1++)
          {
            if (d1 == d) {ma[d1] = a[d1] + 1;}
            else {ma[d1] = a[d1];}
          }
          for (i = 0; i < Npad; i++)
          {
            for (d1 = 0; d1 < Nesp; d1++)
            {
              Vaux[d1][i] = V[d1][i] + ma[d1] * PE[i][d1];
            }
          }
          caux = 0;
          for (l = 0; l < Nrec; l++)
          {
            aux = 0;
            for (d1 = 0; d1 < Nesp; d1++)
            {
              for (i = 0; i < Npad; i++)
              {
                aux = aux + Vaux[d1][i] * L[d1][i][l];
              }
            }
            if (aux < NL[l]) {caux = caux + (NL[l] - aux) * LO[l];}
            if (aux > NL[l]) {caux = caux + (aux - NL[l]) * LE[l];}
            if (aux > LK[l]) {caux = caux + (aux - LK[l]) * LR[l];}
          }
          comp1 = caux;
          if (comp1 < comp)
          {
            comp = comp1;
            i1 = d;
          }
        }
      }
      if (i1 >= 0)
      {
        for (d1 = 0; d1 < Nesp; d1++)
        {
          if (d1 == i1) {a[d1] = a[d1] + 1;}
        }
      }
    }
  }
}

/*---------------------------------------------------------*/
/*          gera_acao_nearest_neighbor                     */
/*---------------------------------------------------------*/

void gera_acao_nearest_neighbor()
{
  int lmi, lmx, d;
  for (d = 0; d < Nesp; d++)
  {
    lmi = A[e][d] - R[d];
    lmx = A[e][d] + R[d];
    if (lmi < 0)
    {
      lmi = 0;
    }
    if (lmx > Smx[d])
```

```
    {
      lmx = Smx[d];
    }
    gera_ale();
    if (u < 1)
    {
      a[d] = (int)floor(lmi+ (lmx-lmi +1) * u);
    }
    if (u == 0.999)
    {
      a[d] = lmx;
    }
  }
}

/*---------------------------------------------------------*/
/*                   compara_vetor                         */
/*---------------------------------------------------------*/

int compara_vetor(int v1, int v2)
{
  int d;
  for (d = 0; d < Nesp; d++)
  {
    if (A[v1][d] != VCstp[v2].b[d])
    {
      return 0;
    }
  }
  return 1;
}

/*---------------------------------------------------------*/
/*      gera proximos estados sem admissao e               */
/*      quantidade esperada de pacientes                   */
/*      em cada padrao                                     */
/*---------------------------------------------------------*/

void permuta(int nivel)
{
  int    j, i, l, d, cheque, sum, sum1;
  double prob, caux2, cust;

  if (prob1 - 1 >= 0) {return;}
  if (nivel < auxint)
  {
    for (j = 0; j <= auxmax; j++)
    {
      npermuta[nivel] = j;
      permuta(nivel + 1);
    }
  }
  else
  {
    j = 0;
    cheque = 0;
    for (d = 0; d < Nesp; d++)
    {
      sum = 0; sum1 = 0;
      for (i = 0; i < Npad; i++)
      {
        xxx[i][d] = npermuta[j];
        sum = sum + npermuta[j];
```

```
        sum1 = sum1 + xx[i][d];
        j = j + 1;
      }
      sum1 = sum1 + a[d] + RE[d];
      if (sum != sum1 - xx[Npad -1][d] ||
          xxx[0][d] + xxx[1][d] < a[d] ||
          xxx[0][d] + xxx[1][d] + xxx[2][d] < a[d] + RE[d])
      {
        cheque = 1;
      }
    }
    if (cheque < 1)
    {
      prob = gera_probabilidade(a);
      prob1 = prob1 + prob;
      cust = 0;
      for (l = 0; l < Nrec; l++)
      {
        caux2 = 0;
        for (d = 0; d < Nesp; d++)
        {
          for (i = 0; i < Npad; i++)
          {
            caux2 = caux2 + xxx[i][d] * L[d][i][l];
          }
        }
        if (caux2 < NL[l]) {cust = cust + (NL[l] - caux2) * LO[l];}
        if (caux2 > NL[l]) {cust = cust + (caux2 - NL[l]) * LE[l];}
        if (caux2 > LK[l]) {cust = cust + (caux2 - LK[l]) * LR[l];}
      }
      custtot = custtot + cust * prob;
      for (i = 0; i < Npad; i++)
      {
        for (d = 0; d < Nesp; d++)
        {
          V[d][i] = V[d][i] + xxx[i][d] * prob;
        }
      }
    }
  }
}

/*---------------------------------------------------------*/
/*      Fatorial                                           */
/*---------------------------------------------------------*/

double fat(int arg)
{
  double n, Fatx, x;
  if (arg < 0 || arg > 170)
  {
    printf("\n");
    printf("Valor negativo ou muito grande no fatorial, arg", arg);
    printf("\n");
    getchar();
  }
  Fatx = 1;
  x = (double) arg;
  for (n=x; n>1; n--)
  {
    Fatx*=n;
  }
  return Fatx;
```

```
}

/*-------------------------------------------------------*/
/*   gera probabilidade de transicao entre dois estados */
/*-------------------------------------------------------*/

double gera_probabilidade(int ac[2])
{
  int    d, i, y0, y1, y2, y3, y4, x00, x01, x02, x03,
         x10, x11, x12, x13, x20, x21, x22, x23,
         infy0, supy0, infy1, supy1, infy2, supy2, infy3, supy3, infy4, supy4,
         infx00, supx00, infx01, supx01, infx02, supx02, infx03, supx03,
         infx10, supx10, infx11, supx11, infx12, supx12, infx13, supx13,
         infx20, supx20, infx21, supx21, infx22, supx22,
         infx23, supx23, cont;
  double Pesp, Ptot;

 /*-------------------------------------------------------*/
 /*  Determinando os limites inferiores e superiores das */
 /*  somatorias                                           */
 /*-------------------------------------------------------*/

 Ptot = 1;
 for (i = 0; i < Nesp; i++)
 {
   cont = 0;
   Pesp = 0;
   if (xxx[0][i] - (xx[0][i] + xx[1][i] + xx[2][i] + RE[i] - xxx[3][i]) > 0)
   {
     infy0 = xxx[0][i] - (xx[0][i] + xx[1][i] + xx[2][i] + RE[i] - xxx[3][i]);
   }
   else
   {
     infy0 = 0;
   }
   if (xxx[0][i] < ac[i])
   {
     supy0 = xxx[0][i];
   }
   else
   {
     supy0 = ac[i];
   }
   for (y0 = infy0; y0 < supy0 + 1; y0++)
   {
     if (xxx[0][i] - (xx[0][i] + xx[1][i] + xx[2][i] + y0 - xxx[3][i]) > 0)
     {
       infy2 = xxx[0][i] - (xx[0][i] + xx[1][i] + xx[2][i] + y0 - xxx[3][i]);
     }
     else
     {
       infy2 = 0;
     }
     if (xxx[0][i] - y0 < RE[i])
     {
       supy2 = xxx[0][i] - y0;
     }
     else
     {
       supy2 = RE[i];
     }
     for (y2 = infy2; y2 < supy2 + 1; y2++)
     {
```

```
if (xxx[0][i] - (xx[1][i] + xx[2][i] + y0 + y2) > 0)
{
  infx00 = xxx[0][i] - (xx[1][i] + xx[2][i] + y0 + y2);
}
else
{
  infx00 = 0;
}
if (xxx[0][i] - (y0 + y2) < xx[0][i])
{
  supx00 = xxx[0][i] - (y0 + y2);
}
else
{
  supx00 = xx[0][i];
}
for (x00 = infx00; x00 < supx00 + 1; x00++)
{
  if (xxx[0][i] - (xx[2][i] + x00 + y0 + y2) > 0)
  {
    infx10 = xxx[0][i] - (xx[2][i] + x00 + y0 + y2);
  }
  else
  {
    infx10 = 0;
  }
  if (xxx[0][i] - (x00 + y0 + y2) < xx[1][i])
  {
    supx10 = xxx[0][i] - (x00 + y0 + y2);
  }
  else
  {
    supx10 = xx[1][i];
  }
  for (x10 = infx10; x10 < supx10 + 1; x10++)
  {
    if (xxx[0][i] - (x10 + x00 + y0 + y2) > 0)
    {
      infx20 = xxx[0][i] - (x10 + x00 + y0 + y2);
    }
    else
    {
      infx20 = 0;
    }
    if (xxx[0][i] - (x10 + x00 + y0 + y2) < xx[2][i])
    {
      supx20 = xxx[0][i] - (x10 + x00 + y0 + y2);
    }
    else
    {
      supx20 = xx[2][i];
    }
    for (x20 = infx20; x20 < supx20 + 1; x20++)
    {
      y1 = ac[i] - y0;
      if (xxx[1][i] - (xx[0][i] + xx[1][i] + xx[2][i] - x00 - x10 - x20 + y1 –
          xxx[3][i]) > 0)
      {
        infy3 = xxx[1][i] - (xx[0][i] + xx[1][i] + xx[2][i] - x00 - x10 - x20 +
                y1 - xxx[3][i]);
      }
      else
      {
```

```
  infy3 = 0;
}
if (xxx[1][i] - y1 < RE[i] - y2)
{
  supy3 = xxx[1][i] - y1;
}
else
{
  supy3 = RE[i] - y2;
}
for (y3 = infy3; y3 < supy3 + 1; y3++)
{
  if (xxx[1][i] - (xx[1][i] + xx[2][i] - x10 - x20 + y1 + y3) > 0)
  {
    infx01 = xxx[1][i] - (xx[1][i] + xx[2][i] - x10 - x20 + y1 + y3);
  }
  else
  {
    infx01 = 0;
  }
  if (xxx[1][i] - (y1 + y3) < xx[0][i] - x00)
  {
    supx01 = xxx[1][i] - (y1 + y3);
  }
  else
  {
    supx01 = xx[0][i] - x00;
  }
  for (x01 = infx01; x01 < supx01 + 1; x01++)
  {
    if (xxx[1][i] - (xx[2][i] - x20 + x01 + y1 + y3) > 0)
    {
      infx11 = xxx[1][i] - (xx[2][i] - x20 + x01 + y1 + y3);
    }
    else
    {
      infx11 = 0;
    }
    if (xxx[1][i] - (x01 + y1 + y3) < xx[1][i] - x10)
    {
      supx11 = xxx[1][i] - (x01 + y1 + y3);
    }
    else
    {
      supx11 = xx[1][i] - x10;
    }
    for (x11 = infx11; x11 < supx11 + 1; x11++)
    {
      if (xxx[1][i] - (x11 + x01 + y1 + y3) > 0)
      {
        infx21 = xxx[1][i] - (x11 + x01 + y1 + y3);
      }
      else
      {
        infx21 = 0;
      }
      if (xxx[1][i] - (x11 + x01 + y1 + y3) < xx[2][i] - x20)
      {
        supx21 = xxx[1][i] - (x11 + x01 + y1 + y3);
      }
      else
      {
        supx21 = xx[2][i] - x20;
```

```
}
for (x21 = infx21; x21 < supx21 + 1; x21++)
{
  y4 = RE[i] - (y2 + y3);
  if (xxx[2][i] - (xx[1][i] + xx[2][i] - x10 - x11 - x20 - x21 +
       y4) > 0)
  {
    infx02 = xxx[2][i] - (xx[1][i] + xx[2][i] - x10 - x11 - x20 –
             x21 + y4);
  }
  else
  {
    infx02 = 0;
  }
  if (xxx[2][i] - y4 < xx[0][i] - x00 - x01)
  {
    supx02 = xxx[2][i] - y4;
  }
  else
  {
    supx02 = xx[0][i] - x00 - x01;
  }
  for (x02 = infx02; x02 < supx02 + 1; x02++)
  {
    if (xxx[2][i] - (xx[2][i] - x20 - x21 + x02 + y4) > 0)
    {
      infx12 = xxx[2][i] - (xx[2][i] - x20 - x21 + x02 + y4);
    }
    else
    {
      infx12 = 0;
    }
    if (xxx[2][i] - (x02 + y4) < xx[1][i] - x10 - x11)
    {
      supx12 = xxx[2][i] - (x02 + y4);
    }
    else
    {
      supx12 = xx[1][i] - x10 - x11;
    }
    for (x12 = infx12; x12 < supx12 + 1; x12++)
    {
      if (xxx[2][i] - (x02 + x12 + y4) > 0)
      {
        infx22 = xxx[2][i] - (x02 + x12 + y4);
      }
      else
      {
        infx22 = 0;
      }
      if (xxx[2][i] - (x02 + x12 + y4) < xx[2][i] - x20 - x21)
      {
        supx22 = xxx[2][i] - (x02 + x12 + y4);
      }
      else
      {
        supx22 = xx[2][i] - x20 - x21;
      }
      for (x22 = infx22; x22 < supx22 + 1; x22++)
      {
        if (xxx[3][i] - (xx[1][i] + xx[2][i] - x10 - x11 - x12 –
            x20 - x21 - x22) > 0)
        {
```

```
                              infx03 = xxx[3][i] - (xx[1][i] + xx[2][i] - x10 - x11 –
                                       x12 - x20 - x21 - x22);
                            }
                            else
                            {
                              infx03 = 0;
                            }
                            if (xxx[3][i] < xx[0][i] - x00 - x01 - x02)
                            {
                              supx03 = xxx[3][i];
                            }
                            else
                            {
                              supx03 = xx[0][i] - x00 - x01 - x02;
                            }
                            for (x03 = infx03; x03 < supx03 + 1; x03++)
                            {
                              if (xxx[3][i] - (xx[2][i] - x20 - x21 - x22 + x03) > 0)
                              {
                                infx13 = xxx[3][i] - (xx[2][i] - x20 - x21 - x22 + x03);
                              }
                              else
                              {
                                infx13 = 0;
                              }
                              if (xxx[3][i] - x03 < xx[1][i] - x10 - x11 - x12)
                              {
                                supx13 = xxx[3][i] - x03;
                              }
                              else
                              {
                                supx13 = xx[1][i] - x10 - x11 - x12;
                              }
                              for (x13 = infx13; x13 < supx13 + 1; x13++)
                              {
                                if (xxx[3][i] - (x03 + x13) > 0)
                                {
                                  infx23 = xxx[3][i] - (x03 + x13);
                                }
                                else
                                {
                                  infx23 = 0;
                                }
                                if (xxx[3][i] - (x03 + x13) < xx[2][i] - x20 - x21 - x22)
                                {
                                  supx23 = xxx[3][i] - (x03 + x13);
                                }
                                else
                                {
                                  supx23 = xx[2][i] - x20 - x21 - x22;
                                }
                                for (x23 = infx23; x23 < supx23 + 1; x23++)
                                {
Pesp = Pesp + (fat(ac[i])/(fat(y0)*fat(y1)))*
      (pow(PE[0][i],y0)*pow(PE[1][i],y1))*
              (fat(RE[i])/(fat(y2)*fat(y3)*fat(y4)))*
      (pow(PRE[0][i],y2)*pow(PRE[1][i],y3)*pow(PRE[2][i],y4))*
              (fat(xx[0][i])/(fat(x00)*fat(x01)*fat(x02)*fat(x03)))*
      (pow(P[i][0][0],x00)*pow(P[i][0][1],x01)*pow(P[i][0][2],x02)*pow(P[i][0][3],x03))*
              (fat(xx[1][i])/(fat(x10)*fat(x11)*fat(x12)*fat(x13)))*
      (pow(P[i][1][0],x10)*pow(P[i][1][1],x11)*pow(P[i][1][2],x12)*pow(P[i][1][3],x13))*
              (fat(xx[2][i])/(fat(x20)*fat(x21)*fat(x22)*fat(x23)))*
      (pow(P[i][2][0],x20)*pow(P[i][2][1],x21)*pow(P[i][2][2],x22)*pow(P[i][2][3],x23));
```

```
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
    Ptot = Ptot * Pesp;
  }
  return Ptot;
}

/*---------------------------------------------------------*/
/*         gera_acao_gulosa 1                              */
/*---------------------------------------------------------*/

void gera_acao_gulosa1()
{
  double V[2][4];
  int    d, i, j, l, cont, a1, a2, ma[2];
  double aux, mcust;

  /*-----------------------------------------------------*/
  /* Gera uma ação inicial que não admite pacientes   */
  /* no próximo período e gera matrizes de estado     */
  /* auxiliar.                                           */
  /*-----------------------------------------------------*/

  for (d = 0; d < Nesp; d++)
  {
    a[d] = 0;
  }

  /*-------------------------------------------------------------------------*/
  /* Teste inicial: se o consumo medio estimado do proximo periodo de um dos  */
  /* recursos for maior que a quantidade disponivel, a unica acao permitida e */
  /* "nao admitir pacientes no proximo periodo".                              */
  /*-------------------------------------------------------------------------*/

  for (i = 0; i < Npad; i++)
  {
    for (d = 0; d < Nesp; d++)
    {
      V[d][i] = 0;
      for (j = 0; j < Npad; j++)
      {
        V[d][i] = V[d][i] + xx[j][d] * P[d][j][i];
      }
      V[d][i] = V[d][i] + RE[d] * PRE[i][d];
    }
  }
  cont = 0;
  for (l = 0; l < Nrec; l++)
  {
    aux = 0;
```

```
    for (d = 0; d< Nesp; d++)
    {
      for (i = 0; i < Npad; i++)
      {
        aux = aux + V[d][i] * L[d][i][l];
      }
    }
    if (aux > LK[l]) {cont = 1;}
  }
  if (cont == 0)
  {

  /*-------------------------------------------------*/
  /*     Iteracoes para gerar a acao gulosa          */
  /*-------------------------------------------------*/

  mcust = 1000000;
  for (a1 = 0; a1 < Smx[0] + 1; a1++)
  {
    for (a2 = 0; a2 < Smx[1] + 1; a2++)
    {
      a[0] = a1; a[1] = a2;
      auxmax = 0;
      for (d = 0; d < Nesp; d++)
      {
        auxint = 0;
        for (i = 0; i < Npad - 1; i++)
        {
          auxint = auxint + xx[i][d];
        }
        auxint = auxint + a[d] + RE[d];
        if (auxint > auxmax) {auxmax = auxint;}
      }
      auxint = Npad * Nesp;
      for (i = 0; i < auxint; i++)
      {
        npermuta[i] = 0;
      }
      for (d = 0; d < Nesp; d++)
      {
        for (i = 0; i < Npad; i++)
        {
          V[d][i] = 0;
        }
      }
      custtot = 0; prob1 = 0;
      nn = 0;
      permuta(0);
      if (custtot < mcust)
      {
        ma[0] = a[0]; ma[1] = a[1];
        mcust = custtot;
      }
    }
  }
  a[0] = ma[0]; a[1] = ma[1];
}
}
```

```
/*-------------------------------------------------------*/
/*  DeDefinicao do mehra4.h                              */
/*-------------------------------------------------------*/

#ifndef _MEHRA_H
#define _MEHRA_H
/* _Nesp: Numero de especialidades médicas*/
/* ========>>>>>>> PROBLEMAS NO STRUCT    <<<<<<<========== */
#define _Nesp 3
/* _Npad: Numero de padroes de consumo */
#define _Npad 4
/* _Nrec: Numero de recursos */
#define _Nrec 2
/* _N: Tamanho da populacao de acoes a ser avaliada a cada iteracao, N >1 */
#define _N  20
/* Definicao de null... nao presente em C */
#define bilhao 1000000000
#define null 0
#define _Nalemx 100
#define _Na 20
#define _C 40
#define _K 1
#define _H 30
struct Cstp
{
  int    b[3];
  double som;
  double med;
  int    N1;
};
struct Custos
{
  int    y[4][2];
  int    aa[2];
  double custo;
};
#endif
```

B.3 Determinação de ações, via função de recompensa simulada

Dado um estado observado, apresenta-se, a seguir, o código em linguagem C para determinação de uma ação pelos método simulados guloso G1S e EHRA1S (Subseção 4.3.6 do Capítulo 4), considerando o PMD modelado para o controle de admissão de pacientes eletivos (Capítulo 2), com as modificações, as probabilidades e os parâmetros definidos na Seção 4.3 do Capítulo 4.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <time.h>
#include <math.h>
#include "mehra4.h"

/*-------------------------------------------------------*/
/* Criacao e alocacao de valores banais pras variaveis */
/*-------------------------------------------------------*/

int Nesp   = 0;// Numeros de especialidades
int Npad   = 0;// Numero de padroes
int Nrec   = 0;// Numero de recursos
int N      = 0;// Tamanho da populacao de acoes
int Nalemx = 0;         // Quantidade msxima de numeros aleatorios gerados
int Na     = 0;         // Tamanho do vetor de acoes para criterio de parada
int Nr     = 0;         // Acoes com menores custos a serem mantidas na pop.
int C      = 0;         // Numero de amostras de caminhos de funcionamento
int K      = 0;         // Politicas para o metodo parallel rollout
double GAMA   = 0.0;    // Fator de desconto
double Q      = 0.0;    // Probabilidade de busca local ou global
int Stp    = 0;         // Numero de repeticoes de uma acao para parada
int Nale   = 0;         // Contador para o numero de numeros aleatorios
int H      = 0;// Tamanho do horizonte de planejamento
int RA     = 0;// Simulacoes para estimativa do custo em um período

/*-------------------------------------------------------*/
/*        Processo de inicializacao efetivas...         */
/*-------------------------------------------------------*/

/* segundo Processo: Variaveis dependentes - Vetores*/

double    LE[2];// Custo de excesso sobre o nivel desejado para cada recurso
double    LR[2];// Custo de excesso sobre a capacidade maxima p/cada recurso
double    LO[2];// Custo de ociosidade para cada recurso
double    NL[2];// Nivel de consumo desejado para cada recurso
double    LK[2];// Capacidade maxima de consumo de cada recurso
double L[2][4][2];// Consumo: para cada padrao, consumo de cada recurso
int    x[4][2];// Estado: para cada padrao, paciente por especialidade
int    a[2];// Acao: quantos pacientes de cada especialidade
int    y[2];// Acao: quantos pacientes de cada especialidade
int    Smx[2];// Capacidade maxima de admissao para cada especialidade
double P[2][4][4];  // Por especialidade: prob de transicoes entre padroes
double PE[4][2];    // Por padrao: prob de entrada de cada especialidade
double PRE[4][2];    // Por padrao: prob de reentrada de cada especialidade
int    A[40][2];    // Matriz: armazena as acoes de uma populacao de acoes
int    R[2];        // Amplitude para a alteracao de acoes na busca local
struct Cstp VCstp[10]; // Criada em mehra4.h: para o genetico e para a parada
int    xx[4][2];// Auxiliar: mesma definicao que x[4][2]
int    xxx[4][2];// Auxiliar: mesma definicao que x[4][2]
int    RE[2];// Pacientes que retomam seus tratamentos

/*-------------------------------------------------------*/
/*      Variaveis Auxiliares                             */
/*-------------------------------------------------------*/

double CLT;        // Custo
double u;          // Numero aleatorio
int e;             // Acao de elite
```

```
/*---------------------------------------------------------*/
/*             Prototipos                                  */
/*---------------------------------------------------------*/

void gera_acao_aleatoria(); // Gera acao aleatoria para iniciar as interacoes
void gera_ale();            // Gera um numero aleatorio: distribuicao uniforme
void gera_proximo_estado_custo();  // Simulacao: proximo estado e custo
int compara_vetor(int v1, int v2);  // Compara dois vetores
void gera_acao_nearest_neighbor();  // Gera acao na vizinhanca de uma dada acao
void gera_custo_novo();    // Dado um estado e uma acao simula o custo esperado
void gera_acao_gulosa1A();    // Dado um estado, gera uma acao gulosa1 amostrada

/*---------------------------------------------------------*/
/*           main                                          */
/*---------------------------------------------------------*/

int main(int argc, char** argv)
{

int FIM;
int e, cont;
double aux0, naux, naux1, time1, time2, time3, timeG1, timeG2;
double caux, caux1, auxstp, raux;
int n, n1, n2, na, nr;
int c;
int i, j, k, d, h, t, l;
int ggg, g, g1, f, f1, f2, f3, f4, f5;


//Iniciar o Programa:

 time1 = time(null);

/*---------------------------------------------------------*/
/*      Parametros do modelo                               */
/*---------------------------------------------------------*/

Nesp = 2;

Npad = 4;

Nrec = 2;

LE[0] = 1.25; LE[1] = 1.0;

LR[0] = 1.25; LR[1] = 1.0;

LO[0] = 1.6; LO[1] = 1.28;

NL[0] = 7; NL[1] = 13;

LK[0] = 8; LK[1] = 14;

Smx[0] = 2; Smx[1] = 2;

RE[0] = 2; RE[1] = 2;

P[0][0][0] = 0.0160; P[0][0][1] = 0.0080; P[0][0][2] = 0.1820; P[0][0][3] = 0.7940;
P[0][1][0] = 0.0178; P[0][1][1] = 0.0210; P[0][1][2] = 0.0226; P[0][1][3] = 0.9386;
P[0][2][0] = 0.0305; P[0][2][1] = 0.0152; P[0][2][2] = 0.5488; P[0][2][3] = 0.4055;
P[0][3][0] = 0.0000; P[0][3][1] = 0.0000; P[0][3][2] = 0.0000; P[0][3][3] = 1.0000;
```

```
P[1][0][0] = 0.0106; P[1][0][1] = 0.0192; P[1][0][2] = 0.0750; P[1][0][3] = 0.8952;
P[1][1][0] = 0.0360; P[1][1][1] = 0.0420; P[1][1][2] = 0.0220; P[1][1][3] = 0.9000;
P[1][2][0] = 0.0234; P[1][2][1] = 0.0252; P[1][2][2] = 0.3143; P[1][2][3] = 0.6371;
P[1][3][0] = 0.0000; P[1][3][1] = 0.0000; P[1][3][2] = 0.0000; P[1][3][3] = 1.0000;

PE[0][0] = 0.3247; PE[0][1] = 0.4044;
PE[1][0] = 0.6753; PE[1][1] = 0.5956;
PE[2][0] = 0.0000; PE[2][1] = 0.0000;
PE[3][0] = 0.0000; PE[3][1] = 0.0000;

PRE[0][0] = 0.3500; PRE[0][1] = 0.3000;
PRE[1][0] = 0.5610; PRE[1][1] = 0.5470;
PRE[2][0] = 0.0890; PRE[2][1] = 0.1530;
PRE[3][0] = 0.0000; PRE[3][1] = 0.0000;

L[0][0][0] = 1.078; L[0][0][1] = 1.698;
L[0][1][0] = 1.032; L[0][1][1] = 0.000;
L[0][2][0] = 0.000; L[0][2][1] = 3.643;
L[0][3][0] = 0.000; L[0][3][1] = 0.000;

L[1][0][0] = 1.118; L[1][0][1] = 1.286;
L[1][1][0] = 1.057; L[1][1][1] = 0.000;
L[1][2][0] = 0.000; L[1][2][1] = 2.454;
L[1][3][0] = 0.000; L[1][3][1] = 0.000;

/*---------------------------------------------------------*/
/*      Parametros do EHRA                                 */
/*---------------------------------------------------------*/

H = 4;

C = 4;

K = 1;

GAMA = 0.8;

N = 4;

Na = 5;

Nr = 1;

Q = 0.9;

R[0] = 1; R[1] = 1;

Stp = 16;

Nalemx = 30000;

srand((unsigned int)time(null));

Nale = 0;

RA = 50000;
```

```
/*--------------------------------------------------------------------------*/
/*         Estado observado                                                 */
/*--------------------------------------------------------------------------*/

x[0][0] = 1; x[0][1] = 1;
x[1][0] = 1; x[1][1] = 1;
x[2][0] = 1; x[2][1] = 1;
x[3][0] = 0; x[3][1] = 0;

/*--------------------------------------------------------------------------*/
/*     Gera acao EHRA1                                                      */
/*--------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------*/
/*  Teste primario: se o consumo medio estimado do proximo periodo de um dos*/
/*  recursos for maior que a quantidade disponivel, a unica acao permitida  */
/*  e "nao admitir pacientes no proximo periodo". E o programa se encerra.  */
/*--------------------------------------------------------------------------*/

 for (d = 0; d < Nesp; d++)
 {
    for (i = 0; i < Npad; i++)
    {
       xx[i][d] = x[i][d];
    }
 }


/*-----------------------------------------------------*/
/*      Geracao de uma populacao inicial               */
/*      Mantendo as ações das políticas de base        */
/*-----------------------------------------------------*/

for (n = 0; n < N; n++ )
{
   if (n < N - K && n > 0)
   {
      gera_acao_aleatoria();
      for (d = 0; d < Nesp; d++)
      {
         A[n][d] = a[d];
      }
   }
   if (n == N - 1)
   {
      time3 = time(null);
      gera_acao_gulosa1A();
      timeG2 = time(null) - time3;
      for (d = 0; d < Nesp; d++)
      {
         A[n][d] = a[d];
      }
   }
}

/*-----------------------------------------------------*/
/*  Limpeza dos registros para o criterio de parada */
/*-----------------------------------------------------*/

for (na = 0; na < Na; na++)
{
   for (d = 0; d < Nesp; d++)
   {
```

```
      VCstp[na].b[d]  = Smx[d] + 1;
   }
   VCstp[na].som   = 0;
   VCstp[na].med   = bilhao;
   VCstp[na].N1    = 0;
}

/*-------------------------------------------------*/
/*       Inicio do processo iterativo              */
/*-------------------------------------------------*/

t = 0;
FIM = 0;
while (FIM == 0)
{
   //Determinacao de uma acao de elite via "paralellel rollout":
   t = t + 1;
   printf("iteracao: %d \n", t);
   e = 0;
   naux1 = bilhao;
   for (n = 0; n < N; n++)
   {
      time3 = time(null);
      naux = 0;
      for (c = 0; c < C; c++)
      {
         caux1 = bilhao;
         for (k = 0 ; k < K; k++)
         {
            for (i = 0; i < Npad; i++)
            {
               for (d = 0; d < Nesp; d++)
               {
                  xx[i][d] = x[i][d];
               }
            }
            for (d = 0; d < Nesp; d++)
            {
               a[d] = A[n][d];
            }
            gera_proximo_estado_custo();
            caux = CLT;
            for (h = 0; h < H; h++)
            {
               gera_acao_gulosa1A();
               gera_proximo_estado_custo();
               caux = caux + pow(GAMA,(h+1)) * CLT;
            }
            if (caux < caux1)
            {
               caux1 = caux;
            }
         }
         naux = naux + caux1;
      }
      naux = naux / C;
      if (naux < naux1)
      {
         naux1 = naux;
         e = n;
      }
      printf("  tempo: %f\n", time(null) - time3);
   }
```

```
/*-----------------------------------------------*/
/*     O índice da acao de elite e dado por e    */
/*-----------------------------------------------*/


/*-----------------------------------------------*/
/*     Verificacao do criterio de parada         */
/*-----------------------------------------------*/

g = 0;
f = 0;
while ((g < Na) && (f == 0))
{
   if ( compara_vetor(e, g) == 1)
   {
      ggg = g;
      f = 1;
      VCstp[g].som = VCstp[g].som + naux1;
      VCstp[g].N1++;
      VCstp[g].med = VCstp[g].som / VCstp[g].N1;
      if (VCstp[g].N1 == Stp)
      {
         FIM = 1;
      }
   }
   g = g + 1;
}
if (f == 0)
{
   g1 = 0;
   for (na = 0; na < Na - 1; na++)
   {
      if (VCstp[g1].med < VCstp[na + 1].med)
      {
         g1 = na + 1;
      }
   }
   for (d = 0; d < Nesp; d++)
   {
      VCstp[g1].b[d] = A[e][d];
   }
   VCstp[g1].som = naux1;
   VCstp[g1].N1  = 1;
   VCstp[g1].med = naux1;
}

/*-----------------------------------------------*/
/* Determinação de uma nova populacao de acoes   */
/* via ERPS, partindo da acao de elite A[e],     */
/* mantendo as acoes das políticas de base,      */
/* e evitando repeticao de acoes.                */
/*-----------------------------------------------*/

if (FIM == 0)
{

 /*-----------------------------------------------*/
 /* Incluir as NR acoes com menores custos,       */
 /* componentes do conjunto de acoes em           */
 /* avaliacao no criterio de parada.              */
 /*-----------------------------------------------*/
```

```
raux = 0.0;
n2 = 0;
nr = 1;
while (nr <= Nr && n2 ==0)
{
   g1 = 0;
   for (na = 0; na < Na - 1; na++)
   {
      if ((VCstp[g1].med > VCstp[na + 1].med) &&
          (VCstp[na + 1].med > raux)) {g1 = na + 1;}
   }
   if ((VCstp[g1].med == bilhao) ||
       (VCstp[g1].med <= raux))
   {
      n2 = 1;
      nr = nr - 1;
   }
   if ((VCstp[g1].med < bilhao) &&
       (VCstp[g1].med > raux))
   {

      /*-------------------------------------------------*/
      /*  Evita repeticao de acoes identicas as acoes   */
      /*  das politicas de base                          */
      /*-------------------------------------------------*/

      f4 = 1;
      for (n1 = (N - K); n1 < N; n1++)
      {
         f5 = 0;
         for (d = 0; d < Nesp; d++)
         {
            if (VCstp[g1].b[d] == A[n1][d]) {f5 = f5 + 1;}
         }
         if (f5 == Nesp) {f4 = 0;}
      }
      if (f4 == 1)
      {
         for (d = 0; d < Nesp; d++)
         {
            A[N - K - nr][d] = VCstp[g1].b[d];
         }
      }
      if (f4 == 0)
      {
         f1 = 0;
         while (f1 == 0)
         {
            f1 = 1;
            gera_ale();
            if (u > Q)
            {
               gera_acao_nearest_neighbor();
               for (d = 0; d < Nesp; d++)
               {
                  A[N - K - nr][d] = a[d];
               }
            }
            if (u <= Q)
            {
               gera_acao_aleatoria();
               for (d = 0; d < Nesp; d++)
               {
```

```
                  A[N - K - nr][d] = a[d];
               }
            }
            for (n1 = (N - K); n1 < N; n1++)
            {
               f3 = 0;
               for (d = 0; d < Nesp; d++)
               {
                  if (A[N - K - nr][d] == A[n1][d]) {f3 = f3 + 1;}
               }
               if (f3 == Nesp) {f1 = 0;}
            }
         }
      }
      raux = VCstp[g1].med;
      nr = nr + 1;
   }
}

/*-------------------------------------------------*/
/* Completar com N - K - nr acoes                  */
/* a nova populacao de acoes.                      */
/*                                                 */
/* A primeira acao e a ação de elite da ultima     */
/* iteracao.                                       */
/*                                                 */
/* Evita-se repeticoes de acoes na nova populacao.*/
/*-------------------------------------------------*/

f2 = 1;
for (d = 0; d < Nesp; d++)
{
   A[0][d] = A[e][d];
}
for (n1 = (N - K - nr); n1 < N; n1++)
{
   f3 = 0;
   for (d = 0; d < Nesp; d++)
   {
      if (A[0][d] == A[n1][d]) {f3 = f3 + 1;}
   }
   if (f3 == Nesp) {f2 = 0;}
}
for (n = f2; n < N - K - nr; n++)
{
   f1 = 0;
   while (f1 == 0)
   {
      f1 = 1;
      gera_ale();
      if (u > Q)
      {
         gera_acao_nearest_neighbor();
         for (d = 0; d < Nesp; d++)
         {
            A[n][d] = a[d];
         }
      }
      if (u <= Q)
      {
         gera_acao_aleatoria();
         for (d = 0; d < Nesp; d++)
         {
```

```
                  A[n][d] = a[d];
               }
            }
            if (n > 0)
            {
               for (n1 = 0; n1 < n; n1++)
               {
                  f3 = 0;
                  for (d = 0; d < Nesp; d++)
                  {
                     if (A[n][d] == A[n1][d]) {f3 = f3 + 1;}
                  }
                  if (f3 == Nesp) {f1 = 0;}
               }
            }
            if (f1 == 1)
            {
               for (n1 = (N - K - nr); n1 < N; n1++)
               {
                  f3 = 0;
                  for (d = 0; d < Nesp; d++)
                  {
                     if (A[n][d] == A[n1][d]) {f3 = f3 + 1;}
                  }
                  if (f3 == Nesp) {f1 = 0;}
               }
            }
         }
      }
   }
}

/*-------------------------------------------------*/
/*     “Resultados do processo iterativo.”         */
/*-------------------------------------------------*/

 time2 = time(null);
    printf("\n");
    printf("\n");
    printf("struct:\n");
    for (n = 0; n < Na; n++)
    {
       printf("\n");
       printf("\n");
       printf("posicao n:, %d, \n", n);
       for (d = 0; d < Nesp; d++)
       {
          printf("admitir, %d, pacientes da especialidade, %d \n",
                  VCstp[n].b[d], d);
       }
       printf("VCstp[n].som, %f \n", VCstp[n].som);
       printf("VCstp[n].N1, %d \n", VCstp[n].N1);
       printf("VCstp[n].med, %f \n", VCstp[n].med);
    }
    printf("\n");
    printf("Estado inicial do sistema:\n");
    for (i = 0; i < Npad - 1; i++)
    {
       printf("pad: %d", i);
       printf("\n");
       for (d = 0; d < Nesp; d++)
       {
          printf(" esp: %d pac: %d\n", d, x[i][d]);
```

```
      }
   }
   printf("\n");
   printf("Considerando-se um horizonte de planejamento: %d\n", H);
   printf("\n");
   printf("Acao com maior numero de repeticoes:\n");
   for (d = 0; d < Nesp; d++)
   {
      printf("admitir, %d, pacientes da especialidade %d\n",
              VCstp[ggg].b[d], d);
   }
   printf("\n");
   printf("Custo total descontado estimado de operacao do sistema: %f\n",
              VCstp[ggg].med);
   printf("Repeticoes: %d\n", VCstp[ggg].N1);
   printf("Posicao na populacao de acoes   : %d\n", ggg);
   printf("\n");
   printf("Acao gulosa 1A:\n");
   for (d = 0; d < Nesp; d++)
   {
      printf("admitir, %d, pacientes da especialidade %d\n", A[N-1][d], d);
   }
   printf("\n");
   printf("Tempo de processamento G1A: %f", timeG2);
   printf("\n");
   printf("\n");
   g1 = 0;
   for (na = 0; na < Na - 1; na++)
   {
      if (VCstp[g1].med > VCstp[na + 1].med)
      {
         g1 = na + 1;
      }
   }
   printf("\n");
   printf("Acao com menor custo:\n");
   for (d = 0; d < Nesp; d++)
   {
      printf("admitir, %d, pacientes da especialidade %d\n", VCstp[g1].b[d], d);
   }
   printf("\n");
   printf("Custo total descontado estimado de operacao do sistema em H: %f\n",
              VCstp[g1].med);
   printf("Repeticoes: %d\n", VCstp[g1].N1);
   printf("Posicao na populacao de acoes   : %d", g1);
   printf("\n");
   printf("\n");
   time2 = time2 - time1;
   printf("Tempo de processamento: %f", time2);
   printf("\n");

 getchar();

 return (EXIT_SUCCESS);
}

/*-------------------------------------------------------*/
/*           gera_acao_aleatoria                         */
/*-------------------------------------------------------*/

void gera_acao_aleatoria()
{
   int d;
```

```
   for (d = 0; d < Nesp; d++)
   {
      gera_ale();
      if (u <1)
      {
         a[d] = (int)floor(( Smx[d] +1) * u);
      }
      if (u == 0.999)
      {
         a[d] = Smx[d];
      }
   }
}

/*-------------------------------------------------------*/
/*           gera numero aleatorio  U (0,1)              */
/*-------------------------------------------------------*/

void gera_ale()
{
   if (Nale >= Nalemx)
   {
      srand((unsigned int)time(NULL));
      Nale = 0;
   }
   u = rand();
   u = u / RAND_MAX;
   if (u == 0.0)
   {
      u = 0.001;
   }
   if (u == 1)
   {
      u = 0.999;
   }
   Nale = Nale + 1;
}

/*-------------------------------------------------------*/
/*           gera_proximo_estado_custo                   */
/*-------------------------------------------------------*/

void gera_proximo_estado_custo()
{
int      d, i, l, j;
int      aux, cont, cont1;
int      xaux[4][2];
double   aux2;

/*-------------------------------------------------*/
/* Gera uma matriz auxiliar contendo o estado "sem  */
/* pacientes"                                       */
/*-------------------------------------------------*/

   for (d = 0; d < Nesp; d++)
   {
      for (i = 0; i < Npad; i++)
      {
         xaux[i][d] = 0;
      }
   }
```

```
/*-------------------------------------------------*/
/* Gera, de acordo com uma distribuicao Multinomial,*/
/* o numero esperado de pacientes, de cada          */
/* especialidade em cada padrao de consumo, no      */
/* proximo periodo de funcionamento do sistema      */
/*-------------------------------------------------*/

   for (d = 0; d < Nesp; d++)
   {
      for (i = 0; i < Npad - 1; i++)
      {
         aux = xx[i][d];
         cont = 0;
         while (cont < aux)
         {
            j = -1;
            gera_ale();
            if (u == 0) {j = 0;}
            aux2 = 0;
            while (u > aux2)
            {
               j = j + 1;
               aux2 = aux2 + P[d][i][j];
            }
            xaux[j][d] = xaux[j][d] + 1;
            cont = cont + 1;
         }
      }
      aux = a[d];
      cont = 0;
      while (cont < aux)
      {
         j = -1;
         gera_ale();
         if (u == 0) {j = 0;}
         aux2 = 0;
         while (u > aux2)
         {
            j = j + 1;
            aux2 = aux2 + PE[j][d];
         }
         xaux[j][d] = xaux[j][d] + 1;
         cont = cont + 1;
      }
      aux = RE[d];
      cont = 0;
      while (cont < aux)
      {
         j = -1;
         gera_ale();
         if (u == 0) {j = 0;}
         aux2 = 0;
         while (u > aux2)
         {
            j = j + 1;
            aux2 = aux2 + PRE[j][d];
         }
         xaux[j][d] = xaux[j][d] + 1;
```

```
         cont = cont + 1;
      }
   }


/*-----------------------------------------------*/
/* Gera custo esperado, dado o estado xx[i,d]    */
/* e a ação a[d]                                 */
/*-----------------------------------------------*/

   gera_custo_novo();

/*-------------------------------------------------*/
/*     Gera uma matriz contendo o novo estado      */
/*-------------------------------------------------*/

   for (d = 0; d < Nesp; d++)
   {
      for (i = 0; i < Npad; i++)
      {
         xx[i][d] = xaux[i][d];
      }
   }
}

/*---------------------------------------------------------*/
/*          gera_acao_nearest_neighbor                     */
/*---------------------------------------------------------*/

void gera_acao_nearest_neighbor()
{
int lmi, lmx;
int d;
   for (d = 0; d < Nesp; d++)
   {
      lmi = A[e][d] - R[d];
      lmx = A[e][d] + R[d];
      if (lmi < 0)
      {
         lmi = 0;
      }
      if (lmx > Smx[d])
      {
         lmx = Smx[d];
      }
      gera_ale();
      if (u < 1)
      {
         a[d] = (int)floor(lmi+ (lmx-lmi +1) * u);
      }
      if (u == 0.999)
      {
         a[d] = lmx;
      }
   }
}

/*---------------------------------------------------------*/
/*                   compara_vetor                         */
/*---------------------------------------------------------*/

int compara_vetor(int v1, int v2)
{
```

```
   int d;
   for (d = 0; d < Nesp; d++)
   {
      if (A[v1][d] != VCstp[v2].b[d])
      {
         return 0;
      }
   }
   return 1;
}

/*---------------------------------------------------------*/
/*             gera_custo_novo                             */
/*---------------------------------------------------------*/

void gera_custo_novo()
{
 int      d, i, l, j, k;
 int      cont;
 int      xaux[4][2];
 double   aux, aux2, caux;

/*-----------------------------------------------------*/
/*  Gera uma matriz auxiliar contendo o estado "sem */
/*  pacientes"                                         */
/*-----------------------------------------------------*/

   CLT = 0;
   for (k = 0; k < RA; k++)
   {
      for (d = 0; d < Nesp; d++)
      {
         for (i = 0; i < Npad; i++)
         {
            xaux[i][d] = 0;
         }
      }

/*-----------------------------------------------------*/
/* Gera, de acordo com uma distribuicao Multinomial,*/
/* o numero esperado de pacientes, de cada          */
/* especialidade em cada padrao de consumo, no      */
/* proximo periodo de funcionamento do sistema      */
/*-----------------------------------------------------*/

      for (d = 0; d < Nesp; d++)
      {
         for (i = 0; i < Npad - 1; i++)
         {
            aux = xx[i][d];
            cont = 0;
            while (cont < aux)
            {
               j = -1;
               gera_ale();
               if (u == 0) {j = 0;}
               aux2 = 0;
               while (u > aux2)
               {
                  j = j + 1;
                  aux2 = aux2 + P[d][i][j];
               }
               xaux[j][d] = xaux[j][d] + 1;
```

```
               cont = cont + 1;
            }
         }
         aux = a[d];
         cont = 0;
         while (cont < aux)
         {
            j = -1;
            gera_ale();
            if (u == 0) {j = 0;}
            aux2 = 0;
            while (u > aux2)
            {
               j = j + 1;
               aux2 = aux2 + PE[j][d];
            }
            xaux[j][d] = xaux[j][d] + 1;
            cont = cont + 1;
         }
         aux = RE[d];
         cont = 0;
         while (cont < aux)
         {
            j = -1;
            gera_ale();
            if (u == 0) {j = 0;}
            aux2 = 0;
            while (u > aux2)
            {
               j = j + 1;
               aux2 = aux2 + PRE[j][d];
            }
            xaux[j][d] = xaux[j][d] + 1;
            cont = cont + 1;
         }
      }

    /*-----------------------------------------------*/
    /* Gera custo, dado o estado xaux[i,d]           */
    /* e a ação a[d]                                 */
    /*-----------------------------------------------*/

      caux = 0;
      for (l = 0; l < Nrec; l++)
      {
         aux = 0;
         for (d = 0; d < Nesp; d++)
         {
            for (i = 0; i < Npad; i++)
            {
               aux = aux + xaux[i][d] * L[d][i][l];
            }
         }
         if (aux < NL[l]) {caux = caux + (NL[l] - aux) * LO[l];}
         if (aux > NL[l]) {caux = caux + (aux - NL[l]) * LE[l];}
         if (aux > LK[l]) {caux = caux + (aux - LK[l]) * LR[l];}
      }
      CLT = CLT + caux;
   }
   CLT = CLT / RA;
}
```

```
/*-------------------------------------------------------*/
/*     gera_acao_gulosa1A amostrada                      */
/*-------------------------------------------------------*/

void gera_acao_gulosa1A()
{
  int d, i, j, l, cont, a1, a2, ma[2];
  double  aux, mcust;

/*-------------------------------------------------*/
/* Gera uma ação inicial que não admite pacientes  */
/* no próximo período e gera matrizes de estado    */
/* auxiliar.                                       */
/*-------------------------------------------------*/

   for (d = 0; d < Nesp; d++)
   {
      a[d] = 0;
   }
   mcust = 1000000;
   for (a1 = 0; a1 < Smx[0] + 1; a1++)
   {
      for (a2 = 0; a2 < Smx[1] + 1; a2++)
      {
         a[0] = a1; a[1] = a2;
         gera_custo_novo();
         if (CLT < mcust)
         {
            ma[0] = a[0]; ma[1] = a[1];
            mcust = CLT;
         }
      }
   }
   a[0] = ma[0]; a[1] = ma[1];
   CLT = mcust;
}




/*-------------------------------------------------------*/
/*  DeDefinicao do mehra4.h                              */
/*-------------------------------------------------------*/

#ifndef _MEHRA_H
#define _MEHRA_H
/* _Nesp: Numero de especialidades médicas*/
/* ========>>>>>>> PROBLEMAS NO STRUCT    <<<<<<<========== */
/* Definicao de null... nao presente em C */
#define bilhao 1000000000
#define null 0
#define _Nalemx 100
struct Cstp
{
  int    b[3];
  double som;
  double med;
  int    N1;
};
#endif
```